User Guide

AWS CloudHSM

AWS CloudHSM: User Guide

Table of Contents

What is AWS CloudHSM?

AWS CloudHSM combines the benefits of the AWS cloud with the security of hardware security modules (HSMs). A hardware security module (HSM) is a computing device that processes cryptographic operations and provides secure storage for cryptographic keys. With AWS CloudHSM, you have complete control over high availability HSMs that are in the AWS Cloud, have low-latency access, and a secure root of trust that automates HSM management (including backups, provisioning, configuration, and maintenance).

AWS CloudHSM offers customers a variety of benefits:

Access to FIPS and non-FIPS clusters

AWS CloudHSM offers clusters in two modes: *FIPS* and *non-FIPS*. In FIPS mode, only Federal Information Processing Standard (FIPS) validated keys and algorithms can be used. Non-FIPS mode offers all the keys and algorithms that are supported by AWS CloudHSM, regardless of FIPS approval. For more information, see AWS CloudHSM cluster modes.

HSMs are general purpose, single tenant, and either FIPS 140-2 level-3 or FIPS 140-3 level-3 validated for clusters in FIPS mode

AWS CloudHSM uses general purpose HSMs that provide more flexibility when compared to the fully-managed AWS services that have predetermined algorithms and key lengths for your application. We offer HSMs that are standards-compliant, single-tenant, and are either FIPS 140-2 level-3 or FIPS 140-3 level-3 validated for clusters in FIPS mode. For customers with use cases outside the restrictions of FIPS 140-2 or FIPS 140-3 level-3 validation, AWS CloudHSM also offers clusters in non-FIPS mode. See AWS CloudHSM clusters for more information.

E2E encryption is not visible to AWS

Because your data plane is end-to-end (E2E) encrypted and not visible to AWS, you control your own user management (outside of IAM roles). The trade off for this control is you have more responsibility than if you used a managed AWS service.

Full control of your keys, algorithms, and application development

AWS CloudHSM gives you full control of the algorithms and keys you use. You can generate, store, import, export, manage, and use cryptographic keys (including, session keys, token keys, symmetric keys and asymmetric key pairs). Additionally, AWS CloudHSM SDKs give you full control over application development, application language, threading, and where your applications physically exist.

Migrate your cryptographic workloads to the cloud

Customers migrating public key infrastructure that use Public Key Cryptography Standards #11 (PKCS #11), Java Cryptographic Extension (JCE), Cryptography API: Next Generation (CNG), or Key Storage Provider (KSP) can migrate to AWS CloudHSM with fewer changes to their application.

To learn more about what you can do with AWS CloudHSM, see the following topics. When you are ready to get started with AWS CloudHSM, see Getting started.

> **Note**
>
> If you want a managed service for creating and controlling your encryption keys but you don't want or need to operate your own HSMs, consider using AWS Key Management Service.
> If you are looking for an elastic service that manages payment HSMs and keys for payment processing applications in the cloud, consider using AWS Payment Cryptography.

Contents

AWS CloudHSM use cases

AWS CloudHSM can be used to accomplish a variety of goals. The content in this topic provides an overview of what you can do with AWS CloudHSM.

Achieve regulatory compliance

Businesses that need to align with enterprise security standards can use AWS CloudHSM to manage private keys that protect highly confidential data. The HSMs provided by AWS CloudHSM are FIPS 140-2 level 3 certified and comply with PCI DSS. Additionally, AWS CloudHSM is PCI PIN compliant and PCI-3DS compliant. For more information, see Compliance.

Encrypt and decrypt data

Use AWS CloudHSM to manage private keys that protect highly confidential data, encryption in transit, and encryption at rest. Additionally, AWS CloudHSM offers standards-compliant integration with multiple cryptographic SDKs.

Sign and verify documents with private and public keys

In cryptography, using a private key to **sign** a document allows recipients to use a public key to **verify** that you (and not someone else) actually sent the document. Use AWS CloudHSM to create asymmetric public and private key pairs that are specifically designed for this purpose.

Authenticate messages using HMACs and CMACs

In cryptography, Cipher Message Authentication Codes (CMACs) and Hash-based Message Authentication Codes (HMACs) are used to authenticate and ensure the integrity of messages sent over unsafe networks. With AWS CloudHSM, you can securely create and manage symmetric keys that support HMACs and CMACs.

Leverage the benefits of AWS CloudHSM and AWS Key Management Service

Customers can combine AWS CloudHSM and AWS KMS to store key material in a single-tenant environment while also getting the key management, scaling, and cloud integration benefits of AWS KMS. For details on how to do this, see AWS CloudHSM key stores in the *AWS Key Management Service Developer Guide*.

Offload SSL/TLS processing for web servers

To securely send data over the internet, web servers use public–private key pairs and SSL/TLS public key certificate to establish HTTPS sessions. This process involves a lot of computation for web servers, but you can reduce computational burden while providing extra security by offloading some of this to your AWS CloudHSM cluster. For information about setting up SSL/TLS offload with AWS CloudHSM, see SSL/TLS offload.

Enable transparent data encryption (TDE)

Transparent Data Encryption (TDE) is used to encrypt database files. Using TDE, database software encrypts data before storing it on disk. You can achieve greater security by storing the

TDE master encryption key in HSMs in your AWS CloudHSM. For information about setting up Oracle TDE with AWS CloudHSM, see Oracle database encryption.

Manage the private keys of an issuing certificate authority (CA)

A certificate authority (CA) is a trusted entity that issues digital certificates that bind a public key to an identity (a person or organization). To operate a CA, you must maintain trust by protecting the private key that signs certificates issued by your CA. You can store such private keys in your AWS CloudHSM cluster and then use your HSMs to perform cryptographic signing operations.

Generate random numbers

Generating random numbers to create encryption keys is core to online security. AWS CloudHSM can be used to securely generate random numbers in HSMs you control and are only visible to you.

How AWS CloudHSM works

This topic provides an overview of the basic concepts and architecture you use to securely encrypt data and perform cryptographic operations in HSMs. AWS CloudHSM operates in your own Amazon Virtual Private Cloud (VPC). Before you can use AWS CloudHSM, you first create a cluster, add HSMs to it, create users and keys, and then use Client SDKs to integrate your HSMs with your application. Once this is done, you use Client SDK logs, AWS CloudTrail, audit logs, and Amazon CloudWatch to monitor AWS CloudHSM.

Learn AWS CloudHSM's basic concepts and how they work together to help protect your data.

Topics

- AWS CloudHSM clusters
- Users in AWS CloudHSM
- Keys in AWS CloudHSM
- Client SDKs for AWS CloudHSM
- AWS CloudHSM cluster backups
- Supported Regions for AWS CloudHSM

AWS CloudHSM clusters

Making individual HSMs work together in a synchronized, redundant, and highly-available way can be difficult, but AWS CloudHSM does the heavy lifting for you by providing hardware security modules (HSMs) in *clusters*. A cluster is a collection of individual HSMs that AWS CloudHSM keeps in sync. When you perform a task or operation on one HSM in a cluster, the other HSMs in that cluster are automatically kept up to date.

AWS CloudHSM offers clusters in two modes: *FIPS* and *non-FIPS*. In FIPS mode, only Federal Information Processing Standard (FIPS) validated keys and algorithms can be used. Non-FIPS mode offers all the keys and algorithms that are supported by AWS CloudHSM, regardless of FIPS approval. AWS CloudHSM also offers two types of HSMs: *hsm1.medium* and *hsm2m.medium*. For details on the differences between each HSM type and cluster mode, see AWS CloudHSM cluster modes. The *hsm1.medium* HSM type is reaching end of support so new clusters cannot be created with this type. For more information, see Deprecation notifications for details.

To meet your availability, durability, and scalability goals, you set the number of HSMs in your cluster across multiple availability zones. You can create a cluster that has 1 to 28 HSMs (the default limit is 6 HSMs per AWS account per AWS Region). You can place the HSMs in different Availability Zones in an AWS region. Adding more HSMs to a cluster provides higher performance. Spreading clusters across Availability Zones provides redundancy and high availability.

For more information about clusters, see Clusters in AWS CloudHSM.

To create a cluster, see Getting started.

Users in AWS CloudHSM

Unlike most AWS services and resources, you do not use AWS Identity and Access Management (IAM) users or IAM policies to access resources within your AWS CloudHSM cluster. Instead, you use *HSM users* directly on HSMs in your AWS CloudHSM cluster.

HSM users are distinct from IAM users. IAM users who have the correct credentials can create HSMs by interacting with resources through the AWS API. Since E2E encryption is not visible to AWS, you must use HSM user credentials to authenticate operations on the HSM because credentials takes place directly on the HSM. The HSM authenticates each HSM user by means of credentials that you define and manage. Each HSM user has a *type* that determines which operations that user can perform on the HSM. Each HSM authenticates each HSM user by means of credentials that you define using CloudHSM CLI.

If you are using the previous SDK version series, then you will use CloudHSM Management Utility (CMU).

Keys in AWS CloudHSM

AWS CloudHSM allows you to securely generate, store, and manage your encryption keys in single-tenant HSMs that are in your AWS CloudHSM cluster. Keys can be symmetric or asymmetric, can be session keys (ephemeral keys) for single sessions, token keys (persistent keys) for long-term use, and can be exported from and imported into AWS CloudHSM. Keys can also be used to complete common cryptographic tasks and functions:

- Perform cryptographic data signing and signature verification with both symmetric and asymmetric encryption algorithms.
- Work with hash functions to compute message digests and hash-based message authentication codes (HMACs).
- Wrap and protect other keys.
- Access cryptographically secure random data.

The maximum keys a cluster can have depends on the type of HSMs that are in the cluster. For example, hsm2m.medium stores more keys than hsm1,medium. For a comparison, see AWS CloudHSM quotas.

Additionally, AWS CloudHSM follows a few foundational principles for key usage and management:

Many key types and algorithms to choose from

To allow you to customize your own solutions, AWS CloudHSM provides many key types and algorithms to choose from. algorithms support a range of key sizes. For more information, refer to the attributes and mechanism pages of each Offload operations with AWS CloudHSM Client SDKs.

How you manage keys

AWS CloudHSM keys are managed through SDKs and command line tools. For information on how to use these tools to manage keys, see Keys in AWS CloudHSM and Best practices for AWS CloudHSM.

Who owns keys

In AWS CloudHSM, the crypto user (CU) who creates the key owns it. The owner can use the **key share** and **key unshare** commands to share and unshare the key with other CUs. For more information, see Share and unshare keys using CloudHSM CLI.

Access and usage can be controlled with attribute-based encryption

AWS CloudHSM allows you to use attribute-based encryption, a form of encryption that lets you use key attributes to control who can decrypt data based on policies.

Client SDKs for AWS CloudHSM

When using AWS CloudHSM, you perform cryptographic operations with AWS CloudHSM Client Software Development Kits (SDKs). AWS CloudHSM Client SDKs include:

- Public Key Cryptography Standards #11 (PKCS #11)
- JCE provider
- OpenSSL Dynamic Engine
- Key Storage Provider (KSP) for Microsoft Windows

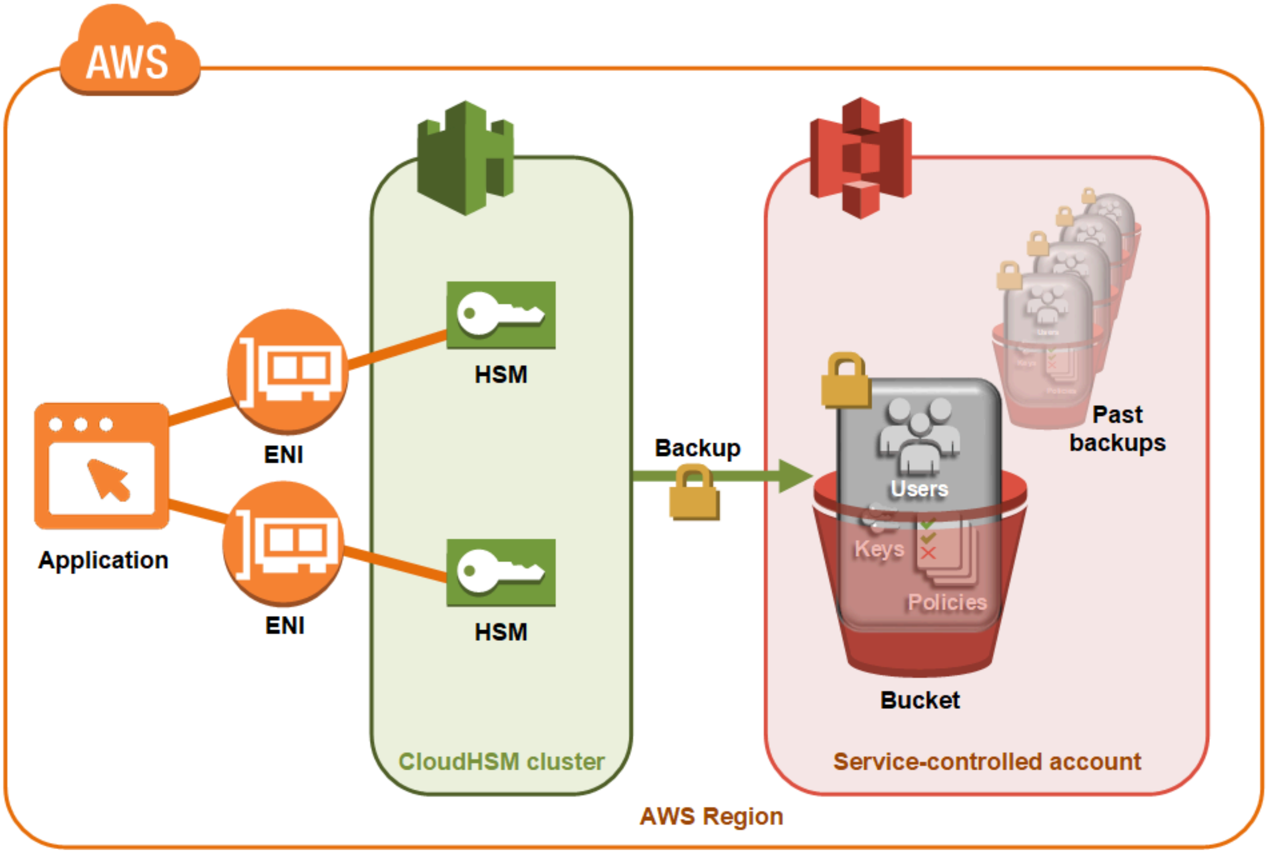
You can use any or all of these SDKS in your AWS CloudHSM cluster. Write your application code to use these SDKs to perform cryptographic operations in your HSMs. To see what platforms and HSM types support each SDK, see AWS CloudHSM Client SDK 5 supported platforms

Utility and command line tools are needed not only to use SDKs but also to configure the credentials, policies, and settings of your application. For more information, refer to AWS CloudHSM command line tools.

For more information about installing and using the Client SDK or the security of the client connection, see Client SDKs and End-to-end encryption.

AWS CloudHSM cluster backups

AWS CloudHSM makes periodic backups of the users, keys, and policies in the cluster. Backups are secure, durable, and updated on a predictable schedule. The following illustration shows the relationship of your backups to the cluster.



For more information about working with backups, see Cluster backups.

Security

When AWS CloudHSM makes a backup from the HSM, the HSM encrypts all of its data before sending it to AWS CloudHSM. The data never leaves the HSM in plaintext form. Additionally, backups cannot be decrypted by AWS because AWS doesn't have access to key used to decrypt the backups. For more information, see Security of cluster backups

Durability

AWS CloudHSM stores backups in a service-controlled Amazon Simple Storage Service (Amazon S3) bucket in the same region as your cluster. Backups have a 99.999999999% durability level, the same as any object stored in Amazon S3.

Supported Regions for AWS CloudHSM

For information about the supported Regions for AWS CloudHSM, see AWS CloudHSM Regions and Endpoints in the *AWS General Reference*, or the Region Table.

AWS CloudHSM might not be available in all Availability Zones in a given Region. However, this should not affect performance, as AWS CloudHSM automatically load balances across all HSMs in a cluster.

Like most AWS resources, clusters and HSMs are regional resources. You cannot reuse or extend a cluster across Regions. You must perform all the required steps listed in Getting started with AWS CloudHSM to create a cluster in a new Region.

For disaster recovery purposes, AWS CloudHSM allows you to copy backups of your AWS CloudHSM Cluster from one region to another. For more information, see AWS CloudHSM cluster backups.

Pricing for AWS CloudHSM

With AWS CloudHSM, you pay by the hour with no long-term commitments or upfront payments. For more information, see AWS CloudHSM Pricing on the AWS website.

Getting started with AWS CloudHSM

The following topics help you create, initialize, and activate a cluster in AWS CloudHSM. After you complete these procedures, you'll be ready to manage users, manage clusters, and use the included software libraries to perform cryptographic operations. For the best experience, follow the topics in the listed order.

Contents

Create IAM administrative groups for AWS CloudHSM

The first step to getting started with AWS CloudHSM is to set up IAM permissions.

As a best practice, don't use your AWS account root user to interact with AWS, including AWS CloudHSM. Instead, use AWS Identity and Access Management (IAM) to create an IAM user, IAM role, or federated user. Follow the steps in the section Create an IAM user and administrator group to create an administrator group and attach the **AdministratorAccess** policy to it. Then create a new administrator user and add the user to the group. Add additional users to the group as needed. Each user you add inherits the **AdministratorAccess** policy from the group.

Another best practice is to create an AWS CloudHSM administrator group that has only the permissions required to run AWS CloudHSM. Add individual users to this group as needed. Each user inherits the limited permissions that are attached to the group rather than full AWS access. The Customer managed policies for AWS CloudHSM section that follows contains the policy that you should attach to your AWS CloudHSM administrator group.

AWS CloudHSM defines a service–linked role for your AWS account. The service–linked role currently defines permissions that allow your account to log AWS CloudHSM events. The role can be created automatically by AWS CloudHSM or manually by you. You cannot edit the role, but you can delete it. For more information, see Service-linked roles for AWS CloudHSM.

Create an IAM user and administrator group

Start by creating an IAM user along with an administrator group for that user.

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

  For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.

2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

For example policies for AWS CloudHSM that you can attach to your IAM user group, see Identity and access management for AWS CloudHSM.

Create a virtual private cloud (VPC) for AWS CloudHSM

You need a virtual private cloud (VPC) for your cluster in AWS CloudHSM. If you don't already have one, follow the steps in this topic to create a VPC.

> **Note**
>
> Following these steps will result in the creation of public and private subnets.

To create a VPC

1. Open the Amazon VPC console at https://console.aws.amazon.com/vpc/.
2. On the navigation bar, use the region selector to choose one of the AWS Regions where AWS CloudHSM is currently supported.
3. Select the **Create VPC** button.
4. For **Resources to create**, choose **VPC and more**.
5. For **Name tag auto-generation**, type an identifiable name such as **`CloudHSM`**.
6. For **IPv6 CIDR block**, select **Amazon-provided IPv6 CIDR block** to use IPv6 connectivity for your HSMs and have AWS allocate an IPv6 CIDR block for your cluster. This setting supports the dual-stack Network Type. Keep the default setting if you don't need IPv6 connectivity.
7. Leave all other options set to their defaults.
8. Choose **Create VPC**.
9. After the VPC is created, select **View VPC** to view the VPC you just created.

Create a cluster in AWS CloudHSM

A cluster is a collection of individual hardware security modules (HSMs). AWS CloudHSM synchronizes the HSMs in each cluster so that they function as a logical unit. AWS CloudHSM offers two types of HSMs: *hsm1.medium* and *hsm2m.medium*. When you create a cluster, you choose which of the two will be in your cluster. For details on the differences between each HSM type and cluster mode, see AWS CloudHSM cluster modes.

When you create a cluster, AWS CloudHSM creates a security group for the cluster on your behalf. This security group controls network access to the HSMs in the cluster. It allows inbound connections only from Amazon Elastic Compute Cloud (Amazon EC2) instances that are in the security group. By default, the security group doesn't contain any instances. Later, you launch a client instance and configure the cluster's security group to allow communication and connections with the HSM.

Important

When you create a cluster, AWS CloudHSM creates a service-linked role named AWSServiceRoleForCloudHSM. If AWS CloudHSM cannot create the role or the role does not already exist, you may not be able to create a cluster. For more information, see Resolving AWS CloudHSM cluster creation failures. For more information about service–linked roles, see Service-linked roles for AWS CloudHSM.

Important

If you are using the AWS CloudHSM dual-stack endpoint (that is, cloudhsmv2.`<region>`.api.aws), ensure that your IAM policies are updated to handle IPv6. For more information, see the Upgrade IAM policies to IPv6 section under Security.

You can create a cluster from the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.

Note

For details on cluster arguments and APIs, see **create-cluster** in the AWS CLI Command Reference.

Console

To create a cluster (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. On the navigation bar, use the region selector to choose one of the AWS Regions where AWS CloudHSM is currently supported.
3. Choose **Create cluster**.
4. In the **Cluster configuration** section, do the following:

    a. For **VPC**, select the VPC that you created in Create a virtual private cloud (VPC) for AWS CloudHSM.

    b. For **Availability Zone(s)**, next to each Availability Zone, choose the private subnet that you created.

    > **Note**
    >
    > Even if AWS CloudHSM is not supported in a given Availability Zone, performance should not be affected, as AWS CloudHSM automatically load balances across all HSMs in a cluster. See AWS CloudHSM Regions and Endpoints in the *AWS General Reference* to see Availability Zone support for AWS CloudHSM.

    c. For **HSM type**, select the HSM type that can be created in your cluster along with the desired mode of the cluster. To see what HSM types are supported in each region, see the AWS CloudHSM pricing calculator.

    > **Important**
    >
    > After the cluster is created, the cluster mode cannot be changed. For information on which type and mode is right for your use case, see AWS CloudHSM cluster modes.

    d. For **Network Type**, choose the IP address protocols for accessing your HSMs. IPv4 limits communication between your application and HSMs to IPv4 only. This is the default option. Dual-stack enables both IPv4 and IPv6 communication. To use dual-stack, add both IPv4 and IPv6 CIDRs to your VPC and subnet configurations. The Network Type is difficult to change after initial setup. To modify it, create a backup of your existing cluster and restore a new cluster with the desired Network Type. For more information, see Creating AWS CloudHSM clusters from backups

e. For **Cluster source**, specify whether you want to create a new cluster or restore one from an existing backup.

- Backups of clusters in non-FIPS mode can only be used to restore clusters that are in non-FIPS mode.
- Backups of clusters in FIPS mode can only be used to restore clusters that are in FIPS mode.

5. Choose **Next**.
6. Specify how long the service should retain backups.

> **Note**
>
> Accept the default retention period of 90 days or type a new value between 7 and 379 days. The service will automatically delete backups in this cluster older than the value you specify here. You can change this later. For more information, see Configure backup retention.

7. Choose **Next**.
8. (Optional) Type a tag key and an optional tag value. To add more than one tag to the cluster, choose **Add tag**.
9. Choose **Review**.
10. Review your cluster configuration, and then choose **Create cluster**.

If your attempts to create a cluster fail, it might be related to problems with the AWS CloudHSM service-linked roles. For help on resolving the failure, see Resolving AWS CloudHSM cluster creation failures.

AWS CLI

To create a cluster (AWS CLI)

- At a command prompt, run the **create-cluster** command. Specify the HSM instance type, the backup retention period, and the subnet IDs of the subnets where you plan to create HSMs. Use the subnet IDs of the private subnets that you created. Specify only one subnet per Availability Zone.

```
$ aws cloudhsmv2 create-cluster --hsm-type hsm2m.medium \
                    --backup-retention-policy Type=DAYS,Value=<number of days> \
                    --subnet-ids <subnet ID> \
                    --mode <FIPS> \
                    --network-type <IPV4>

{
    "Cluster": {
        "BackupPolicy": "DEFAULT",
        "BackupRetentionPolicy": {
            "Type": "DAYS",
            "Value": 90
         },
        "VpcId": "vpc-50ae0636",
        "SubnetMapping": {
            "us-west-2b": "subnet-49a1bc00",
            "us-west-2c": "subnet-6f950334",
            "us-west-2a": "subnet-fd54af9b"
        },
        "SecurityGroup": "sg-6cb2c216",
        "HsmType": "hsm2m.medium",
        "NetworkType": "IPV4",
        "Certificates": {},
        "State": "CREATE_IN_PROGRESS",
        "Hsms": [],
        "ClusterId": "cluster-igklspoyj5v",
        "ClusterMode": "FIPS",
        "CreateTimestamp": 1502423370.069
    }
}
```

Note

`ClusterMode` is a required parameter for all hsm types except hsm1.medium.--mode:

```
$ aws cloudhsmv2 create-cluster --hsm-type hsm2m.medium \
      --backup-retention-policy Type=DAYS,Value=<number of days> \
      --subnet-ids <subnet ID> \
    --mode NON_FIPS
```

If your attempts to create a cluster fail, it might be related to problems with the AWS CloudHSM service-linked roles. For help on resolving the failure, see Resolving AWS CloudHSM cluster creation failures.

AWS CloudHSM API

To create a cluster (AWS CloudHSM API)

- Send a CreateCluster request. Specify the HSM instance type, the backup retention policy, and the subnet IDs of the subnets where you plan to create HSMs. Use the subnet IDs of the private subnets that you created. Specify only one subnet per Availability Zone.

If your attempts to create a cluster fail, it might be related to problems with the AWS CloudHSM service-linked roles. For help on resolving the failure, see Resolving AWS CloudHSM cluster creation failures.

Review the security group for your cluster in AWS CloudHSM

When you create a cluster, AWS CloudHSM creates a security group with the name `cloudhsm-cluster-<clusterID>-sg`. This security group contains a preconfigured TCP rule that allows inbound and outbound communication within the cluster security group over ports 2223-2225. This SG allows your EC2 instances to use your VPC to talk to HSMs in your cluster.

Warning

- Do not delete or modify the preconfigured TCP rule, which is populated in the cluster security group. This rule can prevent connectivity issues and unauthorized access to your HSMs.
- The cluster security group prevents unauthorized access to your HSMs. Anyone that can access instances in the security group can access your HSMs. Most operations require a user to log in to the HSM. However, it's possible to zeroize HSMs without authentication, which destroys the key material, certificates, and other data. If this happens, data created or modified after the most recent backup is lost and unrecoverable. To prevent unauthorized access, ensure that only trusted administrators can modify or access the instances in the default security group.

- The hsm2m.medium clusters introduces mTLS feature to restrict unauthorized users from connecting to the cluster. Unauthorized users will require a valid mTLS credentials to successfully connect to cluster before attempting zeroization.

In the next step, you can launch an Amazon EC2 instance and connect it to your HSMs by attaching the cluster security group to it.

Launch an Amazon EC2 client instance for interacting with AWS CloudHSM

To interact with and manage your AWS CloudHSM cluster and HSM instances, you must be able to communicate with the elastic network interfaces of your HSMs. The easiest way to do this is to use an EC2 instance in the same VPC as your cluster. You can also use the following AWS resources to connect to your cluster:

- Amazon VPC Peering
- AWS Direct Connect
- VPN Connections

Note

This guide provides a simplified example of how to connect an EC2 instance to your AWS CloudHSM cluster. For best practices around secure network configurations, refer to Secure access to your cluster.

The AWS CloudHSM documentation typically assumes that you are using an EC2 instance in the same VPC and Availability Zone (AZ) in which you create your cluster.

To create an EC2 instance

1. Open the **EC2 Dashboard** at https://console.aws.amazon.com/ec2/.
2. Select **Launch instance**. From the drop-down menu, choose **Launch instance**.
3. In the **Name** field, enter a name for your EC2 instance.

4. In the **Applications and OS Images (Amazon Machine Image)** section, choose an Amazon Machine Image (AMI) that corresponds to a platform CloudHSM supports. For more information, see AWS CloudHSM Client SDK 5 supported platforms.
5. In the **Instance Type** section, choose an instance type.
6. In the **Key pair** section, use an existing key pair or select **Create new key pair** and complete the following steps:
   a. For **Key pair name**, enter a name for the key pair.
   b. For **Key pair type**, choose a key pair type.
   c. For **Private key file format**, choose the private key file format.
   d. Select **Create key pair**.
   e. Download and save the private key file.

   > **Important**
   >
   > This is your only chance to save the private key file. Download and store the file in a safe place. You must provide the name of your key pair when you launch an instance. Additionally, you must provide the corresponding private key each time you connect to the instance and choose the key pair that you created when setting up.

7. In **Network settings**, select **Edit**.
8. For **VPC**, choose the VPC that you previously created for your cluster.
9. For **Subnet**, choose the public subnet that you created for the VPC.
10. For **Auto-assign Public IP**, choose **Enable**.
11. For **Auto-assign IPv6 IP**, choose **Enable** to use IPv6 connectivity with your clusters and the Dual-stack NetworkType. If you enable this option, update your Amazon EC2 instance's security group rules, VPC and subnet route tables, and network ACLs to allow IPv6 outbound traffic from the instance to the HSMs.
12. Choose **Select an existing security group**.
13. In **Common security groups**, select the default security group from the drop-down menu.
14. In **Configure Storage**, use the drop-down menus to choose a storage configuration.
15. In the **Summary** window, select **Launch instance**.

> **Note**
>
> Completing this step will start the process for creating your EC2 instance.

For more information about creating a Linux Amazon EC2 client, see Getting Started with Amazon EC2 Linux Instances. For information about connecting to the running client, see the following topics:

- Connecting to Your Linux Instance Using SSH
- Connecting to Your Linux Instance from Windows Using PuTTY

The Amazon EC2 user guide contains detailed instructions for setting up and using your Amazon EC2 instances. The following list provides an overview of available documentation for Linux and Windows Amazon EC2 clients:

- To create a Linux Amazon EC2 client, see Getting Started with Amazon EC2 Linux Instances.

  For information about connecting to the running client, see the following topics:
  - Connecting to your Linux Instance Using SSH
  - Connecting to Your Linux Instance from Windows Using PuTTY
- To create a Windows Amazon EC2 client, see Getting Started with Amazon EC2 Windows Instances. For more information about connecting to your Windows client, see Connect to Your Windows Instance.

> **Note**
>
> Your EC2 instance can run all of the AWS CLI commands contained in this guide. If the AWS CLI is not installed, you can download it from AWS Command Line Interface. If you are using Windows, you can download and run a 64-bit or 32-bit Windows installer. If you are using Linux or macOS, you can install the CLI using pip.

Configure the Client Amazon EC2 instance security groups for AWS CloudHSM

When you launched an Amazon EC2 instance for your cluster in AWS CloudHSM, you associated it with a default Amazon VPC security group. This topic explains how to associate the cluster security group with the EC2 instance. This association allows the AWS CloudHSM client running on your EC2 instance to communicate with your HSMs. To connect your EC2 instance to your AWS CloudHSM cluster, you must properly configure the VPC default security group *and* associate the cluster security group with the instance.

Use the following steps to complete the configuration changes.

Topics

Step 1. Modify the default security group

You need to modify the default security group to permit the SSH or RDP connection so that you can download and install client software, and interact with your HSM.

To modify the default security group

1. Open the **EC2 Dashboard** at https://console.aws.amazon.com/ec2/.
2. Select **Instances (running)** and then select the check box next to the EC2 instance you want to install the AWS CloudHSM client.
3. Under the **Security** tab, choose the security group named **Default**.
4. At the top of the page, choose **Actions**, and then **Edit Inbound Rules**.
5. Select **Add Rule**.
6. For **Type**, do one of the following:
   - For a Windows Server Amazon EC2 instance, choose **RDP**. The port 3389 is automatically populated.
   - For a Linux Amazon EC2 instance, choose **SSH**. The port range 22 is automatically populated.

7. For either option, set **Source** to **My IP** to allow you to communicate with your Amazon EC2 instance.

> **Important**
>
> Do not specify 0.0.0.0/0 as the CIDR range to avoid allowing anyone to access your instance.

8. Choose **Save**.

Step 2. Connect the Amazon EC2 instance to the AWS CloudHSM cluster

You must attach the cluster security group to the EC2 instance so that the EC2 instance can communicate with HSMs in your cluster. The cluster security group contains a preconfigured rule that allows inbound communication over ports 2223-2225.

To connect the EC2 instance to the AWS CloudHSM cluster

1. Open the **EC2 Dashboard** at https://console.aws.amazon.com/ec2/.
2. Select **Instances (running)** and then select the check box for the EC2 instance on which you want to install the AWS CloudHSM client.
3. At the top of the page, choose **Actions**, **Security**, and then **Change Security Groups**.
4. Select the security group with the group name that matches your cluster ID, such as `cloudhsm-cluster-<clusterID>-sg`.
5. Choose **Add Security Groups**.
6. Select **Save**.

> **Note**
>
> You can assign a maximum of five security groups to an Amazon EC2 instance. If you have reached the maximum limit, you must modify the default security group of the Amazon EC2 instance and the cluster security group:
>
> In the default security group, do the following:
>
> - Add an inbound rule to permit traffic using the TCP protocol over ports `2223-2225` from the cluster security group.

In the cluster security group, do the following:

- Add an inbound rule to permit traffic using the TCP protocol over ports `2223-2225` from the default security group.

Create an HSM in AWS CloudHSM

After you create a cluster in AWS CloudHSM, you can create a hardware security module (HSM). However, before you can create an HSM in your cluster, the cluster must be in the uninitialized state. To determine the cluster's state, view the clusters page in the AWS CloudHSM console, use the AWS CLI to run the **describe-clusters** command, or send a DescribeClusters request in the AWS CloudHSM API. You can create an HSM from the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.

Console

To create an HSM (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Select the radio button next to the ID of the cluster you want to create an HSM for.
3. Select **Actions**. From the drop down menu, choose **Initialize**.
4. Choose an Availability Zone (AZ) for the HSM that you are creating.
5. Select **Create**.

After you create a cluster and HSM, you can optionally verify the identity of the HSM, or proceed directly to Initialize the cluster.

AWS CLI

To create an HSM (AWS CLI)

- At a command prompt, run the **create-hsm** command. Specify the cluster ID of the cluster that you created previously and an Availability Zone for the HSM. Specify the Availability Zone in the form of `us-west-2a`, `us-west-2b`, etc.

```
$ aws cloudhsmv2 create-hsm --cluster-id <cluster ID> --availability-
zone <Availability Zone>

{
    "Hsm": {
        "HsmId": "hsm-ted36yp5b2x",
        "EniIp": "10.0.1.12",
        "EniIpV6": "2600:113f:404:be09:310e:ed34:3412:f733",
        "AvailabilityZone": "us-west-2a",
        "ClusterId": "cluster-igklspoyj5v",
        "EniId": "eni-5d7ade72",
        "SubnetId": "subnet-fd54af9b",
        "State": "CREATE_IN_PROGRESS"
    }
}
```

After you create a cluster and HSM, you can optionally verify the identity of the HSM, or proceed directly to Initialize the cluster.

AWS CloudHSM API

To create an HSM (AWS CloudHSM API)

- Send a CreateHsm request. Specify the cluster ID of the cluster that you created previously and an Availability Zone for the HSM.

After you create a cluster and HSM, you can optionally verify the identity of the HSM, or proceed directly to Initialize the cluster.

Verify the identity and authenticity of your cluster's HSM in AWS CloudHSM (optional)

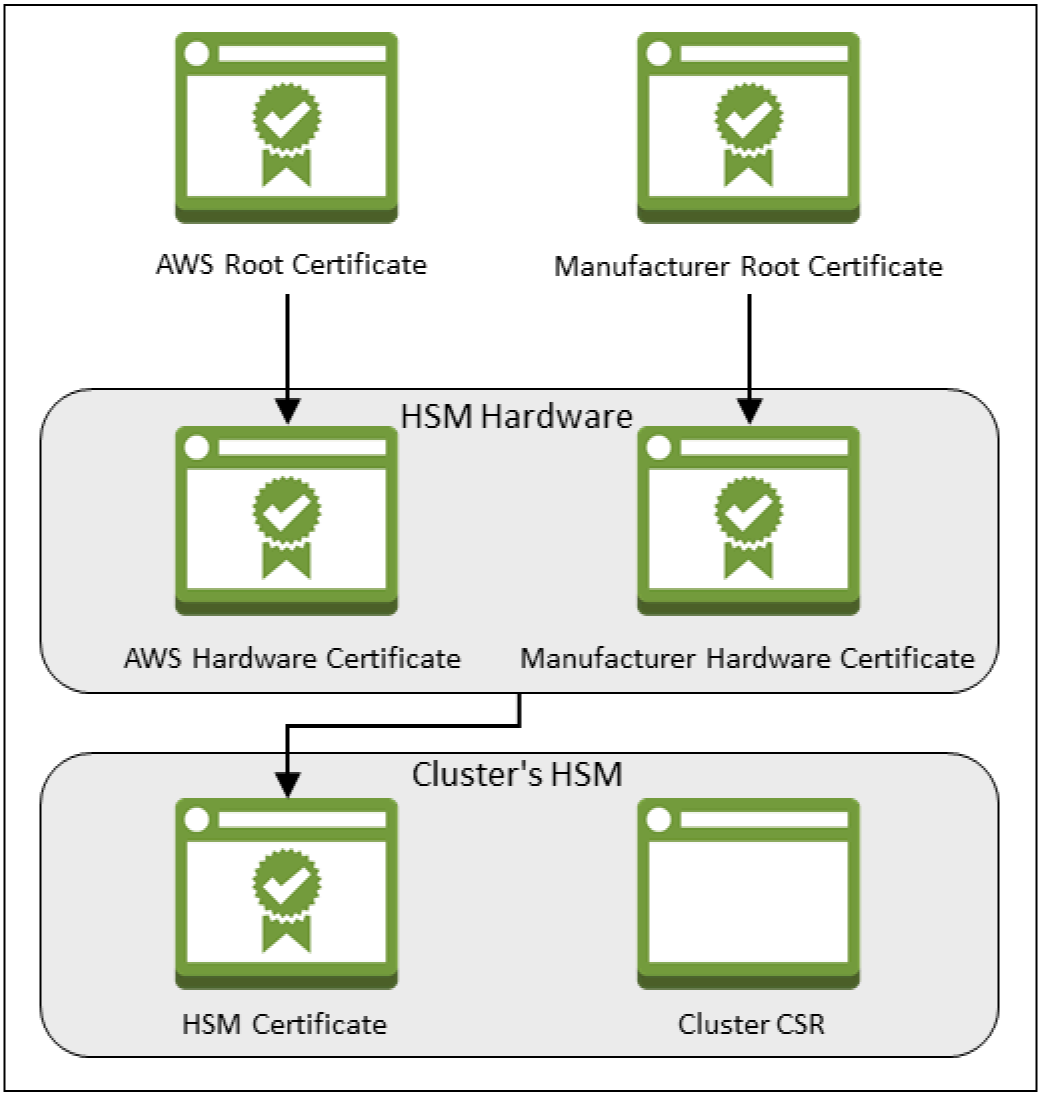
To initialize your cluster in AWS CloudHSM, you sign a certificate signing request (CSR) generated by the cluster's first hardware security module (HSM). Before you do this, you might want to verify the identity and authenticity of the HSM.

> **Note**
>
> This process is optional. However, it works only until a cluster is initialized. After the cluster is initialized, you cannot use this process to get the certificates or verify the HSMs.

To verify the identity of your cluster's first HSM, complete the following steps:

1. Get the certificates and CSR – In this step, you get three certificates and a CSR from the HSM. You also get two root certificates, one from AWS CloudHSM and one from the HSM hardware manufacturer.
2. Verify the certificate chains – In this step, you construct two certificate chains, one to the AWS CloudHSM root certificate and one to the manufacturer root certificate. Then you verify the HSM certificate with these certificate chains to determine that AWS CloudHSM and the hardware manufacturer both attest to the identity and authenticity of the HSM.
3. Compare public keys – In this step, you extract and compare the public keys in the HSM certificate and the cluster CSR, to ensure that they are the same. This should give you confidence that the CSR was generated by an authentic, trusted HSM.

The following diagram shows the CSR, the certificates, and their relationship to each other. The subsequent list defines each certificate.



AWS Root Certificate

This is AWS CloudHSM's root certificate.

Manufacturer Root Certificate

This is the hardware manufacturer's root certificate.

AWS Hardware Certificate

AWS CloudHSM created this certificate when the HSM hardware was added to the fleet. This certificate asserts that AWS CloudHSM owns the hardware.

Manufacturer Hardware Certificate

The HSM hardware manufacturer created this certificate when it manufactured the HSM hardware. This certificate asserts that the manufacturer created the hardware.

HSM Certificate

The HSM certificate is generated by the FIPS-validated hardware when you create the first HSM in the cluster. This certificate asserts that the HSM hardware created the HSM.

Cluster CSR

The first HSM creates the cluster CSR. When you sign the cluster CSR, you claim the cluster. Then, you can use the signed CSR to initialize the cluster.

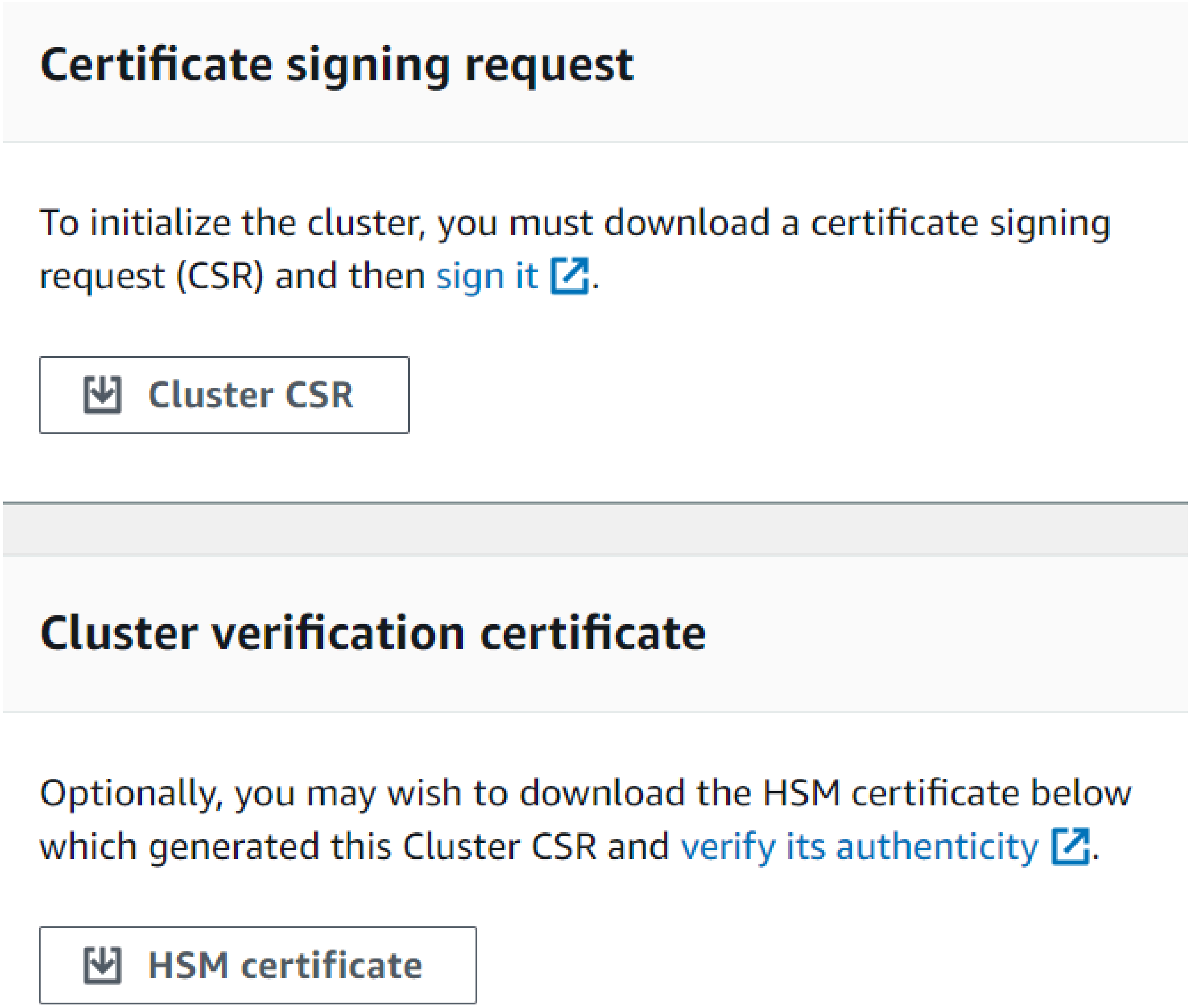
Step 1. Get certificates from the HSM

To verify the identity and authenticity of your HSM, start by getting a CSR and five certificates. You get three of the certificates from the HSM, which you can do with the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.

Console

To get the CSR and HSM certificates (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Select the radio button next to the cluster ID with the HSM you want to verify.
3. Select **Actions**. From the drop down menu, choose **Initialize**.
4. If you did not complete the previous step to create an HSM, choose an Availability Zone (AZ) for the HSM that you are creating. Then select **Create**.
5. When the certificates and CSR are ready, you see links to download them.



6. Choose each link to download and save the CSR and certificates. To simplify the subsequent steps, save all of the files to the same directory and use the default file names.

AWS CLI

To get the CSR and HSM certificates ([AWS CLI](#))

- At a command prompt, run the **[describe-clusters](#)** command four times, extracting the CSR and different certificates each time and saving them to files.

  a. Issue the following command to extract the cluster CSR. Replace *<cluster ID>* with the ID of the cluster that you created previously.

```
$ aws cloudhsmv2 describe-clusters --filters clusterIds=<cluster ID> \
                                   --output text \
                                   --query
 'Clusters[].Certificates.ClusterCsr' \
                                   > <cluster ID>_ClusterCsr.csr
```

b. Issue the following command to extract the HSM certificate. Replace `<cluster ID>` with the ID of the cluster that you created previously.

```
$ aws cloudhsmv2 describe-clusters --filters clusterIds=<cluster ID> \
                                   --output text \
                                   --query
 'Clusters[].Certificates.HsmCertificate' \
                                   > <cluster ID>_HsmCertificate.crt
```

c. Issue the following command to extract the AWS hardware certificate. Replace `<cluster ID>` with the ID of the cluster that you created previously.

```
$ aws cloudhsmv2 describe-clusters --filters clusterIds=<cluster ID> \
                                   --output text \
                                   --query
 'Clusters[].Certificates.AwsHardwareCertificate' \
                                   > <cluster ID>_AwsHardwareCertificate.crt
```

d. Issue the following command to extract the manufacturer hardware certificate. Replace `<cluster ID>` with the ID of the cluster that you created previously.

```
$ aws cloudhsmv2 describe-clusters --filters clusterIds=<cluster ID> \
                                   --output text \
                                   --query
 'Clusters[].Certificates.ManufacturerHardwareCertificate' \
                                   > <cluster
 ID>_ManufacturerHardwareCertificate.crt
```

AWS CloudHSM API

To get the CSR and HSM certificates (AWS CloudHSM API)

- Send a DescribeClusters request, then extract and save the CSR and certificates from the response.

Step 2. Get the root certificates

Follow these steps to get the root certificates for AWS CloudHSM and the manufacturer. Save the root certificate files to the directory that contains the CSR and HSM certificate files.

To get the AWS CloudHSM and manufacturer root certificates

1. Download the AWS CloudHSM root certificate: AWS_CloudHSM_Root-G1.zip
2. Download the right manufacturer root certificate for your HSM type:
   - hsm1.medium manufacturer root certificate: liquid_security_certificate.zip
   - hsm2m.medium manufacturer root certificate: liquid_security_certificate.zip

   > **Note**
   >
   > To download each certificate from its landing page, use the following links:
   >
   > - Landing page for hsm1.medium's manufacturer root certificate
   > - Landing page for hsm2m.medium's manufacturer root certificate
   >
   > You might need to right-click the **Download Certificate** link and then choose **Save Link As...** to save the certificate file.

3. After you download the files, extract (unzip) the contents.

Step 3. Verify certificate chains

In this step, you construct two certificate chains, one to the AWS CloudHSM root certificate and one to the manufacturer root certificate. Then use OpenSSL to verify the HSM certificate with each certificate chain.

To create the certificate chains, open a Linux shell. You need OpenSSL, which is available in most Linux shells, and you need the root certificate and HSM certificate files that you downloaded. However, you do not need the AWS CLI for this step, and the shell does not need to be associated with your AWS account.

To verify the HSM certificate with the AWS CloudHSM root certificate

1. Navigate to the directory where you saved the root certificate and HSM certificate files that you downloaded. The following commands assume that all of the certificates are in the current directory and use the default file names.

   Use the following command to create a certificate chain that includes the AWS hardware certificate and the AWS CloudHSM root certificate, in that order. Replace *<cluster ID>* with the ID of the cluster that you created previously.

```
$ cat <cluster ID>_AwsHardwareCertificate.crt \
     AWS_CloudHSM_Root-G1.crt \
     > <cluster ID>_AWS_chain.crt
```

2. Use the following OpenSSL command to verify the HSM certificate with the AWS certificate chain. Replace *<cluster ID>* with the ID of the cluster that you created previously.

```
$ openssl verify -CAfile <cluster ID>_AWS_chain.crt <cluster ID>_HsmCertificate.crt
<cluster ID>_HsmCertificate.crt: OK
```

To verify the HSM certificate with the manufacturer root certificate

1. Use the following command to create a certificate chain that includes the manufacturer hardware certificate and the manufacturer root certificate, in that order. Replace *<cluster ID>* with the ID of the cluster that you created previously.

```
$ cat <cluster ID>_ManufacturerHardwareCertificate.crt \
     liquid_security_certificate.crt \
     > <cluster ID>_manufacturer_chain.crt
```

2. Use the following OpenSSL command to verify the HSM certificate with the manufacturer certificate chain. Replace *<cluster ID>* with the ID of the cluster that you created previously.

```
$ openssl verify -CAfile <cluster ID>_manufacturer_chain.crt <cluster
 ID>_HsmCertificate.crt
<cluster ID>_HsmCertificate.crt: OK
```

Step 4. Extract and compare public keys

Use OpenSSL to extract and compare the public keys in the HSM certificate and the cluster CSR, to ensure that they are the same.

To compare the public keys, use your Linux shell. You need OpenSSL, which is available in most Linux shells, but you do not need the AWS CLI for this step. The shell does not need to be associated with your AWS account.

To extract and compare the public keys

1. Use the following command to extract the public key from the HSM certificate.

```
$ openssl x509 -in <cluster ID>_HsmCertificate.crt -pubkey -noout > <cluster ID>_HsmCertificate.pub
```

2. Use the following command to extract the public key from the cluster CSR.

```
$ openssl req -in <cluster ID>_ClusterCsr.csr -pubkey -noout > <cluster ID>_ClusterCsr.pub
```

3. Use the following command to compare the public keys. If the public keys are identical, the following command produces no output.

```
$ diff <cluster ID>_HsmCertificate.pub <cluster ID>_ClusterCsr.pub
```

After you verify the identity and authenticity of the HSM, proceed to Initialize the cluster.

Initialize the cluster in AWS CloudHSM

After you create your cluster and add your hardware security module (HSM) in AWS CloudHSM, you can initialize the cluster. Complete the steps in the following topics to initialize your cluster.

> **Note**
>
> Before you initialize the cluster, review the process by which you can verify the identity and authenticity of the HSMs. This process is optional and works only until a cluster is

initialized. After the cluster is initialized, you cannot use this process to get your certificates or verify the HSMs.

Topics

Step 1. Get the cluster CSR

Before you can initialize the cluster, you must download and sign a certificate signing request (CSR) that is generated by the cluster's first HSM. If you followed the steps to verify the identity of your cluster's HSM, you already have the CSR and you can sign it. Otherwise, get the CSR now by using the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.

Important

To initialize your cluster, your trust anchor must comply with RFC 5280 and meet the following requirements:

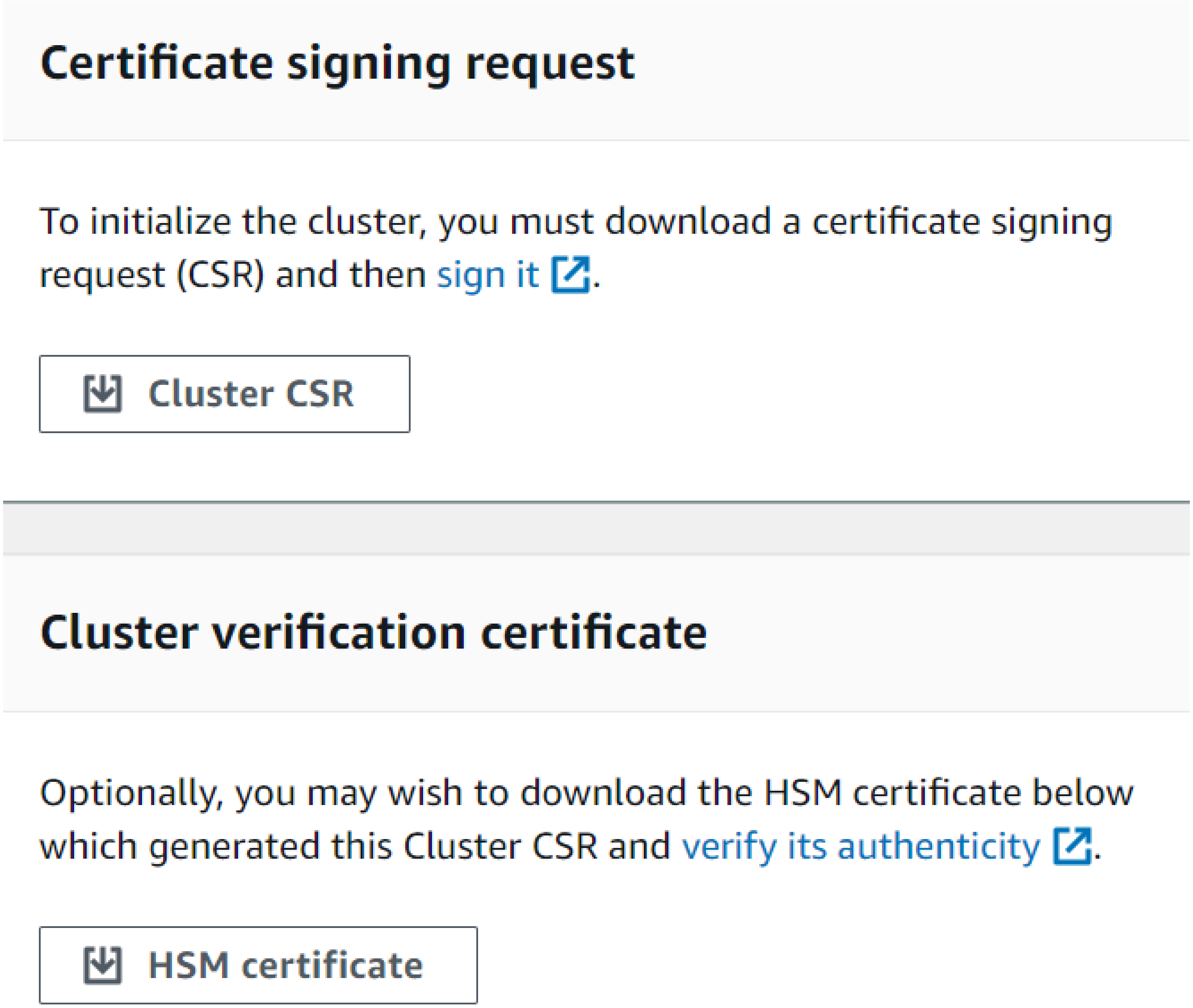
- If using X509v3 extensions, the X509v3 Basic Constraints extension must be present.
- The trust anchor must be a self-signed certificate.
- Extension values must not conflict with each other.

Console

To get the CSR (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Select the radio button next to the cluster ID with the HSM you want to verify.
3. Select **Actions**. From the drop down menu, choose **Initialize**.
4. If you did not complete the previous step to create an HSM, choose an Availability Zone (AZ) for the HSM that you are creating. Then select **Create**.

5. When the CSR is ready, you see a link to download it.



6. Choose **Cluster CSR** to download and save the CSR.

AWS CLI

To get the CSR ([AWS CLI](#))

- At a command prompt, run the following [describe-clusters](#) command, which extracts the CSR and saves it to a file. Replace *<cluster ID>* with the ID of the cluster that you [created previously](#).

```
$ aws cloudhsmv2 describe-clusters --filters clusterIds=<cluster ID> \
```

```
                                --output text \
                                --query 'Clusters[].Certificates.ClusterCsr'
\
                                > <cluster ID>_ClusterCsr.csr
```

AWS CloudHSM API

To get the CSR (AWS CloudHSM API)

1. Send a DescribeClusters request.
2. Extract and save the CSR from the response.

Step 2. Sign the CSR

Currently, you must create a self-signed signing certificate and use it to sign the CSR for your cluster. You do not need the AWS CLI for this step, and the shell does not need to be associated with your AWS account. To sign the CSR, you must do the following:

1. Complete the previous section (see Step 1. Get the cluster CSR).
2. Create a private key.
3. Use the private key to create a signing certificate.
4. Sign your cluster CSR.

Create a private key

Note

For a production cluster, the key you are about to create should be created in a secure manner using a trusted source of randomness. We recommend that you use a secured offsite and offline HSM or the equivalent. Store the key safely. The key establishes the identity of the cluster and your sole control over the HSMs it contains.

For development and testing, you can use any convenient tool (such as OpenSSL) to create and sign the cluster certificate. The following example shows you how to create a key. After you have used the key to create a self-signed certificate (see below), you should store it in a safe manner. To sign into your AWS CloudHSM instance, the certificate must be present, but the private key does not.

Use the following command to create a private key. When initializing an AWS CloudHSM cluster, you must use the RSA 2048 certificate or the RSA 4096 certificate.

```
$ openssl genrsa -aes256 -out customerCA.key 2048
Generating RSA private key, 2048 bit long modulus
........+++
............+++
e is 65537 (0x10001)
Enter pass phrase for customerCA.key:
Verifying - Enter pass phrase for customerCA.key:
```

Use the private key to create a self-signed certificate

The trusted hardware that you use to create the private key for your production cluster should also provide a software tool to generate a self-signed certificate using that key. The following example uses OpenSSL and the private key that you created in the previous step to create a signing certificate. The certificate is valid for 10 years (3652 days). Read the on-screen instructions and follow the prompts.

```
$ openssl req -new -x509 -days 3652 -key customerCA.key -out customerCA.crt
Enter pass phrase for customerCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

This command creates a certificate file named `customerCA.crt`. Put this certificate on every host from which you will connect to your AWS CloudHSM cluster. If you give the file a different name or store it in a path other than the root of your host, you should edit your client configuration file accordingly. Use the certificate and the private key you just created to sign the cluster certificate signing request (CSR) in the next step.

Sign the cluster CSR

The trusted hardware that you use to create your private key for your production cluster should also provide a tool to sign the CSR using that key. The following example uses OpenSSL to sign the cluster's CSR. The example uses your private key and the self-signed certificate that you created in the previous step.

```
$ openssl x509 -req -days 3652 -in <cluster ID>_ClusterCsr.csr \
                              -CA customerCA.crt \
                              -CAkey customerCA.key \
                              -CAcreateserial \
                              -out <cluster ID>_CustomerHsmCertificate.crt
Signature ok
subject=/C=US/ST=CA/O=Cavium/OU=N3FIPS/L=SanJose/CN=HSM:<HSM
 identifier>:PARTN:<partition number>, for FIPS mode
Getting CA Private Key
Enter pass phrase for customerCA.key:
```

This command creates a file named *<cluster ID>*`_CustomerHsmCertificate.crt`. Use this file as the signed certificate when you initialize the cluster.

Step 3. Initialize the cluster

Use your signed HSM certificate and your signing certificate to initialize your cluster. You can use the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.

Console

To initialize a cluster (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Select the radio button next to the cluster ID with the HSM you want to verify.
3. Select **Actions**. From the drop down menu, choose **Initialize**.
4. If you did not complete the previous step to create an HSM, choose an Availability Zone (AZ) for the HSM that you are creating. Then select **Create**.
5. On the **Download certificate signing request** page, choose **Next**. If **Next** is not available, first choose one of the CSR or certificate links. Then choose **Next**.

6. On the **Sign certificate signing request (CSR)** page, choose **Next**.
7. On the **Upload the certificates** page, do the following:

   a. Next to **Cluster certificate**, choose **Upload file**. Then locate and select the HSM certificate that you signed previously. If you completed the steps in the previous section, select the file named `<cluster ID>_CustomerHsmCertificate.crt`.

   b. Next to **Issuing certificate**, choose **Upload file**. Then select your signing certificate. If you completed the steps in the previous section, select the file named `customerCA.crt`.

   c. Choose **Upload and initialize**.

AWS CLI

To initialize a cluster (AWS CLI)

- At a command prompt, run the initialize-cluster command. Provide the following:
  - The ID of the cluster that you created previously.
  - The HSM certificate that you signed previously. If you completed the steps in the previous section, it's saved in a file named `<cluster ID>_CustomerHsmCertificate.crt`.
  - Your signing certificate. If you completed the steps in the previous section, the signing certificate is saved in a file named `customerCA.crt`.

```
$ aws cloudhsmv2 initialize-cluster --cluster-id <cluster ID> \
                                    --signed-cert file://<cluster
 ID>_CustomerHsmCertificate.crt \
                                    --trust-anchor file://customerCA.crt
{
    "State": "INITIALIZE_IN_PROGRESS",
    "StateMessage": "Cluster is initializing. State will change to INITIALIZED
 upon completion."
}
```

AWS CloudHSM API

To initialize a cluster (AWS CloudHSM API)

- Send an InitializeCluster request with the following:
  - The ID of the cluster that you created previously.
  - The HSM certificate that you signed previously.
  - Your signing certificate.

Install and configure CloudHSM CLI

To interact with the HSM in your AWS CloudHSM cluster, you need the CloudHSM CLI.

Connect to your client instance and run the following commands to download and install the AWS CloudHSM command line tools. For more information, see Launch an Amazon EC2 client instance for interacting with AWS CloudHSM.

Amazon Linux 2023

Amazon Linux 2023 on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/
cloudhsm-cli-latest.amzn2023.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.amzn2023.x86_64.rpm
```

Amazon Linux 2023 on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/
cloudhsm-cli-latest.amzn2023.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.amzn2023.aarch64.rpm
```

Amazon Linux 2

Amazon Linux 2 on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-cli-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el7.x86_64.rpm
```

Amazon Linux 2 on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-cli-latest.el7.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el7.aarch64.rpm
```

RHEL 9 (9.2+)

RHEL 9 on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-cli-latest.el9.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el9.x86_64.rpm
```

RHEL 9 on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-cli-latest.el9.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el9.aarch64.rpm
```

RHEL 8 (8.3+)

RHEL 8 on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-cli-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el8.x86_64.rpm
```

Ubuntu 24.04 LTS

Ubuntu 24.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Noble/cloudhsm-cli_latest_u24.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u24.04_amd64.deb
```

Ubuntu 24.04 LTS on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Noble/cloudhsm-cli_latest_u24.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u24.04_arm64.deb
```

Ubuntu 22.04 LTS

Ubuntu 22.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/cloudhsm-cli_latest_u22.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u22.04_amd64.deb
```

Ubuntu 22.04 LTS on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/cloudhsm-cli_latest_u22.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u22.04_arm64.deb
```

Ubuntu 20.04 LTS

Ubuntu 20.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Focal/cloudhsm-cli_latest_u20.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u20.04_amd64.deb
```

Windows Server 2022

For Windows Server 2022 on x86_64 architecture, open PowerShell as an administrator and run the following command:

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Windows/
AWSCloudHSMCLI-latest.msi -Outfile C:\AWSCloudHSMCLI-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMCLI-latest.msi /
quiet /norestart /log C:\client-install.txt' -Wait
```

Windows Server 2019

For Windows Server 2019 on x86_64 architecture, open PowerShell as an administrator and run the following command:

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Windows/
AWSCloudHSMCLI-latest.msi -Outfile C:\AWSCloudHSMCLI-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMCLI-latest.msi /
quiet /norestart /log C:\client-install.txt' -Wait
```

Windows Server 2016

For Windows Server 2016 on x86_64 architecture, open PowerShell as an administrator and run the following command:

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Windows/
AWSCloudHSMCLI-latest.msi -Outfile C:\AWSCloudHSMCLI-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMCLI-latest.msi /
quiet /norestart /log C:\client-install.txt' -Wait
```

Use the following commands to configure CloudHSM CLI.

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of the HSM(s) in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-cli -a <The ENI IPv4 / IPv6 addresses of the
 HSMs>
```

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of the HSM(s) in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" -a <The ENI
 IPv4 / IPv6 addresses of the HSMs>
```

Activate the cluster in AWS CloudHSM

When you activate an AWS CloudHSM cluster, the cluster's state changes from initialized to active. You can then manage the hardware security module (HSM) users and use the HSM.

> **Important**
>
> Before you can activate the cluster, you must first copy the issuing certificate to the default location for the platform on each EC2 instance that connects to the cluster (you create the issuing certificate when you initialize the cluster).
>
> Linux
>
> ```
> /opt/cloudhsm/etc/customerCA.crt
> ```
>
> Windows
>
> ```
> C:\ProgramData\Amazon\CloudHSM\customerCA.crt
> ```

After placing the issuing certificate, install CloudHSM CLI and run the **cluster activate** command on your first HSM. You will notice the admin account on the first HSM in your cluster has the unactivated-admin role. This a temporary role that only exists prior to cluster activation. When you activate your cluster, the unactivated-admin role changes to admin.

To activate a cluster

1. Connect to the client instance that you previously launched in. For more information, see Launch an Amazon EC2 client instance for interacting with AWS CloudHSM. You can launch a Linux instance or a Windows Server.
2. Run the CloudHSM CLI in interactive mode.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```
3. (Optional) Use the **user list** command to display the existing users.

   ```
   aws-cloudhsm > user list
   {
     "error_code": 0,
     "data": {
       "users": [
         {
           "username": "admin",
           "role": "unactivated-admin",
           "locked": "false",
           "mfa": [],
           "cluster-coverage": "full"
         },
         {
           "username": "app_user",
           "role": "internal(APPLIANCE_USER)",
           "locked": "false",
           "mfa": [],
           "cluster-coverage": "full"
         }
       ]
     }
   }
   ```
4. Use the **cluster activate** command to set the initial admin password.

```
aws-cloudhsm > cluster activate
Enter
 password:<NewPassword>
Confirm password:<NewPassword>
{
  "error_code": 0,
  "data": "Cluster activation successful"
}
```

We recommend that you write down the new password on a password worksheet. Do not lose the worksheet. We recommend that you print a copy of the password worksheet, use it to record your critical HSM passwords, and then store it in a secure place. We also recommended that you store a copy of this worksheet in secure off-site storage.

5. (Optional) Use the **user list** command to verify that the user's type changed to admin/CO.

```
aws-cloudhsm > user list
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      },
       {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

6. Use the **quit** command to stop the CloudHSM CLI tool.

```
aws-cloudhsm > quit
```

For more information about working with CloudHSM CLI or the CMU, see Understanding HSM Users and Understanding HSM User Management with CMU.

Set up mutual TLS between client and AWS CloudHSM (recommended)

The following topics describe the steps that you must complete to enable the mutual TLS (mTLS) between client and AWS CloudHSM. Currently this feature is exclusively available on hsm2m.medium. For more information about HSM type, see AWS CloudHSM cluster modes .

Topics

Step 1. Create and register a trust anchor onto the HSM

A trust anchor must be created and registered onto the HSM before enabling mTLS. This is a two-step process:

Topics

Create a private key and self-signed root certificate

> **Note**
>
> For a production cluster, the key you are about to create should be created in a secure manner using a trusted source of randomness. We recommend that you use a secured offsite and offline HSM or the equivalent. Store the key safely.

For development and testing, you can use any convenient tool (such as OpenSSL) to create the key and self-sign a root certificate. You will need the key and root certificate to sign the client certificate in the enable mTLS for AWS CloudHSM.

The following examples show how to create a private key and self-signed root certificate with OpenSSL.

Example – Create a private key with OpenSSL

Use the following command to create a 4096-bit RSA key encrypted with the AES-256 algorithm. To use this example, replace *<mtls_ca_root_1.key>* with the name of the file where you want to store the key.

```
$ openssl genrsa -out <mtls_ca_root_1.key> -aes256 4096
Generating RSA private key, 4096 bit long modulus
.....................................+++
.+++
e is 65537 (0x10001)
Enter pass phrase for mtls_ca_root_1.key:
Verifying - Enter pass phrase for mtls_ca_root_1.key:
```

Example – Create a self-signed root certificate with OpenSSL

Use the following command to create a self-signed root certificate named `mtls_ca_root_1.crt` from the private key you just created. The certificate is valid for 25 years (9130 days). Read the on-screen instructions and follow the prompts.

```
$ openssl req -new -x509 -days 9130 -key mtls_ca_root_1.key -out mtls_ca_root_1.crt
Enter pass phrase for mtls_ca_root_1.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
```

```
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

Register the trust anchor onto the HSM

After creating a self-signed root certificate, the admin must register it as the trust anchor with the AWS CloudHSM cluster.

To register a trust anchor with the HSM

1. Use the following command to start CloudHSM CLI:

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Using CloudHSM CLI, log in as an admin.

   ```
   aws-cloudhsm > login --username <admin> --role admin
   Enter password:
   {
     "error_code": 0,
     "data": {
       "username": "<admin>",
       "role": "admin"
     }
   }
   ```

3. Use the Register a trust anchor with CloudHSM CLI command to register the trust anchor. For more information, see the following example or use the **help cluster mtls register-trust-anchor** command.

Example – Register a trust anchor with AWS CloudHSM cluster

The following example shows how to use the **cluster mtls register-trust-anchor** command in CloudHSM CLI to register an trust anchor onto the HSM. To use this command, the admin must be logged in to the HSM. Replace these values with your own:

```
aws-cloudhsm > cluster mtls register-trust-anchor --path </path/mtls_ca_root_1.crt>
{
  "error_code": 0,
  "data": {
    "trust_anchor": {
      "certificate-reference": "0x01",
      "certificate": "<PEM Encoded Certificate>",
      "cluster-coverage": "full"
    }
  }
}
```

Note

AWS CloudHSM supports registering intermediate certificates as trust anchor. In such cases, the entire PEM-encoded certificate chain file needs to be registered onto the HSM, with the certificates in hierarchical order.
AWS CloudHSM supports a certificate chain of 6980 bytes.

After successfully registering the trust anchor, you can run the **cluster mtls list-trust-anchors** command to check the current registered trust anchors, as shown below:

```
aws-cloudhsm > cluster mtls list-trust-anchors
{
  "error_code": 0,
  "data": {
    "trust_anchors": [
      {
        "certificate-reference": "0x01",
        "certificate": "<PEM Encoded Certificate>",
        "cluster-coverage": "full"
      }
    ]
  }
}
```

Note

The maximum number of trust anchors can be registered onto hsm2m.medium is two (2).

Step 2. Enable mTLS for AWS CloudHSM

To enable the mTLS for AWS CloudHSM, you need to create a private key and a client certificate signed by the root certificate we generated in Create and register a trust anchor onto the HSM, and then use any of the Client SDK 5 configure tool to setup the private key path and client certificate chain path.

Topics

- Create a private key and client certificate chain
- Configure mTLS for Client SDK 5

Create a private key and client certificate chain

Example – Create a private key with OpenSSL

Use the following command to create a 4096-bit RSA key. To use this example, replace *<ssl-client.key>* with the name of the file where you want to store the key.

```
$ openssl genrsa -out <ssl-client.key> 4096
Generating RSA private key, 4096 bit long modulus
.....................................+++
.+++
e is 65537 (0x10001)
```

Example – Generate a certificate signing request (CSR) with OpenSSL

Use the following command to generate a certificate signing request (CSR) from the private key you just created. Read the on-screen instructions and follow the prompts.

```
$ openssl req -new -key <ssl-client.key> -out <ssl-client.csr>
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
```

```
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

Example – Sign the CSR with the root certificate

Use the following command to sign the CSR with the root certificate we created and registered in Create and register a trust anchor onto the HSM and create a client certificate named `ssl-client.crt`. The certificate is valid for 5 years (1826 days).

```
$ openssl x509 -req -days 1826 -in <ssl-client.csr> -CA <mtls_ca_root_1.crt> -CAkey <mtls_ca_root_1.key> -CAcreateserial -out <ssl-client.crt>
```

Example – Create a client certificate chain

Use the following command to combine the client certificate and root certificate we created and registered in Create and register a trust anchor onto the HSM and create a client certificate chain named `ssl-client.pem`, which will be used to configure in next step.

```
$ cat <ssl-client.crt> <mtls_ca_root_1.crt> > <ssl-client.pem>
```

Note

If you registered intermediate certificates in Create and register a trust anchor onto the HSM as trust anchor, make sure to combine the client certificate with the entire certificate chain to create a client certificate chain.

Configure mTLS for Client SDK 5

Use any of the Client SDK 5 configure tools to enable the mutual TLS by providing the right client key path and client certificate chain path. For more information about configure tool for Client SDK 5, see ??? .

PKCS #11 library

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

```
$ sudo cp ssl-client.pem </opt/cloudhsm/etc>
$ sudo cp ssl-client.key </opt/cloudhsm/etc>
```

2. Use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 \
            --client-cert-hsm-tls-file </opt/cloudhsm/etc/ssl-client.pem> \
            --client-key-hsm-tls-file </opt/cloudhsm/etc/ssl-client.key>
```

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Windows

1. Copy your key and certificate to the appropriate directory.

```
cp ssl-client.pem <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem>
cp ssl-client.key <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

2. With a PowerShell interpreter, use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" `
            --client-cert-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-
client.pem> `
            --client-key-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-
client.key>
```

OpenSSL Dynamic Engine

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

```
$ sudo cp ssl-client.pem </opt/cloudhsm/etc>
sudo cp ssl-client.key </opt/cloudhsm/etc>
```

2. Use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
$ sudo /opt/cloudhsm/bin/configure-dyn \
            --client-cert-hsm-tls-file </opt/cloudhsm/etc/ssl-client.pem> \
            --client-key-hsm-tls-file </opt/cloudhsm/etc/ssl-client.key>
```

Key Storage Provider (KSP)

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Windows

1. Copy your key and certificate to the appropriate directory.

```
cp ssl-client.pem <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem>
cp ssl-client.key <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

2. With a PowerShell interpreter, use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" `
            --client-cert-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-
client.pem> `
            --client-key-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-
client.key>
```

JCE provider

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

```
$ sudo cp ssl-client.pem </opt/cloudhsm/etc>
sudo cp ssl-client.key </opt/cloudhsm/etc>
```

2. Use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
$ sudo /opt/cloudhsm/bin/configure-jce \
            --client-cert-hsm-tls-file </opt/cloudhsm/etc/ssl-client.pem> \
            --client-key-hsm-tls-file </opt/cloudhsm/etc/ssl-client.key>
```

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Windows

1. Copy your key and certificate to the appropriate directory.

```
cp ssl-client.pem <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem>
cp ssl-client.key <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

2. With a PowerShell interpreter, use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" `
            --client-cert-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem> `
            --client-key-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

CloudHSM CLI

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

```
$ sudo cp ssl-client.pem </opt/cloudhsm/etc>
sudo cp ssl-client.key </opt/cloudhsm/etc>
```

2. Use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
$ sudo /opt/cloudhsm/bin/configure-cli \
            --client-cert-hsm-tls-file </opt/cloudhsm/etc/ssl-client.pem> \
            --client-key-hsm-tls-file </opt/cloudhsm/etc/ssl-client.key>
```

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Windows

1. Copy your key and certificate to the appropriate directory.

```
cp ssl-client.pem <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem>
cp ssl-client.key <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

2. With a PowerShell interpreter, use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" `
            --client-cert-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-
client.pem> `
            --client-key-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-
client.key>
```

Step 3. Set the mTLS enforcement for AWS CloudHSM

After configuring with any of the Client SDK 5 configure tools, connection between client and AWS CloudHSM will be mutual TLS in the cluster. However, removing the private key path and client certificate chain path from the config file will turn the connection into regular TLS again. You can use CloudHSM CLI to set the mtls enforcement in the cluster by completing the following steps:

1. Use the following command to start CloudHSM CLI:

   Linux

```
$ /opt/cloudhsm/bin/cloudhsm-cli interactive
```

   Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
```

2. Using CloudHSM CLI, log in as an admin.

```
aws-cloudhsm > login --username <admin> --role admin
Enter password:
{
  "error_code": 0,
```

```
  "data": {
    "username": "<admin>",
    "role": "admin"
  }
}
```

> **Note**
>
> 1. Make sure you have configured the CloudHSM CLI and start the CloudHSM CLI under a mTLS connection.
> 2. You must be logged in as the default admin user with username as **admin** before set mTLS enforcement.

3. Use the **Set the mTLS enforcement level with CloudHSM CLI** command to set the enforcement. For more information, see the following example or use the **help cluster mtls set-enforcement** command.

   Example – Set mTLS enforcement with AWS CloudHSM cluster

   The following example shows how to use the **cluster mtls set-enforcement** command in CloudHSM CLI to set the mTLS enforcement with the HSM. To use this command, the admin with username as admin must be logged in to the HSM.

```
aws-cloudhsm > cluster mtls set-enforcement --level cluster
{
  "error_code": 0,
  "data": {
    "message": "Mtls enforcement level set to Cluster successfully"
  }
}
```

> **Warning**
>
> After you enforce mTLS usage in the cluster, all existing non-mTLS connections will be dropped and you can only connect to the cluster with mTLS certificates.

Create and use keys in AWS CloudHSM

Before you can create and use keys in your new cluster, create a hardware security module (HSM) user with the AWS CloudHSM CLI For more information, see Understanding HSM User Management Tasks, Getting started with AWS CloudHSM Command Line Interface (CLI), and How to Manage HSM Users.

> **Note**
>
> If using Client SDK 3, use CloudHSM Management Utility (CMU) instead of CloudHSM CLI.

After you create HSM users, you can sign in to the HSM and manage keys using any of these options:

- Use key management utility, a command line tool
- Build a C application using the PKCS #11 library
- Build a Java application using the JCE provider
- Use the OpenSSL Dynamic Engine directly from the command line
- Use the OpenSSL Dynamic Engine for TLS offload with NGINX and Apache web servers
- Use the Key Storage Provider (KSP) for AWS CloudHSM with Microsoft Windows Server Certificate Authority (CA)
- Use the Key Storage Provider (KSP) for AWS CloudHSM with Microsoft Sign Tool
- Use the Key Storage Provider (KSP) for TLS offload with Internet Information Server (IIS) web server

Best practices for AWS CloudHSM

Perform the best practices in this topic to effectively use AWS CloudHSM.

Contents

AWS CloudHSM cluster management best practices

Follow the best practices in this section when creating, accessing, and managing your AWS CloudHSM cluster.

Scale your cluster to handle peak traffic

Several factors can influence the maximum throughput that your cluster can handle, including client instance size, cluster size, network topography, and the cryptographic operations you require for your use case.

As a starting point, refer to the topic AWS CloudHSM performance information for performance estimates on common cluster sizes and configurations. We recommend you load test your cluster with the peak load you anticipate to determine whether your current architecture is resilient and at the right scale.

Architect your cluster for high availability

Add redundancy to account for maintenance: AWS may replace your HSM for scheduled maintenance or if it detects a problem. As a general rule, your cluster size should have at least +1 redundancy. For example, if you require two HSMs for your service to operate at peak times, your ideal cluster size will then be three. If you follow the best practices relating to availability, these HSM replacements should not impact your service. However, in-progress operations on the replaced HSM may fail and must be retried.

Spread your HSMs across many Availability Zones: Consider how your service will be able to operate during an Availability Zone outage. AWS recommends that you spread your HSMs across as many Availability Zones as possible. For a cluster with three HSMs, you should spread HSMs across three Availability Zones. Depending on your system, you may require additional redundancy.

Have at least three HSMs to ensure durability for newly generated keys

For applications that require durability of newly generated keys, we recommend having at least three HSMs spread across different Availability Zones in a region.

Secure access to your cluster

Use private subnets to limit access to your instance: Launch your HSMs and client instances in the private subnets of your VPC. This limits access to your HSMs from the outside world.

Use VPC endpoints to access APIs: The AWS CloudHSM data plane was designed to operate without needing access to the internet or AWS APIs. If your client instance requires access to the AWS CloudHSM API, you can use VPC endpoints to access the API without requiring internet access on your client instance. See AWS CloudHSM and VPC endpoints for more information.

Reduce costs by scaling to your needs

There are no upfront costs to use AWS CloudHSM. You pay an hourly fee for each HSM you launch until you terminate the HSM. If your service does not require continuous usage of AWS CloudHSM, you can reduce costs by scaling down (deleting) your HSMs to zero when they are not needed. When HSMs are again needed, you can restore your HSMs from a backup. If, for example, you have a workload requiring you to sign code once a month, specifically on the last day of the month, you can scale up your cluster before, scale it down by deleting your HSMs after the work is completed, and then restore your cluster to perform signing operations again at the end of the next month.

AWS CloudHSM automatically makes periodic backups of the HSMs in the cluster. When adding a new HSM at a later date, AWS CloudHSM will restore the latest backup onto the new HSM so that you can resume usage from the same place you left it. To calculate your AWS CloudHSM architecture costs, see AWS CloudHSM Pricing.

Related resources:

- General overview of backups
- Backup retention policy
- Copying AWS CloudHSM cluster backups across AWS Regions

AWS CloudHSM user management best practices

Follow the best practices in this section to effectively manage users in your AWS CloudHSM cluster. HSM users are distinct from IAM users. IAM users and entities that have an identity-based policy with the appropriate permissions can create HSMs by interacting with resources through the AWS API. After the HSM is created, you must use HSM user credentials to authenticate operations on the HSM. For a detailed guide of HSM users, see HSM users in AWS CloudHSM.

Protect your HSM users' credentials

It is imperative to keep the credentials of your HSM users securely protected as HSM users are the entities that can access and perform cryptographic and management operations on your HSM. AWS CloudHSM does not have access to your HSM user credentials, and will be unable to assist you if you lose access to them.

Have at least two admins to prevent lockout

To avoid being locked out of your cluster, we recommend you have at least two admins in case one admin password is lost. In the event this happens, you can use the other admin to reset the password.

> **Note**
>
> *Admins* in Client SDK 5 are synonymous with *crypto officers* (COs) in Client SDK 3.

Enable quorum for all user management operations

Quorum allows you to set a min number of admins that must approve a user management operation before that operation can take place. Due to the privilege that admins have, we recommend that you enable quorum for all user management operations. This can limit the potential for impact if one of your admin passwords is compromised. For more information, see Managing Quorum.

Create multiple crypto users, each with limited permissions

By separating the responsibilities of crypto users, no one user has total control over the entire system. For this reason, we recommend you create multiple crypto users and limit the permissions of each. Typically, this is done by giving different crypto users distinctly different responsibilities and actions they perform (for example, having one crypto user who is responsible for generating and sharing keys with other crypto users who then utilize them in your application).

Related resources:

- Share a key using CloudHSM CLI
- Unshare a key using CloudHSM CLI

AWS CloudHSM key management best practices

Follow the best practices in this section when managing keys in AWS CloudHSM.

Choose the right key type

When using a session key, your transactions per second (TPS) will be limited to one HSM where the key exists. Extra HSMs in your cluster will not increase the throughput of requests for that key. If you use a token key for the same application, your requests will be load balanced across all available HSMs in your cluster. For more information, see Key synchronization and durability settings in AWS CloudHSM.

Manage key storage limits

HSMs have limits on the maximum number of token and session keys that can be stored on an HSM at a single time. For information on key storage limits, see AWS CloudHSM quotas. If your application requires more than the limit, you can use one or more of the following strategies to effectively manage keys:

Use trusted wrapping to store your keys in an external data store: Using trusted key wrapping, you can overcome the key storage limit by storing all of your keys wrapped inside an external data store. When you are required to use this key, you can unwrap the key into the HSM as a session key, use the key for your required operation, and then discard the session key. The original key data remains safely stored in your data store for use whenever you need it. Using trusted keys to do this maximizes your protection.

Distribute keys across clusters: Another strategy for overcoming the key storage limit is storing your keys in multiple clusters. In this approach, you maintain a mapping of the keys that are stored in each cluster. Use this mapping to route your client requests to the cluster with the required key. For information on how to connect to multiple clusters from the same client application, see the following topics:

- Connecting to multiple AWS CloudHSM clusters with the JCE provider
- Multiple slot configuration with PKCS #11 library for AWS CloudHSM

Managing and securing key wrapping

Keys may be marked either extractable or non-extractable through the `EXTRACTABLE` attribute. By default, HSM keys are marked as extractable.

Extractable keys are keys that are permitted to be exported from the HSM through key wrapping. Keys that are wrapped are encrypted, and must be unwrapped using the same wrapping key before they can be used. Non-extractable keys may not be exported from the HSM under any circumstance. There is no way to make a non-extractable key extractable. For this reason, it is important to consider whether you require your keys to be extractable or not and to set the corresponding key attribute accordingly.

If you require key wrapping in your application, you should utilize trusted key wrapping to limit the ability of your HSM users to only wrap/unwrap keys which have been explicitly marked as trusted by an admin. For more information, see topics on trusted key wrapping in Keys in AWS CloudHSM.

Related resources

- Wrap and Unwrap functions
- Cipher functions for JCE
- Supported Java key attributes for AWS CloudHSM Client SDK 5
- Key attributes for CloudHSM CLI

AWS CloudHSM application integration best practices

Follow the best practices in this section to optimize how your application integrates with your AWS CloudHSM cluster.

Bootstrap your Client SDK

Before your client SDK can connect to your cluster, it must be bootstrapped. When bootstrapping IP addresses to your cluster, we recommend using the `--cluster-id` parameter when possible. This method populates your config with all HSM IP addresses in your cluster without needing to keep track of each individual address. Doing this adds extra resilience to your application initialization in the event an HSM is undergoing maintenance or during an Availability Zone outage. For more details, see Bootstrap the Client SDK.

Authenticate to perform operations

In AWS CloudHSM, you must authenticate to your cluster before you are able to perform most operations such as cryptographic operations.

Authenticate with CloudHSM CLI: You can authenticate with CloudHSM CLI using either its single command mode or interactive mode. Use the Log in to an HSM using CloudHSM CLI command to authenticate in interactive mode. To authenticate in single command mode, you must set the environmental variables `CLOUDHSM_ROLE` and `CLOUDHSM_PIN`. For details on doing this, refer to Single Command mode. AWS CloudHSM recommends securely storing your HSM credentials when not being used by your application.

Authenticate with PKCS #11: In PKCS #11, you login using the C_Login API after opening a session using C_OpenSession. You only need to perform one C_Login per slot (cluster). After you have successfully logged in, you can open additional sessions using C_OpenSession without the need to perform additional login operations. For examples on authenticating to PKCS #11, see Code samples for the PKCS #11 library for AWS CloudHSM Client SDK 5.

Authenticate with JCE: The AWS CloudHSM JCE Provider supports both implicit and explicit login. The method that works for you depends on your use case. When possible, we recommend using Implicit Login because the SDK will automatically handle authentication if your application becomes disconnected from your cluster and needs to be re-authenticated. Using implicit login also allows you to provide credentials to your application when using an integration that doesn't allow you to have control over your application code. For more about login methods, see Step 2: Provide credentials to the JCE provider.

Authenticate with OpenSSL: With the OpenSSL Dynamic Engine, you provide credentials through environment variables. AWS CloudHSM recommends securely storing your HSM credentials when not being used by your application. If possible, you should configure your environment to systematically retrieve and set these environment variables without manual entry. For details on authenticating with OpenSSL, see Install the OpenSSL Dynamic Engine for AWS CloudHSM Client SDK 5.

Authenticate with KSP: You can authenticate with Key Storage Provider (KSP) using either Windows credential manager or environment variables, see Install the Key storage provider (KSP) for AWS CloudHSM Client SDK 5.

Effectively manage keys in your application

Use key attributes to control what keys can do: When generating a key, use key attributes to define a set of permissions that will allow or deny specific types of operations for that key. We recommend that keys be generated with the least amount of attributes needed to complete their task. For example, an AES key used for encryption should not also be allowed to wrap keys out of the HSM. For more information, see our attributes pages for the following Client SDKs:

- PKCS #11 key attributes
- JCE key attributes

When possible, cache key objects to minimize latency: Key find operations will query every HSM in your cluster. This operation is expensive and does not scale with HSM count in your cluster.

- With PKCS #11, you find keys using the `C_FindObjects` API.
- With JCE, you find keys using the KeyStore.

For optimal performance, AWS recommends that you utilize key find commands (like Search for AWS CloudHSM keys by attributes using KMU and List keys for a user with CloudHSM CLI) only once during your application start-up and cache the key object returned in application memory. If you require this key object later on, you should retrieve the object from your cache instead of querying for this object for each operation which will add significant performance overhead.

Use multi-threading

AWS CloudHSM supports multi-threaded applications, but there are certain things to keep in mind with multi-threaded applications.

With PKCS #11, you should initialize the PKCS #11 library (calling `C_Initialize`) only once. Each thread should be assigned its own session (`C_OpenSession`). Using the same session in multiple threads is not recommended.

With JCE, the AWS CloudHSM provider should be initialized only once. Do not share instances of SPI objects across threads. For example, Cipher, Signature, Digest, Mac, KeyFactory or KeyGenerator objects should only be utilized in the context of their own thread.

Handle throttling errors

You may experience HSM throttling errors under the following circumstances:

- Your cluster is not properly scaled to manage peak traffic.
- Your cluster is not sized with a +1 redundancy during maintenance events.
- Availability Zone outages result in a reduced number of available HSMs in your cluster.

See HSM throttling for information on how to best handle this scenario.

To ensure your cluster is adequately sized and will not be throttled, AWS recommends you load test in your environment with your expected peak traffic.

Integrate retries on cluster operations

AWS may replace your HSM for operational or maintenance reasons. In order to make your application resilient to such situations, AWS recommends that you implement client-side retry logic on all operations that are routed to your cluster. Subsequent retries on failed operations due to replacements are expected to succeed.

Implement disaster recovery strategies

In response to an event, it may be necessary to shift your traffic away from an entire cluster or region. The following sections describe multiple strategies for doing this.

Use VPC peering to access your cluster from another account or region: You can utilize VPC peering to access your AWS CloudHSM cluster from another account or region. For information on how to set this up, see What is VPC peering? in the *VPC Peering Guide*. Once you have established your peering connections and configured your security groups appropriately, you can communicate with HSM IP addresses in the same way as you normally would.

Connect to multiple clusters from the same application: The JCE provider, PKCS #11 library, and CloudHSM CLI in Client SDK 5 support connecting to multiple clusters from the same application. For example, you can have two active clusters, each in different regions, and your application can connect to both at once and load balance between the two as part of normal operations. If your application is not using Client SDK 5 (the latest SDK), then you cannot connect to multiple clusters from the same application. Alternatively, you can keep another cluster up and running and, in the event there is a regional outage, shift your traffic to the other cluster to minimize downtime. See the respective pages for details:

- Multiple slot configuration with PKCS #11 library for AWS CloudHSM
- Connecting to multiple AWS CloudHSM clusters with the JCE provider
- Connecting to multiple clusters with CloudHSM CLI

Restore a cluster from a backup: You can create a new Cluster from a backup of an existing Cluster. For more information, see Cluster backups in AWS CloudHSM.

AWS CloudHSM monitoring best practices

This section describes multiple mechanisms you can use to monitor your cluster and application. For additional details on monitoring, see Monitoring AWS CloudHSM.

Monitor client logs

Every Client SDK writes logs that you can monitor. For information on client logging, see Working with AWS CloudHSM client SDK logs.

On platforms that are designed to be ephemeral, such as Amazon ECS and AWS Lambda, collecting client logs from a file can be difficult. In these situations, it is a best practice to configure your Client SDK logging to write logs to the console. Most services will automatically collect this output and publish it to Amazon CloudWatch logs for you to keep and view.

If you are using any third-party integration on top of the AWS CloudHSM Client SDK, you should ensure that you configure that software package to log its output to the console as well. The output from the AWS CloudHSM Client SDK may be captured by this package and written to its own log file otherwise.

See the AWS CloudHSM Client SDK 5 configure tool for information on how to configure logging options in your application.

Monitor audit logs

AWS CloudHSM publishes audit logs to your Amazon CloudWatch account. Audit logs come from the HSM and track certain operations for auditing purposes.

You can use audit logs to keep track of any management commands that are invoked on your HSM. For example, you can trigger an alarm when you notice an unexpected management operation being performed.

See How HSM audit logging works for more details.

Monitor AWS CloudTrail

AWS CloudHSM is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS CloudHSM. AWS CloudTrail captures all API calls for AWS CloudHSM as events. The calls captured include calls from the AWS CloudHSM console and code calls to the AWS CloudHSM API operations.

You can use AWS CloudTrail to audit any API call that is made to the AWS CloudHSM control plane to ensure that no unwanted activity is taking place in your account.

See Working with AWS CloudTrail and AWS CloudHSM for details.

Monitor Amazon CloudWatch metrics

You can use Amazon CloudWatch metrics to monitor your AWS CloudHSM cluster in real time. The metrics can be grouped by region, cluster ID, or HSM ID *and* cluster ID.

Using Amazon CloudWatch metrics, you can configure Amazon CloudWatch alarms to alert you of any potential issue that may arise that could impact your service. We recommend configuring alarms to monitor the following:

- Approaching your key limit on an HSM
- Approaching the HSM session count limit on an HSM
- Approaching the HSM user count limit on an HSM
- Differences in HSM user or key count to identify synchronization issues
- Unhealthy HSMs to scale your cluster up until AWS CloudHSM can resolve the issue

For more details, see Working with Amazon CloudWatch Logs and AWS CloudHSM Audit Logs.

Clusters in AWS CloudHSM

A cluster is a collection of individual hardware security modules (HSM) that AWS CloudHSM keeps in sync. When you perform a task or operation on one HSM in a cluster, the other HSMs in that cluster are automatically kept up to date.

You can manage your AWS CloudHSM clusters from the AWS CloudHSM console or one of the AWS SDKs or command line tools. For more information, see the following topics.

To create a cluster, see Getting started.

The following topics provide more information about clusters.

Topics

AWS CloudHSM cluster architecture

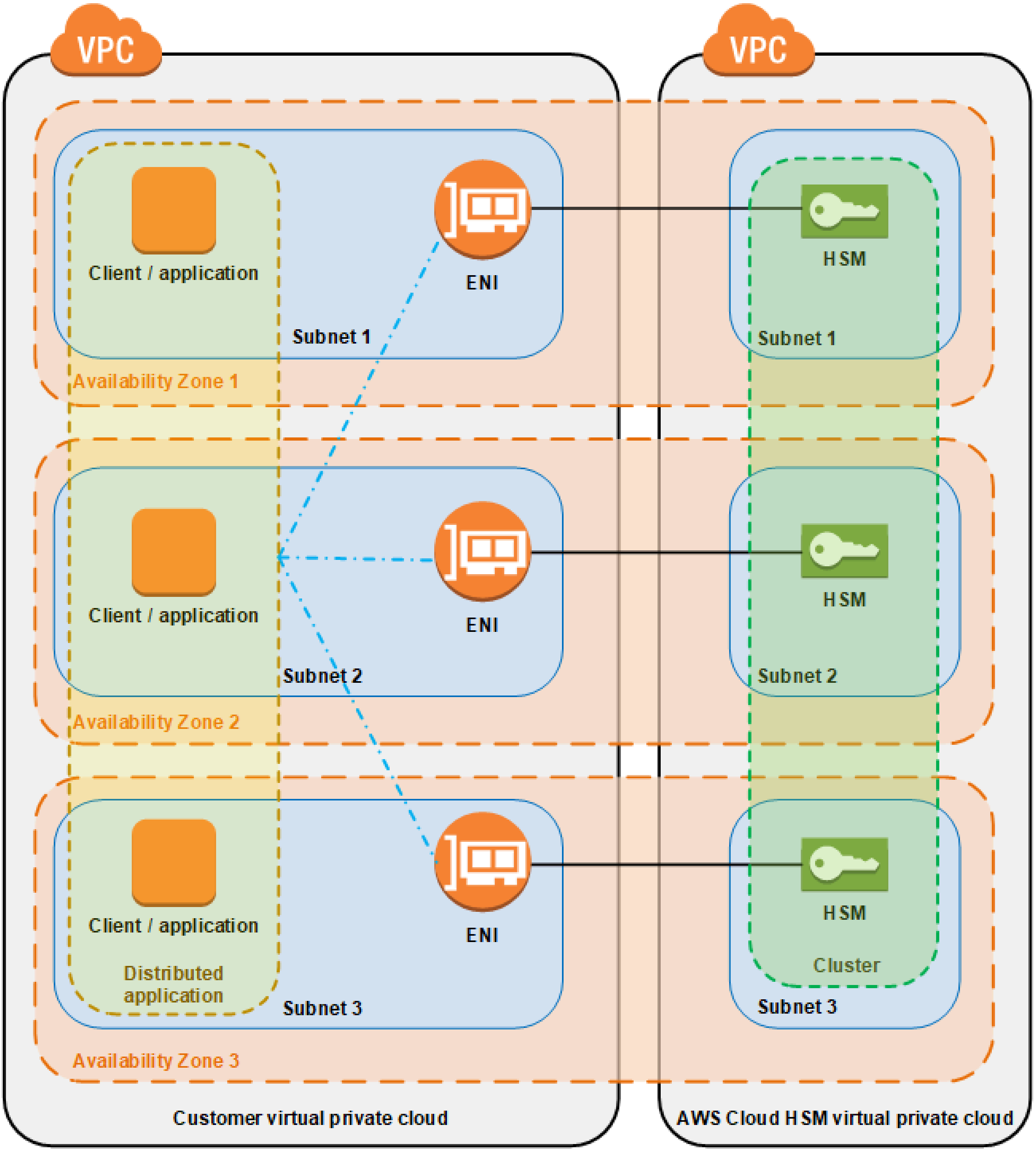
When you create a cluster, you specify an Amazon Virtual Private Cloud (VPC) in your AWS account and one or more subnets in that VPC. We recommend that you create one subnet in each Availability Zone (AZ) in your chosen AWS Region. You can create private subnets when you create a VPC. To learn more, see Create a virtual private cloud (VPC) for AWS CloudHSM.

Each time you create an HSM, you specify the cluster and Availability Zone for the HSM. By putting the HSMs in different Availability Zones, you achieve redundancy and high availability in case one Availability Zone is unavailable.

When you create an HSM, AWS CloudHSM puts an elastic network interface (ENI) in the specified subnet in your AWS account. The elastic network interface is the interface for interacting with the HSM. The HSM resides in a separate VPC in an AWS account that is owned by AWS CloudHSM. The HSM and its corresponding network interface are in the same Availability Zone.

To interact with the HSMs in a cluster, you need the AWS CloudHSM client software. Typically you install the client on Amazon EC2 instances, known as *client instances*, that reside in the same VPC as the HSM ENIs, as shown in the following figure. That's not technically required though; you can install the client on any compatible computer, as long as it can connect to the HSM ENIs. The client communicates with the individual HSMs in your cluster through their ENIs.

The following figure represents an AWS CloudHSM cluster with three HSMs, each in a different Availability Zone in the VPC.



AWS CloudHSM cluster synchronization

In an AWS CloudHSM cluster, AWS CloudHSM keeps the keys on the individual HSMs in sync. You don't need to do anything to synchronize the keys on your HSMs. To keep the users and policies on each HSM in sync, update the AWS CloudHSM client configuration file before you manage HSM users. For more information, see Keep HSM users in sync.

When you add a new HSM to a cluster, AWS CloudHSM makes a backup of all keys, users, and policies on an existing HSM. It then restores that backup onto the new HSM. This keeps the two HSMs in sync.

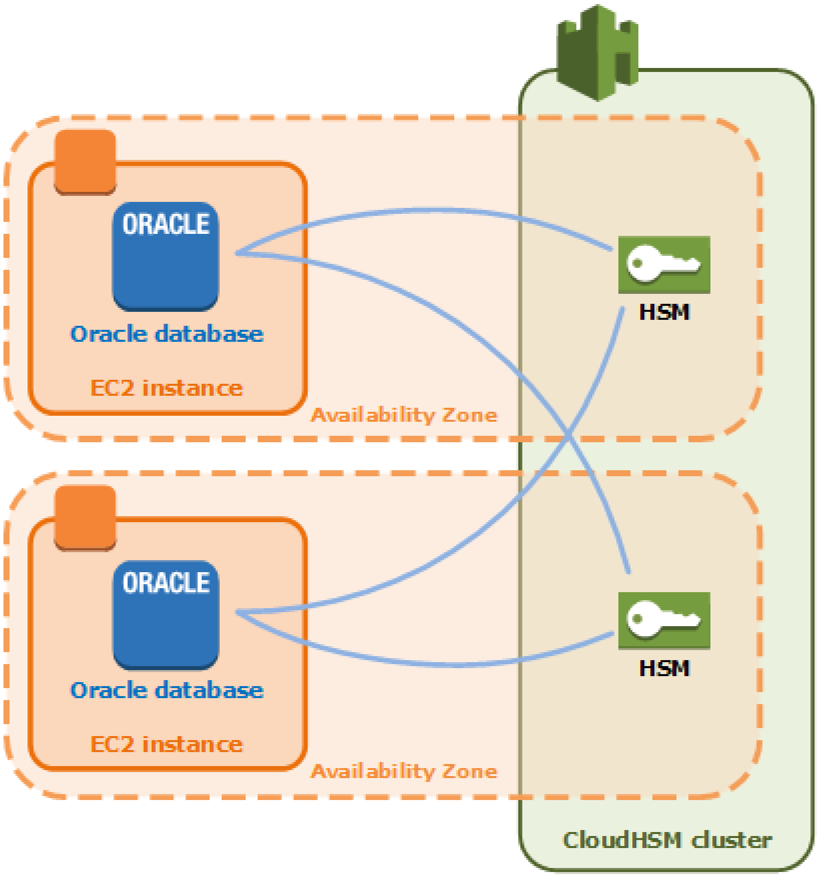
If the HSMs in a cluster fall out of synchronization, AWS CloudHSM automatically resynchronizes them. To enable this, AWS CloudHSM uses the credentials of the appliance user. This user exists on all HSMs provided by AWS CloudHSM and has limited permissions. It can get a hash of objects on the HSM and can extract and insert masked (encrypted) objects. AWS cannot view or modify your users or keys and cannot perform any cryptographic operations using those keys.

AWS CloudHSM cluster high availability and load balancing

When you create an AWS CloudHSM cluster with more than one HSM, you automatically get load balancing. Load balancing means that the AWS CloudHSM client distributes cryptographic operations across all HSMs in the cluster based on each HSM's capacity for additional processing.

When you create the HSMs in different AWS Availability Zones, you automatically get high availability. High availability means that you get higher reliability because no individual HSM is a single point of failure. We recommend that you have a minimum of two HSMs in each cluster, with each HSM in different Availability Zones within an AWS Region.

For example, the following figure shows an Oracle database application that is distributed to two different Availability Zones. The database instances store their master keys in a cluster that includes an HSM in each Availability Zone. AWS CloudHSM automatically synchronizes the keys to both HSMs so that they are immediately accessible and redundant.



AWS CloudHSM cluster modes

AWS CloudHSM offers clusters in two modes: *FIPS* and *non-FIPS*. In FIPS mode, only Federal Information Processing Standard (FIPS) validated keys and algorithms can be used. Non-FIPS mode offers all the keys and algorithms that are supported by AWS CloudHSM, regardless of FIPS approval.

Review the details on this page before deciding which cluster mode and HSM type is right for your needs.

Note

All clusters created before June 10, 2024 are in FIPS mode and have HSM type hsm1.medium.

To see your cluster's mode and HSM type, use the describe-clusters command.

The following table lists the major differences between each cluster mode:

Differentiating feature	FIPS mode	Non-FIPS mode
HSM type compatibility	Available with hsm1.medium and hsm2m.medium.	Available with hsm2m.med ium.
Backup compatibility	Can only be used to backup restore clusters in FIPS mode.	Can only be used to backup restore clusters in non-FIPS mode.
Key selection	Supports generating and using keys with mechanisms that are FIPS approved[1].	Supports generating and using keys with all FIPS-validated mechanisms, in addition to other non-valid ated mechanisms.
Algorithms	Supports AWS CloudHSM algorithms that are FIPS approved[1].	Supports AWS CloudHSM algorithms that are both FIPS approved and not FIPS approved.

[1] See Deprecation notifications for details.

Before choosing a cluster mode, note that a cluster's mode (FIPS or non-FIPS) cannot be changed after it is created, so ensure you select the right mode for your needs.

HSM types in AWS CloudHSM

AWS CloudHSM also offers two hardware security module (HSM) types: *hsm1.medium* and *hsm2m.medium*. Review the details on this page before deciding which HSM type is right for your needs.

In addition to cluster modes, AWS CloudHSM offers two HSM types: *hsm1.medium* and *hsm2m.medium*. Each HSM type uses different hardware, and each cluster can only contain one type of HSM. The following table lists the major differences between the two:

Differentiating feature	hsm1.medium	hsm2m.medium
Cluster mode compatibility	Available for clusters in FIPS mode.	Available for clusters in FIPS or non-FIPS mode.
Network type compatibility	Not available	Available for clusters in FIPS or non-FIPS mode.
Backup compatibility	Can be used to backup and restore to **hsm1.medium** and **hsm2m.medium** clusters in FIPS mode.	Can only be used to backup and restore **hsm2m.medium** clusters.
Key capacity	3,300 per cluster.	16,666 total keys, with asymmetric keys having a maximum of 3,333 per cluster.
Client SDKs	Supports all Client SDKs.	Supports all Client SDKs.
Client SDK versions	Compatible with SDK version 3.1.0 and later.	Compatible with Client SDK version 5.9.0 and later.
Region availability	CloudHSM no longer supports creating new clusters in any AWS Region. For more information, see Deprecation notifications for details.	Available in AWS Regions that CloudHSM is available.
Performance	To see the performance of each HSM type, refer to AWS CloudHSM performance information.	
Certification	FIPS 140-2, PCI DSS, PCI PIN, SOC2, and PCI-3DS compliant .	FIPS 140-3, PCI DSS, PCI PIN, and SOC2 compliant.

Connect the client SDK to the AWS CloudHSM cluster

To connect to the cluster with either Client SDK 5 or Client SDK 3, you must first do two things:

- Have an issuing certificate in place on the EC2 instance
- Bootstrap the Client SDK to the cluster

Place the issuing certificate on each EC2 instance

You create the issuing certificate when you initialize the cluster. Copy the issuing certificate to the default location for the platform on each EC2 instance that connects to the cluster.

Linux

```
/opt/cloudhsm/etc/customerCA.crt
```

Windows

```
C:\ProgramData\Amazon\CloudHSM\customerCA.crt
```

Specify the location of the issuing certificate

With Client SDK 5, you use the configure tool to specify the location of the issuing certificate.

PKCS #11 library

To place the issuing certificate on Linux for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --hsm-ca-cert <customerCA certificate file>
```

To place the issuing certificate on Windows for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" --hsm-ca-cert <customerCA certificate file>
```

OpenSSL Dynamic Engine

To place the issuing certificate on Linux for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --hsm-ca-cert <customerCA certificate file>
```

Key Storage Provider (KSP)

To place the issuing certificate on Windows for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" --hsm-ca-cert <customerCA certificate file>
```

JCE provider

To place the issuing certificate on Linux for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
$ sudo /opt/cloudhsm/bin/configure-jce --hsm-ca-cert <customerCA certificate file>
```

To place the issuing certificate on Windows for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" --hsm-ca-cert <customerCA certificate file>
```

CloudHSM CLI

To place the issuing certificate on Linux for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
$ sudo /opt/cloudhsm/bin/configure-cli --hsm-ca-cert <customerCA certificate
 file>
```

To place the issuing certificate on Windows for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" --hsm-ca-
cert <customerCA certificate file>
```

For more information, see Configure Tool.

For more information about initializing the cluster or creating and signing the certificate, see Initialize the Cluster.

Bootstrap the Client SDK

The bootstrap process is different depending on the version of the Client SDK you're using, but you must have the IP address of one of the hardware security modules (HSM) in the cluster. You can use the IP address of any HSM attached to your cluster. After the Client SDK connects, it retrieves the IP addresses of any additional HSMs and performs load balancing and client-side key synchronization operations.

To get an IP address for the cluster

To get an IP address for an HSM (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To open the cluster detail page, in the cluster table, choose the cluster ID.
4. To get the IP address, go to the HSMs tab. For IPv4 clusters, choose an address listed under **ENI IPv4 address**. For dual-stack clusters use either the ENI IPv4 or the **ENI IPv6 address**.

To get an IP address for an HSM (AWS CLI)

- Get the IP address of an HSM by using the [describe-clusters](#) command from the AWS CLI. In the output from the command, the IP address of the HSMs are the values of `EniIp` and `EniIpV6` (if it is a dual-stack cluster).

```
$ aws cloudhsmv2 describe-clusters
{
    "Clusters": [
        { ... }
            "Hsms": [
                {
...
                    "EniIp": "10.0.0.9",
...
                },
                {
...
                    "EniIp": "10.0.1.6",
                    "EniIpV6": "2600:113f:404:be09:310e:ed34:3412:f733",
...
```

For more information about bootstrapping, see Configure Tool.

To bootstrap Client SDK 5

PKCS #11 library

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of an HSM in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 -a <HSM IP addresses>
```

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of an HSM in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" -a <HSM IP addresses>
```

OpenSSL Dynamic Engine

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of an HSM in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-dyn -a <HSM IP addresses>
```

Key Storage Provider (KSP)

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of an HSM in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" -a <HSM IP addresses>
```

JCE provider

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of an HSM in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-jce -a <HSM IP addresses>
```

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of an HSM in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" -a <HSM IP addresses>
```

CloudHSM CLI

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of the HSM(s) in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-cli -a <The ENI IPv4 / IPv6 addresses of the
 HSMs>
```

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of the HSM(s) in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" -a <The ENI
 IPv4 / IPv6 addresses of the HSMs>
```

Note

you can use the `--cluster-id` parameter in place of `-a <HSM_IP_ADDRESSES>`. To see requirements for using `--cluster-id`, see AWS CloudHSM Client SDK 5 configure tool.

To bootstrap Client SDK 3

To bootstrap a Linux EC2 instance for Client SDK 3

- Use **configure** to specify the IP address of an HSM in your cluster.

```
sudo /opt/cloudhsm/bin/configure -a <IP address>
```

To bootstrap a Windows EC2 instance for Client SDK 3

- Use **configure** to specify the IP address of an HSM in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" -a <HSM IP address>
```

For more information about configure, see ???.

Scaling HSMs in an AWS CloudHSM cluster

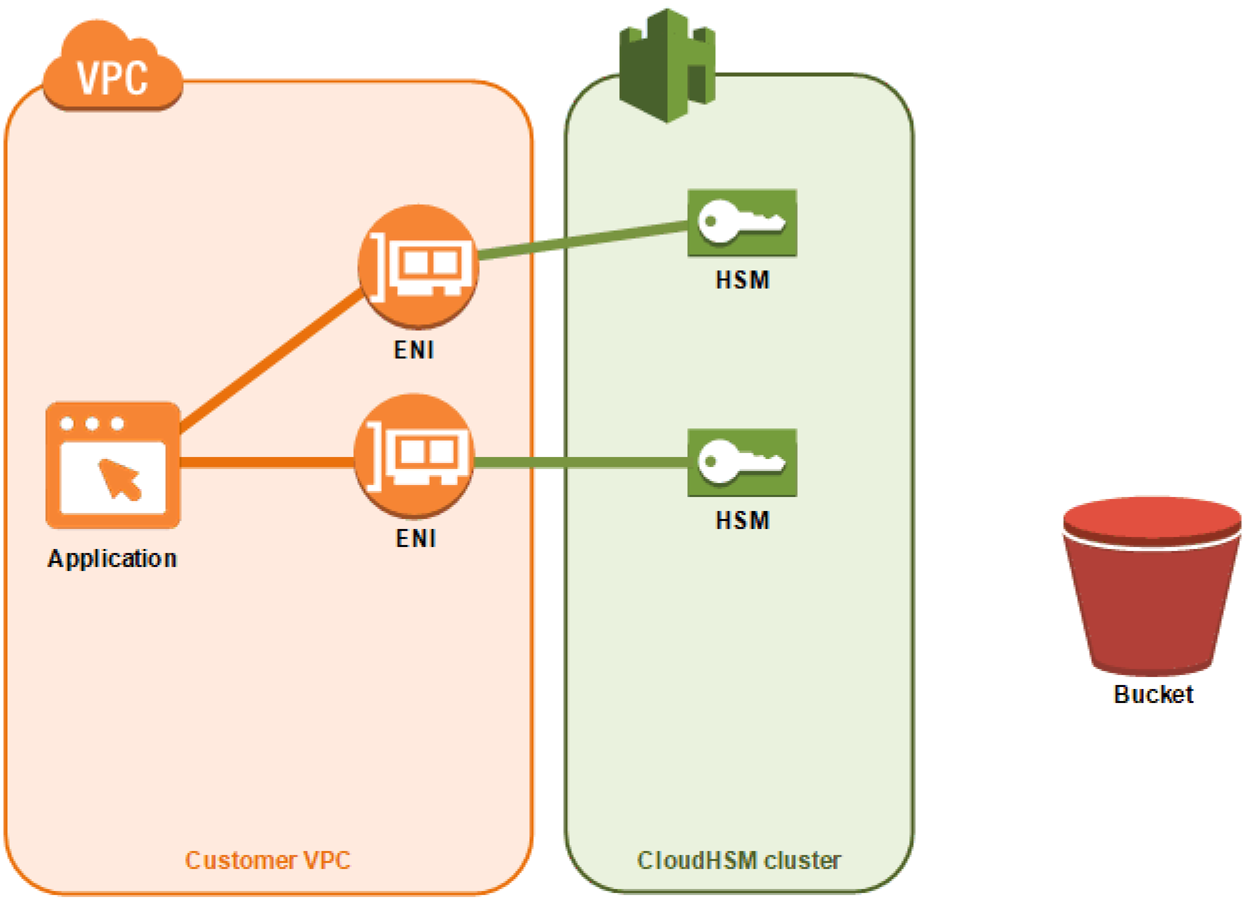
To scale up or down your AWS CloudHSM cluster, add or remove HSMs by using the AWS CloudHSM console or one of the AWS SDKs or command line tools. We recommend load testing your cluster to determine the peak load you should anticipate, and then add one more HSM to it to ensure high availability.

Topics

- Adding an HSM to an AWS CloudHSM cluster
- Removing an HSM from an AWS CloudHSM cluster

Adding an HSM to an AWS CloudHSM cluster

The following figure illustrates the events that occur when you add an HSM to a cluster.



1. You add a new HSM to a cluster. The following procedures explain how to do this from the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), and the AWS CloudHSM API.

   This is the only action that you take. The remaining events occur automatically.
2. AWS CloudHSM makes a backup copy of an existing HSM in the cluster. For more information, see Backups.
3. AWS CloudHSM restores the backup onto the new HSM. This ensures that the HSM is in sync with the others in the cluster.
4. The existing HSMs in the cluster notify the AWS CloudHSM client that there's a new HSM in the cluster.
5. The client establishes a connection to the new HSM.

To add an HSM (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Choose a cluster for the HSM that you are adding.
3. On the **HSMs** tab, choose **Create HSM**.
4. Choose an Availability Zone (AZ) for the HSM that you are creating. Then choose **Create**.

To add an HSM (AWS CLI)

- At a command prompt, issue the **create-hsm** command, specifying a cluster ID and an Availability Zone for the HSM that you are creating. If you don't know the cluster ID of your preferred cluster, issue the **describe-clusters** command. Specify the Availability Zone in the form of `us-east-2a`, `us-east-2b`, etc.

```
$ aws cloudhsmv2 create-hsm --cluster-id <cluster ID> --availability-
zone <Availability Zone>
{
    "Hsm": {
        "State": "CREATE_IN_PROGRESS",
        "ClusterId": "cluster-5a73d5qzrdh",
        "HsmId": "hsm-lgavqitns2a",
        "SubnetId": "subnet-0e358c43",
        "AvailabilityZone": "us-east-2c",
        "EniId": "eni-bab18892",
        "EniIp": "10.0.3.10",
```

```
        "EniIpV6": "2600:113f:404:be09:310e:ed34:3412:f733"
    }
}
```

To add an HSM (AWS CloudHSM API)

- Send a CreateHsm request, specifying the cluster ID and an Availability Zone for the HSM that you are creating.

Removing an HSM from an AWS CloudHSM cluster

You can remove an HSM by using the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.

To remove an HSM (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Choose the cluster that contains the HSM that you are removing.
3. On the **HSMs** tab, choose the HSM that you are removing. Then choose **Delete HSM**.
4. Confirm that you want to delete the HSM. Then choose **Delete**.

To remove an HSM (AWS CLI)

- At a command prompt, issue the **delete-hsm** command. Pass the ID of the cluster that contains the HSM that you are deleting and one of the following HSM identifiers:
  - The HSM ID (`--hsm-id`)
  - The HSM IP address (`--eni-ip`)
  - The HSM's elastic network interface ID (`--eni-id`)

  If you don't know the values for these identifiers, issue the **describe-clusters** command.

```
$ aws cloudhsmv2 delete-hsm --cluster-id <cluster ID> --eni-ip <HSM IP address>
{
    "HsmId": "hsm-lgavqitns2a"
}
```

To remove an HSM (AWS CloudHSM API)

- Send a DeleteHsm request, specifying the cluster ID and an identifier for the HSM that you are deleting.

Deleting an AWS CloudHSM cluster

Before you can delete a cluster, you must remove all HSMs from the cluster. For more information, see Removing an HSM from an AWS CloudHSM cluster.

After you remove all HSMs, you can delete a cluster by using the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.

To delete a cluster (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Choose the cluster that you are deleting. Then choose **Delete cluster**.
3. Confirm that you want to delete the cluster, then choose **Delete**.

To delete a cluster (AWS CLI)

- At a command prompt, issue the **delete-cluster** command, passing the ID of the cluster that you are deleting. If you don't know the cluster ID, issue the **describe-clusters** command.

```
$ aws cloudhsmv2 delete-cluster --cluster-id <cluster ID>
{
    "Cluster": {
        "Certificates": {
            "ClusterCertificate": "<certificate string>"
        },
        "SourceBackupId": "backup-rtq2dwi2gq6",
        "SecurityGroup": "sg-40399d28",
        "CreateTimestamp": 1504903546.035,
        "SubnetMapping": {
            "us-east-2a": "subnet-f1d6e798",
            "us-east-2c": "subnet-0e358c43",
            "us-east-2b": "subnet-40ed9d3b"
        },
        "ClusterId": "cluster-kdmrayrc7gi",
        "VpcId": "vpc-641d3c0d",
```

```
            "State": "DELETE_IN_PROGRESS",
            "HsmType": "hsm1.medium",
            "StateMessage": "The cluster is being deleted.",
            "Hsms": [],
            "BackupPolicy": "DEFAULT"
        }
}
```

To delete a cluster (AWS CloudHSM API)

- Send a DeleteCluster request, specifying the ID of the cluster that you are deleting.

Creating AWS CloudHSM clusters from backups

To restore an AWS CloudHSM cluster from a backup, follow the steps in this topic. Your cluster will contain the same users, key material, certificates, configuration, and policies that were in the backup. For more information about managing backups, see Cluster backups.

Create clusters from backups (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Choose **Create cluster**.
3. In the **Cluster configuration** section, do the following:
   a. For **VPC**, choose a VPC for the cluster that you are creating.
   b. For **AZ(s)**, choose a private subnet for each Availability Zone that you are adding to the cluster.
   c. For **Network type**, choose the IP protocol your HSMs will use for connections.
4. In the **Cluster source** section, do the following:
   a. Choose **Restore cluster from existing backup**.
   b. Choose the backup that you are restoring.
5. Choose **Next: Review**.
6. Review your cluster configuration, then choose **Create cluster**.
7. Specify how long the service should retain backups.

Accept the default retention period of 90 days or type a new value between 7 and 379 days. The service will automatically delete backups in this cluster older than the value you specify here. You can change this later. For more information, see Configure backup retention.

8. Choose **Next**.
9. (Optional) Type a tag key and an optional tag value. To add more than one tag to the cluster, choose **Add tag**.
10. Choose **Review**.
11. Review your cluster configuration, and then choose **Create cluster**.

> **Tip**
>
> To create an HSM in this cluster that contains the same users, key material, certificates, configuration, and policies that were in the backup that you restored, add an HSM to the cluster.

Create clusters from backups (AWS CLI)

To determine the backup ID, issue the **describe-backups** command.

- At a command prompt, issue the **create-cluster** command. Specify the HSM instance type, the subnet IDs of the subnets where you plan to create HSMs, and the backup ID of the backup that you are restoring.

```
$ aws cloudhsmv2 create-cluster --hsm-type hsm2m.medium \
                                --subnet-ids <subnet ID 1> <subnet ID 2> <subnet ID
 N> \
                                --source-backup-id <backup ID>
                                --mode <FIPS> \
                                --network-type <IPV4>
{
    "Cluster": {
        "HsmType": "hsm2m.medium",
        "VpcId": "vpc-641d3c0d",
        "Hsms": [],
        "State": "CREATE_IN_PROGRESS",
        "SourceBackupId": "backup-rtq2dwi2gq6",
```

```
        "BackupPolicy": "DEFAULT",
        "BackupRetentionPolicy": {
            "Type": "DAYS",
            "Value": 90
         },
        "NetworkType": "IPV4",
        "SecurityGroup": "sg-640fab0c",
        "CreateTimestamp": 1504907311.112,
        "SubnetMapping": {
            "us-east-2c": "subnet-0e358c43",
            "us-east-2a": "subnet-f1d6e798",
            "us-east-2b": "subnet-40ed9d3b"
        },
        "Certificates": {
            "ClusterCertificate": "<certificate string>"
        },
        "ClusterId": "cluster-jxhlf7644ne"
    }
}
```

Create clusters from backups (AWS CloudHSM API)

Refer to the following topic to learn how to create clusters from backups by using the API.

- CreateCluster

Cluster HSM type migration

AWS CloudHSM offers the ability to change the HSM type of an existing cluster. Review the table on this page to determine whether the HSM type modification is allowed.

For more information on the types of HSMs supported and their features please refer to HSM types in AWS CloudHSM.

> **Note**
>
> You cannot change the FIPS mode of a cluster during this operation.

From	To	Comment
hsm1.medium	**hsm2m.medium**	**Allowed**
hsm2m.medium	**hsm1.medium**	**Conditional**. You can roll back from hsm2m.medium to hsm1.medium within 24 hours of the start of a migration.

Topics

Migrating from hsm1.medium to hsm2m.medium

You can migrate your AWS CloudHSM cluster from hsm1.medium to hsm2m.medium. This topic describes the prerequisites, migration process, and rollback procedures.

Before starting the migration, make sure your application follows the recommendations in Architect your cluster for high availability. This helps avoid downtime during the process.

Overview of the hsm1.medium to hsm2m.medium migration process

You can start the migration using the AWS CloudHSM Console, the AWS CLI, or the AWS CloudHSM API. No matter where you initiate it, the AWS CloudHSM cluster migration uses the `modify-cluster` API endpoint. Once the migration starts, your entire cluster enters a limited-write mode. For more information, see Cluster limited-write mode.

To minimize impact, AWS CloudHSM changes HSMs from hsm1.medium to hsm2m.medium one at a time. The replacement HSMs maintain the same IP addresses, thereby requiring no configuration changes during or after migration.

Here's how the migration works:

1. Before migrating the first HSM, AWS CloudHSM creates a full backup of the entire cluster.
2. Using this backup, AWS CloudHSM creates a new HSM of the requested type (hsm2m.medium) to replace the first HSM.

3. Before migrating each subsequent HSM, AWS CloudHSM creates a new full backup of the entire cluster.
4. AWS CloudHSM repeats steps 3 and 4 for each HSM in the cluster, migrating one HSM at a time.
5. Each individual HSM migration takes approximately 30 minutes.

AWS CloudHSM monitors cluster health and performs validations throughout the migration process. If AWS CloudHSM detects an increase in errors or a validation check fails, it will automatically stop the migration and revert the cluster to its original HSM type. You can also roll back manually for up to 24 hours after starting the migration. Before rolling back, see HSM type rollback considerations.

Prerequisites for migrating to hsm2m.medium

Your existing AWS CloudHSM cluster must meet these requirements to migrate to hsm2m.medium. If any condition isn't met during validation checks, AWS CloudHSM automatically reverts the cluster to its original HSM type.

For a list of known migration issues, see ???

- In the last 7 days:
  - All client connections have used SDK 5.9 or higher.
    - If performing ECDSA Verify, all client connections have used SDK 5.13 or higher.
  - AWS CloudHSM instances have used only supported (and none of the deprecated) functionalities. See Deprecation notifications for details.
  - You must have used an SDK to connect with at least one HSM in the cluster in the past 7 days.
- The cluster is in an ACTIVE state.
- The cluster has 27 HSMs or fewer.
- The error rate for HSM operations doesn't increase during migration.

Note

The previous restriction which prevented customers with token key workloads from migrating has been removed.

Cluster limited-write mode

When you start the cluster migration, it enters a limited-write mode. Operations that can change the HSM state are rejected. All read operations remain unaffected.

During migration, your application receives an error from the HSM when attempting these operations:

- Token key generation and deletion (session key workloads continue to operate).
- All user creation, deletion, or modification.
- Quorum operations.
- Modification of keys within the HSM, such as changing key attributes.
- mTLS registration.

AWS CloudHSM also places your cluster in a `MODIFY_IN_PROGRESS` state during migration. During this time, you can't add or remove HSMs from the cluster.

Starting the migration

The cluster migration process replaces individual HSMs in your cluster one at a time. The duration depends on the number of HSMs in your cluster. On average, this process takes about 30 minutes per HSM. You can track progress by monitoring the HSM type of individual HSMs in the cluster to see how many have been migrated to the new type.

Console

To change the HSM type (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Select the radio button next to the ID of the cluster you want to change
3. From the Actions menu, choose `Modify HSM Type` and select the desired HSM type

This procedure puts your cluster into the `MODIFY_IN_PROGRESS` state. After migration, your cluster returns to the `ACTIVE` state.

AWS CLI

To change the HSM type (AWS CLI)

- At a command prompt, run the modify-cluster command. Specify the cluster ID and the desired HSM type.

```
$ aws cloudhsmv2 modify-cluster --cluster-id <cluster ID> --hsm-type <HSM Type>


{
    "Cluster": {
        "BackupPolicy": "DEFAULT",
        "BackupRetentionPolicy": {
            "Type": "DAYS",
            "Value": 90
         },
        "VpcId": "vpc-50ae0636",
        "SubnetMapping": {
            "us-west-2b": "subnet-49a1bc00",
            "us-west-2c": "subnet-6f950334",
            "us-west-2a": "subnet-fd54af9b"
        },
        "SecurityGroup": "sg-6cb2c216",
        "HsmType": "hsm2m.medium",
        "HsmTypeRollbackExpiration": 1730383180.000,
        "Certificates": {},
        "State": "MODIFY_IN_PROGRESS",
        "Hsms": [],
        "ClusterId": "cluster-igklspoyj5v",
        "ClusterMode": "FIPS",
        "CreateTimestamp": 1502423370.069
    }
}
```

This procedure puts your cluster into the `MODIFY_IN_PROGRESS` state. After migration, your cluster returns to the `ACTIVE` state.

AWS CloudHSM API

To change the HSM type (AWS CloudHSM API)

- Send a ModifyCluster request. Specify the cluster ID and the desired HSM type for the cluster.

This procedure puts your cluster into the `MODIFY_IN_PROGRESS` state. After migration, your cluster returns to the `ACTIVE` state.

Rolling back the migration

AWS CloudHSM monitors for elevated error rates and performs continuous validation checks throughout the migration. If AWS CloudHSM detects a decrease in service quality or any validation failures, it automatically initiates a rollback to your cluster's original HSM type. During a rollback, for each HSM in the cluster:

- AWS CloudHSM uses the backup taken at the start of that HSM's migration.
- It replaces one HSM at a time until all HSMs are returned to the original type.
- Your cluster remains in limited-write mode throughout the process.

You can roll back the migration within 24 hours of starting it. To check the rollback deadline:

1. Run the describe-clusters command.
2. Look for the `HsmTypeRollbackExpiration` value. This timestamp is your rollback deadline.

If you decide to roll back, do it before this deadline. The rollback uses the latest backup of your original HSM type.

> **Warning**
>
> Be cautious about rolling back after a migration is complete. If you complete a migration and then use AWS CloudHSM to create new keys or users, rolling back can result in data loss. See Synchronizing Data After a Rollback to learn how to mitigate data loss after a rollback.

Console

To roll back your HSM type (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Select the ID of the cluster you want to roll back.
3. From the Actions menu, choose `Modify HSM Type` and select the original HSM type

This procedure puts your cluster into the `ROLLBACK_IN_PROGRESS` state. After rollback, your cluster returns to the `ACTIVE` state.

AWS CLI

To roll back your HSM type (AWS CLI)

- At a command prompt, run the modify-cluster command. Specify the cluster ID and the original HSM type.

```
$ aws cloudhsmv2 modify-cluster --cluster-id <cluster ID> --hsm-type <HSM Type>

{
 "Cluster": {
     "BackupPolicy": "DEFAULT",
     "BackupRetentionPolicy": {
         "Type": "DAYS",
         "Value": 90
      },
     "VpcId": "vpc-50ae0636",
     "SubnetMapping": {
         "us-west-2b": "subnet-49a1bc00",
         "us-west-2c": "subnet-6f950334",
         "us-west-2a": "subnet-fd54af9b"
     },
     "SecurityGroup": "sg-6cb2c216",
     "HsmType": "hsm1.medium",
     "HsmTypeRollbackExpiration": 1730383180.000,
     "Certificates": {},
     "State": "ROLLBACK_IN_PROGRESS",
     "Hsms": [],
     "ClusterId": "cluster-igklspoyj5v",
     "ClusterMode": "FIPS",
     "CreateTimestamp": 1502423370.069
```

```
  }
}
```

This procedure puts your cluster into the ROLLBACK_IN_PROGRESS state. After rollback, your cluster returns to the ACTIVE state.

AWS CloudHSM API

To roll back your HSM type (AWS CloudHSM API)

- Send a ModifyCluster request. Specify the cluster ID and the original HSM type for the cluster.

This procedure puts your cluster into the ROLLBACK_IN_PROGRESS state. After rollback, your cluster returns to the ACTIVE state.

Synchronizing data after a rollback

During migration, HSMs are in limited-write mode, preventing changes to HSM state. If you roll back during this time (while the cluster is MODIFY_IN_PROGRESS), it results in a cluster with content identical to the original cluster.

After your cluster returns to the ACTIVE state, limited-write mode is lifted. If you create a key or user while in ACTIVE state and then roll back, that key or user won't be present in your rolled back cluster.

To resolve this, use the CloudHSM CLI's key replicate command to replicate a key between two clusters. If you haven't installed it, see the instructions in ???.

To synchronize keys after rollback

Follow these steps after completing the rollback. We'll use these terms:

- "cluster-1": Your rolled back cluster (now hsm1.medium)
- "cluster-2": A new temporary hsm2m.medium cluster that you will create

1. Create a new hsm2m.medium cluster (cluster-2) using the latest hsm2m.medium backup from cluster-1:

```
aws cloudhsmv2 create-cluster --hsm-type hsm2m.medium \
                                --subnet-ids <subnet ID 1> <subnet ID 2> <subnet ID
 N> \
                                --source-backup-id <backup ID>
                                --mode <FIPS>
```

2. Create an HSM in cluster-2:

```
aws cloudhsmv2 create-hsm --cluster-id <cluster-2 ID>
```

3. List keys in cluster-2 that need replication:

```
cloudhsm-cli key list --cluster-id <cluster-2 ID>
```

4. Replicate each key from cluster-2 to cluster-1:

```
cloudhsm-cli key replicate --source-cluster-id <cluster-2 ID> \
                        --destination-cluster-id <cluster-1 ID> \
                        --filter attr.label=<key ID>
```

5. Repeat step 4 for each key that needs copying.
6. Delete the HSM in cluster-2:

```
aws cloudhsmv2 delete-hsm --cluster-id <cluster-2 ID> --hsm-id <HSM ID>
```

7. Delete cluster-2:

```
aws cloudhsmv2 delete-cluster --cluster-id <cluster-2 ID>
```

HSM users in AWS CloudHSM

Before you can use your AWS CloudHSM cluster for cryptoprocessing, you must create users and keys on the hardware security modules (HSM) in your cluster.

> **Note**
>
> HSM users are distinct from IAM users. IAM users who have the correct credentials can create HSMs by interacting with resources through the AWS API. After the HSM is created, you must use HSM user credentials to authenticate operations on the HSM.

In AWS CloudHSM, you must use CloudHSM CLI or CloudHSM Management Utility (CMU) command line tools to create and manage the users on your HSM. CloudHSM CLI is designed to be used with the latest SDK version series, while the CMU is designed to be used with the previous SDK version series.

See the following topics for more information about managing HSM users in AWS CloudHSM. You can also learn how to use quorum authentication (also known as M of N access control).

Topics

- HSM user management with CloudHSM CLI
- HSM user management with CloudHSM Management Utility (CMU)

HSM user management with CloudHSM CLI

To manage hardware security module (HSM) users in AWS CloudHSM, you must log in to the HSM with the user name and password of an admin. Only admins can manage users. The HSM contains a default admin named admin. You set the password for admin when you activated the cluster.

This topic provides step-by-step instruction on and detail about managing HSM users with CloudHSM CLI.

Topics

- Prerequisites for user management in CloudHSM CLI
- HSM user types for CloudHSM CLI

Prerequisites for user management in CloudHSM CLI

Before you use CloudHSM CLI to manage hardware security modules (HSM) users in AWS CloudHSM, you must complete these prerequisites. The following topics describe getting started with the CloudHSM CLI.

Topics

Get the IP address of an HSM in AWS CloudHSM

To use CloudHSM CLI, you must use the configure tool to update the local configuration. For instructions on running the configure tool with CloudHSM CLI, see Getting started with AWS CloudHSM Command Line Interface (CLI). The `-a` parameter requires you to add the IP address of an HSM in your cluster. If you have multiple HSMs, you can use any IP address. This ensures CloudHSM CLI can propagate any changes you make across the entire cluster. Remember that CloudHSM CLI uses its local file to track cluster information. If the cluster has changed since the last time you used CloudHSM CLI from a particular host, you must add those changes to the local configuration file stored on that host. Never remove an HSM while you're using CloudHSM CLI.

To get an IP address for an HSM (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To open the cluster detail page, in the cluster table, choose the cluster ID.

4. To get the IP address, go to the HSMs tab. For IPv4 clusters, choose an address listed under **ENI IPv4 address**. For dual-stack clusters use either the ENI IPv4 or the **ENI IPv6 address**.

To get an IP address for an HSM (AWS CLI)

- Get the IP address of an HSM by using the **describe-clusters** command from the AWS CLI. In the output from the command, the IP address of the HSMs are the values of `EniIp` and `EniIpV6` (if it is a dual-stack cluster).

```
$ aws cloudhsmv2 describe-clusters
{
    "Clusters": [
        { ... }
            "Hsms": [
                {
...
                    "EniIp": "10.0.0.9",
...
                },
                {
...
                    "EniIp": "10.0.1.6",
                    "EniIpV6": "2600:113f:404:be09:310e:ed34:3412:f733",
...
```

Download CloudHSM CLI

The latest version of CloudHSM CLI is available for HSM user management tasks for Client SDK 5. To download and install CloudHSM CLI, follow the instructions in Install and configure CloudHSM CLI.

HSM user types for CloudHSM CLI

Most operations that you perform on the hardware security module (HSM) require the credentials of an AWS CloudHSM *HSM user*. The HSM authenticates each HSM user and each HSM user has a *type* that determines which operations you can perform on the HSM as that user.

> **Note**
>
> HSM users are distinct from IAM users. IAM users who have the correct credentials can create HSMs by interacting with resources through the AWS API. After the HSM is created, you must use HSM user credentials to authenticate operations on the HSM.

User types

- Unactivated admin
- Admin
- Crypto user (CU)
- Appliance user (AU)

Unactivated admin

In CloudHSM CLI, The unactivated admin is a temporary user that exists only on the first HSM in an AWS CloudHSM cluster that has never been activated. To activate a cluster, run the **cluster activate** command in CloudHSM CLI. After running this command, unactivated admin are prompted to change the password. After changing the password, the unactivated admin becomes an admin.

Admin

In CloudHSM CLI, admin can perform user management operations. For example, they can create and delete users and change user passwords. For more information about admins, see the HSM user permissions table for CloudHSM CLI.

Crypto user (CU)

A crypto user (CU) can perform the following key management and cryptographic operations.

- **Key management** – Create, delete, share, import, and export cryptographic keys.
- **Cryptographic operations** – Use cryptographic keys for encryption, decryption, signing, verifying, and more.

For more information, see the HSM user permissions table for CloudHSM CLI.

Appliance user (AU)

The appliance user (AU) can perform cloning and synchronization operations on your cluster's HSMs. AWS CloudHSM uses the AU to synchronize the HSMs in an AWS CloudHSM cluster. The AU exists on all HSMs provided by AWS CloudHSM, and has limited permissions. For more information, see the HSM user permissions table for CloudHSM CLI.

AWS cannot perform any operations on your HSMs . AWS cannot view or modify your users or keys and cannot perform any cryptographic operations using those keys.

HSM user permissions table for CloudHSM CLI

The following table lists hardware security module (HSM) operations sorted by the type of HSM user or session that can perform the operation in AWS CloudHSM.

	Admin	Crypto User (CU)	Appliance User (AU)	Unauthenticated Session
Get basic cluster info[1]	Yes	Yes	Yes	Yes
Change own password	Yes	Yes	Yes	Not applicable
Change any user's password	Yes	No	No	No
Add, remove users	Yes	No	No	No

	Admin	Crypto User (CU)	Appliance User (AU)	Unauthent icated Session
Get sync status[2]	Yes	Yes	Yes	No
Extract, insert masked objects[3]	Yes	Yes	Yes	No
Key managemen t functions[4]	No	Yes	No	No
Encrypt, decrypt	No	Yes	No	No
Sign, verify	No	Yes	No	No
Generate digests and HMACs	No	Yes	No	No

- [1] Basic cluster information includes the number of HSMs in the cluster and each HSM's IP address, model, serial number, device ID, firmware ID, etc.

- [2] The user can get a set of digests (hashes) that correspond to the keys on the HSM. An application can compare these sets of digests to understand the synchronization status of HSMs in a cluster.
- [3] Masked objects are keys that are encrypted before they leave the HSM. They cannot be decrypted outside of the HSM. They are only decrypted after they are inserted into an HSM that is in the same cluster as the HSM from which they were extracted. An application can extract and insert masked objects to synchronize the HSMs in a cluster.
- [4] Key management functions include creating, deleting, wrapping, unwrapping, and modifying the attributes of keys.

Create an HSM user admin using CloudHSM CLI

Follow these steps to create a hardware security module (HSM) admin user using the CloudHSM CLI.

1. Use the following command to start CloudHSM CLI interactive mode.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Use the **login** command and log in to the cluster as the admin.

   ```
   aws-cloudhsm > login --username <username> --role admin
   ```

3. The system prompts you for your password. Enter the password, and the output shows that the command was successful.

   ```
   Enter password:
   {
     "error_code": 0,
     "data": {
       "username": "<username>",
       "role": "admin"
     }
   ```

```
}
```

4. Enter the following command to create an admin:

```
aws-cloudhsm > user create --username <username> --role admin
```

5. Enter the password for the new user.
6. Re-enter the password to confirm the password you entered is correct.

Create an HSM crypto user using CloudHSM CLI

Follow these steps to create a hardware security module (HSM) crypto user (CU) using the CloudHSM CLI.

1. Use the following command to start CloudHSM CLI interactive mode.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Use the **login** command and log in to the cluster as the admin.

```
aws-cloudhsm > login --username <username> --role admin
```

3. The system prompts you for your password. Enter the password, and the output shows that the command was successful.

```
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "<USERNAME>",
    "role": "admin"
  }
}
```

4. Enter the following command to create a crypto user:

```
aws-cloudhsm > user create --username <username> --role crypto-user
```

5. Enter the password for the new crypto user.
6. Re-enter the password to confirm the password you entered is correct.

List all HSM users in the cluster using CloudHSM CLI

Use **user list** command in the CloudHSM CLI to list all the users in the AWS CloudHSM cluster. You do not have to log in to run **user list**. All user types can list users.

Follow these steps to list all users in the cluster

1. Use the following command to start CloudHSM CLI interactive mode.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Enter the following command to list all the users in the cluster:

```
aws-cloudhsm > user list
```

For more information about **user list**, see user list.

Change HSM user passwords using CloudHSM CLI

Use the **user change-password** command in the CloudHSM CLI to change a hardware security module (HSM) user's password.

User types and passwords are case sensitive, but user names are not case sensitive.

Admin, crypto user (CU), and appliance user (AU) can change their own password. To change the password of another user, you must log in as an admin. You cannot change the password of a user who is currently logged in.

To change your own password

1. Use the following command to start CloudHSM CLI interactive mode.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Use the **login** command and log in as the user with the password you want to change.

```
aws-cloudhsm > login --username <username> --role <role>
```

3. Enter the user's password.

```
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "<username>",
    "role": "<role>"
  }
}
```

4. Enter the **user change-password** command.

```
aws-cloudhsm > user change-password --username <username> --role <role>
```

5. Enter the new password.
6. Re-enter the new password.

To change the password of another user

1. Use the following command to start CloudHSM CLI interactive mode.

Linux

```
$ /opt/cloudhsm/bin/cloudhsm-cli interactive
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
```

2. Using CloudHSM CLI, log in as an admin.

```
aws-cloudhsm > login --username <admin> --role admin
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "<admin>",
    "role": "admin"
  }
}
```

3. Enter the **user change-password** command along with the username of the user whose password you want to change.

```
aws-cloudhsm > user change-password --username <username> --role <role>
```

4. Enter the new password.

5. Re-enter the new password.

For more information about **user change-password**, see user change-password.

Delete HSM users using CloudHSM CLI

Use **user delete** in the CloudHSM CLI to delete a hardware security module (HSM) user. You must log in as an admin to delete another user.

> **Tip**
>
> You can't delete crypto users (CU) that own keys.

To delete a user

1. Use the following command to start CloudHSM CLI interactive mode.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Use the **login** command and log in to the cluster as the admin.

   ```
   aws-cloudhsm > login --username <username> --role admin
   ```

3. The system prompts you for your password. Enter the password, and the output shows that the command was successful.

   ```
   Enter password:
   {
     "error_code": 0,
     "data": {
       "username": "<username>",
       "role": "admin"
     }
   }
   ```

4. Use the **user delete** command to delete the user.

   ```
   aws-cloudhsm > user delete --username <username> --role <role>
   ```

For more information about **user delete**, see deleteUser.

Manage MFA for HSM users using CloudHSM CLI

For increased security, you can configure multi-factor authentication (MFA) for users to help protect the AWS CloudHSM cluster.

When you log in to a cluster with an MFA enabled hardware security module (HSM) user account, you provide the CloudHSM CLI with your password—the first factor, what you know—and CloudHSM CLI provides you with a token and prompts you to have the token signed.

To provide the second factor—what you have—you sign the token with a private key from a key pair you've already created and associated with the HSM user. To access the cluster, you provide the signed token to CloudHSM CLI.

For more information on setting up MFA for a user see [Set up MFA for CloudHSM CLI]

The following topics provide more information about working with quorum authentication in AWS CloudHSM.

Topics

Quorum authentication and MFA in AWS CloudHSM clusters using CloudHSM CLI

The AWS CloudHSM cluster uses the same key for quorum authentication and for multi-factor authentication (MFA). This means a user with MFA enabled is effectively registered for MofN or quorum access control. To successfully use MFA and quorum authentication for the same HSM user, consider the following points:

- If you are using quorum authentication for a user today, you should use the same key pair you created for the quorum user to enable MFA for the user.
- If you add the MFA requirement for a non-MFA user who is not a quorum authentication user, then you register that user as a quorum (MofN) registered user with MFA authentication.
- If you remove the MFA requirement or change the password for an MFA user who is also a registered quorum authentication user, you will also remove the user's registration as a quorum (MofN) user.

- If you remove the MFA requirement or change the password for an MFA user who is also a quorum authentication user, *but you still want that user to participate in quorum authentication*, then you must register that user again as a Quorum (MofN) user.

For more information about quorum authentication, see Manage quorum authentication (M of N).

MFA key pair requirements for AWS CloudHSM using CloudHSM CLI

To enable multi-factor authentication (MFA) for a hardware security module (HSM) user in AWS CloudHSM, you can create a new key pair or use an existing key that meets the following requirements:

- **Key type:** Asymmetric
- **Key usage:** Sign and verify
- **Key spec:** RSA_2048
- **Signing algorithm includes:** sha256WithRSAEncryption

> **Note**
>
> If you are using quorum authentication or plan to use quorum authentication, see Quorum authentication and MFA in AWS CloudHSM clusters using CloudHSM CLI

You can use CloudHSM CLI and the key pair to create a new admin user with MFA enabled.

Set up MFA for CloudHSM CLI

Follow these steps to set up multi-factor authentication (MFA) for CloudHSM CLI.

1. To setup MFA using the Token Sign Strategy you must first generate a 2048 bit RSA private key and associated public key.

```
$ openssl genrsa -out officer1.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
...........................................................+++++
....................................................................+++++
e is 65537 (0x010001)

$ openssl rsa -in officer1.key -outform PEM -pubout -out officer1.pub
```

```
writing RSA key
```

2. Use the following command to start the CLI in interactive mode.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

3. Using the CloudHSM CLI, login to your user account.

   ```
   aws-cloudhsm > login --username <admin> --role <admin> --cluster-id <cluster ID>
   Enter password:
   {
     "error_code": 0,
     "data": {
       "username": "<admin>",
       "role": "<admin>"
     }
   }
   ```

4. Next, execute the command to change you MFA strategy. You must provide the parameter `--token`. This parameter specifies a file that will have unsigned tokens written to it.

   ```
   aws-cloudhsm > user change-mfa token-sign --token unsigned-tokens.json --
   username <username> --role crypto-user --change-quorum
   Enter password:
   Confirm password:
   ```

5. You now have a file with unsigned tokens that need to be signed: `unsigned-tokens.json`. The number of tokens in this file depends on the number of HSMs in your cluster. Each token represents one HSM. This file is JSON formatted and contains tokens that need to be signed to prove you have a private key.

   ```
   $ cat unsigned-tokens.json
   {
     "version": "2.0",
     "tokens": [
   {
   ```

```
    {
      "unsigned": "Vtf/9QOFY45v/E1osvpEMr59JsnP/hLDm4ItOO2vqL8=",
      "signed": ""
    },
    {
      "unsigned": "wVbC0/5IKwjyZK2NBpdFLyI7BiayZ24YcdUdlcxLwZ4=",
      "signed": ""
    },
    {
      "unsigned": "z6aW9RzErJBL5KqFG5h8lhTVt9oLbxppjod0Ebysydw=",
      "signed": ""
    }
  ]
}
```

6. The next step is to sign these tokens with the private key created in step 1. Place the signatures back in the file. First, you have to extract and decode the base64 encoded tokens.

```
$ echo "Vtf/9QOFY45v/E1osvpEMr59JsnP/hLDm4ItOO2vqL8=" > token1.b64
$ echo "wVbC0/5IKwjyZK2NBpdFLyI7BiayZ24YcdUdlcxLwZ4=" > token2.b64
$ echo "z6aW9RzErJBL5KqFG5h8lhTVt9oLbxppjod0Ebysydw=" > token3.b64
$ base64 -d token1.b64 > token1.bin
$ base64 -d token2.b64 > token2.bin
$ base64 -d token3.b64 > token3.bin
```

7. Now, you have binary tokens that you can sign using the RSA private key created in step 1.

```
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token1.bin \
      -out token1.sig.bin
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token2.bin \
      -out token2.sig.bin
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
```

```
      -in token3.bin \
      -out token3.sig.bin
```

8. Now, you have binary signatures of the tokens. You must encode them using base64, and then place them back in your token file.

```
$ base64 -w0 token1.sig.bin > token1.sig.b64
$ base64 -w0 token2.sig.bin > token2.sig.b64
$ base64 -w0 token3.sig.bin > token3.sig.b64
```

9. Finally, you can copy and paste the base64 values back into your token file:


```
{
  "version": "2.0",
  "tokens": [
    {
      "unsigned": "1jqwxb9bJOUUQLiNb7mxXS1uBJsEXh0B9nj05BqnPsE=",
      "signed": "eiw3fZeCKIY50C4zPeg9Rt90M1Qlq3WlJh6Yw7xXm4nF6e9ETLE39+9M
+rUqDWMRZjaBfaMbg5d9yDkz5p13U7ch2tlF9LoYabsWutkT014KRq/rcYMvFsU9n/Ey/
TK0PVaxLN42X+pebV4juwMhN4mK4CzdFAJgM+UGBOj4yB9recpOBB9K8QFSpJZALSEdDgUc/
mS1eDq3rU0int6+4NKuLQjpR
+LSEIWRZ6g6+MND2vXGskxHjadCQ09L7Tz8VcWjKDbxJcBiGKvkqyozl9zrGo8fA3WHBmwiAgS61Merx77ZGY4PFR37
YMSC14prCN15DtMRv2xA1SGSb4w=="
    },
    {
      "unsigned": "LMMFc34ASPnvNPFzBbMbr9FProS/Zu2P8zF/xzk5hVQ=",
      "signed": "HBImKnHmw+6R2TpFEpfiAg4+hu2pFNwn43ClhKPkn2higbEhUD0JVi
+4MerSyvU/NN79iWVxDvJ9Ito+jpiRQjTfTGEoIteyuAr1v/Bzh+HjmrO53OQpZaJ/VXGIgApD0myuu/
ZGNKQTCSkkL7+V81FG7yR1Nm22jUeGa735zvm/E+cenvZdy0VVx6A7WeWrl3JEKKBweHbi+7BwbaW
+PTdCuIRd4Ug76Sy+cFhsvcG1k7cMwDh8MgXzIZ2m1f/hdy2j8qAxORTLlmwyUOYvPYOvUhc
+s83hx36QpGwGcD7RA0bPT5OrTx7PHd0N1CL+Wwy91We8yIOFBS6nxo1R7w=="
    },
    {
      "unsigned": "dzeHbwhiVXQqcUGj563z51/7sLUdxjL93SbOUyZRjH8=",
      "signed": "VgQPvrTsvGljVBFxHnswduq16x8ZrnxfcYVYGf/
N7gEzI4At3GDs2EVZWTRdvS0uGHdkFYp1apHgJZ7PDVmGcTkIXVD2lFYppcgNlSzkYlftr5EOjqS9ZjYEqgGuB4g//
MxaBaRbJai/6BlcE92NIdBusTtreIm3yTpjIXNAVoeRSnkfuw7wZcL96QoklNb1WUuSHw
+psUyeIVtIwFMHEfFoRC0t
+VhmnlnFnkjGPb9W3Aprw2dRRvFM3R2ZTDvMCiOYDzUCd43GftGq2LfxH3qSD51oFHglHQVOY0jyVzzlAvub5HQdtOQ
    }
  ]
}
```

10. Now that your token file has all the required signatures, you can proceed. Enter the name of the file containing the signed tokens and press the enter key. Finally, enter the path of your public key.

```
Enter signed token file path (press enter if same as the unsigned token file):
Enter public key PEM file path:officer1.pub
{
  "error_code": 0,
  "data": {
    "username": "<username>",
    "role": "crypto-user"
  }
}
```

Now you have setup your user with MFA.

```
{
    "username": "<username>",
    "role": "crypto-user",
    "locked": "false",
    "mfa": [
      {
        "strategy": "token-sign",
        "status": "enabled"
      }
    ],
    "cluster-coverage": "full"
},
```

Create users with MFA enabled for CloudHSM CLI

Follow these steps to create AWS CloudHSM users with multi-factor authentication (MFA) enabled.

1. Use CloudHSM CLI to log in to the HSM as an admin.
2. Use the **user create** command to create a user of your choice. Then follow the steps in Set up MFA for CloudHSM CLI to setup MFA for the user.

Log in users with MFA enabled for CloudHSM CLI

Follow these steps to log in AWS CloudHSM users with multi-factor authentication (MFA) enabled.

1. Use the **login mfa-token-sign** command in CloudHSM CLI to start the login process with MFA for a user who has MFA enabled.

```
aws-cloudhsm > login --username <username> --role <role> mfa-token-sign --
token <unsigned-tokens.json>
Enter password:
```

2. Enter your password. You will then be prompted to enter the path to the token file which contains unsigned/signed token pairs, where signed tokens are those generated by using your private key.

```
aws-cloudhsm > login --username <username> --role <role> mfa-token-sign --
token <unsigned-tokens.json>
Enter password:
Enter signed token file path (press enter if same as the unsigned token file):
```

3. While prompted to enter the signed token file path, you can inspect the unsigned token file in a separate terminal. Identify the file with unsigned tokens that need to be signed: *<unsigned-tokens.json>*. The number of tokens in this file depends on the number of HSMs in your cluster. Each token represents one HSM. This file is JSON formatted and contains tokens that need to be signed to prove you have a private key.

```
$ cat <unsigned-tokens.json>
{
  "version": "2.0",
  "tokens": [
    {
      "unsigned": "Vtf/9QOFY45v/E1osvpEMr59JsnP/hLDm4ItOO2vqL8=",
      "signed": ""
    },
    {
      "unsigned": "wVbC0/5IKwjyZK2NBpdFLyI7BiayZ24YcdUdlcxLwZ4=",
      "signed": ""
    },
    {
      "unsigned": "z6aW9RzErJBL5KqFG5h8lhTVt9oLbxppjod0Ebysydw=",
      "signed": ""
    }
```

```
  ]
}
```

4. Sign the unsigned tokens with the private key created in step 2. First you have to extract and decode the base64 encoded tokens.

```
$ echo "Vtf/9QOFY45v/E1osvpEMr59JsnP/hLDm4ItOO2vqL8=" > token1.b64
$ echo "wVbC0/5IKwjyZK2NBpdFLyI7BiayZ24YcdUdlcxLwZ4=" > token2.b64
$ echo "z6aW9RzErJBL5KqFG5h8lhTVt9oLbxppjod0Ebysydw=" > token3.b64
$ base64 -d token1.b64 > token1.bin
$ base64 -d token2.b64 > token2.bin
$ base64 -d token3.b64 > token3.bin
```

5. You now have binary tokens. Sign them using the RSA private key you previously created in step 1 of MFA setup.

```
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token1.bin \
      -out token1.sig.bin
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token2.bin \
      -out token2.sig.bin
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token3.bin \
      -out token3.sig.bin
```

6. You now have binary signatures of the tokens. Encode them using base64, and place them back in your token file.

```
$ base64 -w0 token1.sig.bin > token1.sig.b64
$ base64 -w0 token2.sig.bin > token2.sig.b64
$ base64 -w0 token3.sig.bin > token3.sig.b64
```

7. Finally, copy and paste the base64 values back into your token file:

```
{
  "version": "2.0",
  "tokens": [
    {
      "unsigned": "1jqwxb9bJOUUQLiNb7mxXS1uBJsEXh0B9nj05BqnPsE=",
      "signed": "eiw3fZeCKIY50C4zPeg9Rt90M1Qlq3WlJh6Yw7xXm4nF6e9ETLE39+9M
+rUqDWMRZjaBfaMbg5d9yDkz5p13U7ch2tlF9LoYabsWutkT014KRq/rcYMvFsU9n/Ey/
TK0PVaxLN42X+pebV4juwMhN4mK4CzdFAJgM+UGBOj4yB9recpOBB9K8QFSpJZALSEdDgUc/
mS1eDq3rU0int6+4NKuLQjpR
+LSEIWRZ6g6+MND2vXGskxHjadCQ09L7Tz8VcWjKDbxJcBiGKvkqyozl9zrGo8fA3WHBmwiAgS61Merx77ZGY4PFR37
YMSC14prCN15DtMRv2xA1SGSb4w=="
    },
    {
      "unsigned": "LMMFc34ASPnvNPFzBbMbr9FProS/Zu2P8zF/xzk5hVQ=",
      "signed": "HBImKnHmw+6R2TpFEpfiAg4+hu2pFNwn43ClhKPkn2higbEhUD0JVi
+4MerSyvU/NN79iWVxDvJ9Ito+jpiRQjTfTGEoIteyuAr1v/Bzh+HjmrO53OQpZaJ/VXGIgApD0myuu/
ZGNKQTCSkkL7+V81FG7yR1Nm22jUeGa735zvm/E+cenvZdy0VVx6A7WeWrl3JEKKBweHbi+7BwbaW
+PTdCuIRd4Ug76Sy+cFhsvcG1k7cMwDh8MgXzIZ2m1f/hdy2j8qAxORTLlmwyUOYvPYOvUhc
+s83hx36QpGwGcD7RA0bPT5OrTx7PHd0N1CL+Wwy91We8yIOFBS6nxo1R7w=="
    },
    {
      "unsigned": "dzeHbwhiVXQqcUGj563z51/7sLUdxjL93SbOUyZRjH8=",
      "signed": "VgQPvrTsvGljVBFxHnswduq16x8ZrnxfcYVYGf/
N7gEzI4At3GDs2EVZWTRdvS0uGHdkFYp1apHgJZ7PDVmGcTkIXVD2lFYppcgNlSzkYlftr5EOjqS9ZjYEqgGuB4g//
MxaBaRbJai/6BlcE92NIdBusTtreIm3yTpjIXNAVoeRSnkfuw7wZcL96QoklNb1WUuSHw
+psUyeIVtIwFMHEfFoRC0t
+VhmnlnFnkjGPb9W3Aprw2dRRvFM3R2ZTDvMCiOYDzUCd43GftGq2LfxH3qSD51oFHglHQVOY0jyVzzlAvub5HQdtOQ
    }
  ]
}
```


8. Now that your token file has all the required signatures, you can proceed. Enter the name of the file containing the signed tokens and press the enter key. You should now log in successfully.

```
aws-cloudhsm > login --username <username> --role <role> mfa-token-sign --
token <unsigned-tokens.json>
Enter password:
Enter signed token file path (press enter if same as the unsigned token file):
{
  "error_code": 0,
  "data": {
```

```
    "username": "<username>",
    "role": "<role>"
  }
}
```

Rotate keys for users with MFA enabled for CloudHSM CLI

Follow these steps to rotate keys for AWS CloudHSM users with multi-factor authentication (MFA) enabled.

<result>

You have signed the generated JSON formatted token file with your private key and registered a new MFA public key.
</result>

1. Use CloudHSM CLI to log in to the HSM as any admin or as the specific user who has MFA enabled (see Log in users with MFA enabled for details).
2. Next, execute the command to change you MFA strategy. You must provide the parameter **--token**. This parameter specifies a file that will have unsigned tokens written to it.

```
aws-cloudhsm > user change-mfa token-sign --token unsigned-tokens.json --
username <username> --role crypto-user --change-quorum
Enter password:
Confirm password:
```

3. Identify the file with unsigned tokens that need to be signed: `unsigned-tokens.json`. The number of tokens in this file depends on the number of HSMs in your cluster. Each token represents one HSM. This file is JSON formatted and contains tokens that need to be signed to prove you have a private key. This will be the new private key from the new RSA public/private key pair you wish to use for rotating the currently registered public key.

```
$ cat unsigned-tokens.json
{
  "version": "2.0",
  "tokens": [
    {
      "unsigned": "Vtf/9QOFY45v/E1osvpEMr59JsnP/hLDm4ItOO2vqL8=",
      "signed": ""
    },
```

```
    {
      "unsigned": "wVbC0/5IKwjyZK2NBpdFLyI7BiayZ24YcdUdlcxLwZ4=",
      "signed": ""
    },
    {
      "unsigned": "z6aW9RzErJBL5KqFG5h8lhTVt9oLbxppjod0Ebysydw=",
      "signed": ""
    }
  ]
}
```

4. Sign these tokens with the private key you previously created during setup. First we have to extract and decode the base64 encoded tokens.

```
$ echo "Vtf/9QOFY45v/E1osvpEMr59JsnP/hLDm4ItOO2vqL8=" > token1.b64
$ echo "wVbC0/5IKwjyZK2NBpdFLyI7BiayZ24YcdUdlcxLwZ4=" > token2.b64
$ echo "z6aW9RzErJBL5KqFG5h8lhTVt9oLbxppjod0Ebysydw=" > token3.b64
$ base64 -d token1.b64 > token1.bin
$ base64 -d token2.b64 > token2.bin
$ base64 -d token3.b64 > token3.bin
```

5. You now have binary tokens. Sign them using the RSA private key you previously created during setup.

```
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token1.bin \
      -out token1.sig.bin
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token2.bin \
      -out token2.sig.bin
$ openssl pkeyutl -sign \
      -inkey officer1.key \
      -pkeyopt digest:sha256 \
      -keyform PEM \
      -in token3.bin \
      -out token3.sig.bin
```

6. You now have binary signatures of the tokens. Encode them using base64, and place them back in your token file.

```
$ base64 -w0 token1.sig.bin > token1.sig.b64
$ base64 -w0 token2.sig.bin > token2.sig.b64
$ base64 -w0 token3.sig.bin > token3.sig.b64
```

7. Finally, copy and paste the base64 values back into your token file:


```
{
  "version": "2.0",
  "tokens": [
    {
      "unsigned": "1jqwxb9bJOUUQLiNb7mxXS1uBJsEXh0B9nj05BqnPsE=",
      "signed": "eiw3fZeCKIY50C4zPeg9Rt90M1Qlq3WlJh6Yw7xXm4nF6e9ETLE39+9M
+rUqDWMRZjaBfaMbg5d9yDkz5p13U7ch2tlF9LoYabsWutkT014KRq/rcYMvFsU9n/Ey/
TK0PVaxLN42X+pebV4juwMhN4mK4CzdFAJgM+UGBOj4yB9recpOBB9K8QFSpJZALSEdDgUc/
mS1eDq3rU0int6+4NKuLQjpR
+LSEIWRZ6g6+MND2vXGskxHjadCQ09L7Tz8VcWjKDbxJcBiGKvkqyozl9zrGo8fA3WHBmwiAgS61Merx77ZGY4PFR37
YMSC14prCN15DtMRv2xA1SGSb4w=="
    },
    {
      "unsigned": "LMMFc34ASPnvNPFzBbMbr9FProS/Zu2P8zF/xzk5hVQ=",
      "signed": "HBImKnHmw+6R2TpFEpfiAg4+hu2pFNwn43ClhKPkn2higbEhUD0JVi
+4MerSyvU/NN79iWVxDvJ9Ito+jpiRQjTfTGEoIteyuAr1v/Bzh+HjmrO53OQpZaJ/VXGIgApD0myuu/
ZGNKQTCSkkL7+V81FG7yR1Nm22jUeGa735zvm/E+cenvZdy0VVx6A7WeWrl3JEKKBweHbi+7BwbaW
+PTdCuIRd4Ug76Sy+cFhsvcG1k7cMwDh8MgXzIZ2m1f/hdy2j8qAxORTLlmwyUOYvPYOvUhc
+s83hx36QpGwGcD7RA0bPT5OrTx7PHd0N1CL+Wwy91We8yIOFBS6nxo1R7w=="
    },
    {
      "unsigned": "dzeHbwhiVXQqcUGj563z51/7sLUdxjL93SbOUyZRjH8=",
      "signed": "VgQPvrTsvGljVBFxHnswduq16x8ZrnxfcYVYGf/
N7gEzI4At3GDs2EVZWTRdvS0uGHdkFYp1apHgJZ7PDVmGcTkIXVD2lFYppcgNlSzkYlftr5EOjqS9ZjYEqgGuB4g//
MxaBaRbJai/6BlcE92NIdBusTtreIm3yTpjIXNAVoeRSnkfuw7wZcL96QoklNb1WUuSHw
+psUyeIVtIwFMHEfFoRC0t
+VhmnlnFnkjGPb9W3Aprw2dRRvFM3R2ZTDvMCiOYDzUCd43GftGq2LfxH3qSD51oFHglHQVOY0jyVzzlAvub5HQdtOQ
    }
  ]
}
```


8. Now that your token file has all the required signatures, you can proceed. Enter the name of the file containing the signed tokens and press the enter key. Finally, enter the path of your new public key. Now you will see the following as part of the output of [user list](#).

```
Enter signed token file path (press enter if same as the unsigned token file):
Enter public key PEM file path:officer1.pub
{
  "error_code": 0,
  "data": {
    "username": "<username>",
    "role": "crypto-user"
  }
}
```

Now we have setup our user with MFA.

```
{
    "username": "<username>",
    "role": "crypto-user",
    "locked": "false",
    "mfa": [
      {
        "strategy": "token-sign",
        "status": "enabled"
      }
    ],
    "cluster-coverage": "full"
},
```

Deregister an MFA public key using CloudHSM CLI

Follow these steps to deregister a multi-factor authentication (MFA) public key for AWS CloudHSM admin users when MFA public key is registered.

1. Use CloudHSM CLI to log in to the HSM as an admin with MFA enabled.
2. Use the **user change-mfa token-sign** command to remove MFA for a user.

```
aws-cloudhsm > user change-mfa token-sign --username <username> --role admin --
deregister --change-quorum
Enter password:
Confirm password:
{
  "error_code": 0,
  "data": {
```

```
    "username": "<username>",
    "role": "admin"
  }
}
```

Token file reference for MFA with CloudHSM CLI

The token file generated when either registering a multi-factor authentication (MFA) public key or when attempting to login to the CloudHSM CLI using MFA consists of the following:

- **Tokens:** An array base64 encoded unsigned/signed token pairs in the form of JSON object literals.
- **Unsigned:** A base64 encoded and SHA256 hashed token.
- **Signed:** A base64 encoded signed token (signature) of the unsigned token, using the RSA 2048-bit private key.


```
{
  "version": "2.0",
  "tokens": [
    {
      "unsigned": "1jqwxb9bJOUUQLiNb7mxXS1uBJsEXh0B9nj05BqnPsE=",
      "signed": "eiw3fZeCKIY50C4zPeg9Rt90M1Qlq3WlJh6Yw7xXm4nF6e9ETLE39+9M
+rUqDWMRZjaBfaMbg5d9yDkz5p13U7ch2tlF9LoYabsWutkT014KRq/rcYMvFsU9n/Ey/TK0PVaxLN42X
+pebV4juwMhN4mK4CzdFAJgM+UGBOj4yB9recpOBB9K8QFSpJZALSEdDgUc/mS1eDq3rU0int6+4NKuLQjpR
+LSEIWRZ6g6+MND2vXGskxHjadCQ09L7Tz8VcWjKDbxJcBiGKvkqyozl9zrGo8fA3WHBmwiAgS61Merx77ZGY4PFR37+j/
YMSC14prCN15DtMRv2xA1SGSb4w=="
    },
    {
      "unsigned": "LMMFc34ASPnvNPFzBbMbr9FProS/Zu2P8zF/xzk5hVQ=",
      "signed": "HBImKnHmw+6R2TpFEpfiAg4+hu2pFNwn43ClhKPkn2higbEhUD0JVi
+4MerSyvU/NN79iWVxDvJ9Ito+jpiRQjTfTGEoIteyuAr1v/Bzh+HjmrO53OQpZaJ/VXGIgApD0myuu/
ZGNKQTCSkkL7+V81FG7yR1Nm22jUeGa735zvm/E+cenvZdy0VVx6A7WeWrl3JEKKBweHbi+7BwbaW
+PTdCuIRd4Ug76Sy+cFhsvcG1k7cMwDh8MgXzIZ2m1f/hdy2j8qAxORTLlmwyUOYvPYOvUhc
+s83hx36QpGwGcD7RA0bPT5OrTx7PHd0N1CL+Wwy91We8yIOFBS6nxo1R7w=="
    },
    {
      "unsigned": "dzeHbwhiVXQqcUGj563z51/7sLUdxjL93SbOUyZRjH8=",
      "signed": "VgQPvrTsvGljVBFxHnswduq16x8ZrnxfcYVYGf/
N7gEzI4At3GDs2EVZWTRdvS0uGHdkFYp1apHgJZ7PDVmGcTkIXVD2lFYppcgNlSzkYlftr5EOjqS9ZjYEqgGuB4g//
MxaBaRbJai/6BlcE92NIdBusTtreIm3yTpjIXNAVoeRSnkfuw7wZcL96QoklNb1WUuSHw
```

```
+psUyeIVtIwFMHEfFoRC0t
+VhmnlnFnkjGPb9W3Aprw2dRRvFM3R2ZTDvMCiOYDzUCd43GftGq2LfxH3qSD51oFHglHQVOY0jyVzzlAvub5HQdtOQdErI
      }
    ]
}
```


Manage quorum authentication (M of N access control) using CloudHSM CLI

AWS CloudHSM clusters support quorum authentication, also known as M of N access control. This feature requires HSM users to cooperate for certain operations, adding an extra layer of protection.

With quorum authentication, no single user on the HSM can perform quorum-controlled operations on the HSM. Instead, a minimum number of HSM users (at least 2) must cooperate to do these operations.

Quorum authentication can control the following operations:

- HSM user management by admin: Creating and deleting HSM users or changing a different HSM user's password. For more information, see User management with quorum authentication enabled for AWS CloudHSM using CloudHSM CLI.

Key points about quorum authentication in AWS CloudHSM.

- An HSM user can sign their own quorum token—that is, providing one of the required approvals for quorum authentication.
- You choose the minimum number of quorum approvers, which ranges from two (2) to eight (8).
- HSMs can store up to 1024 quorum tokens. When this limit is reached, the HSM purges an expired token to create a new one.
- Tokens expire ten minutes after creation by default.
- For clusters with MFA enabled, the same key is used for quorum authentication and multi-factor authentication (MFA). See Using CloudHSM CLI to manage MFA for more information.
- Each HSM can contain one token per Admin service and multiple tokens per Crypto User service.

The following topics provide more information about quorum authentication in AWS CloudHSM.

Topics

- Quorum authentication process for CloudHSM CLI
- Supported AWS CloudHSM service names and types for quorum authentication with CloudHSM CLI
- Set up quorum authentication for AWS CloudHSM admins using CloudHSM CLI
- User management with quorum authentication enabled for AWS CloudHSM using CloudHSM CLI
- Change the quorum minimum value for AWS CloudHSM using CloudHSM CLI

Quorum authentication process for CloudHSM CLI

The following steps summarize the quorum authentication processes for CloudHSM CLI. For the specific steps and tools, see User management with quorum authentication enabled for AWS CloudHSM using CloudHSM CLI.

1. Each hardware security module (HSM) user creates an asymmetric key for signing. Users do this outside of the HSM, taking care to protect the key appropriately.
2. Each HSM user logs in to the HSM and registers the public part of their signing key (the public key) with the HSM.
3. When an HSM user wants to do a quorum-controlled operation, the same user logs in to the HSM and gets a *quorum token*.
4. The HSM user gives the quorum token to one or more other HSM users and asks for their approval.
5. The other HSM users approve by using their keys to cryptographically sign the quorum token. This occurs outside the HSM.
6. When the HSM user has the required number of approvals, the same user logs in to the HSM and runs the quorum-controlled operation with the **--approval** argument, supplying the signed quorum token file, which contains all necessary approvals (signatures).
7. The HSM uses the registered public keys of each signer to verify the signatures. If the signatures are valid, the HSM approves the token and the quorum-controlled operation is performed.

Supported AWS CloudHSM service names and types for quorum authentication with CloudHSM CLI

Admin Services: Quorum authentication is used for admin privileged services like creating users, deleting users, changing user passwords, setting quorum values, and deactivating quorum and MFA capabilities.

Crypto User Services: Quorum authentication is used for crypto-user privileged services associated with a specific key like signing with a key, sharing/unsharing a key, wrapping/unwrapping a key, and setting a key's attribute. The quorum value of an associated key is configured when the key is generated, imported, or unwrapped. The quorum value must be equal to or less than the number of users that the key is associated with, which includes users that the key is shared with and the key owner.

Each service type is further broken down into a qualifying service name, which contains a specific set of quorum supported service operations that can be performed.

Service name	Service type	Service operations
user	Admin	• user create • user delete • user change-password • user change-mfa
quorum	Admin	• quorum token-sign set-quorum-value
cluster[1]	Admin	• cluster mtls register-trust-anchor • cluster mtls deregister-trust-anchor • cluster mtls set-enforcement
key-management	Crypto User	• key wrap • key unwrap • key share • key unshare • key set-attribute
key-usage	Crypto User	• key sign

[1] Cluster service is exclusively available on hsm2m.medium

Set up quorum authentication for AWS CloudHSM admins using CloudHSM CLI

The following topics describe the steps that you must complete to configure your hardware security module (HSM) so that AWS CloudHSM admins can use quorum authentication. You need to do these steps only once when you first configure quorum authentication for admins. After you complete these steps, see User management with quorum authentication enabled for AWS CloudHSM using CloudHSM CLI.

Topics

Prerequisites

To understand this example, you should be familiar with CloudHSM CLI.

Step 1. Create and register a key for signing

To use quorum authentication, each admin must complete *all* of the following steps:

Topics

Create an RSA key pair

There are many different ways to create and protect a key pair. The following examples show how to do it with OpenSSL.

Example – Create a private key with OpenSSL

The following example demonstrates how to use OpenSSL to create a 2048-bit RSA key. To use this example, replace *<admin.key>* with the name of the file where you want to store the key.

```
$ openssl genrsa -out <admin.key>
```

```
Generating RSA private key, 2048 bit long modulus
.....................................+++
.+++
e is 65537 (0x10001)
```

Next, generate the public key using the private key that you just created.

Example – Create a public key with OpenSSL

The following example demonstrates how to use OpenSSL to create a public key from the private key you just created.

```
$ openssl rsa -in admin.key -outform PEM -pubout -out admin1.pub
writing RSA key
```

Create and sign a registration token

You create a token and sign it with the private key you just generated in the previous step.

Example – Create a registration token

1. Use the following command to start the CloudHSM CLI:

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Create a registration token by running the quorum token-sign generate command:

   ```
   aws-cloudhsm > quorum token-sign generate --service registration --token /path/
   tokenfile
   {
     "error_code": 0,
     "data": {
       "path": "/path/tokenfile"
     }
   }
   ```

3. The quorum token-sign generate command generates a registration token at the specified file path. Inspect the token file:

```
$ cat /path/tokenfile
{
  "version": "2.0",
  "tokens": [
    {
      "approval_data": <approval data in base64 encoding>,
      "unsigned": <unsigned token in base64 encoding>,
      "signed": ""
    }
  ]
}
```

The token file consists of the following:

- **approval_data**: A base64 encoded randomized data token whose raw data doesn't exceed the maximum of 245 bytes.
- **unsigned**: A base64 encoded and SHA256 hashed token of the approval_data.
- **signed**: A base64 encoded signed token (signature) of the unsigned token, using the RSA 2048-bit private key previously generated with OpenSSL.

You sign the unsigned token with the private key to demonstrate that you have access to the private key. You will need the registration token file fully populated with a signature and the public key to register the admin as a quorum user with the AWS CloudHSM cluster.

Example – Sign the unsigned registration token

1. Decode the base64 encoded unsigned token and place it into a binary file:

```
$ echo -n '6BMUj6mUjjko6ZLCEdzGlWpR5sILhFJfqhW1ej3Oq1g=' | base64 -d > admin.bin
```

2. Use OpenSSL and the private key to sign the now binary unsigned registration token and create a binary signature file:

```
$ openssl pkeyutl -sign \
-inkey admin.key \
-pkeyopt digest:sha256 \
```

```
-keyform PEM \
-in admin.bin \
-out admin.sig.bin
```

3. Encode the binary signature into base64:

```
$ base64 -w0 admin.sig.bin > admin.sig.b64
```

4. Copy and paste the base64 encoded signature into the token file:

```
{
  "version": "2.0",
  "tokens": [
    {
      "approval_data": <approval data in base64 encoding>,
      "unsigned": <unsigned token in base64 encoding>,
      "signed": <signed token in base64 encoding>
    }
  ]
}
```

Register the public key with the HSM

After creating a key, the admin must register the public key with the AWS CloudHSM cluster.

To register a public key with the HSM

1. Use the following command to start CloudHSM CLI:

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Using CloudHSM CLI, log in as an admin.

```
aws-cloudhsm > login --username <admin> --role admin
Enter password:
```

```
{
  "error_code": 0,
  "data": {
    "username": "<admin>",
    "role": "admin"
  }
}
```

3. Use the Register a user's token-sign quorum strategy using CloudHSM CLI command to register the public key. For more information, see the following example or use the **help user change-quorum token-sign register** command.

Example – Register a public key with AWS CloudHSM cluster

The following example shows how to use the **user change-quorum token-sign register** command in CloudHSM CLI to register an admin' public key with the HSM. To use this command, the admin must be logged in to the HSM. Replace these values with your own:

```
aws-cloudhsm > user change-quorum token-sign register --public-key </path/admin.pub> --signed-token </path/tokenfile>
{
  "error_code": 0,
  "data": {
    "username": "admin",
    "role": "admin"
  }
}
```

> **Note**
>
> **/path/admin.pub**: The filepath to the public key PEM file
> **Required**: Yes
> **/path/tokenfile**: The filepath with token signed by user private key
> **Required**: Yes

After all admins register their public keys, the output from the **user list** command shows this in the quorum field, stating the enabled quorum strategy in use, as shown below:

```
aws-cloudhsm > user list
```

```
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "admin2",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "admin3",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
```

```
          "username": "admin4",
          "role": "admin",
          "locked": "false",
          "mfa": [],
          "quorum": [
            {
              "strategy": "token-sign",
              "status": "enabled"
            }
          ],
          "cluster-coverage": "full"
        },
        {
          "username": "app_user",
          "role": "internal(APPLIANCE_USER)",
          "locked": "false",
          "mfa": [],
          "quorum": [],
          "cluster-coverage": "full"
        }
      ]
    }
}
```

In this example, the AWS CloudHSM cluster has two HSMs, each with the same admins, as shown in the following output from the **user list** command. For more information about creating users, see User management with CloudHSM CLI

Step 2. Set the quorum minimum value on the HSM

To use quorum authentication, an admin must log in to the HSM and then set the *quorum minimum value*. This is the minimum number of admin approvals that are required to perform HSM user management operations. Any admin on the HSM can set the quorum minimum value, including admins who have not registered a key for signing. You can change the quorum minimum value at any time. For more information, see Change the minimum value.

To set the quorum minimum value on the HSM

1. Use the following command to start CloudHSM CLI:

Linux

```
$ /opt/cloudhsm/bin/cloudhsm-cli interactive
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
```

2. Using CloudHSM CLI, log in as an admin.

```
aws-cloudhsm > login --username <admin> --role admin
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "<admin>",
    "role": "admin"
  }
}
```

3. Use the [Update a quorum value using CloudHSM CLI]() command to set the quorum minimum value. The `--service` flag identifies the HSM service that you're setting values for. See the following example or use the **help quorum token-sign set-quorum-value** command for more information.

Example – Set the quorum minimum value on the HSM

This example uses a quorum minimum value of two (2). You can choose any value from two (2) to eight (8), up to the total number of admins on the HSM. In this example, the HSM has four (4) admins, so the maximum possible value is four (4).

To use the following example command, replace the final number (*<2>*) with the preferred quorum minimum value.

```
aws-cloudhsm > quorum token-sign set-quorum-value --service user --value <2>
{
  "error_code": 0,
  "data": "Set quorum value successful"
}
```

In this example, the **Show quorum values using CloudHSM CLI** command lists the HSM service types, names, and descriptions that are included in the service.

Quorum minimum values

To get the quorum minimum value for a service, use the **quorum token-sign list-quorum-values** command:

```
aws-cloudhsm > quorum token-sign list-quorum-values
{
  "error_code": 0,
  "data": {
    "user": 2,
    "quorum": 1
  }
}
```

The output from the preceding **quorum token-sign list-quorum-values** command shows that the quorum minimum value for HSM user service, responsible for user management operations, is now two (2). After you complete these steps, see User management with quorum (M of N).

Admin Services: Quorum authentication is used for admin privileged services like creating users, deleting users, changing user passwords, setting quorum values, and deactivating quorum and MFA capabilities.

Crypto User Services: Quorum authentication is used for crypto-user privileged services associated with a specific key like signing with a key, sharing/unsharing a key, wrapping/unwrapping a key, and setting a key's attribute. The quorum value of an associated key is configured when the key is generated, imported, or unwrapped. The quorum value must be equal to or less than the number of users that the key is associated with, which includes users that the key is shared with and the key owner.

Each service type is further broken down into a qualifying service name, which contains a specific set of quorum supported service operations that can be performed.

Service name	Service type	Service operations
user	Admin	• user create • user delete

Service name	Service type	Service operations
		• user change-password • user change-mfa
quorum	Admin	• quorum token-sign set-quorum-value
cluster[1]	Admin	• cluster mtls register-trust-anchor • cluster mtls deregister-trust-anchor • cluster mtls set-enforcement
key-management	Crypto User	• key wrap • key unwrap • key share • key unshare • key set-attribute
key-usage	Crypto User	• key sign

[1] Cluster service is exclusively available on hsm2m.medium

User management with quorum authentication enabled for AWS CloudHSM using CloudHSM CLI

An AWS CloudHSM admin on the hardware security module (HSM) can configure quorum authentication for the following operations in the AWS CloudHSM cluster:

- **Create an AWS CloudHSM user with CloudHSM CLI**
- **Delete an AWS CloudHSM user with CloudHSM CLI**
- **Change a user's password with CloudHSM CLI**
- **The user change-mfa category in CloudHSM CLI**

After the AWS CloudHSM cluster is configured for quorum authentication, admins cannot perform HSM user management operations on their own. The following example shows the output when an admin attempts to create a new user on the HSM. The command fails with an error, stating that quorum authentication is required.

```
aws-cloudhsm > user create --username user1 --role crypto-user
Enter password:
Confirm password:
{
  "error_code": 1,
  "data": "Quorum approval is required for this operation"
}
```

To perform an HSM user management operation, an admin must complete the following tasks:

Topics

Step 1. Get a quorum token

First, the admin must use CloudHSM CLI to request a *quorum token*.

To get a quorum token

1. Use the following command to start CloudHSM CLI.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Using CloudHSM CLI, log in as an admin.

```
aws-cloudhsm > login --username <admin> --role admin
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "<admin>",
    "role": "admin"
  }
}
```

3. Use the **quorum token-sign generate** command to generate a quorum token. For more information, see the following example or use the **help quorum token-sign generate** command.

Example – Generate a quorum token

This example gets a quorum token for the admin with user name `admin` and saves the token to a file named `admin.token`. To use the example command, replace these values with your own:

- *`<admin>`* – The name of the admin who is getting the token. This must be the same admin who is logged in to the HSM and is running this command.
- *`<admin.token>`* – The name of the file to use for storing the quorum token.

In the following command, `user` identifies the *service name* for which you can use the token that you are generating. In this case, the token is for HSM user management operations (`user` service). .

```
aws-cloudhsm > login --username <admin> --role admin --password <password>
{
  "error_code": 0,
  "data": {
    "username": "<admin>",
    "role": "admin"
  }
}
```

```
aws-cloudhsm > quorum token-sign generate --service user --token </path/admin.token>
{
  "error_code": 0,
  "data": {
```

```
    "path": "/home/tfile"
  }
}
```

The **quorum token-sign generate** command generates a user service quorum token at the specified file path. The token file can be inspected:

```
$ cat </path/admin.token>
{
  "version": "2.0",
  "service": "user-management",
  "approval_data": "AAEAAwAAABgAAAAAAAAAJ9eFkfcP3mNzJAlfK
+OWbNhZG1pbgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABj5vbeAAAAAAAAAAAAAQADAAAAFQAAAAAAAAAAW/
v5Euk83amq1fij0zyvD2FkbWluAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGPm9t4AAAAAAAAAAAAABAAMAAAAUA
+b23gAAAAAAAAA",
  "token": "0l2LZkmAHZyAc1hPhyckOoVW33aGrgG77qmDHWQ3CJ8=",
  "signatures": []
}
```

The token file consists of the following:

- **service**: An identifier for the quorum service the token is associated with.
- **approval_data**: A base64 encoded raw data token generated by the HSM.
- **token**: A base64 encoded and SHA-256 hashed token of the approval_data
- **signatures**: An array of base64 encoded signed tokens (signatures) of the unsigned token, where each signature of an approver is in the form of a JSON object literal:

```
{
      "username": "<APPROVER_USERNAME>",
      "role": "<APPROVER_ROLE>",
      "signature": "<APPROVER_RSA2048_BIT_SIGNATURE>"
}
```

  Each signature is created from the result of an approver using their corresponding RSA 2048-bit private key whose public key was registered with the HSM.

The generated user service quorum token can be confirmed to exist on the CloudHSM cluster by running the **quorum token-sign list** command:

```
aws-cloudhsm > quorum token-sign list
{
  "error_code": 0,
  "data": {
    "tokens": [
      {
        "username": "admin",
        "service": "user",
        "approvals-required": {
          "value": 2
        },
        "number-of-approvals": {
          "value": 0
        },
        "token-timeout-seconds": {
          "value": 597
        },
        "cluster-coverage": "full"
      }
    ]
  }
}
```

The `token-timeout-seconds` time indicates the timeout period in seconds for a generated token to be approved before it expires.

Step 2. Get signatures from approving admins

An admin who has a quorum token must get the token approved by other admins. To give their approval, the other admins use their signing key to cryptographically sign the token. They do this outside the HSM.

There are many different ways to sign the token. The following example shows how to do it with OpenSSL. To use a different signing tool, make sure that the tool uses the admin's private key (signing key) to sign a SHA-256 digest of the token.

Example – Get signatures from approving admins

In this example, the admin that has the token (`admin`) needs at least two (2) approvals. The following example commands show how two (2) admins can use OpenSSL to cryptographically sign the token.

1. Decode the base64 encoded unsigned token and place it into a binary file:

```
$ echo -n '0l2LZkmAHZyAc1hPhyckOoVW33aGrgG77qmDHWQ3CJ8=' | base64 -d > admin.bin
```

2. Use OpenSSL and the respective private key of the approver (`admin3`) to sign the now binary quorum unsigned token for the user service and create a binary signature file:

```
$ openssl pkeyutl -sign \
-inkey admin3.key \
-pkeyopt digest:sha256 \
-keyform PEM \
-in admin.bin \
-out admin.sig.bin
```

3. Encode the binary signature into base64:

```
$ base64 -w0 admin.sig.bin > admin.sig.b64
```

4. Finally, copy and paste the base64 encoded signature into the token file, following the JSON object literal format specified earlier for approver signature:


```
{
  "version": "2.0",
  "approval_data": "AAEAAwAAABgAAAAAAAAAAJ9eFkfcP3mNzJAlfK
+OWbNhZG1pbgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABj5vbeAAAAAAAAAAAAAQADAAAAFQAAAAAAAAAAAW
v5Euk83amq1fij0zyvD2FkbWluAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGPm9t4AAAAAAAAAAAAABAAMAA
+b23gAAAAAAAAAA",
  "token": "0l2LZkmAHZyAc1hPhyckOoVW33aGrgG77qmDHWQ3CJ8=",
  "signatures": [
    {
      "username": "admin2",
      "role": "admin",
      "signature": "O6qx7/mUaVkYYVr1PW7l8JJko+Kh3e8zBIqdk3tAiNy+1rW
+OsDtvYujhEU4aOFVLcrUFmyB/CX9OQmgJLgx/pyK+ZPEH+GoJGqk9YZ7X1nOXwZRP9g7hKV
+7XCtg9TuDFtHYWDpBfz2jWiu2fXfX4/
jTs4f2xIfFPIDKcSP8fhxjQ63xEcCf1jzGha6rDQMu4xUWWdtDgfT7um7EJ9dXNoHqLB7cTzphaubNaEFbFPXQ1siGm
ssktwyruGFLpXs1n0tJOEglGhx2qbYTs+omKWZdORl5WIWEXW3IXw/
Dg5vVObrNpvG0eZKO8nSMc27+cyPySc+ZbNw=="
    },
    {
      "username": "admin3",
      "role": "admin",
```

```
      "signature": "O6qx7/mUaVkYYVr1PW7l8JJko+Kh3e8zBIqdk3tAiNy+1rW
+OsDtvYujhEU4aOFVLcrUFmyB/CX9OQmgJLgx/pyK+ZPEH+GoJGqk9YZ7X1nOXwZRP9g7hKV
+7XCtg9TuDFtHYWDpBfz2jWiu2fXfX4/
jTs4f2xIfFPIDKcSP8fhxjQ63xEcCf1jzGha6rDQMu4xUWWdtDgfT7um7EJ9dXNoHqLB7cTzphaubNaEFbFPXQ1siGm
ssktwyruGFLpXs1n0tJOEglGhx2qbYTs+omKWZdORl5WIWEXW3IXw/
Dg5vVObrNpvG0eZKO8nSMc27+cyPySc+ZbNw=="
    }
  ]
}
```

Step 3. Approve the token on the AWS CloudHSM cluster and execute a user management operation

After an admin has the necessary approvals/signatures, as detailed in the previous section, the admin can supply that token to the AWS CloudHSM cluster along with one of the following user management operations:

- **create**
- **delete**
- **change-password**
- **user change-mfa**

For more information about using these commands, see User management with CloudHSM CLI.

During the transaction, the token will be approved within the AWS CloudHSM cluster and execute the requested user management operation. The success of the user management operation is contingent upon both a valid approved quorum token and a valid user management operation.

The admin can use the token for only one operation. When that operation succeeds, the token is no longer valid. To do another HSM user management operation, the admin must repeat the above outlined process. That is, the admin must generate a new quorum token, get new signatures from approvers, and then approve and consume the new token on the HSM with the requested user management operation.

> **Note**
>
> The quorum token is only valid as long as your current login session is open. If you log out of CloudHSM CLI or if the network disconnects, the token is no longer valid. Similarly, an

authorized token can only be used within CloudHSM CLI. It cannot be used to authenticate in a different application.

Example Creating a new user as an admin

In the following example, a logged in admin creates a new user on the HSM:

```
aws-cloudhsm > user create --username user1 --role crypto-user --approval /path/
admin.token
Enter password:
Confirm password:
{
  "error_code": 0,
  "data": {
    "username": "user1",
    "role": "crypto-user"
  }
}
```

The admin then enters the **user list** command to confirm the creation of the new user:

```
aws-cloudhsm > user list
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "admin2",
        "role": "admin",
```

```
      "locked": "false",
      "mfa": [],
      "quorum": [
        {
          "strategy": "token-sign",
          "status": "enabled"
        }
      ],
      "cluster-coverage": "full"
    },
    {
      "username": "admin3",
      "role": "admin",
      "locked": "false",
      "mfa": [],
      "quorum": [
        {
          "strategy": "token-sign",
          "status": "enabled"
        }
      ],
      "cluster-coverage": "full"
    },
    {
      "username": "admin4",
      "role": "admin",
      "locked": "false",
      "mfa": [],
      "quorum": [
        {
          "strategy": "token-sign",
          "status": "enabled"
        }
      ],
      "cluster-coverage": "full"
    },
    {
      "username": "user1",
      "role": "crypto-user",
      "locked": "false",
      "mfa": [],
      "quorum": [],
      "cluster-coverage": "full"
    },
```

```
      {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

If the admin tries to perform another HSM user management operation, it fails with a quorum authentication error:

```
aws-cloudhsm > user delete --username user1 --role crypto-user
{
  "error_code": 1,
  "data": "Quorum approval is required for this operation"
}
```

As shown below, the **quorum token-sign list** command shows that the admin has no approved tokens. To perform another HSM user management operation, the admin must generate a new quorum token, get new signatures from approvers, and execute the desired user management operation with the --approval argument to supply the quorum token to be approved and consumed during execution of the user management operation.

```
aws-cloudhsm > quorum token-sign list
{
  "error_code": 0,
  "data": {
    "tokens": []
  }
}
```

Change the quorum minimum value for AWS CloudHSM using CloudHSM CLI

After setting the quorum minimum value for CloudHSM admins, you might need to adjust the quorum minimum value. The HSM allows changes to the quorum minimum value only when the number of approvers meets or exceeds the current value. For example, with a quorum minimum value of two (2), at least two (2) admins must approve any changes.

> **Note**
>
> The quorum value of the user service must always be less than the quorum value of the quorum service. For information on service names, see Supported AWS CloudHSM service names and types for quorum authentication with CloudHSM CLI.

To get quorum approval to change the quorum minimum value, you need a *quorum token* for the **quorum service** using the **quorum token-sign set-quorum-value** command. To generate a quorum token for the for the **quorum service** using the **quorum token-sign set-quorum-value** command, the quorum service must be higher than one (1). This means that before you can change the quorum minimum value for *user service*, you might need to change the quorum minimum value for *quorum service*.

Steps to change the quorum minimum value for admins

1. Start the CloudHSM CLI interactive mode.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Using CloudHSM CLI, log in as an admin.

   ```
   aws-cloudhsm > login --username <admin> --role admin
   Enter password:
   {
     "error_code": 0,
     "data": {
       "username": "<admin>",
       "role": "admin"
     }
   }
   ```

3. Check current quorum minimum values:

```
aws-cloudhsm > quorum token-sign list-quorum-values
```

4. If the quorum minimum value for the quorum service is lower than the value for the user service, change the *quorum service* value:

```
aws-cloudhsm > quorum token-sign set-quorum-value --service quorum --value <3>
```

5. Generate a quorum token for the quorum service.
6. Get approvals (signatures) from other admins.
7. Approve the token on the CloudHSM cluster and execute a user management operation..
8. Change the quorum minimum value for the *user service*:

```
aws-cloudhsm > quorum token-sign set-quorum-value
```

Example Adjusting *quorum service* minimum values

1. **Check current values**. The example shows that the quorum minimum value for *user service* is currently two (2).

```
aws-cloudhsm > quorum token-sign list-quorum-values
{
  "error_code": 0,
  "data": {
    "user": 2,
    "quorum": 1
  }
}
```

2. **Change quorum service value**. Set the quorum minimum value for *quorum service* to a value that is the same or higher than the value for *user service*. This example sets the quorum minimum value for *quorum service* to two (2), the same value that was set for *user service* in the previous example.

```
aws-cloudhsm > quorum token-sign set-quorum-value --service quorum --value 2
{
  "error_code": 0,
  "data": "Set quorum value successful"
}
```

3. **Verify the changes**. This example shows that the quorum minimum value is now two (2) for *user service* and *quorum service*.

```
aws-cloudhsm > quorum token-sign list-quorum-values
{
  "error_code": 0,
  "data": {
    "user": 2,
    "quorum": 2
  }
}
```

HSM user management with CloudHSM Management Utility (CMU)

To manage hardware security module (HSM) users in AWS CloudHSM, you must log in to the HSM with the user name and password of a cryptographic officer (CO). Only COs can manage users. The HSM contains a default CO named admin. You set the password for admin when you activated the cluster.

This topic provides step-by-step instruction on and detail about managing HSM users with AWS CloudHSM Management Utility (CMU).

Topics

Prerequisites for user management in AWS CloudHSM Management Utility

Before you use AWS CloudHSM Management Utility (CMU) to manage hardware security module (HSM) users in AWS CloudHSM, you must complete these prerequisites. The following topics describe getting started with the CMU.

Sections

Get the IP address of an HSM in AWS CloudHSM

To use CMU, you must use the configure tool to update the local configuration. CMU creates its own connection to the cluster and this connection is *not* cluster aware. To track cluster information, CMU maintains a local configuration file. This means that *each time* you use CMU, you should first update the configuration file by running the configure command line tool with the `--cmu` parameter. If you are using Client SDK 3.2.1 or earlier, you must use a different parameter than `--cmu`. For more information, see the section called "Using CMU with Client SDK 3.2.1 and earlier".

The `--cmu` parameter requires you to add the IP address of an HSM in your cluster. If you have multiple HSMs, you can use any IP address. This ensures CMU can propagate any changes you make across the entire cluster. Remember that CMU uses its local file to track cluster information. If the cluster has changed since the last time you used CMU from a particular host, you must add those changes to the local configuration file stored on that host. Never add or remove an HSM while you're using CMU.

To get an IP address for an HSM (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To open the cluster detail page, in the cluster table, choose the cluster ID.
4. To get the IP address, go to the HSMs tab. For IPv4 clusters, choose an address listed under **ENI IPv4 address**. For dual-stack clusters use either the ENI IPv4 or the **ENI IPv6 address**.

To get an IP address for an HSM (AWS CLI)

- Get the IP address of an HSM by using the **describe-clusters** command from the AWS CLI. In the output from the command, the IP address of the HSMs are the values of `EniIp` and `EniIpV6` (if it is a dual-stack cluster).

```
$ aws cloudhsmv2 describe-clusters
{
    "Clusters": [
        { ... }
            "Hsms": [
                {
...
                    "EniIp": "10.0.0.9",
...
                },
                {
...
                    "EniIp": "10.0.1.6",
                    "EniIpV6": "2600:113f:404:be09:310e:ed34:3412:f733",
...
```

Using CMU with Client SDK 3.2.1 and earlier

With Client SDK 3.3.0, AWS CloudHSM added support for the `--cmu` parameter, which simplifies the process of updating the configuration file for CMU. If you're using a version of CMU from Client SDK 3.2.1 or earlier, you must continue to use the `-a` and `-m` parameters to update the configuration file. For more information about these parameters, see Configure Tool.

Download CloudHSM Management Utility

The latest version of CMU is available for HSM user management tasks whether you are using Client SDK 5 and Client SDK 3.

To download and install CMU

- Download and install CMU.

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-mgmt-util-latest.el6.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-mgmt-util-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-mgmt-util-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-mgmt-util-latest.el7.x86_64.rpm
```

CentOS 7.8+

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-mgmt-util-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-mgmt-util-latest.el7.x86_64.rpm
```

CentOS 8.3+

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-mgmt-util-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-mgmt-util-latest.el8.x86_64.rpm
```

RHEL 7 (7.8+)

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-mgmt-util-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-mgmt-util-latest.el7.x86_64.rpm
```

RHEL 8 (8.3+)

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-mgmt-util-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-mgmt-util-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-mgmt-util_latest_amd64.deb
```

```
$ sudo apt install ./cloudhsm-mgmt-util_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-mgmt-util_latest_u18.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-mgmt-util_latest_u18.04_amd64.deb
```

Windows Server 2012

1. Download CloudHSM Management Utility.
2. Run the CMU installer (**AWSCloudHSMManagementUtil-latest.msi**) with Windows administrative privilege.

Windows Server 2012 R2

1. Download CloudHSM Management Utility.
2. Run the CMU installer (**AWSCloudHSMManagementUtil-latest.msi**) with Windows administrative privilege.

Windows Server 2016

1. Download CloudHSM Management Utility.

2. Run the CMU installer (**AWSCloudHSMManagementUtil-latest.msi**) with Windows administrative privilege.

HSM user types for AWS CloudHSM Management Utility

Most operations that you perform on the hardware security module (HSM) require the credentials of an AWS CloudHSM *HSM user*. The HSM authenticates each HSM user and each HSM user has a *type* that determines which operations you can perform on the HSM as that user.

> **Note**
>
> HSM users are distinct from IAM users. IAM users who have the correct credentials can create HSMs by interacting with resources through the AWS API. After the HSM is created, you must use HSM user credentials to authenticate operations on the HSM.

User types

- Precrypto officer (PRECO)
- Crypto officer (CO)
- Crypto user (CU)
- Appliance user (AU)

Precrypto officer (PRECO)

In both the cloud management utility (CMU) and the key management utility (KMU), the PRECO is a temporary user that exists only on the first HSM in an AWS CloudHSM cluster. The first HSM in a new cluster contains an PRECO user indicating that this cluster has never been activated. To activate a cluster, you execute the cloudhsm-cli and run the **cluster activate** command. Log in to the HSM and change the PRECO's password. When you change the password, this user becomes the crypto officer (CO).

Crypto officer (CO)

In both the cloud management utility (CMU) and the key management utility (KMU), a crypto officer (CO) can perform user management operations. For example, they can create and delete users and change user passwords. For more information about CO users, see the HSM user permissions table for AWS CloudHSM Management Utility. When you activate a new cluster, the user changes from a Precrypto Officer (PRECO) to a crypto officer (CO).-->

Crypto user (CU)

A crypto user (CU) can perform the following key management and cryptographic operations.

- **Key management –** Create, delete, share, import, and export cryptographic keys.
- **Cryptographic operations –** Use cryptographic keys for encryption, decryption, signing, verifying, and more.

For more information, see the HSM user permissions table for AWS CloudHSM Management Utility.

Appliance user (AU)

The appliance user (AU) can perform cloning and synchronization operations on your cluster's HSMs. AWS CloudHSM uses the AU to synchronize the HSMs in an AWS CloudHSM cluster. The AU exists on all HSMs provided by AWS CloudHSM, and has limited permissions. For more information, see the HSM user permissions table for AWS CloudHSM Management Utility.

AWS cannot perform any operations on your HSMs . AWS cannot view or modify your users or keys and cannot perform any cryptographic operations using those keys.

HSM user permissions table for AWS CloudHSM Management Utility

The following table lists hardware security module (HSM(operations sorted by the type of HSM user or session that can perform the operation in AWS CloudHSM.

	Crypto officer (CO)	Crypto User (CU)	Appliance User (AU)	Unauthenticated Session
Get basic cluster info[1]	Yes	Yes	Yes	Yes

	Crypto officer (CO)	Crypto User (CU)	Appliance User (AU)	Unauthenticated Session
Change own password	Yes	Yes	Yes	Not applicable
Change any user's password	Yes	No	No	No
Add, remove users	Yes	No	No	No
Get sync status[2]	Yes	Yes	Yes	No
Extract, insert masked objects[3]	Yes	Yes	Yes	No
Key management functions[4]	No	Yes	No	No
Encrypt, decrypt	No	Yes	No	No

	Crypto officer (CO)	Crypto User (CU)	Appliance User (AU)	Unauthenticated Session
Sign, verify	No	Yes	No	No
Generate digests and HMACs	No	Yes	No	No

- [1] Basic cluster information includes the number of HSMs in the cluster and each HSM's IP address, model, serial number, device ID, firmware ID, etc.
- [2] The user can get a set of digests (hashes) that correspond to the keys on the HSM. An application can compare these sets of digests to understand the synchronization status of HSMs in a cluster.
- [3] Masked objects are keys that are encrypted before they leave the HSM. They cannot be decrypted outside of the HSM. They are only decrypted after they are inserted into an HSM that is in the same cluster as the HSM from which they were extracted. An application can extract and insert masked objects to synchronize the HSMs in a cluster.
- [4] Key management functions include creating, deleting, wrapping, unwrapping, and modifying the attributes of keys.

Create HSM users using AWS CloudHSM Management Utility

Use **createUser** in AWS CloudHSM Management Utility (CMU) to create new users on the hardware security module (HSM). You must log in as a CO to create a user.

To create a new CO user

1. Use the configure tool to update the CMU configuration.

Linux

```
$ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" --cmu <IP address>
```

2. Start CMU.

Linux

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\cloudhsm_mgmt_util.exe" C:
\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

3. Log in to the HSM as a CO user.

```
aws-cloudhsm > loginHSM CO admin co12345
```

Make sure the number of connections CMU lists match the number of HSMs in the cluster. If not, log out and start over.

4. Use **createUser** to create a CO user named **`example_officer`** with a password of **`password1`**.

```
aws-cloudhsm > createUser CO example_officer password1
```

CMU prompts you about the create user operation.

```
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************
```

```
Do you want to continue(y/n)?
```

5. Type **y**.

To create a new CU user

1. Use the configure tool to update the CMU configuration.

   Linux

   ```
   $ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" --cmu <IP address>
   ```

2. Start CMU.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\cloudhsm_mgmt_util.exe" C:
   \ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
   ```

3. Log in to the HSM as a CO user.

   ```
   aws-cloudhsm > loginHSM CO admin co12345
   ```

   Make sure the number of connections CMU lists match the number of HSMs in the cluster. If not, log out and start over.

4. Use **createUser** to create a CU user named **`example_user`** with a password of **`password1`**.

   ```
   aws-cloudhsm > createUser CU example_user password1
   ```

   CMU prompts you about the create user operation.

```
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)?
```

5. Type **y**.

For more information about **createUser**, see createUser.

List all HSM users in the cluster using AWS CloudHSM Management Utility

Use **listUsers** command in the AWS CloudHSM Management Utility (CMU) to list all the users in the AWS CloudHSM cluster. You do not have to log in to run **listUsers** and all user types can list users.

To list all users on the cluster

1. Use the configure tool to update the CMU configuration.

   Linux

   ```
   $ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" --cmu <IP address>
   ```

2. Start CMU.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\cloudhsm_mgmt_util.exe" C:
\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

3. Use **listUsers** to list all the users on the cluster.

```
aws-cloudhsm > listUsers
```

CMU lists all the users on the cluster.

```
Users on server 0(10.0.2.9):
Number of users found:4

    User Id             User Type       User Name                            
 MofnPubKey    LoginFailureCnt         2FA
         1              AU              app_user                             NO
               0               NO
         2              CO              example_officer                      NO
               0               NO
         3              CU              example_user                         NO
               0               NO
Users on server 1(10.0.3.11):
Number of users found:4

    User Id             User Type       User Name                            
 MofnPubKey    LoginFailureCnt         2FA
         1              AU              app_user                             NO
               0               NO
         2              CO              example_officer                      NO
               0               NO
         3              CU              example_user                         NO
               0               NO
Users on server 2(10.0.1.12):
Number of users found:4

    User Id             User Type       User Name                            
 MofnPubKey    LoginFailureCnt         2FA
         1              AU              app_user                             NO
               0               NO
         2              CO              example_officer                      NO
               0               NO
```

```
        3              CU               example_user                          NO
               0               NO
```

For more information about **listUsers**, see listUsers.

Change HSM user passwords using AWS CloudHSM Management Utility

Use **changePswd** in the AWS CloudHSM Management Utility (CMU) to change a hardware security module (HSM) user's password.

User types and passwords are case sensitive, but user names are not case sensitive.

CO, Crypto user (CU), and appliance user (AU) can change their own password. To change the password of another user, you must log in as a CO. You cannot change the password of a user who is currently logged in.

To change your own password

1. Use the configure tool to update the CMU configuration.

   Linux

   ```
   $ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" --cmu <IP address>
   ```

2. Start CMU.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\cloudhsm_mgmt_util.exe" C:
   \ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
   ```

3. Log in to the HSM.

```
aws-cloudhsm > loginHSM CO admin co12345
```

Make sure the number of connections CMU lists match the number of HSMs in the cluster. If not, log out and start over.

4. Use **changePswd** to change your own password.

```
aws-cloudhsm > changePswd CO example_officer <new password>
```

CMU prompts you about the change password operation.

```
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)?
```

5. Type **y**.

CMU prompts you about the change password operation.

```
Changing password for example_officer(CO) on 3 nodes
```

To change the password of another user

1. Use the configure tool to update the CMU configuration.

Linux

```
$ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" --cmu <IP address>
```

2. Start CMU.

Linux

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\cloudhsm_mgmt_util.exe" C:
\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

3. Log in to the HSM as a CO user.

```
aws-cloudhsm > loginHSM CO admin co12345
```

   Make sure the number of connections CMU lists match the number of HSMs in the cluster. If not, log out and start over.

4. Use **changePswd** to change the password of another user.

```
aws-cloudhsm > changePswd CU example_user <new password>
```

   CMU prompts you about the change password operation.

```
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)?
```

5. Type **y**.

   CMU prompts you about the change password operation.

```
Changing password for example_user(CU) on 3 nodes
```

For more information about **changePswd**, see changePswd.

Delete HSM users using AWS CloudHSM Management Utility

Use **deleteUser** in the AWS CloudHSM Management Utility (CMU) to delete a hardware security module (HSM) user. You must log in as a CO to delete another user.

> **Tip**
>
> You can't delete crypto users (CU) that own keys.

To delete a user

1. Use the configure tool to update the CMU configuration.

   Linux

   ```
   $ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" --cmu <IP address>
   ```

2. Start CMU.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\cloudhsm_mgmt_util.exe" C:
   \ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
   ```

3. Log in to the HSM as a CO user.

   ```
   aws-cloudhsm > loginHSM CO admin co12345
   ```

   Make sure the number of connections CMU lists match the number of HSMs in the cluster. If not, log out and start over.

4. Use **deleteUser** to delete a user.

```
aws-cloudhsm > deleteUser CO example_officer
```

CMU deletes the user.

```
Deleting user example_officer(CO) on 3 nodes
deleteUser success on server 0(10.0.2.9)
deleteUser success on server 1(10.0.3.11)
deleteUser success on server 2(10.0.1.12)
```

For more information about **deleteUser**, see deleteUser.

Manage 2FA for users using AWS CloudHSM Management Utility

For increased security, you can configure two-factor authentication (2FA) to help protect the AWS CloudHSM cluster. You can only enable 2FA for crypto officers (CO).

When you log in to a cluster with a 2FA-enabled hardware service module (HSM) account, you provide cloudhsm_mgmt_util (CMU) with your password—the first factor, what you know—and CMU provides you with a token and prompts you to have the token signed. To provide the second factor—what you have—you sign the token with a private key from a key pair you've already created and associated with the HSM user. To access the cluster, you provide the signed token to CMU.

> **Note**
>
> You cannot enable 2FA for crypto users (CU) or applications. Two-factor authentication (2FA) is only for CO users.

Topics

- Quorum authentication and 2FA in AWS CloudHSM clusters using AWS CloudHSM Management Utility
- 2FA key pair requirements for AWS CloudHSM using AWS CloudHSM Management Utility
- Create users with 2FA enabled for AWS CloudHSM Management Utility users
- Manage 2FA for HSM users using AWS CloudHSM Management Utility

- Disable 2FA for HSM users using AWS CloudHSM Management Utility
- Configuration reference for 2FA with AWS CloudHSM Management Utility

Quorum authentication and 2FA in AWS CloudHSM clusters using AWS CloudHSM Management Utility

The cluster uses the same key for quorum authentication and for two-factor authentication 2FA). This means a user with 2FA enabled is effectively registered for M-of-N-access-control (MofN). To successfully use 2FA and quorum authentication for the same HSM user, consider the following points:

- If you are using quorum authentication for a user today, you should use the same key pair you created for the quorum user to enable 2FA for the user.
- If you add the 2FA requirement for a non-2FA user that is not a quorum authentication user, then you register that user as an MofN user with 2FA authentication.
- If you remove the 2FA requirement or change the password for a 2FA user that is also a quorum authentication user, you will also remove the registration of the quorum user as an MofN user.
- If you remove the 2FA requirement or change the password for a 2FA user that is also a quorum authentication user, but you *still want that user to participate in quorum authentication*, then you must register that user again as an MofN user.

For more information about quorum authentication, see Using CMU to manage quorum authentication.

2FA key pair requirements for AWS CloudHSM using AWS CloudHSM Management Utility

To enable two-factor authentication (2FA) for an AWS CloudHSM hardware security module (HSM) user, use a key that meets the following requirements.

You can create a new key pair or use an existing key that meets the following requirements.

- Key type: Asymmetric
- Key usage: Sign and Verify
- Key spec: RSA_2048
- Signing algorithm includes:

- `sha256WithRSAEncryption`

> **Note**
>
> If you are using quorum authentication or plan to use quorum authentication, see the section called “Quorum authentication”.

Create users with 2FA enabled for AWS CloudHSM Management Utility users

Use AWS CloudHSM Management Utility CMU (CMU) and the key pair to create a new crypto office (CO) user with two-factor authentication (2FA) enabled.

To create CO users with 2FA enabled

1. In one terminal, perform the following steps:

   a. Access your HSM and log in to the CloudHSM Management utility:

   ```
   /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

   b. Log in as a CO and use the following command to create a new user MFA with 2FA:

   ```
   aws-cloudhsm > createUser CO MFA <CO USER NAME> -2fa /home/ec2-user/authdata
   *************************CAUTION********************************
   This is a CRITICAL operation, should be done on all nodes in the
   cluster. AWS does NOT synchronize these changes automatically with the
   nodes on which this operation is not executed or failed, please
   ensure this operation is executed on all nodes in the cluster.
   ****************************************************************

   Do you want to continue(y/n)? y

   Creating User exampleuser3(CO) on 1 nodesAuthentication data written to: "/
   home/ec2-user/authdata"Generate Base64-encoded signatures for SHA256 digests in
    the authentication datafile.
   To generate the signatures, use the RSA private key, which is the second factor
    ofauthentication for this user. Paste the signatures and the corresponding
    public keyinto the authentication data file and provide
   ```

```
the file path below.Leave this field blank to use the path initially
 provided.Enter filename:
```

c. Leave the above terminal in this state. Do not press enter or enter any filename.

2. In another terminal, perform the following steps:

   a. Access your HSM and log in to the CloudHSM Management utility:

```
/opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

   b. Generate a public-private key-pair using the following commands:

```
openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt
 rsa_keygen_bits:2048
```

```
openssl rsa -pubout -in private_key.pem -out public_key.pem
```

   c. Run the following command to install a json querying feature for extracting the Digest from authdata file:

```
sudo yum install jq
```

   d. To extract the digest value, first find the following data in the authdata file:

```
{
  "Version":"1.0",
  "PublicKey":"",
  "Data":[
    {
      "HsmId": <"HSM ID">,
      "Digest": <"DIGEST">,
      "Signature": ""
    }
  ]
}
```

Note

The obtained Digest is base64 encoded, however to sign the digest, you need the file to be decoded first and then signed. The following command will decode the digest and store the decoded content in ‘digest1.bin’

```
cat authdata | jq '.Data[0].Digest' | cut -c2- | rev | cut -c2- | rev |
 base64 -d > digest1.bin
```

e. Convert the public key content, adding "\n" and removing spaces as shown here:

```
-----BEGIN PUBLIC KEY-----\n<PUBLIC KEY>\n-----END PUBLIC KEY-----
```

Important

The above command shows how "\n" is added immediately after **BEGIN PUBLIC KEY-----**, spaces between "\n" and the first character of the public key are removed, "\n" is added before **-----END PUBLIC KEY**, and spaces are removed between "\n" and the end of the public key.

This is the PEM format for public key which is accepted in the authdata file.

f. Paste the public key pem format content in the public key section in the authdata file.

```
vi authdata
```

```
{
  "Version":"1.0",
  "PublicKey":"-----BEGIN PUBLIC KEY-----\n<"PUBLIC KEY">\n-----END PUBLIC
 KEY-----",
  "Data":[
    {
      "HsmId":<"HSM ID">,
      "Digest":<"DIGEST">,
      "Signature": ""
    }
  ]
```

```
}
```

g. Sign the token file using the following command:

```
openssl pkeyutl -sign -in digest1.bin -inkey private_key.pem -pkeyopt
 digest:sha256 | base64
Output Expected:
<"THE SIGNATURE">
```

> **Note**
>
> As shown in the above command, use **openssl pkeyutl** instead of **openssl dgst** for signing.

h. Add the signed digest in the Authdata File in "Signature" field.

```
vi authdata
```

```
{
    "Version": "1.0",
    "PublicKey": "-----BEGIN PUBLIC KEY----- ... -----END PUBLIC KEY-----",
    "Data": [
        {
            "HsmId": <"HSM ID">,
            "Digest": <"DIGEST">,
            "Signature": <"Kkdl ... rkrvJ6Q==">
        },
        {
            "HsmId": <"HSM ID">,
            "Digest": <"DIGEST">,
            "Signature": <"K1hxy ... Q261Q==">
        }
    ]
}
```

3. Go back to the first terminal and press **`Enter`**:

```
Generate Base64-encoded signatures for SHA256 digests in the authentication
 datafile. To generate the signatures, use the RSA private key,
```

```
which is the second factor ofauthentication for this user. Paste the signatures and
 the corresponding public keyinto the authentication data file and provide the file
 path below. Leave this field blank to use the path initially provided.
Enter filename: >>>>> Press Enter here

createUser success on server 0(10.0.1.11)
```

Manage 2FA for HSM users using AWS CloudHSM Management Utility

Use **changePswd** in AWS CloudHSM Management Utility (CMU) to modify two-factor authentication (2FA) for a user. Each time you enable 2FA, you must provide a public key for 2FA logins.

changePswd performs any of the following scenarios:

- Change the password for a 2FA user
- Change the password for a non-2FA user
- Add 2FA to a non-2FA user
- Remove 2FA from a 2FA user
- Rotate the key for a 2FA user

You can also combine tasks. For example, you can remove 2FA from a user and change the password at the same time, or you might rotate the 2FA key and change the user password.

To change passwords or rotate keys for CO users with 2FA enabled

1. Use CMU to log in to the HSM as a CO with 2FA enabled.
2. Use **changePswd** to change the password or rotate the key from CO users with 2FA enabled. Use the `-2fa` parameter and include a location in the file system for the system to write the `authdata` file. This file includes a digest for each HSM in the cluster.

   ```
   aws-cloudhsm > changePswd CO example-user <new-password> -2fa /path/to/authdata
   ```

   CMU prompts you to use the private key to sign the digests in the `authdata` file and return the signatures with the public key.

3. Use the private key to sign the digests in the `authdata` file, add the signatures and the public key to the JSON formatted `authdata` file and then provide CMU with the location of the `authdata` file. For more information, see the section called “Configuration reference”.

> **Note**
>
> The cluster uses the same key for quorum authentication and 2FA. If you are using quorum authentication or plan to use quorum authentication, see the section called “Quorum authentication”.

Disable 2FA for HSM users using AWS CloudHSM Management Utility

Use the AWS CloudHSM Management Utility (CMU) to disable two-factor authentication (2FA) for hardware security module HSM) users in AWS CloudHSM.

To disable 2FA for CO users with 2FA enabled

1. Use CMU to log in to the HSM as a CO with 2FA enabled.
2. Use **changePswd** to remove 2FA from CO users with 2FA enabled.

```
aws-cloudhsm > changePswd CO example-user <new password>
```

   CMU prompts you to confirm the change password operation.

> **Note**
>
> If you remove the 2FA requirement or change the password for a 2FA user that is also a quorum authentication user, you will also remove the registration of the quorum user as an MofN user. For more information about quorum users and 2FA, see the section called “Quorum authentication”.

3. Type **y**.

   CMU confirms the change password operation.

Configuration reference for 2FA with AWS CloudHSM Management Utility

The following is an example of the two-factor authentication (2FA) properties in the `authdata` file for both the AWS CloudHSM Management Utility (CMU) generated request and your responses.

```
{
    "Version": "1.0",
    "PublicKey": "-----BEGIN PUBLIC KEY----- ... -----END PUBLIC KEY-----",
    "Data": [
        {
            "HsmId": "hsm-lgavqitns2a",
            "Digest": "k5O1p3f6foQRVQH7S8Rrjcau6h3TYqsSdr16A54+qG8=",
            "Signature": "Kkdl ... rkrvJ6Q=="
        },
        {
            "HsmId": "hsm-lgavqitns2a",
            "Digest": "IyBcx4I5Vyx1jztwvXinCBQd9lDx8oQe7iRrWjBAi1w=",
            "Signature": "K1hxy ... Q261Q=="
        }
    ]
}
```

Data

Top-level node. Contains a subordinate node for each HSM in the cluster. Appears in requests and responses for all 2FA commands.

Digest

This is what you must sign to provide the second factor of authentication. CMU generated in requests for all 2FA commands.

HsmId

The ID of your HSM. Appears in requests and responses for all 2FA commands.

PublicKey

The public key portion of the key pair you generated inserted as PEM-formatted string. You enter this in responses for **createUser** and **changePswd**.

Signature

The Base 64 encoded signed digest. You enter this in responses for all 2FA commands.

Version

The version of the authentication data JSON formatted file. Appears in requests and responses for all 2FA commands.

Using CloudHSM Management Utility (CMU) to manage quorum authentication (M of N access control)

The HSMs in your AWS CloudHSM cluster support quorum authentication, which is also known as M of N access control. With quorum authentication, no single user on the HSM can do quorum-controlled operations on the HSM. Instead, a minimum number of HSM users (at least 2) must cooperate to do these operations. With quorum authentication, you can add an extra layer of protection by requiring approvals from more than one HSM user.

Quorum authentication can control the following operations:

- HSM user management by crypto officers (COs) – Creating and deleting HSM users, and changing a different HSM user's password. For more information, see User management with quorum authentication enabled for AWS CloudHSM Management Utility.

Note the following additional information about using quorum authentication in AWS CloudHSM.

- An HSM user can sign their own quorum token—that is, the requesting user can provide one of the required approvals for quorum authentication.
- You choose the minimum number of quorum approvers for quorum-controlled operations. The smallest number you can choose is two (2), and the largest number you can choose is eight (8).
- The HSM can store up to 1024 quorum tokens. If the HSM already has 1024 tokens when you try to create a new one, the HSM purges one of the expired tokens. By default, tokens expire ten minutes after their creation.
- The cluster uses the same key for quorum authentication and for two-factor authentication (2FA). For more information about using quorum authentication and 2FA, see Quorum Authentication and 2FA.

The following topics provide more information about quorum authentication in AWS CloudHSM.

Topics

- Quorum authentication process for AWS CloudHSM Management Utility
- Set up quorum authentication for AWS CloudHSM crypto officers
- User management with quorum authentication enabled for AWS CloudHSM Management Utility
- Change the quorum minimum value with AWS CloudHSM Management Utility

Quorum authentication process for AWS CloudHSM Management Utility

The following steps summarize the quorum authentication processes. For the specific steps and tools, see User management with quorum authentication enabled for AWS CloudHSM Management Utility.

1. Each HSM user creates an asymmetric key for signing. They do this outside of the HSM, taking care to protect the key appropriately.
2. Each HSM user logs in to the HSM and registers the public part of their signing key (the public key) with the HSM.
3. When an HSM user wants to do a quorum-controlled operation, each user logs in to the HSM and gets a *quorum token*.
4. The HSM user gives the quorum token to one or more other HSM users and asks for their approval.
5. The other HSM users approve by using their keys to cryptographically sign the quorum token. This occurs outside the HSM.
6. When the HSM user has the required number of approvals, the same user logs in to the HSM and gives the quorum token and approvals (signatures) to the HSM.
7. The HSM uses the registered public keys of each signer to verify the signatures. If the signatures are valid, the HSM approves the token.
8. The HSM user can now do a quorum-controlled operation.

Set up quorum authentication for AWS CloudHSM crypto officers

The following topics describe the steps that you must complete to configure your hardware security module (HSM) so that AWS CloudHSM crypto officers (COs) can use quorum authentication. You need to do these steps only once when you first configure quorum authentication for COs. After you complete these steps, see User management with quorum authentication enabled for AWS CloudHSM Management Utility.

Topics

Prerequisites

To understand this example, you should be familiar with the cloudhsm_mgmt_util (CMU) command line tool. In this example, the AWS CloudHSM cluster has two HSMs, each with the same COs, as shown in the following output from the **listUsers** command. For more information about creating users, see HSM users.

```
aws-cloudhsm > listUsers
Users on server 0(10.0.2.14):
Number of users found:7

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                                    NO
           0               NO
         2              AU              app_user                                 NO
           0               NO
         3              CO              officer1                                 NO
           0               NO
         4              CO              officer2                                 NO
           0               NO
         5              CO              officer3                                 NO
           0               NO
         6              CO              officer4                                 NO
           0               NO
         7              CO              officer5                                 NO
           0               NO
Users on server 1(10.0.1.4):
Number of users found:7

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                                    NO
           0               NO
         2              AU              app_user                                 NO
           0               NO
```

```
          3              CO              officer1                                  NO
            0               NO
          4              CO              officer2                                  NO
            0               NO
          5              CO              officer3                                  NO
            0               NO
          6              CO              officer4                                  NO
            0               NO
          7              CO              officer5                                  NO
            0               NO
```

Step 1. Create and register a key for signing

To use quorum authentication, each CO must do *all* of the following steps:

Topics

Create an RSA key pair

There are many different ways to create and protect a key pair. The following examples show how to do it with OpenSSL.

Example – Create a private key with OpenSSL

The following example demonstrates how to use OpenSSL to create a 2048-bit RSA key that is protected by a pass phrase. To use this example, replace *officer1.key* with the name of the file where you want to store the key.

```
$ openssl genrsa -out <officer1.key> -aes256 2048
        Generating RSA private key, 2048 bit long modulus
........................................+++
.+++
e is 65537 (0x10001)
Enter pass phrase for officer1.key:
Verifying - Enter pass phrase for officer1.key:
```

Next, generate the public key using the private key that you just created.

Example – Create a public key with OpenSSL

The following example demonstrates how to use OpenSSL to create a public key from the private key you just created.

```
$ openssl rsa -in officer1.key -outform PEM -pubout -out officer1.pub
Enter pass phrase for officer1.key:
writing RSA key
```

Create and sign a registration token

You create a token and sign it with the private key you just generated in the previous step.

Example – Create a token

The registration token is just a file with any random data that doesn't exceed the maximum size of 245 bytes. You sign the token with the private key to demonstrate that you have access to the private key. The following command uses echo to redirect a string to a file.

```
$ echo <token to be signed> > officer1.token
```

Sign the token and save it to a signature file. You will need the signed token, the unsigned token, and the public key to register the CO as an MofN user with the HSM.

Example – Sign the token

Use OpenSSL and the private key to sign the registration token and create the signature file.

```
$ openssl dgst -sha256 \
    -sign officer1.key \
    -out officer1.token.sig officer1.token
```

Register the public key with the HSM

After creating a key, the CO must register the public part of the key (the public key) with the HSM.

To register a public key with the HSM

1. Use the following command to start the cloudhsm_mgmt_util command line tool.

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

2. Use the **loginHSM** command to log in to the HSM as a CO. For more information, see ???.
3. Use the registerQuorumPubKey command to register the public key. For more information, see the following example or use the **help registerQuorumPubKey** command.

Example – Register a public key with the HSM

The following example shows how to use the **registerQuorumPubKey** command in the cloudhsm_mgmt_util command line tool to register a CO's public key with the HSM. To use this command, the CO must be logged in to the HSM. Replace these values with your own:

```
aws-cloudhsm > registerQuorumPubKey
 CO <officer1> <officer1.token> <officer1.token.sig> <officer1.pub>
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)? y
registerQuorumPubKey success on server 0(10.0.2.14)
```

<officer1.token>

The path to a file that contains an unsigned registration token. Can have any random data of max file size of 245 bytes.

Required: Yes

<officer1.token.sig>

The path to a file that contains the SHA256_PKCS mechanism signed hash of the registration token.

Required: Yes

<officer1.pub>

The path to the file that contains the public key of an asymmetric RSA-2048 key pair. Use the private key to sign the registration token.

Required: Yes

After all COs register their public keys, the output from the **listUsers** command shows this in the `MofnPubKey` column, as shown in the following example.

```
aws-cloudhsm > listUsers
Users on server 0(10.0.2.14):
Number of users found:7

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                                    NO
           0               NO
         2              AU              app_user                                 NO
           0               NO
         3              CO              officer1                                YES
           0               NO
         4              CO              officer2                                YES
           0               NO
         5              CO              officer3                                YES
           0               NO
         6              CO              officer4                                YES
           0               NO
         7              CO              officer5                                YES
           0               NO
Users on server 1(10.0.1.4):
Number of users found:7

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                                    NO
           0               NO
         2              AU              app_user                                 NO
           0               NO
         3              CO              officer1                                YES
           0               NO
         4              CO              officer2                                YES
           0               NO
         5              CO              officer3                                YES
           0               NO
         6              CO              officer4                                YES
           0               NO
         7              CO              officer5                                YES
           0               NO
```

Step 2. Set the quorum minimum value on the HSM

To use quorum authentication for COs, a CO must log in to the HSM and then set the *quorum minimum value*, also known as the *m value*. This is the minimum number of CO approvals that are required to perform HSM user management operations. Any CO on the HSM can set the quorum minimum value, including COs that have not registered a key for signing. You can change the quorum minimum value at any time; for more information, see Change the minimum value.

To set the quorum minimum value on the HSM

1. Use the following command to start the cloudhsm_mgmt_util command line tool.

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

2. Use the **loginHSM** command to log in to the HSM as a CO. For more information, see ???.
3. Use the **setMValue** command to set the quorum minimum value. For more information, see the following example or use the **help setMValue** command.

Example – Set the quorum minimum value on the HSM

This example uses a quorum minimum value of two. You can choose any value from two (2) to eight (8), up to the total number of COs on the HSM. In this example, the HSM has six COs, so the maximum possible value is six.

To use the following example command, replace the final number (*2*) with the preferred quorum minimum value.

```
aws-cloudhsm > setMValue 3 <2>
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)? y
Setting M Value(2) for 3 on 2 nodes
```

In the preceding example, the first number (3) identifies the *HSM service* whose quorum minimum value you are setting.

The following table lists the HSM service identifiers along with their names, descriptions, and the commands that are included in the service.

Service Identifier	Service Name	Service Description	HSM Commands
3	USER_MGMT	HSM user management	• **createUser** • **deleteUser** • **changePswd** (applies only when changing the password of a different HSM user)
4	MISC_CO	Miscellaneous CO service	• **setMValue**

To get the quorum minimum value for a service, use the **getMValue** command, as in the following example.

```
aws-cloudhsm > getMValue 3
MValue of service 3[USER_MGMT] on server 0 : [2]
MValue of service 3[USER_MGMT] on server 1 : [2]
```

The output from the preceding **getMValue** command shows that the quorum minimum value for HSM user management operations (service 3) is now two.

After you complete these steps, see User management with quorum authentication enabled for AWS CloudHSM Management Utility.

User management with quorum authentication enabled for AWS CloudHSM Management Utility

An AWS CloudHSM crypto officer (CO) on the hardware security module (HSM) can configure quorum authentication for the following operations on the HSM:

- Creating HSM users
- Deleting HSM users

- Changing another HSM user's password

After the HSM is configured for quorum authentication, COs cannot perform HSM user management operations on their own. The following example shows the output when a CO attempts to create a new user on the HSM. The command fails with a `RET_MXN_AUTH_FAILED` error, which indicates that quorum authentication failed.

```
aws-cloudhsm > createUser CU user1 password
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)? y
Creating User user1(CU) on 2 nodes
createUser failed: RET_MXN_AUTH_FAILED
creating user on server 0(10.0.2.14) failed

Retry/Ignore/Abort?(R/I/A): A
```

To perform an HSM user management operation, a CO must complete the following tasks:

1. Get a *quorum token*.
2. Get approvals (signatures) from other COs.
3. Approve the token on the HSM.
4. Perform the HSM user management operation.

If you have not yet configured the HSM for quorum authentication for COs, do that now. For more information, see First time setup.

Step 1. Get a quorum token

First the CO must use the cloudhsm_mgmt_util command line tool to request a *quorum token*.

To get a quorum token

1. Use the following command to start the cloudhsm_mgmt_util command line tool.

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

2. Use the **loginHSM** command to log in to the HSM as a CO. For more information, see ???.
3. Use the **getToken** command to get a quorum token. For more information, see the following example or use the **help getToken** command.

Example – Get a quorum token

This example gets a quorum token for the CO with user name officer1 and saves the token to a file named `officer1.token`. To use the example command, replace these values with your own:

- *`officer1`* – The name of the CO who is getting the token. This must be the same CO who is logged in to the HSM and is running this command.
- *`officer1.token`* – The name of the file to use for storing the quorum token.

In the following command, 3 identifies the *service* for which you can use the token that you are getting. In this case, the token is for HSM user management operations (service 3). For more information, see Step 2. Set the quorum minimum value on the HSM.

```
aws-cloudhsm > getToken 3 officer1 officer1.token
getToken success on server 0(10.0.2.14)
Token:
Id:1
Service:3
Node:1
Key Handle:0
User:officer1
getToken success on server 1(10.0.1.4)
Token:
Id:1
Service:3
Node:0
Key Handle:0
User:officer1
```

Step 2. Get signatures from approving COs

A CO who has a quorum token must get the token approved by other COs. To give their approval, the other COs use their signing key to cryptographically sign the token. They do this outside the HSM.

There are many different ways to sign the token. The following example shows how to do it with OpenSSL. To use a different signing tool, make sure that the tool uses the CO's private key (signing key) to sign a SHA-256 digest of the token.

Example – Get signatures from approving COs

In this example, the CO that has the token (officer1) needs at least two approvals. The following example commands show how two COs can use OpenSSL to cryptographically sign the token.

In the first command, officer1 signs his or her own token. To use the following example commands, replace these values with your own:

- *officer1.key* and *officer2.key* – The name of the file that contains the CO's signing key.
- *officer1.token.sig1* and *officer1.token.sig2* – The name of the file to use for storing the signature. Make sure to save each signature in a different file.
- *officer1.token* – The name of the file that contains the token that the CO is signing.

```
$ openssl dgst -sha256 -sign officer1.key -out officer1.token.sig1 officer1.token
Enter pass phrase for officer1.key:
```

In the following command, officer2 signs the same token.

```
$ openssl dgst -sha256 -sign officer2.key -out officer1.token.sig2 officer1.token
Enter pass phrase for officer2.key:
```

Step 3. Approve the signed token on the HSM

After a CO gets the minimum number of approvals (signatures) from other COs, he or she must approve the signed token on the HSM.

To approve the signed token on the HSM

1. Create a token approval file. For more information, see the following example.

2. Use the following command to start the cloudhsm_mgmt_util command line tool.

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

3. Use the **loginHSM** command to log in to the HSM as a CO. For more information, see ???.
4. Use the **approveToken** command to approve the signed token, passing the token approval file. For more information, see the following example.

Example – Create a token approval file and approve the signed token on the HSM

The token approval file is a text file in a particular format that the HSM requires. The file contains information about the token, its approvers, and the approvers' signatures. The following shows an example token approval file.

```
# For "Multi Token File Path", type the path to the file that contains
# the token. You can type the same value for "Token File Path", but
# that's not required. The "Token File Path" line is required in any
# case, regardless of whether you type a value.
Multi Token File Path = officer1.token;
Token File Path = ;

# Total number of approvals
Number of Approvals = 2;

# Approver 1
# Type the approver's type, name, and the path to the file that
# contains the approver's signature.
Approver Type = 2; # 2 for CO, 1 for CU
Approver Name = officer1;
Approval File = officer1.token.sig1;

# Approver 2
# Type the approver's type, name, and the path to the file that
# contains the approver's signature.
Approver Type = 2; # 2 for CO, 1 for CU
Approver Name = officer2;
Approval File = officer1.token.sig2;
```

After creating the token approval file, the CO uses the cloudhsm_mgmt_util command line tool to log in to the HSM. The CO then uses the **approveToken** command to approve the token, as shown in the following example. Replace *approval.txt* with the name of the token approval file.

```
aws-cloudhsm > approveToken approval.txt
approveToken success on server 0(10.0.2.14)
approveToken success on server 1(10.0.1.4)
```

When this command succeeds, the HSM has approved the quorum token. To check the status of a token, use the **listTokens** command, as shown in the following example. The command's output shows that the token has the required number of approvals.

The token validity time indicates how long the token is guaranteed to persist on the HSM. Even after the token validity time elapses (zero seconds), you can still use the token.

```
aws-cloudhsm > listTokens
=====================
    Server 0(10.0.2.14)
=====================
-------- Token - 0 ----------
Token:
Id:1
Service:3
Node:1
Key Handle:0
User:officer1
Token Validity: 506 sec
Required num of approvers : 2
Current num of approvals : 2
Approver-0: officer1
Approver-1: officer2
Num of tokens = 1

=====================
    Server 1(10.0.1.4)
=====================
-------- Token - 0 ----------
Token:
Id:1
Service:3
Node:0
Key Handle:0
User:officer1
Token Validity: 506 sec
Required num of approvers : 2
Current num of approvals : 2
```

```
Approver-0: officer1
Approver-1: officer2
Num of tokens = 1

listTokens success
```

Step 4. Use the token for user management operations

After a CO has a token with the required number of approvals, as shown in the previous section, the CO can perform one of the following HSM user management operations:

- Create an HSM user with the createUser command
- Delete an HSM user with the **deleteUser** command
- Change a different HSM user's password with the **changePswd** command

For more information about using these commands, see HSM users.

The CO can use the token for only one operation. When that operation succeeds, the token is no longer valid. To do another HSM user management operation, the CO must get a new quorum token, get new signatures from approvers, and approve the new token on the HSM.

> **Note**
>
> The MofN token is only valid as long as your current login session is open. If you log out of cloudhsm_mgmt_util or the network connection disconnects, the token is no longer valid. Similarly, an authorized token can only be used within cloudhsm_mgmt_util, it cannot be used to authenticate in a different application.

In the following example command, the CO creates a new user on the HSM.

```
aws-cloudhsm > createUser CU user1 <password>
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)? y
```

```
Creating User user1(CU) on 2 nodes
```

After the previous command succeeds, a subsequent **listUsers** command shows the new user.

```
aws-cloudhsm > listUsers
Users on server 0(10.0.2.14):
Number of users found:8

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                                    NO
           0               NO
         2              AU              app_user                                 NO
           0               NO
         3              CO              officer1                                YES
           0               NO
         4              CO              officer2                                YES
           0               NO
         5              CO              officer3                                YES
           0               NO
         6              CO              officer4                                YES
           0               NO
         7              CO              officer5                                YES
           0               NO
         8              CU              user1                                    NO
           0               NO
Users on server 1(10.0.1.4):
Number of users found:8

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                                    NO
           0               NO
         2              AU              app_user                                 NO
           0               NO
         3              CO              officer1                                YES
           0               NO
         4              CO              officer2                                YES
           0               NO
         5              CO              officer3                                YES
           0               NO
         6              CO              officer4                                YES
           0               NO
```

```
         7              CO              officer5                                YES
           0               NO
         8              CU              user1                                    NO
           0               NO
```

If the CO tries to perform another HSM user management operation, it fails with a quorum authentication error, as shown in the following example.

```
aws-cloudhsm > deleteUser CU user1
Deleting user user1(CU) on 2 nodes
deleteUser failed: RET_MXN_AUTH_FAILED
deleteUser failed on server 0(10.0.2.14)

Retry/rollBack/Ignore?(R/B/I): I
deleteUser failed: RET_MXN_AUTH_FAILED
deleteUser failed on server 1(10.0.1.4)

Retry/rollBack/Ignore?(R/B/I): I
```

The **listTokens** command shows that the CO has no approved tokens, as shown in the following example. To perform another HSM user management operation, the CO must get a new quorum token, get new signatures from approvers, and approve the new token on the HSM.

```
aws-cloudhsm > listTokens
=====================
    Server 0(10.0.2.14)
=====================
Num of tokens = 0

=====================
    Server 1(10.0.1.4)
=====================
Num of tokens = 0

listTokens success
```

Change the quorum minimum value with AWS CloudHSM Management Utility

After you set the quorum minimum value so that AWS CloudHSM crypto officers (COs) can use quorum authentication, you might want to change the quorum minimum value. The HSM allows you to change the quorum minimum value only when the number of approvers is the same or higher than the current quorum minimum value. For example, if the quorum minimum value is two, at least two COs must approve to change the quorum minimum value.

To get quorum approval to change the quorum minimum value, you need a *quorum token* for the **setMValue** command (service 4). To get a quorum token for the **setMValue** command (service 4), the quorum minimum value for service 4 must be higher than one. This means that before you can change the quorum minimum value for COs (service 3), you might need to change the quorum minimum value for service 4.

The following table lists the HSM service identifiers along with their names, descriptions, and the commands that are included in the service.

Service Identifier	Service Name	Service Description	HSM Commands
3	`USER_MGMT`	HSM user management	• **createUser** • **deleteUser** • **changePswd** (applies only when changing the password of a different HSM user)
4	`MISC_CO`	Miscellaneous CO service	• **setMValue**

To change the quorum minimum value for crypto officers

1. Use the following command to start the cloudhsm_mgmt_util command line tool.

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

2. Use the **loginHSM** command to log in to the HSM as a CO. For more information, see ???.
3. Use the **getMValue** command to get the quorum minimum value for service 3. For more information, see the following example.
4. Use the **getMValue** command to get the quorum minimum value for service 4. For more information, see the following example.

5. If the quorum minimum value for service 4 is lower than the value for service 3, use the **setMValue** command to change the value for service 4. Change the value for service 4 to one that is the same or higher than the value for service 3. For more information, see the following example.
6. Get a *quorum token*, taking care to specify service 4 as the service for which you can use the token.
7. Get approvals (signatures) from other COs.
8. Approve the token on the HSM.
9. Use the **setMValue** command to change quorum minimum value for service 3 (user management operations performed by COs).

Example – Get quorum minimum values and change the value for service 4

The following example command shows that the quorum minimum value for service 3 is currently two.

```
aws-cloudhsm > getMValue 3
MValue of service 3[USER_MGMT] on server 0 : [2]
MValue of service 3[USER_MGMT] on server 1 : [2]
```

The following example command shows that the quorum minimum value for service 4 is currently one.

```
aws-cloudhsm > getMValue 4
MValue of service 4[MISC_CO] on server 0 : [1]
MValue of service 4[MISC_CO] on server 1 : [1]
```

To change the quorum minimum value for service 4, use the **setMValue** command, setting a value that is the same or higher than the value for service 3. The following example sets the quorum minimum value for service 4 to two (2), the same value that is set for service 3.

```
aws-cloudhsm > setMValue 4 2
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************
```

```
Do you want to continue(y/n)? y
Setting M Value(2) for 4 on 2 nodes
```

The following commands show that the quorum minimum value is now two for service 3 and service 4.

```
aws-cloudhsm > getMValue 3
MValue of service 3[USER_MGMT] on server 0 : [2]
MValue of service 3[USER_MGMT] on server 1 : [2]
```

```
aws-cloudhsm > getMValue 4
MValue of service 4[MISC_CO] on server 0 : [2]
MValue of service 4[MISC_CO] on server 1 : [2]
```

Keys in AWS CloudHSM

Before you can use your AWS CloudHSM cluster for cryptoprocessing, you must create users and keys on the hardware security modules (HSM) in your cluster.

In AWS CloudHSM, use any of the following to manage keys on the HSMs in your cluster:

- PKCS #11 library
- JCE provider
- CNG and KSP providers
- CloudHSM CLI

Before you can manage keys, you must log in to the HSM with the user name and password of a crypto user (CU). Only a CU can create a key. The CU who creates a key owns and manages that key.

See the following topics for more information about managing keys in AWS CloudHSM.

Topics

Key synchronization and durability settings in AWS CloudHSM

AWS CloudHSM synchronizes every token key you create. Key synchronization is mostly an automatic process, but you can use a minimum of two hardware security modules (HSM) in your cluster to make keys more durable. This topic describes key synchronization settings, common issues customers face working with keys on a cluster, and strategies for making keys more durable.

This topic describes key synchronization settings in AWS CloudHSM, common issues customers face working with keys on a cluster, and strategies for making keys more durable.

Topics

AWS CloudHSM key concepts

The following are concepts to be aware of when working with keys in AWS CloudHSM.

Token keys

Persistent keys that you create during key generate, import or unwrap operations. AWS CloudHSM synchronizes token keys across a cluster.

Session keys

Ephemeral keys that exist only on one hardware security module (HSM) in the cluster. AWS CloudHSM does *not* synchronize session keys across a cluster.

Client-side key synchronization

A client-side process that clones token keys you create during key generate, import or unwrap operations. You can make token keys more durable by running a cluster with a minimum of two HSMs.

Server-side key synchronization

Periodically clones keys to every HSM in the cluster. Requires no management.

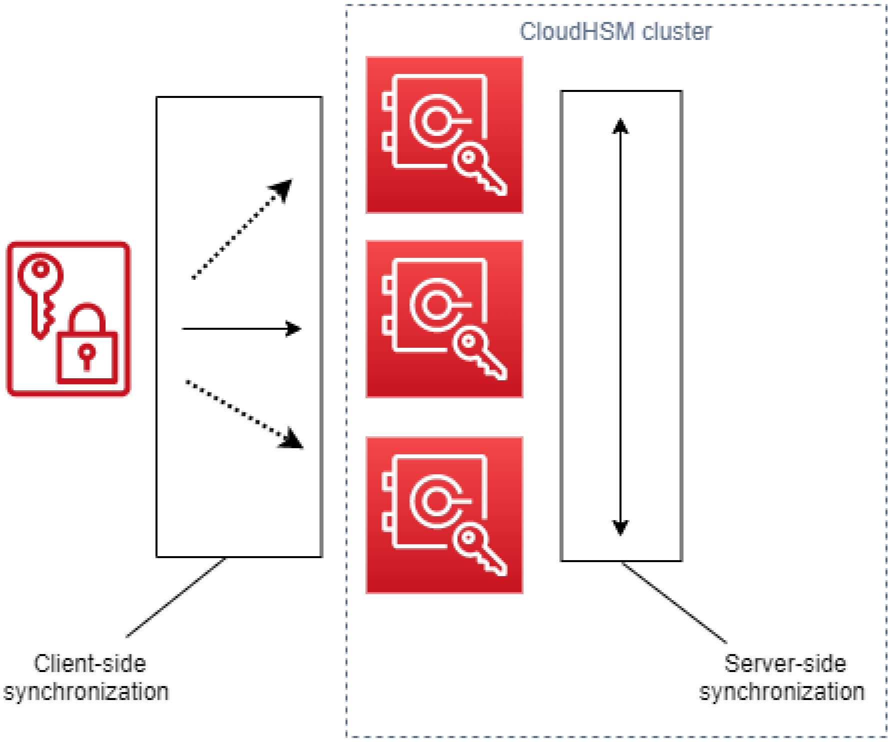
Client key durability settings

Settings you configure on the client that impact key durability. These settings work differently in Client SDK 5 and Client SDK 3.

- In Client SDK 5, use this setting to run a single HSM cluster.
- In Client SDK 3, use this setting to specify the number of HSMs required for key creation operations to succeed.

Understanding AWS CloudHSM key synchronization

AWS CloudHSM uses key synchronization to clone token keys across all the hardware security modules (HSM) in a cluster. You create token keys as persistent keys during key generation, import, or unwrap operations. To distribute these keys across the cluster, CloudHSM offers both client-side and server-side key synchronization.



The goal with key synchronization—both server side and client side—is to distribute new keys across the cluster as quickly as possible after you create them. This is important because the subsequent calls you make to use new keys can get routed to any available HSM in the cluster. If the call you make routes to an HSM without the key, then the call fails. You can mitigate these type failures by specifying that your applications retry subsequent calls made after key creation operations. The time required to synchronize can vary, depending on the workload of your cluster and other intangibles. Use CloudWatch metrics to determine the timing your application should employ in this type situation. For more information, see CloudWatch Metrics.

The challenge with key synchronization in a cloud environment is key durability. You create keys on a single HSM and often begin using those keys immediately. If the HSM on which you create keys should fail before the keys have been cloned to another HSM in the cluster, you lose the keys *and* access to anything encrypted by the keys. To mitigate this risk, we offer *client-side synchronization*. Client side synchronization is a client-side process that clones the keys you create during key generate, import, or unwrap operations. Cloning keys as you create them makes keys more durable. Of course, you can't clone keys in a cluster with a single HSM. To make keys more durable, we also recommend you configure your cluster to use a minimum of two HSMs. With client-side synchronization and a cluster with two HSMs, you can meet the challenge of key durability in a cloud environment.

Change AWS CloudHSM client key durability settings

Key synchronization is mostly an automatic process, but you can manage client-side key durability settings. Client-side key durability settings works differently in Client SDK 5 and Client SDK 3.

- In Client SDK 5, we introduce the concept of *key availability quorums* which requires you to run clusters with a minimum of two HSMs. You can use client-side key durability settings to opt out of the two HSM requirement. For more information about quorums, see the section called "Client SDK 5 concepts".
- In Client SDK 3, you use client-side key durability settings to specify the number of HSMs on which key creation must succeed for the overall operation to be deemed a success.

Client SDK 5 client key durability settings

In Client SDK 5, key synchronization is a fully automatic process. With key availability quorum, newly created keys must exist on two HSMs in the cluster before your application can use the key. To use key availability quorum, your cluster must have a minimum of two HSMs.

If your cluster configuration doesn't meet the key durability requirements, any attempt to create or use a token key will fail with the following error message in the logs:

```
Key <key handle> does not meet the availability requirements - The key must be
 available on at least 2 HSMs before being used.
```

You can use client configuration settings to opt out of key availability quorum. You might want to opt out to run a cluster with a single HSM, for example.

Client SDK 5 concepts

Key Availability Quorum

AWS CloudHSM specifies the number of HSMs in a cluster on which keys must exist before your application can use the key. Requires clusters with a minimum of two HSMs.

Managing client key durability settings

To manage client key durability settings, you must use the configure tool for Client SDK 5.

PKCS #11 library

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --disable-key-availability-check
```

To disable client key durability for Client SDK 5 on Windows

- Use the configure tool to disable client key durability settings.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" --disable-
key-availability-check
```

OpenSSL Dynamic Engine

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --disable-key-availability-check
```

Key Storage Provider (KSP)

To disable client key durability for Client SDK 5 on Windows

- Use the configure tool to disable client key durability settings.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" --disable-
key-availability-check
```

JCE provider

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
$ sudo /opt/cloudhsm/bin/configure-jce --disable-key-availability-check
```

To disable client key durability for Client SDK 5 on Windows

- Use the configure tool to disable client key durability settings.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" --disable-key-availability-check
```

CloudHSM CLI

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
$ sudo /opt/cloudhsm/bin/configure-cli --disable-key-availability-check
```

To disable client key durability for Client SDK 5 on Windows

- Use the configure tool to disable client key durability settings.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" --disable-key-availability-check
```

Client SDK 3 client key durability settings

In Client SDK 3, key synchronization is mostly an automatic process, but you can use the client key durability settings to make keys more durable. You specify the number of HSMs on which key creation must succeed for the overall operation to be deemed a success. Client-side synchronization always makes a best-effort attempt to clone keys to every HSM in the cluster no matter what setting you choose. Your setting enforces key creation on the number of HSMs you specify. If you specify a value and the system cannot replicate the key to that number of HSMs, then the system automatically cleans up any unwanted key material and you can try again.

Important

If you don't set client key durability settings (or if you use the default value of 1), your keys are vulnerable to loss. If your current HSM should fail before the server-side service has cloned that key to another HSM, you lose the key material.

To maximize key durability, consider specifying at least two HSMs for client-side synchronization. Remember that no matter how many HSMs you specify, the workload on your cluster remains the same. Client-side synchronization always makes a best-effort attempt to clone keys to every HSM in the cluster.

Recommendations

- **Minimum**: Two HSMs per cluster
- **Maximum**: One fewer than the total number of HSMs in your cluster

If client-side synchronization fails, the client service cleans up any unwanted keys that may have been created and are now unwanted. This clean up is a best-effort response that may not always work. If cleanup fails, you may have to delete unwanted key material. For more information, see Key Synchronization Failures.

Setting up the configuration file for client key durability

To specify client key durability settings, you must edit `cloudhsm_client.cfg`.

To edit the client configuration file

1. Open `cloudhsm_client.cfg`.

   Linux:

   ```
   /opt/cloudhsm/etc/cloudhsm_client.cfg
   ```

   Windows:

   ```
   C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
   ```

2. In the `client` node of the file, add `create_object_minimum_nodes` and specify a value for the minimum number of HSMs on which AWS CloudHSM must successfully create keys for key creation operations to succeed.

```
"create_object_minimum_nodes" : 2
```

> **Note**
>
> The key_mgmt_util (KMU) command-line tool has an additional setting for client key durability. For more information, see the section called "KMU and client-side synchronization"

Configuration reference

These are the client-side synchronization properties, shown in an excerpt of the `cloudhsm_client.cfg`:

```
{
    "client": {
        "create_object_minimum_nodes" : 2,
        ...
    },

    ...
}
```

create_object_minimum_nodes

Specifies the minimum number of HSMs required to deem key generation, key import, or key unwrap operations a success. If set, the default is 1. This means that for every key create operation, the client-side service attempts to create keys on every HSM in the cluster, but to return a success, only needs to create a *single key* on one HSM in the cluster.

KMU and client-side synchronization

If you create keys with the key_mgmt_util (KMU) command-line tool, you use an optional command line parameter (`-min_srv`) to *limit* the number of HSMs on which to clone keys. If you specify the command-line parameter *and* a value in the configuration file, AWS CloudHSM honors the LARGER of the two values.

For more information, see the following topics:

- genDSAKeyPair
- genECCKeyPair
- genRSAKeyPair
- genSymKey
- importPrivateKey
- importPubKey
- imSymKey
- insertMaskedObject
- unWrapKey

Synchronizing keys across cloned AWS CloudHSM clusters

Client-side and server-side synchronization are only for synchronizing keys within the *same* AWS CloudHSM cluster. If you copy a backup of a cluster to another region, you can use the syncKey command of the cloudhsm_mgmt_util (CMU) for synchronizing keys between clusters. You might use cloned clusters for cross-region redundancy or to simplify your disaster recovery process. For more information, see syncKey.

AES key wrapping in AWS CloudHSM

This topic describes the options for AES key wrapping in AWS CloudHSM. AES key wrapping uses an AES key (the wrapping key) to wrap another key of any type (the target key). You use key wrapping to protect stored keys or transmit keys over insecure networks.

Topics

- Supported algorithms
- Using AES key wrap in AWS CloudHSM

Supported algorithms

AWS CloudHSM offers three options for AES key wrapping, each based on how the target key is padded before being wrapped. Padding is done automatically, in accordance with the algorithm you use, when you call key wrap. The following table lists the supported algorithms and associated details to help you choose an appropriate wrapping mechanism for your application.

AES Key Wrap Algorithm	Specification	Supported Target Key Types	Padding Scheme	AWS CloudHSM Client Availability
AES Key Wrap with Zero Padding	RFC 5649 and SP 800–38F	All	Adds zeros after key bits, if necessary, to block align	SDK 3.1 and later
AES Key Wrap with No Padding	RFC 3394 and SP 800–38F	Block-aligned keys such as AES and 3DES	None	SDK 3.1 and later
AES Key Wrap with PKCS #5 Padding	None	All	At least 8 bytes are added as per PKCS #5 padding scheme to block align	All

To learn how to use the AES key wrap algorithms from the preceding table in your application, see Using AES Key Wrap in AWS CloudHSM.

Understanding initialization vectors in AES key wrap

Prior to wrapping, CloudHSM appends an initialization vector (IV) to the target key for data integrity. Each key wrap algorithm has specific restrictions on what type of IV is allowed. To set the IV in AWS CloudHSM, you have two options:

- Implicit: set the IV to NULL and CloudHSM uses the default value for that algorithm for wrap and unwrap operations (recommended)
- Explicit: set the IV by passing the default IV value to the key wrap function

Important

You must understand what IV you are using in your application. To unwrap the key, you must provide the same IV that you used to wrap the key. If you use an implicit IV to wrap, then use an implicit IV to unwrap. With an implicit IV, CloudHSM will use the default value to unwrap.

The following table describes permitted values for IVs, which the wrapping algorithm specifies.

AES Key Wrap Algorithm	Implicit IV	Explicit IV
AES Key Wrap with Zero Padding	Required Default value: (IV calculated internally based on specification)	Not allowed
AES Key Wrap with No Padding	Allowed (recommended) Default value: `0xA6A6A6A6A6A6A6A6`	Allowed Only this value accepted: `0xA6A6A6A6A6A6A6A6`
AES Key Wrap with PKCS #5 Padding	Allowed (recommended) Default value: `0xA6A6A6A6A6A6A6A6`	Allowed Only this value accepted: `0xA6A6A6A6A6A6A6A6`

Using AES key wrap in AWS CloudHSM

You wrap and unwrap keys as follows:

- In the PKCS #11 library, select the appropriate mechanism for the `C_WrapKey` and `C_UnWrapKey` functions as shown in the following table.
- In the JCE provider, select the appropriate algorithm, mode and padding combination, implementing cipher methods `Cipher.WRAP_MODE` and `Cipher.UNWRAP_MODE` as shown in the following table.

- In the CloudHSM CLI, choose the appropriate algorithm from the list of supported The key wrap command in CloudHSM CLI and The key unwrap command in CloudHSM CLI algorithms as shown in the following table.
- In key_mgmt_util (KMU), use commands Export an AWS CloudHSM key using KMU and Unwrap an AWS CloudHSM key using KMU with appropriate m values as shown in the following table.

AES Key Wrap Algorithm	PKCS #11 Mechanism	Java Method	CloudHSM CLI Sub Command	Key Management Utility (KMU) Argument
AES Key Wrap with Zero Padding	• `CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD` (Vendor Defined Mechanism)	`AESWrap/ECB/ZeroPadding`	aes-zero-pad	m = 6
AES Key Wrap with No Padding	• `CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD` (Vendor Defined Mechanism)	`AESWrap/ECB/NoPadding`	aes-no-pad	m = 5
AES Key Wrap with PKCS #5 Padding	• `CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD` (Vendor Defined Mechanism)	`AESWrap/ECB/PKCS5Padding`	aes-pkcs5-pad	m = 4

Using trusted keys in AWS CloudHSM

AWS CloudHSM supports trusted key wrapping to protect data keys from insider threats. This topic describes how to create trusted keys to secure data.

Topics

Understanding trusted keys in AWS CloudHSM

A *trusted key* is a key that is used to wrap other keys and that admins and cryptographic officers (COs) specifically identify as trusted using the attribute `CKA_TRUSTED`. Additionally, admins and cryptographic officers (COs) use `CKA_UNWRAP_TEMPLATE` and related attributes to specify what actions data keys can do once they are unwrapped by a trusted key. Data keys that are unwrapped by the trusted key must also contain these attributes for the unwrap operation to succeed, which helps ensure that unwrapped data keys are only permitted for the use you intend.

Use the attribute `CKA_WRAP_WITH_TRUSTED` to identify all of the data keys you want to wrap with trusted keys. Doing this allows you to restrict data keys so applications can only use trusted keys to unwrap them. Once you set this attribute on the data keys, the attribute becomes read-only and you cannot change it. With these attributes in place, applications can only unwrap your data keys with the keys you trust, and unwraps always result in data keys with attributes that limit how these keys can be used.

Trusted key attributes in AWS CloudHSM

The following attributes allow you to mark an AWS CloudHSM key as trusted, specify a data key can only be wrapped and unwrapped with a trusted key, and control what a data key can do after it is unwrapped:

- `CKA_TRUSTED`: Apply this attribute (in addition to `CKA_UNWRAP_TEMPLATE`) to the key that will wrap data keys to specify that an admin or crypto officer (CO) has done the necessary diligence and trusts this key. Only an admin or CO can set `CKA_TRUSTED`. The crypto user (CU) owns the key, but only a CO can set its `CKA_TRUSTED` attribute.

- CKA_WRAP_WITH_TRUSTED: Apply this attribute to an exportable data key to specify that you can only wrap this key with keys marked as CKA_TRUSTED. Once you set CKA_WRAP_WITH_TRUSTED to true, the attribute becomes read-only and you cannot change or remove the attribute.
- CKA_UNWRAP_TEMPLATE: Apply this attribute to the wrapping key (in addition to CKA_TRUSTED) to specify which attribute names and values the service must automatically apply to data keys that the service unwraps. When an application submits a key for unwrapping, the application can also provide its own unwrap template. If you specify an unwrap template and the application provides its own unwrap template, the HSM uses both templates to apply attribute names and values to the key. However, if a value in the CKA_UNWRAP_TEMPLATE for the wrapping key conflicts with an attribute provided by the application during the unwrap request, then the unwrap request fails.

For more information about attributes, refer to the following topics:

- PKCS #11 key attributes
- JCE key attributes
- CloudHSM CLI key attributes

How to use trusted keys to wrap data keys in AWS CloudHSM

To use a trusted key to wrap a data key in AWS CloudHSM, you must complete three basic steps:

1. For the data key you plan to wrap with a trusted key, set its CKA_WRAP_WITH_TRUSTED attribute to true.
2. For the trusted key you plan to wrap the data key with, set its CKA_TRUSTED attribute to true.
3. Use the trusted key to wrap the data key.

Step 1: Set the data key's CKA_WRAP_WITH_TRUSTED to true

For the data key you want to wrap, choose one of the following options to set the key’s CKA_WRAP_WITH_TRUSTED attribute to true. Doing this restricts the data key so applications can only use trusted keys to wrap it.

Option 1: If generating a new key, set CKA_WRAP_WITH_TRUSTED to true

Generate a key using PKCS #11, JCE, or CloudHSM CLI. See the following examples for more details.

PKCS #11

To generate a key with PKCS #11, you need to set the key's `CKA_WRAP_WITH_TRUSTED` attribute to true. As shown in the following example, do this by including this attribute in the key's `CK_ATTRIBUTE template` and then setting the attribute to true:

```
CK_BYTE_PTR label = "test_key";
CK_ATTRIBUTE template[] = {
        {CKA_WRAP_WITH_TRUSTED, &true_val,         sizeof(CK_BBOOL)},
        {CKA_LABEL,             label,             strlen(label)},
        ...
};
```

For more information, see our public samples demonstrating key generation with PKCS #11.

JCE

To generate a key with JCE, you need to set the key's `WRAP_WITH_TRUSTED` attribute to true. As shown in the following example, do this by including this attribute in the key's `KeyAttributesMap` and then setting the attribute to true:

```
final String label = "test_key";
final KeyAttributesMap keySpec = new KeyAttributesMap();
keySpec.put(KeyAttribute.WRAP_WITH_TRUSTED, true);
keySpec.put(KeyAttribute.LABEL, label);
...
```

For more information, see our public samples demonstrating key generation with JCE.

CloudHSM CLI

To generate a key with CloudHSM CLI, you need to set the key's `wrap-with-trusted` attribute to true. Do this by including `wrap-with-trusted=true` in the appropriate argument for the key generation command:

- For symmetric keys, add `wrap-with-trusted` to the `attributes` argument.

- For public keys, add `wrap-with-trusted` to the `public-attributes` argument.
- For private keys, add `wrap-with-trusted` to the `private-attributes` argument.

For more information on key pair generation, see The generate-asymmetric-pair category in CloudHSM CLI.

For more information on symmetric key generation, see The generate-symmetric category in CloudHSM CLI.

Option 2: If using an existing key, use CloudHSM CLI to set its CKA_WRAP_WITH_TRUSTED to true

To set an existing key's `CKA_WRAP_WITH_TRUSTED` attribute to true, follow these steps:

1. Use the Log in to an HSM using CloudHSM CLI command to log in as a crypto user (CU).
2. Use the Set the attributes of keys with CloudHSM CLI command to set the key's `wrap-with-trusted` attribute to true.

```
aws-cloudhsm > key set-attribute --filter attr.label=test_key --name wrap-with-trusted --value true
{
  "error_code": 0,
  "data": {
    "message": "Attribute set successfully"
  }
}
```

Step 2: Set the trusted key's CKA_TRUSTED to true

To make a key a trusted key, its `CKA_TRUSTED` attribute must be set to true. You can either use CloudHSM CLI or the CloudHSM Management Utility (CMU) to do this.

- If using CloudHSM CLI to set a key's `CKA_TRUSTED` attribute, see Mark a key as trusted using CloudHSM CLI.
- If using the CMU to set a key's `CKA_TRUSTED` attribute, see How to mark a key as trusted with the AWS CloudHSM Management Utility.

Step 3. Use the trusted key to wrap the data key

To wrap the data key referenced in Step 1 with the trusted key you set in Step 2, refer to the following links for code samples. Each demonstrates how to wrap keys.

- AWS CloudHSM PKCS #11 examples
- AWS CloudHSM JCE examples

How to unwrap a data key with a trusted key for AWS CloudHSM

To unwrap a data key in AWS CloudHSM, you need a trusted key that has `CKA_UNWRAP` set to true. To be such a key, it must also meet the following criteria:

- The key's `CKA_TRUSTED` attribute must be set to true.
- The key must use `CKA_UNWRAP_TEMPLATE` and related attributes to specify what actions data keys can perform once they are unwrapped. If, for example, you want an unwrapped key to be non-exportable, you set `CKA_EXPORTABLE = FALSE` as part of the `CKA_UNWRAP_TEMPLATE`.

> **Note**
>
> `CKA_UNWRAP_TEMPLATE` is only available with PKCS #11.

When an application submits a key to be unwrapped, the application can also provide its own unwrap template. If you specify an unwrap template and the application provides its own unwrap template, the HSM uses both templates to apply attribute names and values to the key. However, if during the unwrap request a value in the trusted key's `CKA_UNWRAP_TEMPLATE` conflicts with an attribute provided by the application, the unwrap request fails.

To see an example on unwrapping a data key with a trusted key, refer to this PKCS #11 example.

Key management with CloudHSM CLI

If using the latest SDK version series, use CloudHSM CLI to manage the keys in your AWS CloudHSM cluster. For more details, see the topics below.

- Using trusted keys describes how to use CloudHSM CLI to create trusted keys to secure data.

- Generating keys includes instructions on creating keys, including symmetric keys, RSA keys, and EC keys.
- Deleting keys describes how key owners delete keys.
- Sharing and unsharing keys details how key owners share and unshare keys.
- Filtering keys offers guidelines on how to use filters to find keys.
- Manage key quorum authentication (M of N) offers guidelines on how to setup and use quorum authentication with keys.

Generate keys with CloudHSM CLI

Before you can generate a key, you must start CloudHSM CLI and log in as a crypto user (CU). To generate keys on the HSM, use the command that corresponds to the type of key that you want to generate.

Topics

- Generate symmetric keys with CloudHSM CLI
- Generate asymmetric keys using CloudHSM CLI
- AWS CloudHSM key related topics

Generate symmetric keys with CloudHSM CLI

Use the commands listed in **The generate-symmetric category in CloudHSM CLI** to generate symmetric keys for AWS CloudHSM. To see all available options, use the **help key generate-symmetric** command.

Generate an AES key

Use the **key generate-symmetric aes** command to generate AES keys. To see all available options, use the **help key generate-symmetric aes** command.

Example

The following example generates a 32-byte AES key.

```
aws-cloudhsm > key generate-symmetric aes \
    --label aes-example \
    --key-length-bytes 32
```

Arguments

<LABEL>

Specifies a user-defined label for the AES key.

Required: Yes

<KEY-LENGTH-BYTES>

Specifies the key length in bytes.

Valid values:

- 16, 24, and 32

Required: Yes

<KEY_ATTRIBUTES>

Specifies a space separated list of key attributes to set for the generated AES key in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (for example, `sign=true`)

For a list of supported AWS CloudHSM key attributes, see Key attributes for CloudHSM CLI.

Required: No

<SESSION>

Creates a key that exists only in the current session. The key cannot be recovered after the session ends. Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use key set-attribute.

By default, when keys are generated they are persistent/token keys. Using <SESSION> changes this, ensuring a key generated with this argument is a session/ephemeral

Required: No

Generate generic secret key

Use the **key generate-symmetric generic-secret** command to generate generic secret keys. To see all available options, use the **help key generate-symmetric generic-secret** command.

Example

The following example generates a 32-byte generic secret key.

```
aws-cloudhsm > key generate-symmetric generic-secret \
    --label generic-secret-example \
    --key-length-bytes 32
```

Arguments

<LABEL>

Specifies a user-defined label for the generic secret key.

Required: Yes

<KEY-LENGTH-BYTES>

Specifies the key length in bytes.

Valid values:

- 1 to 800

Required: Yes

<KEY_ATTRIBUTES>

Specifies a space separated list of key attributes to set for the generated generic secret key in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (for example, `sign=true`)

For a list of supported AWS CloudHSM key attributes, see [Key attributes for CloudHSM CLI](#).

Required: No

<SESSION>

Creates a key that exists only in the current session. The key cannot be recovered after the session ends. Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use [key set-attribute](#).

By default, when keys are generated they are persistent/token keys. Using <SESSION> changes this, ensuring a key generated with this argument is a session/ephemeral

Required: No

Generate asymmetric keys using CloudHSM CLI

Use the commands listed in [The generate-asymmetric-pair category in CloudHSM CLI]() to generate asymmetric key pairs for AWS CloudHSM clusters.

Generate an RSA key

Use the **key generate-asymmetric-pair rsa** command to generate an RSA key pair. To see all available options, use the **help key generate-asymmetric-pair rsa** command.

Example

The following example generates an RSA 2048-bit key pair.

```
aws-cloudhsm > key generate-asymmetric-pair rsa \
    --public-exponent 65537 \
    --modulus-size-bits 2048 \
    --public-label rsa-public-example \
    --private-label rsa-private-example
```

Arguments

<PUBLIC_LABEL>

Specifies a user-defined label for the public-key.

Required: Yes

<PRIVATE_LABEL>

Specifies a user-defined label for the private-key.

Required: Yes

<MODULUS_SIZE_BITS>

Specifies the length of the modulus in bits. The minimum value is 2048.

Required: Yes

<PUBLIC_EXPONENT>

Specifies the public exponent. The value must be an odd number greater than or equal to 65537.

Required: Yes

<PUBLIC_KEY_ATTRIBUTES>

Specifies a space-separated list of key attributes to set for the generated RSA public key in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (for example, `sign=true`).

For a list of supported AWS CloudHSM key attributes, see Key attributes for CloudHSM CLI.

Required: No

<SESSION>

Creates a key that exists only in the current session. The key cannot be recovered after the session ends. Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use key set-attribute.

By default, when keys are generated they are persistent/token keys. Using <SESSION> changes this, ensuring a key generated with this argument is a session/ephemeral

Required: No

Generate EC (elliptic curve cryptography) key pairs

Use the **key generate-asymmetric-pair ec** command to generate an EC key pair. To see all available options,including a list of the supported elliptic curves, use the **help key generate-asymmetric-pair ec** command.

Example

The following example generates an EC key pair using the Secp384r1 elliptic curve.

```
aws-cloudhsm > key generate-asymmetric-pair ec \
```

```
    --curve secp384r1 \
    --public-label ec-public-example \
    --private-label ec-private-example
```

Arguments

<PUBLIC_LABEL>

Specifies a user-defined label for the public-key. The maximum size allowable for `label` is 127 characters for Client SDK 5.11 and after. Client SDK 5.10 and before has a limit of 126 characters.

Required: Yes

<PRIVATE_LABEL>

Specifies a user-defined label for the private-key. The maximum size allowable for `label` is 127 characters for Client SDK 5.11 and after. Client SDK 5.10 and before has a limit of 126 characters.

Required: Yes

<CURVE>

Specifies the identifier for the elliptic curve.

Valid values:

- prime256v1
- secp256r1
- secp224r1
- secp384r1
- secp256k1
- secp521r1

Required: Yes

<PUBLIC_KEY_ATTRIBUTES>

Specifies a space-separated list of key attributes to set for the generated EC public key in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (for example, `verify=true`).

For a list of supported AWS CloudHSM key attributes, see Key attributes for CloudHSM CLI.

Required: No

<PRIVATE_KEY_ATTRIBUTES>

Specifies a space-separated list of key attributes to set for the generated EC private key in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (for example, `sign=true`).

For a list of supported AWS CloudHSM key attributes, see Key attributes for CloudHSM CLI.

Required: No

<SESSION>

Creates a key that exists only in the current session. The key cannot be recovered after the session ends. Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use key set-attribute.

By default, keys that are generated are persistent (token) keys. Passing in <SESSION> changes this, ensuring a key generated with this argument is a session (ephemeral) key.

Required: No

AWS CloudHSM key related topics

See the following sections for additional information about keys in AWS CloudHSM.

- Key attributes for CloudHSM CLI
- The generate-asymmetric-pair category in CloudHSM CLI
- The generate-symmetric category in CloudHSM CLI

Delete keys using CloudHSM CLI

Use the example in this topic to delete a key with CloudHSM CLI. Only key owners can delete keys.

Topics

- Example: Delete a key
- Related topics

Example: Delete a key

1. Run the **key list** command to identify the key you want to delete:

```
aws-cloudhsm > key list --filter attr.label="my_key_to_delete" --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x0000000000540011",
        "key-info": {
          "key-owners": [
            {
              "username": "my_crypto_user",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "my_key_to_delete",
          "id": "",
          "check-value": "0x29bbd1",
          "class": "private-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
```

```
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1217,
          "public-exponent": "0x010001",
          "modulus":
 "0x8b3a7c20618e8be08220ed8ab2c8550b65fc1aad8d4cf04fbf2be685f97eeb78fcbbad9b02cd91a3b15e990
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
```

2. After identifying the key, run the **key delete** with the key's unique `label` attribute to delete the key:

```
aws-cloudhsm > key delete --filter attr.label="my_key_to_delete"
{
  "error_code": 0,
  "data": {
    "message": "Key deleted successfully"
  }
}
```

3. Run the **key list** command with the key's unique `label` attribute and confirm the key has been deleted. As shown in the following example, no key with the label `my_key_to_delete` is in the HSM cluster:

```
aws-cloudhsm > key list --filter attr.label="my_key_to_delete"
{
  "error_code": 0,
  "data": {
    "matched_keys": [],
    "total_key_count": 0,
    "returned_key_count": 0
  }
}
```

Related topics

- Key attributes for CloudHSM CLI
- Delete a key with CloudHSM CLI

Share and unshare keys using CloudHSM CLI

Use the commands in this topic to share and unshare keys in CloudHSM CLI. In AWS CloudHSM, the crypto user (CU) who creates the key owns it. The owner can use the **key share** and **key unshare** commands to share and unshare the key with other CUs. Users with whom the key is shared can use the key in cryptographic operations, but they cannot export the key, delete the key, or share it with other users.

Before you can share a key, you must log in to the HSM as the crypto user (CU) who owns the key.

Topics

- Example: Sharing and unsharing a key
- Related topics

Example: Sharing and unsharing a key

Example

The following example shows how to share and unshare a key with crypto user (CU) `alice`. Along with the **key share** and **key unshare** commands, sharing and unsharing commands also requires a specific key using CloudHSM CLI key filters and the specific username of the user whom the key will be shared or unshared with.

1. Start by running the **key list** command with a filter to return a specific key and see whom the key is already shared with.

```
aws-cloudhsm > key list --filter attr.label="rsa_key_to_share" --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
```

```
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            },
            {
              "username": "cu1",
              "key-coverage": "full"
            },
            {
              "username": "cu4",
              "key-coverage": "full"
            },
            {
              "username": "cu5",
              "key-coverage": "full"
            },
            {
              "username": "cu6",
              "key-coverage": "full"
            },
            {
              "username": "cu7",
              "key-coverage": "full"
            },
          ],
          "key-quorum-values": {
            "manage-key-quorum-value": 0,
            "use-key-quorum-value": 0
          },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "rsa_key_to_share",
          "id": "",
          "check-value": "0xae8ff0",
          "class": "private-key",
```

```
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1219,
          "public-exponent": "0x010001",
          "modulus":
 "0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```

2. View the `shared-users` output to identify whom the key is currently shared with.

3. To share this key with crypto user (CU) `alice`, enter the following command:

```
aws-cloudhsm > key share --filter attr.label="rsa_key_to_share" attr.class=private-
key --username alice --role crypto-user
{
  "error_code": 0,
  "data": {
    "message": "Key shared successfully"
  }
}
```

Note that, along with the **key share** command, this command uses the unique label of the key and the name of the user whom the key will be shared with.

4. Run the **key list** command to confirm that the key has been shared with `alice`:

```
aws-cloudhsm > key list --filter attr.label="rsa_key_to_share" --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            },
            {
              "username": "cu1",
              "key-coverage": "full"
            },
            {
              "username": "cu4",
              "key-coverage": "full"
            },
            {
              "username": "cu5",
              "key-coverage": "full"
            },
            {
              "username": "cu6",
              "key-coverage": "full"
            },
            {
              "username": "cu7",
              "key-coverage": "full"
```

```
           },
           {
             "username": "alice",
             "key-coverage": "full"
           }
         ],
         "key-quorum-values": {
           "manage-key-quorum-value": 0,
           "use-key-quorum-value": 0
         },
         "cluster-coverage": "full"
       },
       "attributes": {
         "key-type": "rsa",
         "label": "rsa_key_to_share",
         "id": "",
         "check-value": "0xae8ff0",
         "class": "private-key",
         "encrypt": false,
         "decrypt": true,
         "token": true,
         "always-sensitive": true,
         "derive": false,
         "destroyable": true,
         "extractable": true,
         "local": true,
         "modifiable": true,
         "never-extractable": false,
         "private": true,
         "sensitive": true,
         "sign": true,
         "trusted": false,
         "unwrap": true,
         "verify": false,
         "wrap": false,
         "wrap-with-trusted": false,
         "key-length-bytes": 1219,
         "public-exponent": "0x010001",
         "modulus":
"0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254
         "modulus-size-bits": 2048
       }
     }
   ],
```

```
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```

5. To unshare the same key with `alice`, run the following **unshare** command:

```
aws-cloudhsm > key unshare --filter attr.label="rsa_key_to_share"
 attr.class=private-key --username alice --role crypto-user
{
  "error_code": 0,
  "data": {
    "message": "Key unshared successfully"
  }
}
```

   Note that, along with the **key unshare** command, this command uses the unique label of the key and the name of the user whom the key will be shared with.

6. Run the **key list** command again and confirm the key was unshared with crypto user `alice`:

```
aws-cloudhsm > key list --filter attr.label="rsa_key_to_share" --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            },
            {
              "username": "cu1",
              "key-coverage": "full"
```

```
          },
          {
            "username": "cu4",
            "key-coverage": "full"
          },
          {
            "username": "cu5",
            "key-coverage": "full"
          },
          {
            "username": "cu6",
            "key-coverage": "full"
          },
          {
            "username": "cu7",
            "key-coverage": "full"
          },
        ],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa_key_to_share",
        "id": "",
        "check-value": "0xae8ff0",
        "class": "private-key",
        "encrypt": false,
        "decrypt": true,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
```

```
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1219,
          "public-exponent": "0x010001",
          "modulus":
 "0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```

Related topics

- [Key attributes for CloudHSM CLI]
- [Share a key using CloudHSM CLI]
- [Unshare a key using CloudHSM CLI]
- [Filter keys using CloudHSM CLI]

Filter keys using CloudHSM CLI

Use the following key commands to utilize the standardized key filtration mechanisms for CloudHSM CLI.

- **key list**
- **key delete**
- **key share**
- **key unshare**
- **key set-attribute**

To select and/or filter keys with CloudHSM CLI, key commands utilize a standardized filtration mechanism based on Key attributes for CloudHSM CLI. A key or set of keys can be specified in key commands by using one or more AWS CloudHSM attributes that can identify a single key or multiple keys. The key filtration mechanism operates only on keys that the currently logged in user owns and shares, as well as all public keys in the AWS CloudHSM cluster.

Topics

- Requirements
- Filtering to find a single key
- Filtration Errors
- Related topics

Requirements

To filter keys, you must be a logged in as a crypto user (CUs).

Filtering to find a single key

Please note that in the following examples, each attribute that is used as a filter must be written in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE`. For example, if you want to filter by the label attribute, you will write `attr.label=my_label`.

Example Use a single attribute to find a single key

This example demonstrates how to filter to a single unique key using only a single identifying attribute.

```
aws-cloudhsm > key list --filter attr.label="my_unique_key_label" --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
```

```
          {
            "username": "alice",
            "key-coverage": "full"
          }
        ],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "my_unique_key_label",
        "id": "",
        "check-value": "0xae8ff0",
        "class": "private-key",
        "encrypt": false,
        "decrypt": true,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": true,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 1219,
        "public-exponent": "0x010001",
        "modulus":
"0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254c8f5
        "modulus-size-bits": 2048
      }
    }
  ],
  "total_key_count": 1,
```

```
    "returned_key_count": 1
  }
}
```

Example Use a multiple attributes to find a single key

The following example demonstrates how to find a single key using multiple key attributes.

```
aws-cloudhsm > key list --filter attr.key-type=rsa attr.class=private-key attr.check-value=0x29bbd1 --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x0000000000540011",
        "key-info": {
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            }
          ],
          "key-quorum-values": {
            "manage-key-quorum-value": 0,
            "use-key-quorum-value": 0
          },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "my_crypto_user",
          "id": "",
          "check-value": "0x29bbd1",
          "class": "my_test_key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
```

```
            "always-sensitive": true,
            "derive": false,
            "destroyable": true,
            "extractable": true,
            "local": true,
            "modifiable": true,
            "never-extractable": false,
            "private": true,
            "sensitive": true,
            "sign": true,
            "trusted": false,
            "unwrap": true,
            "verify": false,
            "wrap": false,
            "wrap-with-trusted": false,
            "key-length-bytes": 1217,
            "public-exponent": "0x010001",
            "modulus":
 "0x8b3a7c20618e8be08220ed8ab2c8550b65fc1aad8d4cf04fbf2be685f97eeb78fcbbad9b02cd91a3b15e990c2a7
            "modulus-size-bits": 2048
          }
        }
      ],
      "total_key_count": 1,
      "returned_key_count": 1
    }
}
```

Example Filtering to find a set of keys

The following example demonstrates how to filter to find a set of private rsa keys.

```
aws-cloudhsm > key list --filter attr.key-type=rsa attr.class=private-key --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
            {
              "username": "my_crypto_user",
              "key-coverage": "full"
```

```
          }
        ],
        "shared-users": [
          {
            "username": "cu2",
            "key-coverage": "full"
          },
          {
            "username": "cu1",
            "key-coverage": "full"
          },
        ],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa_key_to_share",
        "id": "",
        "check-value": "0xae8ff0",
        "class": "private-key",
        "encrypt": false,
        "decrypt": true,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": true,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 1219,
        "public-exponent": "0x010001",
```

```
          "modulus":
 "0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254c8f5
          "modulus-size-bits": 2048
        }
      },
      {
        "key-reference": "0x0000000000540011",
        "key-info": {
          "key-owners": [
            {
              "username": "my_crypto_user",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            }
          ],
          "key-quorum-values": {
            "manage-key-quorum-value": 0,
            "use-key-quorum-value": 0
          },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "my_test_key",
          "id": "",
          "check-value": "0x29bbd1",
          "class": "private-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
```

```
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1217,
          "public-exponent": "0x010001",
          "modulus":
 "0x8b3a7c20618e8be08220ed8ab2c8550b65fc1aad8d4cf04fbf2be685f97eeb78fcbbad9b02cd91a3b15e990c2a7
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 2,
    "returned_key_count": 2
  }
}
```

Filtration Errors

Certain key operations can only be performed on a single key at a time. For these operations, the CloudHSM CLI will provide an error if the filtration criteria is not sufficiently refined and multiple keys match the criteria. One such example is shown below with the key delete.

Example Filtration Error when matching too many keys

```
aws-cloudhsm > key delete --filter attr.key-type=rsa
{
  "error_code": 1,
  "data": "Key selection criteria matched 48 keys. Refine selection criteria to select
 a single key."
}
```

Related topics

- Key attributes for CloudHSM CLI

Mark a key as trusted using CloudHSM CLI

The content in this section provides instructions on using CloudHSM CLI to mark a key as trusted.

1. Using the [CloudHSM CLI **login** command](), log in as a crypto user (CU).
2. Use the **key list** command to identify the key reference of the key you want to mark as trusted. The following example lists the key with the label `key_to_be_trusted`.

```
aws-cloudhsm > key list --filter attr.label=test_aes_trusted
        {
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x0000000000200333",
        "attributes": {
          "label": "test_aes_trusted"
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```

3. Using the Log out of an HSM using CloudHSM CLI command, log out as a crypto user (CU).
4. Using the Log in to an HSM using CloudHSM CLI command, log in as an admin.
5. Using the key set-attribute command with the key reference you identified in step 2, set the key's trusted value to true:

```
aws-cloudhsm > key set-attribute --filter key-reference=<Key Reference> --name
 trusted --value true
{
  "error_code": 0,
  "data": {
    "message": "Attribute set successfully"
  }
}
```

Manage quorum authentication (M of N access control) using CloudHSM CLI

The hardware security modules (HSMs) in your AWS CloudHSM cluster support quorum authentication, also known as M of N access control. With quorum authentication, no single user on the HSM can perform quorum-controlled operations. Instead, a minimum number of HSM users (at least 2) must cooperate to do these operations. Quorum authentication adds an extra layer of protection by requiring approvals from multiple HSM users.

Quorum authentication can control the following operations:

- HSM key usage and management by a crypto-user – Creating signatures with a key, or wrapping, unwrapping, sharing, unsharing, and setting an attribute of a key.

Important considerations

- An HSM user can sign their own quorum token—that is, the requesting user can provide one of the required approvals for quorum authentication.
- You choose the minimum number of quorum approvers for quorum-controlled operations. The smallest number you can choose is two (2), and the largest number you can choose is eight (8).
- The HSM can store up to 1,024 quorum tokens. If the HSM already has 1,024 tokens when you try to create a new one, the HSM purges one of the expired tokens. By default, tokens expire ten minutes after their creation.
- If multi-factor authentication (MFA) is enabled, the cluster uses the same key for quorum authentication and for MFA. For more information about using quorum authentication and MFA, see Using CloudHSM CLI to manage MFA.
- Each HSM can only contain one token per Admin service at a time, but multiple tokens per Crypto User service.

The following topics provide more information about quorum authentication in AWS CloudHSM.

Topics

- Key management and usage with quorum authentication enabled for AWS CloudHSM using CloudHSM CLI

Quorum authentication process for CloudHSM CLI

The following steps summarize the quorum authentication processes for CloudHSM CLI. For the specific steps and tools, see Key management and usage with quorum authentication enabled for AWS CloudHSM using CloudHSM CLI.

1. Each hardware security module (HSM) user creates an asymmetric key for signing. Users do this outside of the HSM, taking care to protect the key appropriately.
2. Each HSM user logs in to the HSM and registers the public part of their signing key (the public key) with the HSM.
3. When an HSM user wants to do a quorum-controlled operation, the same user logs in to the HSM and gets a *quorum token*.
4. The HSM user gives the quorum token to one or more other HSM users and asks for their approval.
5. The other HSM users approve by using their keys to cryptographically sign the quorum token. This occurs outside the HSM.
6. When the HSM user has the required number of approvals, the same user logs in to the HSM and runs the quorum-controlled operation with the **--approval** argument, supplying the signed quorum token file, which contains all necessary approvals (signatures).
7. The HSM uses the registered public keys of each signer to verify the signatures. If the signatures are valid, the HSM approves the token and the quorum-controlled operation is performed.

Supported AWS CloudHSM service names and types for quorum authentication with CloudHSM CLI

Admin Services: Quorum authentication is used for admin privileged services like creating users, deleting users, changing user passwords, setting quorum values, and deactivating quorum and MFA capabilities.

Crypto User Services: Quorum authentication is used for crypto-user privileged services associated with a specific key like signing with a key, sharing/unsharing a key, wrapping/unwrapping a key, and setting a key's attribute. The quorum value of an associated key is configured when the key is generated, imported, or unwrapped. The quorum value must be equal to or less than the number of users that the key is associated with, which includes users that the key is shared with and the key owner.

Each service type is further broken down into a qualifying service name, which contains a specific set of quorum supported service operations that can be performed.

Service name	Service type	Service operations
user	Admin	• user create • user delete • user change-password • user change-mfa
quorum	Admin	• quorum token-sign set-quorum-value
cluster[1]	Admin	• cluster mtls register-trust-anchor • cluster mtls deregister-trust-anchor • cluster mtls set-enforcement
key-management	Crypto User	• key wrap • key unwrap • key share • key unshare • key set-attribute
key-usage	Crypto User	• key sign

[1] Cluster service is exclusively available on hsm2m.medium

Set up quorum authentication for AWS CloudHSM crypto-users using CloudHSM CLI

These topics describe how to configure your CloudHSM for quorum authentication by crypto-users. Perform these steps once during initial setup. For subsequent key management and usage, refer to Key management and usage with quorum authentication enabled for AWS CloudHSM using CloudHSM CLI.

Topics

Prerequisites

- Familiarity with the CloudHSM CLI

Step 1. Create and register a key for signing

To use quorum authentication, each crypto-user must complete *all* of the following steps:

Topics

Create an RSA key pair

There are many different ways to create and protect a key pair. The following examples show how to do it with OpenSSL.

Example – Create a private key with OpenSSL

The following example demonstrates how to use OpenSSL to create a 2048-bit RSA key. To use this example, replace *<crypto_user1.key>* with the name of the file where you want to store the key.

```
$ openssl genrsa -out <crypto_user1.key>
Generating RSA private key, 2048 bit long modulus
.....................................+++
.+++
e is 65537 (0x10001)
```

Next, generate the public key using the private key that you just created.

Example – Create a public key with OpenSSL

The following example demonstrates how to use OpenSSL to create a public key from the private key you just created.

```
$ openssl rsa -in crypto_user1.key -outform PEM -pubout -out crypto_user1.pub
writing RSA key
```

Create a registration token

You create a token and sign it with the private key you just generated in the previous step.

Create a registration token

1. Use the following command to start the CloudHSM CLI:

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Create a registration token by running the quorum token-sign generate command:

   ```
   aws-cloudhsm > quorum token-sign generate --service registration --token /path/
   tokenfile
   {
     "error_code": 0,
     "data": {
       "path": "/path/tokenfile"
     }
   ```

```
}
```

3. The quorum token-sign generate command generates a registration token at the specified file path. Inspect the token file:

```
$ cat /path/tokenfile
{
  "version": "2.0",
  "tokens": [
    {
      "approval_data": <approval data in base64 encoding>,
      "unsigned": <unsigned token in base64 encoding>,
      "signed": ""
    }
  ]
}
```

The token file consists of the following:

- **approval_data**: A base64 encoded randomized data token whose raw data doesn't exceed the maximum of 245 bytes.
- **unsigned**: A base64 encoded and SHA256 hashed token of the approval_data.
- **signed**: A base64 encoded signed token (signature) of the unsigned token, using the RSA 2048-bit private key previously generated with OpenSSL.

You sign the unsigned token with the private key to demonstrate that you have access to the private key. You will need the registration token file fully populated with a signature and the public key to register the crypto-user as a quorum user with the AWS CloudHSM cluster.

Sign the unsigned registration token

1. Decode the base64 encoded unsigned token and place it into a binary file:

```
$ echo -n '6BMUj6mUjjko6ZLCEdzGlWpR5sILhFJfqhW1ej3Oq1g=' | base64 -d >
 crypto_user.bin
```

2. Use OpenSSL and the private key to sign the now binary unsigned registration token and create a binary signature file:

```
$ openssl pkeyutl -sign \
-inkey crypto_user1.key \
-pkeyopt digest:sha256 \
-keyform PEM \
-in crypto_user.bin \
-out crypto_user.sig.bin
```

3. Encode the binary signature into base64:

```
$ base64 -w0 crypto_user.sig.bin > crypto_user.sig.b64
```

4. Copy and paste the base64 encoded signature into the token file:

```
{
  "version": "2.0",
  "tokens": [
    {
      "approval_data": <approval data in base64 encoding>,
      "unsigned": <unsigned token in base64 encoding>,
      "signed": <signed token in base64 encoding>
    }
  ]
}
```

Register the public key with the HSM

After creating a key, the crypto-user must register the public key with the AWS CloudHSM cluster.

1. Start CloudHSM CLI:

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Sign in as the crypto-user whose public key you want to register.

```
aws-cloudhsm > login --username crypto_user1 --role crypto-user
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "crypto_user1",
    "role": "crypto-user"
  }
}
```

3. Register the public key with the [Register a user's token-sign quorum strategy using CloudHSM CLI](). For more information, see the following example or use the **help user change-quorum token-sign register** command.

   Example – Register a public key with AWS CloudHSM cluster

   The following example shows how to use the **user change-quorum token-sign register** command in CloudHSM CLI to register a crypto-user public key with the HSM. To use this command, the crypto-user must be logged in to the HSM. Replace these values with your own:

```
aws-cloudhsm > user change-quorum token-sign register --public-key </path/crypto_user.pub> --signed-token </path/tokenfile>
{
  "error_code": 0,
  "data": {
    "username": "crypto_user1",
    "role": "crypto-user"
  }
}
```

> **Note**
>
> **/path/crypto_user.pub**: The filepath to the public key PEM file
> **Required**: Yes
> **/path/token_file**: The filepath with token signed by user private key
> **Required**: Yes

4. After all crypto-users register their public keys, the output from the **user list** command shows this in the quorum field, stating the enabled quorum strategy in use.

In this example, the AWS CloudHSM cluster has two HSMs, each with the same crypto-users, as shown in the following output from the **user list** command. For more information about creating users, see User management with CloudHSM CLI.

```
aws-cloudhsm > user list
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      },
      {
        "username": "crypto_user1",
        "role": "crypto-user",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "crypto_user2",
        "role": "crypto-user",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
```

```
      },
      {
        "username": "crypto_user3",
        "role": "crypto-user",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

Step 2. Set the key quorum values during key generation

To use quorum authentication, a crypto-user must log in to the HSM and then set the associated *key quorum values*. This is the minimum number of crypto-user approvals that are required to perform HSM key management/usage operations. For more information about the associated key commands associated with either key management or key usage, see Supported services and types.

Generate a key pair with key quorum values set

1. Use the following command to start CloudHSM CLI:

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
```

2. Using CloudHSM CLI, log in as a crypto-user.

```
aws-cloudhsm > login --username crypto_user1 --role crypto-user
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "crypto_user1",
    "role": "crypto-user"
  }
}
```

This example generates an RSA key pair which has key quorum values of two (2) set for both key-management and key-usage operations. You can choose any value from zero (0) to eight (8), up to the total number of crypto-users on the HSM. In this example, the HSM has three (3) crypto-users, so the maximum possible value is three (3). Note that in this example we are sharing the key with *<crypto_user2>* during key generation. Also note that public keys do not have quorum values.

```
aws-cloudhsm > key generate-asymmetric-pair rsa \
--public-exponent 65537 \
--modulus-size-bits 2048 \
--public-label rsa-public-key-example \
--private-label rsa-private-key-example \
--public-attributes verify=true \
--private-attributes sign=true
--share-crypto-users crypto_user2 \
--manage-private-key-quorum-value 2 \
--use-private-key-quorum-value 2
{
  "error_code": 0,
  "data": {
    "public_key": {
      "key-reference": "0x0000000000640006",
      "key-info": {
        "key-owners": [
          {
```

```
            "username": "crypto_user",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa-public-key-example",
        "id": "0x",
        "check-value": "0x218f50",
        "class": "public-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 512,
        "public-exponent": "0x010001",
        "modulus":
  "0xbdf471a3d2a869492f51c767bece8780730ae6479a9a75efffe7cea3594fb28ca518630e7b1d988b45d2fedc830
        "modulus-size-bits": 2048
      }
    },
    "private_key": {
      "key-reference": "0x0000000000640007",
```

```
      "key-info": {
        "key-owners": [
          {
            "username": "crypto_user",
            "key-coverage": "full"
          }
        ],
        "shared-users": [
          {
            "username": "crypto_user2",
            "key-coverage": "full"
          }
        ],
        "key-quorum-values": {
          "manage-key-quorum-value": 2,
          "use-key-quorum-value": 2
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa-private-key-example",
        "id": "0x",
        "check-value": "0x218f50",
        "class": "private-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 1216,
```

```
        "public-exponent": "0x010001",
        "modulus":
 "0xbdf471a3d2a869492f51c767bece8780730ae6479a9a75efffe7cea3594fb28ca518630e7b1d988b45d2fedc830
        "modulus-size-bits": 2048
      }
    }
  }
}
```

When generating a key with quorum controls, the key must be associated with a minimum number of users equal to the largest key quorum value. Associated users include the key owner and Crypto Users with whom the key is shared with. To determine the number of minimum users to share the key with, get the largest quorum value between the key usage quorum value and the key management quorum value and subtract 1 to account for the key owner, who is by default associated with the key. To share the key with more users, use the **Share a key using CloudHSM CLI** command.

Failure to share the key with enough users at key generation will result in failure, as shown below.

```
aws-cloudhsm > key generate-asymmetric-pair rsa \
--public-exponent 65537 \
--modulus-size-bits 2048 \
--public-label rsa-public-key-example \
--private-label rsa-private-key-example \
--public-attributes verify=true \
--private-attributes sign=true
--share-crypto-users crypto_user2 crypto_user3 \
--manage-private-key-quorum-value 3 \
--use-private-key-quorum-value 4
{
  "error_code": 1,
  "data": "Invalid quorum value provided."
}
```

Key management and usage with quorum authentication enabled for AWS CloudHSM using CloudHSM CLI

After you configure quorum authentication for your AWS CloudHSM cluster, crypto users can't perform HSM key management or usage operations on their own if their key has associated quorum values. This topic explains how a crypto-user can get a temporary token to perform an HSM key management or key usage operation.

> **Note**
>
> Each quorum token is valid for one operation. When that operation succeeds, the token is no longer valid and the crypto-user must obtain a new token. A quorum token is only valid during your current login session. If you log out of the CloudHSM CLI or if the network disconnects, the token is no longer valid and you need to get a new token. You can only use a CloudHSM token within the CloudHSM CLI. You can't use it to authenticate in a different application.

The following example shows the output when a crypto-user tries to create a signature with a quorum-associated key on the HSM after quorum authentication is configured. The command fails with a `Quorum Failed` error, which means quorum authentication failed:

```
aws-cloudhsm > crypto sign rsa-pkcs --key-filter attr.label=rsa-private-key-example --
hash-function sha256 --data YWJjMTIz
{
  "error_code": 1,
  "data": "Quorum Failed"
}
```

A crypto-user must complete the following tasks to get a temporary token for performing a key management or key usage operation on the HSM:

Steps

- Step 1. Get a quorum token
- Step 2. Get signatures from approving crypto-users
- Step 3. Approve the token on the CloudHSM; cluster and execute an operation

Step 1. Get a quorum token

1. Start CloudHSM CLI.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
```

2. Log in to the cluster as a crypto-user.

```
aws-cloudhsm > login --username <crypto_user1> --role crypto-user --
password password123
```

This example signs `crypto_user1` into the CloudHSM CLI with the `crypto-user` role. Replace these values with your own.

```
{
  "error_code": 0,
  "data": {
    "username": "crypto_user1",
    "role": "crypto-user"
  }
}
```

3. Generate a quorum token using the **quorum token-sign generate** command.

In the following command, `key-usage` identifies the *service name* where you will use token that you are generating. In this case, the token is for key-usage operations (`key-usage` service) This example uses the `--filter` flag to associate the token with a specific key.

```
aws-cloudhsm > quorum token-sign generate --service key-usage --token </path/
crypto_user1.token> --filter attr.label=rsa-private-key-example
{
  "error_code": 0,
  "data": {
    "path": "/home/crypto_user1.token"
  }
}
```

This example gets a quorum token for the crypto-user with username `crypto_user1` and saves the token to a file named `crypto_user1.token`. To use the example command, replace these values with your own:

The **quorum token-sign generate** command generates a key-usage service quorum token at the specified file path. You can inspect the token file:

```
$ cat </path/crypto_user1.token>
{
  "version": "2.0",
  "service": "key-usage",
  "key_reference": "0x0000000000680006",
  "approval_data":
 "AAIABQAAABkAAAAAAGgABi5CDa9x9VyyRIaFbkSrHgJjcnlwdG9fdXNlcgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
+GJj8gXo9lKuANGNyeXB0b191c2VyAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGc9AEsAAAAAAAAAAA==",
  "token": "5GlgoWOlQU4fw4QIlbxkPGZVOVoDugFGuSKE/k67ncM=",
  "signatures": []
}
```

The token file consists of the following:

- **service**: An identifier for the quorum service the token is associated with.
- **key_reference**: An identifier for the key that this quorum token is associated with.
- **approval_data**: A base64 encoded raw data token generated by the HSM.
- **token**: A base64 encoded and SHA-256 hashed token of the approval_data
- **signatures**: An array of base64 encoded signed tokens (signatures) of the unsigned token. Each approver signature is in the form of a JSON object literal:

  ```
  {
        "username": "<APPROVER_USERNAME>",
        "role": "<APPROVER_ROLE>",
        "signature": "<APPROVER_RSA2048_BIT_SIGNATURE>"
  }
  ```

  Each signature is created from the result of an approver using their corresponding RSA 2048-bit private key whose public key was registered with the HSM.

4. Validate the new user service quorum token. The **quorum token-sign list** command confirms that the token exists on CloudHSM.

```
aws-cloudhsm > quorum token-sign list
{
  "error_code": 0,
```

```
  "data": {
    "tokens": [
      {
        "username": "crypto_user",
        "service": "key-usage",
        "key-reference": "0x0000000000680006",
        "minimum-token-count": 2
      }
    ]
  }
}
```


The `minimum-token-count` presents an aggregated cluster view of the minimum usable number of key tokens corresponding to the username, service, and key-reference that are retrieved from a single HSM in the cluster.

For example, assuming a 2-HSM cluster, if we receive two (2) key-usage tokens generated by user `crypto_user1` for key with reference `0x0000000000680006` from the first HSM in the cluster and we receive one (1) key-usage tokens generated by user `crypto_user1` for key with reference `0x0000000000680006` from the other HSM in the cluster, we will display `"minimum-token-count": 1`.

Step 2. Get signatures from approving crypto-users

An crypto user who has a quorum token must get the token approved by other crypto-users. To give their approval, the other crypto =-users use their signing key to cryptographically sign the token outside the HSM.

There are many different ways to sign the token. The following example shows how to sign the token using OpenSSL. To use a different signing tool, make sure that the tool uses the private key (signing key) of the crypto-user to sign a SHA-256 digest of the token.

In this example, the crypto-user that has the token (`crypto-user`) needs at least two (2) approvals. The following example commands show how two (2) crypto-users can use OpenSSL to cryptographically sign the token.

1. Decode the base64 encoded unsigned token and place it into a binary file:

```
$echo -n '5GlgoWOlQU4fw4QIlbxkPGZVOVoDugFGuSKE/k67ncM=' | base64 -d >
 crypto_user1.bin
```

2. Use OpenSSL and the approver's private key to sign the binary quorum unsigned token for the user service and create a binary signature file:

```
$openssl pkeyutl -sign \
-inkey crypto_user1.key \
-pkeyopt digest:sha256 \
-keyform PEM \
-in crypto_user1.bin \
-out crypto_user1.sig.bin
```

3. Encode the binary signature into base64:

```
$ base64 -w0 crypto_user1.sig.bin > crypto_user1.sig.b64
```

4. Copy and paste the base64 encoded signature into the token file, using the JSON object literal format specified earlier for approver signature:


```
{
  "version": "2.0",
  "service": "key-usage",
  "key_reference": "0x0000000000680006",
  "approval_data":
 "AAIABQAAABkAAAAAAGgABi5CDa9x9VyyRIaFbkSrHgJjcnlwdG9fdXNlcgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
+GJj8gXo9lKuANGNyeXB0b191c2VyAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAGc9AEsAAAAAAAAAAAA==",
  "token": "5GlgoWOlQU4fw4QIlbxkPGZVOVoDugFGuSKE/k67ncM=",
  "signatures": [
    {
      "username": "crypto_user1",
      "role": "crypto-user",
      "signature": "wa7aPzmGwBjcEoZ6jAzYASp841AfgOvcI27Y/
tGlCj1E9DawnFw5Uf0IJT2Ca7T5XD2ThVkUi0B+dhAomdqYNl6aUUFrJyH9GBJ
+E0PmA5jNVm25tzeRWBJzneTg4/
zTeE2reNqrHFHicWnttQLe9jS09J1znuDGWDe0HaBKWUaz2gUInJRqmeXDsZYdSvZksrqUH5dci/
RsaDE2+tGiS9g0RcIkFbsPW4HpGe2e5HVzGsqrV8O3PKlYQv6+fymfcNTTuoxKcHAkOjpl43QSuSIu2gVq7KI8mSmmW
+oiukaNfLJr+MoDKzAvCGDg4cDArg=="
    },
    {
      "username": "crypto_user2",
      "role": "crypto-user",
      "signature": "wa7aPzmGwBjcEoZ6jAzYASp841AfgOvcI27Y/
tGlCj1E9DawnFw5Uf0IJT2Ca7T5XD2ThVkUi0B+dhAomdqYNl6aUUFrJyH9GBJ
+E0PmA5jNVm25tzeRWBJzneTg4/
zTeE2reNqrHFHicWnttQLe9jS09J1znuDGWDe0HaBKWUaz2gUInJRqmeXDsZYdSvZksrqUH5dci/
```

```
RsaDE2+tGiS9g0RcIkFbsPW4HpGe2e5HVzGsqrV8O3PKlYQv6+fymfcNTTuoxKcHAkOjpl43QSuSIu2gVq7KI8mSmmW
+oiukaNfLJr+MoDKzAvCGDg4cDArg=="
    }
  ]
}
```

Step 3. Approve the token on the CloudHSM; cluster and execute an operation

After a crypto-user has the necessary approvals and signatures, they can supply that token to the CloudHSM cluster along with a key management or key usage operation.

Make sure the key operation corresponds to the appropriate quorum service associated with the quorum token. For more information, see Supported services and types for more information.

During the transaction, the token will be approved within the AWS CloudHSM cluster and execute the requested key operation. The success of the key operation is contingent upon both a valid approved quorum token and a valid key operation.

Example Generate a signature with the RSA-PKCS mechanism

In the following example, a logged in crypto-user creates a signature with a key on the HSM:

```
aws-cloudhsm > crypto sign rsa-pkcs --key-filter attr.label=rsa-private-key-example --
hash-function sha256 --data YWJjMTIz --approval /path/crypto_user1.token

{
  "error_code": 0,
  "data": {
    "key-reference": "0x0000000000640007",
    "signature":
 "h6hMqXacBrT3x3MXV13RXHdQno0+IQ6iy0kVrGzo23+eoWT0ZZgrSpBCu5KcuP6IYYHw9goQ5CfPf4jI1nO5m/
IUJtF1A1lmcz0HjEy1CJ7ICXNReDRyeOU8m43dkJzt0OUdkbtkDJGAcxkbKHLZ02uWsGXaQ8bOKhoGwsRAHHF6nldTXquIC
+pZmUS38ythybney94Wj6fzYOER8v7VIY5ijQGa3LfxrjSG4aw6QijEEbno5LSf18ahEaVKmVEnDBL54tylCJBGvGsYSY9H
TDd2wfvP4PaxbFRyyHaw=="
  }
}
```

If the crypto user tries to perform another HSM key usage operation with the same token, it fails:

```
aws-cloudhsm > crypto sign rsa-pkcs --key-filter attr.label=rsa-private-key-example --
hash-function sha256 --data YWJjMTIz --approval /home/crypto_user1.token
```

```
{
  "error_code": 1,
  "data": "Quorum approval is required for this operation"
}
```

To perform another HSM key operation, the crypto user must generate a new quorum token, get new signatures from approvers, and execute the desired key operation with the --approval argument to supply the quorum token.

Use the **quorum token-sign list** to check for available token. This example shows that the crypto-user has no approved tokens.

```
aws-cloudhsm > quorum token-sign list
{
  "error_code": 0,
  "data": {
    "tokens": []
  }
}
```

Key management with the AWS CloudHSM KMU

If using the latest SDK version series, use CloudHSM CLI to manage the keys in your AWS CloudHSM cluster.

If using the previous SDK version series, you can manage keys on the hardware security modules (HSM) in your AWS CloudHSM cluster using the key_mgmt_util (KMU) command line tool. Before you can manage keys, you must start the AWS CloudHSM client, start key_mgmt_util, and log in to the HSMs. For more information, see Getting Started with key_mgmt_util.

- Using trusted keys describes how to use PKCS #11 library attributes and CMU to create trusted keys to secure data.
- Generating keys has instructions on generating keys, including symmetric keys, RSA keys, and EC keys.
- Importing keys provides details on how key owners import keys.
- Exporting keys provides details on how key owners export keys.
- Deleting keys provides details on how key owners delete keys.

- Sharing and unsharing keys details how key owners share and unshare keys.

Generate keys with the AWS CloudHSM KMU

To generate keys on the hardware security module (HSM), use the command in AWS CloudHSM key_mgmt_util (KMU) that corresponds to the type of key that you want to generate.

Topics

Generate symmetric keys with the AWS CloudHSM KMU

Use the genSymKey command in AWS CloudHSM key_mgmt_util (KMU) to generate AES and other types of symmetric keys for AWS CloudHSM. To see all available options, use the **genSymKey -h** command.

The following example creates a 256-bit AES key.

```
Command: genSymKey -t 31 -s 32 -l aes256
Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

Symmetric Key Created.  Key Handle: 524295

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Generate RSA key pairs with the AWS CloudHSM KMU

To generate an RSA key pair for AWS CloudHSM, use the genRSAKeyPair command in AWS CloudHSM key_mgmt_util. To see all available options, use the **genRSAKeyPair -h** command.

The following example generates an RSA 2048-bit key pair.

```
Command: genRSAKeyPair -m 2048 -e 65537 -l rsa2048
```

```
Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

Cfm3GenerateKeyPair:    public key handle: 524294    private key handle: 524296

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Generate ECC (elliptic curve cryptography) key pairs using the AWS CloudHSM KMU

To generate an ECC key pair for AWS CloudHSM, use the genECCKeyPair command in AWS CloudHSM key_mgmt_util. To see all available options, including a list of the supported elliptic curves, use the **genECCKeyPair -h** command.

The following example generates an ECC key pair using the P-384 elliptic curve defined in NIST FIPS publication 186-4.

```
Command: genECCKeyPair -i 14 -l ecc-p384
Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

Cfm3GenerateKeyPair:    public key handle: 524297    private key handle: 524298

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Import keys with the AWS CloudHSM KMU

To import secret keys—that is, symmetric keys and asymmetric private keys—into the hardware security module (HSM) using the AWS CloudHSM key_mgmt_util, you must first create a wrapping key on the HSM. You can import public keys directly without a wrapping key.

Topics

- Import secret keys with the AWS CloudHSM KMU
- Import public keys with the AWS CloudHSM KMU

Import secret keys with the AWS CloudHSM KMU

Complete the following steps to import a secret key into AWS CloudHSM using the key_mgmt_util (KMU). Before you import a secret key, save it to a file. Save symmetric keys as raw bytes, and asymmetric private keys in PEM format.

This example shows how to import a plaintext secret key from a file into the HSM. To import an encrypted key from a file into the HSM, use the unWrapKey command.

To import a secret key

1. Use the genSymKey command to create a wrapping key. The following command creates a 128-bit AES wrapping key that is valid only for the current session. You can use a session key or a persistent key as a wrapping key.

```
Command: genSymKey -t 31 -s 16 -sess -l import-wrapping-key
Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

Symmetric Key Created.  Key Handle: 524299

Cluster Error Status
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

2. Use one of the following commands, depending on the type of secret key that you are importing.

   - To import a symmetric key, use the imSymKey command. The following command imports an AES key from a file named `aes256.key` using the wrapping key created in the previous step. To see all available options, use the **imSymKey -h** command.

```
Command: imSymKey -f aes256.key -t 31 -l aes256-imported -w 524299
Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

Symmetric Key Unwrapped.  Key Handle: 524300

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

```
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

- To import an asymmetric private key, use the importPrivateKey command. The following command imports a private key from a file named `rsa2048.key` using the wrapping key created in the previous step. To see all available options, use the **importPrivateKey -h** command.

```
Command: importPrivateKey -f rsa2048.key -l rsa2048-imported -w 524299
BER encoded key length is 1216

Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

Private Key Unwrapped.  Key Handle: 524301

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Import public keys with the AWS CloudHSM KMU

Use the importPubKey command in the AWS CloudHSM key_mgmt_util (KMU) to import a public key. To see all available options, use the **importPubKey -h** command.

The following example imports an RSA public key from a file named `rsa2048.pub`.

```
Command: importPubKey -f rsa2048.pub -l rsa2048-public-imported
Cfm3CreatePublicKey returned: 0x00 : HSM Return: SUCCESS

Public Key Handle: 524302

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Export keys with the AWS CloudHSM KMU

To export AWS CloudHSM secret keys—that is, symmetric keys and asymmetric private keys—from the hardware security module (HSM) using the AWS CloudHSM key_mgmt_util (KMU), you must first create a wrapping key. You can export public keys directly without a wrapping key.

Only the key owner can export a key. Users with whom the key is shared can use the key in cryptographic operations, but they cannot export it. When running this example, be sure to export a key that you created.

> **Important**
>
> The exSymKey command writes a plaintext (unencrypted) copy of the secret key to a file. The export process requires a wrapping key, but the key in the file is ***not*** a wrapped key. To export a wrapped (encrypted) copy of a key, use the wrapKey command.

Topics

Export secret keys with the AWS CloudHSM KMU

Complete the following steps to export a secret key from AWS CloudHSM using the key_mgmt_util (KMU).

To export a secret key

1. Use the genSymKey command to create a wrapping key. The following command creates a 128-bit AES wrapping key that is valid only for the current session.

```
Command: genSymKey -t 31 -s 16 -sess -l export-wrapping-key
Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

Symmetric Key Created.  Key Handle: 524304

Cluster Error Status
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

2. Use one of the following commands, depending on the type of secret key that you are exporting.

   - To export a symmetric key, use the exSymKey command. The following command exports an AES key to a file named `aes256.key.exp`. To see all available options, use the **exSymKey -h** command.

     ```
     Command: exSymKey -k 524295 -out aes256.key.exp -w 524304
     Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

     Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS


     Wrapped Symmetric Key written to file "aes256.key.exp"
     ```

     Note

     The command's output says that a "Wrapped Symmetric Key" is written to the output file. However, the output file contains a plaintext (not wrapped) key. To export a wrapped (encrypted) key to a file, use the wrapKey command.

   - To export a private key, use the **exportPrivateKey** command. The following command exports a private key to a file named `rsa2048.key.exp`. To see all available options, use the **exportPrivateKey -h** command.

     ```
     Command: exportPrivateKey -k 524296 -out rsa2048.key.exp -w 524304
     Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

     Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS

     PEM formatted private key is written to rsa2048.key.exp
     ```

Export public keys with the AWS CloudHSM KMU

Use the **exportPubKey** command in the AWS CloudHSM key_mgmt_util (KMU) to export a public key. To see all available options, use the **exportPubKey -h** command.

The following example exports an RSA public key to a file named `rsa2048.pub.exp`.

```
Command: exportPubKey -k 524294 -out rsa2048.pub.exp
PEM formatted public key is written to rsa2048.pub.key

Cfm3ExportPubKey returned: 0x00 : HSM Return: SUCCESS
```

Delete keys with KMU and CMU

Use the deleteKey command to delete a key, as in the following example. Only the key owner can delete a key.

```
Command: deleteKey -k 524300
Cfm3DeleteKey returned: 0x00 : HSM Return: SUCCESS

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Share and unshare keys with KMU and CMU

In AWS CloudHSM, the CU who creates the key owns it. The owner manages the key, can export and delete it, and can use the key in cryptographic operations. The owner can also share the key with other CU users. Users with whom the key is shared can use the key in cryptographic operations, but they cannot export or delete the key, or share it with other users.

You can share keys with other CU users when you create the key, such as by using the `-u` parameter of the genSymKey or genRSAKeyPair commands. To share existing keys with a different HSM user, use the cloudhsm_mgmt_util command line tool. This is different from most of the tasks documented in this section, which use the key_mgmt_util command line tool.

Before you can share a key, you must start cloudhsm_mgmt_util, enable end-to-end encryption, and log in to the HSMs. To share a key, log in to the HSM as the crypto user (CU) that owns the key. Only key owners can share a key.

Use the **shareKey** command to share or unshare a key, specifying the handle of the key and the IDs of the user or users. To share or unshare with more than one user, specify a comma-separated list of user IDs. To share a key, use 1 as the command's last parameter, as in the following example. To unshare, use `0`.

```
aws-cloudhsm > shareKey 524295 4 1
```

```
************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)? y
shareKey success on server 0(10.0.2.9)
shareKey success on server 1(10.0.3.11)
shareKey success on server 2(10.0.1.12)
```

The following shows the syntax for the **shareKey** command.

```
aws-cloudhsm > shareKey <key handle> <user ID> <Boolean: 1 for share, 0 for unshare>
```

How to mark a key as trusted with the AWS CloudHSM Management Utility

The content in this section provides instructions on using the AWS CloudHSM management Utility (CMU) to mark a key as trusted.

1. Using the loginHSM command, log in as a crypto officer (CO).
2. Use the Set the attributes of AWS CloudHSM keys using CMU command with `OBJ_ATTR_TRUSTED` (value 134) set to true (1).

   ```
   aws-cloudhsm > setAttribute <Key Handle> 134 1
   ```

Cluster backups in AWS CloudHSM

AWS CloudHSM makes periodic backups of your cluster at least once every 24 hours. Each backup contains encrypted copies of the following data:

- Users (COs, CUs, and AUs)
- Key material and certificates
- Hardware security module (HSM) configuration and policies

You can't instruct the service to make backups, but you can take certain actions that force the service to create a backup. The service makes a backup when you perform any of the following actions:

- Activate a cluster
- Add an HSM to an active cluster
- Remove an HSM from an active cluster

AWS CloudHSM deletes backups based on the backup retention policy you set when you create clusters. For information about managing backup retention policy, see Configure backup retention.

Topics

Working with AWS CloudHSM cluster backups

When you add a hardware security module (HSM) to a cluster in AWS CloudHSM that previously contained one or more active HSMs, the service restores the latest backup onto the new HSM. Use backups to manage HSMs you use infrequently. When you don't need the HSM, delete it to trigger a backup. Later, when you need the HSM, create a new one in the same cluster, and this action will restore the backup you previously created with the delete HSM operation.

Removing expired keys or inactive users

You may want to remove unwanted cryptographic materials from your environment such as expired keys or inactive users. This is a two-step process. First, delete these materials from your HSM. Next, delete all existing backups. Following this process ensures you do not restore deleted information when initializing a new cluster from backup. For more information, see the section called “Delete backups”.

Considering disaster recovery

You can create a cluster from a backup. You might want to do this to set a recovery point for your cluster. Nominate a backup that contains all the users, key material, certificates that you want in your recovery point, and then use that backup to create a new cluster. For more information about creating a cluster from a backup, see Creating clusters from backups.

You can also copy a backup of a cluster into a different region, where you can create a new cluster as a clone of the original. You may want to do this for a number of reasons, including simplification of the disaster recovery process. For more information about copying backups to regions, see Copying backups across Regions.

Delete AWS CloudHSM cluster backups

After you delete an AWS CloudHSM cluster backup, the service holds the backup for seven days, during which time you can restore the backup. After the seven-day period, you can no longer restore the backup. For more information about managing backups, see Cluster backups.

The following table describes how to delete a backup.

Console

To delete a backup (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Backups**.
4. Choose a backup to delete.

5. To delete the selected backup, choose **Actions, Delete**.

   The Delete backups dialog box appears.

6. Choose **Delete**.

   The state of the backup changes to PENDING_DELETE. You can restore a backup that is pending deletion for up to 7 days after you request the deletion.

To list backups (AWS CLI)

- To see a list of all backups in the PENDING_DELETION state, run the **describe-backups** command and include states=PENDING_DELETION as a filter.

```
$ aws cloudhsmv2 describe-backups --filters states=PENDING_DELETION
{
    "Backups": [
        {
            "BackupId": "backup-ro5c4er4aac",
            "BackupState": "PENDING_DELETION",
            "ClusterId": "cluster-dygnwhmscg5",
            "CreateTimestamp": 1534461854.64,
            "DeleteTimestamp": 1536339805.522,
            "HsmType": "hsm2m.medium",
            "Mode": "NON_FIPS",
            "NeverExpires": false,
            "TagList": []
        }
}
```

AWS CLI

Check the status of a backup or find its ID by using the [describe-backups](#) command from the AWS CLI.

To delete a backup (AWS CLI)

- At a command prompt, run the [delete-backup](#) command, passing the ID of the backup to be deleted.

```
$ aws cloudhsmv2 delete-backup --backup-id <backup ID>
```

```
{
    "Backup": {
        "CreateTimestamp": 1534461854.64,
        "ClusterId": "cluster-dygnwhmscg5",
        "BackupId": "backup-ro5c4er4aac",
        "BackupState": "PENDING_DELETION",
        "DeleteTimestamp": 1536339805.522,
        "HsmType": "hsm1.medium",
        "Mode": "FIPS"
    }
}
```

AWS CloudHSM API

Refer to DeleteBackup to learn how to delete backups by using the API.

Restore AWS CloudHSM backups

AWS CloudHSM holds deleted backups for seven days, during which time you can restore the backup. After the seven-day period, you can no longer restore the backup. For more information about managing backups, see Cluster backups.

The following table describes how to delete a backup.

Console

To restore a backup (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Backups**.
4. Choose a backup in the `PENDING_DELETE` state to restore.
5. To restore the selected backup, choose **Actions, Restore**.

AWS CLI

To restore a backup (AWS CLI)

- To restore a backup, issue the **restore-backup** command, passing the ID of a backup that is in the PENDING_DELETION state.

```
$ aws cloudhsmv2 restore-backup --backup-id <backup ID>
{
    "Backup": {
        "ClusterId": "cluster-dygnwhmscg5",
        "CreateTimestamp": 1534461854.64,
        "BackupState": "READY",
        "BackupId": "backup-ro5c4er4aac"
    }
}
```

AWS CloudHSM API

Refer to RestoreBackup to learn how to restore backups by using the API.

Configure AWS CloudHSM backup retention policy

AWS CloudHSM purges backups based on the backup retention policy you set when you create a cluster. Backup retention policy applies to clusters. If you move a backup to a different region, that backup is no longer associated with a cluster and has no backup retention policy. You must manually delete any backups not associated with a cluster. AWS CloudHSM does not delete a cluster's last backup.

AWS CloudTrail reports backups marked for deletion. You can restore backups the service purges just as you would restore manually deleted backups. To prevent a race condition, you should change the backup retention policy for the cluster before you restore a backup deleted by the service. If you want to keep the retention policy the same and preserve select backups, you can specify that the service exclude backups from the cluster backup retention policy.

Managed backup retention

Clusters created before 18 November 2020 have a backup retention policy of 90 days plus the age of the cluster. For example, if you created a cluster on 18 November 2019, the service would assign your cluster a backup retention policy of one year plus 90 days (455 days). You can set this period to any number between 7 and 379 days. AWS CloudHSM does not delete a cluster's last backup. For more information about managing backups, see Cluster backups.

> **Note**
>
> You can opt out of managed backup retention altogether by contacting support (https://aws.amazon.com/support).

The following table describes how to set the backup retention.

Console

To configure backup retention policy (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. Click the cluster ID of a cluster in the Active state to manage the backup retention policy for that cluster.
4. To change the backup retention policy, choose **Actions, Change backup retention period**.

   The Change backup retention period dialog box appears.
5. In **Backup retention period (in days)**, type a value between 7 and 379 days.
6. Choose **Change backup retention period**.

To exclude or include a backup from backup retention policy (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. To view your backups, in the navigation pane choose **Backups**.
3. Click the backup ID of a backup in the Ready state to exclude or include.
4. On the **Backup details** page, take one of the following actions.
   - To exclude a backup with a date in **Expiration time**, choose **Actions, Disable expiration**.
   - To include a backup that does not expire, choose **Actions, Use cluster retention policy**.

AWS CLI

To configure backup retention policy (AWS CLI)

- At a command prompt, issue the **modify-cluster** command. Specify the cluster ID and the backup retention policy.

```
$ aws cloudhsmv2 modify-cluster --cluster-id <cluster ID> \
                                --backup-retention-policy
 Type=DAYS,Value=<number of days to retain backups>
{
   "Cluster": {
      "BackupPolicy": "DEFAULT",
      "BackupRetentionPolicy": {
         "Type": "DAYS",
         "Value": 90
      },
      "Certificates": {},
      "ClusterId": "cluster-kdmrayrc7gi",
      "CreateTimestamp": 1504903546.035,
      "Hsms": [],
      "HsmType": "hsm1.medium",
      "SecurityGroup": "sg-40399d28",
      "State": "ACTIVE",
      "SubnetMapping": {
         "us-east-2a": "subnet-f1d6e798",
         "us-east-2c": "subnet-0e358c43",
         "us-east-2b": "subnet-40ed9d3b"
      },
      "TagList": [
         {
            "Key": "Cost Center",
            "Value": "12345"
         }
      ],
      "VpcId": "vpc-641d3c0d"
   }
}
```

To exclude a backup from backup retention policy (AWS CLI)

- At a command prompt, issue the **modify-backup-attributes** command. Specify the backup ID and set the never-expires flag to preserve the backup.

```
$ aws cloudhsmv2 modify-backup-attributes --backup-id <backup ID> \
                                            --never-expires
{
   "Backup": {
      "BackupId": "backup-ro5c4er4aac",
      "BackupState": "READY",
      "ClusterId": "cluster-dygnwhmscg5",
      "NeverExpires": true
   }
}
```

To include a backup in backup retention policy (AWS CLI)

- At a command prompt, issue the **modify-backup-attributes** command. Specify the backup ID and set the no-never-expires flag to include the backup in backup retention policy, which means the service will eventually delete the backup.

```
$ aws cloudhsmv2 modify-backup-attributes --backup-id <backup ID> \
                                            --no-never-expires
{
   "Backup": {
      "BackupId": "backup-ro5c4er4aac",
      "BackupState": "READY",
      "ClusterId": "cluster-dygnwhmscg5",
      "NeverExpires": false
   }
}
```

AWS CloudHSM API

Refer to the following topics to learn how to manage backup retention by using the API.

- [ModifyCluster]
- [ModifyBackupAttributes]

Copying AWS CloudHSM cluster backups across AWS Regions

You can copy AWS CloudHSM cluster backups across Regions for many reasons, including cross-region resilience, global workloads, and disaster recovery. After you copy backups, they appear in the destination region with a `CREATE_IN_PROGRESS` status. Upon successful completion of the copy, the status of the backup changes to `READY`. If the copy fails, the status of the backup changes to `DELETED`. Check your input parameters for errors and ensure that the specified source backup is not in a `DELETED` state before rerunning the operation. For information about backups or how to create a cluster from a backup, see Cluster backups or Creating clusters from backups.

Note the following:

- To copy a cluster backup to a destination region, your account must have the proper IAM policy permissions. In order to copy the backup to a different region, your IAM policy must allow access to the source region in which the backup is located. Once copied across regions, your IAM policy must allow access to the destination region in order to interact with the copied backup, which includes using the CreateCluster operation. For more information, see Create IAM administrators.
- The original cluster and the cluster that may be built from a backup in the destination region are not linked. You must manage each of these clusters independently. For more information, see Clusters.
- Backups cannot be copied between AWS restricted regions and standard regions. Backups *can* be copied between the AWS GovCloud (US-East) and AWS GovCloud (US-West) regions.

Copy backups to different Regions (console)

To copy backups to different Regions (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Backups**.
4. Choose a backup to copy to a different region.
5. To copy the selected backup, choose **Actions, Copy backup to another region**.

   The Copy backup to another region dialog box appears.
6. In **Destination region**, choose a region from **Select a region**.

7. (Optional) Type a tag key and an optional tag value. To add more than one tag to the cluster, choose **Add tag**.
8. Choose **Copy backup**.

Copy backups to different Regions (AWS CLI)

To determine the backup ID, run the **describe-backups** command.

To copy backups to different regions (AWS CLI)

- At a command prompt, run the **copy-backup-to-region** command. Specify the destination region and the backup ID of the source backup. If you specify a backup ID, the associated backup is copied.

```
$ aws cloudhsmv2 copy-backup-to-region --destination-region <destination region> \
                                        --backup-id <backup ID>
```

Copy backups to different Regions (AWS CloudHSM API)

Refer to the following topic to learn how to copy backups to different regions by using the API.

- CopyBackupToRegion

Working with shared backups in AWS CloudHSM

CloudHSM integrates with AWS Resource Access Manager (AWS RAM) to enable resource sharing. AWS RAM is a service that enables you to share some CloudHSM resources with other AWS accounts or through AWS Organizations. With AWS RAM, you share resources that you own by creating a *resource share*. A resource share specifies the resources to share, and the consumers with whom to share them. Consumers can include:

- Specific AWS accounts inside or outside of its organization in AWS Organizations
- An organizational unit inside its organization in AWS Organizations
- An entire organization in AWS Organizations

For more information about AWS RAM, see the *AWS RAM User Guide*.

This topic explains how to share resources that you own, and how to use resources that are shared with you.

Contents

Prerequisites for sharing backups

- To share a backup, you must own it in your AWS account. This means that the resource must be allocated or provisioned in your account. You cannot share a backup that has been shared with you.
- To share a backup, it must be in the *READY* state.
- To share a backup with your organization or an organizational unit in AWS Organizations, you must enable sharing with AWS Organizations. For more information, see Enable Sharing with AWS Organizations in the *AWS RAM User Guide*.

Sharing a backup

When you share a backup with other AWS accounts, you enable them to restore clusters from the backup which contain the keys and users stored in the backup.

To share a backup, you must add it to a resource share. A resource share is an AWS RAM resource that lets you share your resources across AWS accounts. A resource share specifies the resources to share, and the consumers with whom they are shared. When you share a backup using the CloudHSM console, you add it to an existing resource share. To add the backup to a new resource share, you must first create the resource share using the AWS RAM console.

If you are part of an organization in AWS Organizations and sharing within your organization is enabled, consumers in your organization are automatically granted access to the shared backup.

Otherwise, consumers receive an invitation to join the resource share and are granted access to the shared backup after accepting the invitation.

You can share a backup that you own using the AWS RAM console or AWS CLI.

To share a backup that you own using the AWS RAM console

See Creating a Resource Share in the *AWS RAM User Guide*.

To share a backup that you own (AWS RAM command)

Use the create-resource-share command.

To share a backup that you own (CloudHSM command)

> **Important**
>
> While you can share a backup using the CloudHSM PutResourcePolicy operation, we recommend using AWS Resource Access Manager (AWS RAM) instead. Using AWS RAM provides multiple benefits as it creates the policy for you, allows multiple resources to be shared at one time, and increases the discoverability of shared resources. If you use PutResourcePolicy and want consumers to be able to describe the backups you shared with them, you must promote the backup to a standard AWS RAM Resource Share using the AWS RAM PromoteResourceShareCreatedFromPolicy API operation.

Use the put-resource-policy command.

1. Create a file named `policy.json` and copy the following policy into it.

```
{
  "Version":"2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Principal":{
      "AWS":"<consumer-aws-account-id-or-user>"
    },
    "Action":[
      "cloudhsm:CreateCluster",
      "cloudhsm:DescribeBackups"],
    "Resource":"<arn-of-backup-to-share>"
  }]
```

```
}
```

2. Update `policy.json` with the backup ARN and identifiers to share it with. The following example grants read-only access to the root user for the AWS account identified by 123456789012.

```
{
  "Version":"2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Principal":{
      "AWS": [
        "account-id"
      ]
    },
    "Action":[
      "cloudhsm:CreateCluster",
      "cloudhsm:DescribeBackups"],
    "Resource":"arn:aws:cloudhsm:us-west-2:123456789012:backup/backup-123"
  }]
}
```

> **Important**
>
> You can only grant permissions to DescribeBackups at the account level. When you share a backup with another customer, any principal that has DescribeBackups permission in that account can describe the backup.

3. Run the put-resource-policy command.

```
$ aws cloudhsmv2 put-resource-policy --resource-arn <resource-arn> --policy file://
policy.json
```

> **Note**
>
> At this point, the consumer can use the backup but it will not show up in the DescribeBackups response with the shared parameter. The next steps describe how to promote the AWS RAM resource share in order for the backup to be included in the response.

4. Get the AWS RAM resource share ARN.

```
$ aws ram list-resources --resource-owner SELF --resource-arns <backup-arn>
```

This returns a response similar to this:

```
{
  "resources": [
    {
      "arn": "<project-arn>",
      "type": "<type>",
      "resourceShareArn": "<resource-share-arn>",
      "creationTime": "<creation-time>",
      "lastUpdatedTime": "<last-update-time>"
    }
  ]
}
```

From the response, copy the `<resource-share-arn>` value to use in the next steps.

5. Run the AWS RAM promote-resource-share-created-from-policy command.

```
$ aws ram promote-resource-share-created-from-policy --resource-share-
arn <resource-share-arn>
```

6. To validate that the resource share has been promoted, you can run the AWS RAM get-resource-shares command.

```
$ aws ram get-resource-shares --resource-owner SELF --resource-share-
arns <resource-share-arn>
```

When the policy has been promoted, the `featureSet` listed in the response is `STANDARD`. This also means the backup can be described by the new accounts in the policy.

Unsharing a shared backup

When you unshare a resource, the consumer may no longer use it to restore a cluster. Consumers will still be able to access any clusters that they restored from the shared backup.

To unshare a shared backup that you own, you must remove it from the resource share. You can do this using the AWS RAM console or AWS CLI.

To unshare a shared backup that you own using the AWS RAM console

See [Updating a Resource Share] in the *AWS RAM User Guide*.

To unshare a shared backup that you own (AWS RAM command)

Use the [disassociate-resource-share] command.

To unshare a shared backup that you own (CloudHSM command)

Use the [delete-resource-policy] command.

```
$ aws cloudhsmv2 delete-resource-policy --resource-arn <resource-arn>
```

Identifying a shared backup

Consumers can identify a backup shared with them using the CloudHSM console and AWS CLI.

To identify backups shared with you using the CloudHSM console

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Backups**.
4. In the table, choose the **Shared backups** tab.

To identify backups shared with you using the AWS CLI

Use the [describe-backups] command with the `--shared` parameter to return the backups that are shared with you.

Permissions for shared backups

Permissions for owners

Backup owners can describe and manage a shared backup as well as use it to restore a cluster.

Permissions for consumers

Backup consumers cannot modify a shared backup, but they can describe it and use it to restore a cluster.

Billing and metering

There are no additional charges for sharing backups.

Cloned clusters in AWS CloudHSM

Use AWS CloudHSM Management Utility (CMU) to synchronize a cluster in a remote region, *if the cluster in that region was originally created from the backup of a cluster in another region*. Let's say you copied a cluster to another region (destination) and then later you want to synchronize changes from the original cluster (source). In scenarios like this, you use CMU to synchronize the clusters. You do this by creating a new CMU configuration file, specifying hardware security modules (HSM) from both clusters in the new file, and then using CMU to connect to the cluster with that file.

To use CMU across cloned clusters

1. Create a copy of your current configuration file and change the name of the copy to something else.

   For example, use the following file locations to locate and create a copy of your current configuration file, then change the name of the copy from `cloudhsm_mgmt_config.cfg` to `syncConfig.cfg`.

   - Linux: `/opt/cloudhsm/etc/cloudhsm_mgmt_config.cfg`
   - Windows: `C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_config.cfg`

2. In the renamed copy, add the Elastic Network Interface (ENI) IP of the destination HSM (the HSM in the foreign region that needs to be synced). We recommend that you add the destination HSM *below* the source HSM.

```
{
    ...
    "servers": [
        {
            ...
            "hostname": "<ENI Source IP>",
            ...
        },
        {
            ...
            "hostname": "<ENI Destination IP>",
            ...
        }
    ]
```

```
}
```

For more information about how to get the IP address, see the section called “Get an IP address for an HSM”.

3. Initialize CMU with the new configuration file:

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/userSync.cfg
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\cloudhsm_mgmt_util.exe" C:
   \ProgramData\Amazon\CloudHSM\data\userSync.cfg
   ```

4. Check the status messages returned to ensure that the CMU is connected to all desired HSMs and determine which of the returned ENI IPs corresponds to each cluster. Use syncUser and syncKey to manually synchronize users and keys. For more information, see syncUser and syncKey.

Get an IP address for an HSM

Use this section to obtain an IP address for an HSM.

To get an IP address for an HSM (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To open the cluster detail page, in the cluster table, choose the cluster ID.
4. To get the IP address, go to the HSMs tab. For IPv4 clusters, choose an address listed under **ENI IPv4 address**. For dual-stack clusters use either the ENI IPv4 or the **ENI IPv6 address**.

To get an IP address for an HSM (AWS CLI)

- Get the IP address of an HSM by using the [describe-clusters](#) command from the AWS CLI. In the output from the command, the IP address of the HSMs are the values of `EniIp` and `EniIpV6` (if it is a dual-stack cluster).

```
$ aws cloudhsmv2 describe-clusters
{
    "Clusters": [
        { ... }
            "Hsms": [
                {
...
                    "EniIp": "10.0.0.9",
...
                },
                {
...
                    "EniIp": "10.0.1.6",
                    "EniIpV6": "2600:113f:404:be09:310e:ed34:3412:f733",
...
```

Related topics

- syncUser
- syncKey
- Copying Backups Across Regions

Tag AWS CloudHSM resources

A tag is a label that you assign to an AWS resource. You can assign tags to your AWS CloudHSM clusters. Each tag consists of a tag key and a tag value, both of which you define. For example, the tag key might be **Cost Center** and the tag value might be **12345**. Tag keys must be unique for each cluster.

You can use tags for a variety of purposes. One common use is to categorize and track your AWS costs. You can apply tags that represent business categories (such as cost centers, application names, or owners) to organize your costs across multiple services. When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. You can use this report to view your AWS CloudHSM costs in terms of projects or applications, instead of viewing all AWS CloudHSM costs as a single line item.

For more information about using tags for cost allocation, see Using Cost Allocation Tags in the *AWS Billing User Guide*.

You can use the AWS CloudHSM console or one of the AWS SDKs or command line tools to add, update, list, and remove tags.

Topics

- Add or update tags for AWS CloudHSM resources
- List tags for AWS CloudHSM resources
- Remove tags from AWS CloudHSM resources

Add or update tags for AWS CloudHSM resources

You can add or update tags from the AWS CloudHSM console, the AWS Command Line Interface (AWS CLI), or the AWS CloudHSM API.

To add or update tags (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Choose the cluster that you are tagging.
3. Choose **Tags**.
4. To add a tag, do the following:

a. Choose **Edit Tag** and then choose **Add Tag**.

b. For **Key**, type a key for the tag.

c. (Optional) For **Value**, type a value for the tag.

d. Choose **Save**.

5. To update a tag, do the following:

a. Choose **Edit Tag**.

> **Note**
>
> If you update the tag key for an existing tag, the console deletes the existing tag and creates a new one.

b. Type the new tag value.

c. Choose **Save**.

To add or update tags (AWS CLI)

1. At a command prompt, issue the **tag-resource** command, specifying the tags and the ID of the cluster that you are tagging. If you don't know the cluster ID, issue the **describe-clusters** command.

```
$ aws cloudhsmv2 tag-resource --resource-id <cluster ID> \
                              --tag-list Key="<tag key>",Value="<tag value>"
```

2. To update tags, use the same command but specify an existing tag key. When you specify a new tag value for an existing tag, the tag is overwritten with the new value.

To add or update tags (AWS CloudHSM API)

- Send a TagResource request. Specify the tags and the ID of the cluster that you are tagging.

List tags for AWS CloudHSM resources

You can list tags for a cluster from the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.

To list tags (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Choose the cluster whose tags you are listing.
3. Choose **Tags**.

To list tags (AWS CLI)

- At a command prompt, issue the **list-tags** command, specifying the ID of the cluster whose tags you are listing. If you don't know the cluster ID, issue the **describe-clusters** command.

```
$ aws cloudhsmv2 list-tags --resource-id <cluster ID>
{
    "TagList": [
        {
            "Key": "Cost Center",
            "Value": "12345"
        }
    ]
}
```

To list tags (AWS CloudHSM API)

- Send a ListTags request, specifying the ID of the cluster whose tags you are listing.

Remove tags from AWS CloudHSM resources

You can remove tags from an AWS CloudHSM cluster by using the AWS CloudHSM console, the AWS CLI, or the AWS CloudHSM API.

To remove tags (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. Choose the cluster whose tags you are removing.
3. Choose **Tags**.
4. Choose **Edit Tag** and then choose **Remove tag** for the tag you want to remove.
5. Choose **Save**.

To remove tags (AWS CLI)

- At a command prompt, issue the **untag-resource** command, specifying the tag keys of the tags that you are removing and the ID of the cluster whose tags you are removing. When you use the AWS CLI to remove tags, specify only the tag keys, not the tag values.

```
$ aws cloudhsmv2 untag-resource --resource-id <cluster ID> \
                                --tag-key-list "<tag key>"
```

To remove tags (AWS CloudHSM API)

- Send an UntagResource request in the AWS CloudHSM API, specifying the ID of the cluster and the tags that you are removing.

AWS CloudHSM command line tools

In addition to the AWS Command Line Interface (AWS CLI) that you use for managing your AWS resources, AWS CloudHSM offers command-line tools for creating and managing hardware security module (HSM) users and keys on your HSMs. In AWS CloudHSM, you use the familiar CLI to manage your cluster, and the CloudHSM command-line tools to manage your HSM.

These are the various command-line tools:

To manage HSMs and clusters

CloudHSMv2 commands in AWS CLI and HSM2 PowerShell cmdlets in the AWSPowerShell module

- These tools get, create, delete, and tag AWS CloudHSM clusters and HSMs:
- To use the commands in CloudHSMv2 commands in CLI, you need to install and configure AWS CLI.
- HSM2 PowerShell cmdlets in the AWSPowerShell module are available in a Windows PowerShell module and a cross-platform PowerShell Core module.

To manage HSM users

CloudHSM CLI

- Use CloudHSM CLI to create users, delete users, list users, change user passwords, and update user multi-factor authentication (MFA). It is not included in the AWS CloudHSM client software. For guidance on installing this tool, see Install and configure CloudHSM CLI.

Helper Tools

Two tools help you to use AWS CloudHSM tools and software libraries:

- The configure tool updates your CloudHSM client configuration files. This allows AWS CloudHSM to synchronize the HSMs in a cluster.

  AWS CloudHSM offers two major versions, and Client SDK 5 is the latest. It offers a variety of advantages over Client SDK 3 (the previous series).
- pkpspeed measures the performance of your HSM hardware independent of software libraries.

Tools for previous SDKs

Use the key management tool (KMU) create, delete, import, and export symmetric keys and asymmetric key pairs:

- key_mgmt_util. This tool is included in the AWS CloudHSM client software.

Use the CloudHSM management tool (CMU) to create and delete HSM users, including implementing quorum authentication of user management tasks

- cloudhsm_mgmt_util. This tool is included in the AWS CloudHSM client software.

The following topics further describe the command-line tools available for managing and using AWS CloudHSM.

Topics

AWS CloudHSM configure tool

AWS CloudHSM automatically synchronizes data among all hardware security modules (HSM) in a cluster. The **configure** tool updates the HSM data in the configuration files that the synchronization mechanisms use. Use **configure** to refresh the HSM data before you use the command line tools, especially when the HSMs in the cluster have changed.

AWS CloudHSM includes two major Client SDK versions:

- Client SDK 5: This is our latest and default Client SDK. For information on the benefits and advantages it provides, see Benefits of AWS CloudHSM Client SDK 5.
- Client SDK 3: This is our older Client SDK. It includes a full set of components for platform and language-based applications compatibility and management tools.

For instructions on migrating from Client SDK 3 to Client SDK 5, see Migrating from AWS CloudHSM Client SDK 3 to Client SDK 5.

Topics

AWS CloudHSM Client SDK 5 configure tool

Use the AWS CloudHSM Client SDK 5 configure tool to update client-side configuration files.

Each component in Client SDK 5 includes a configure tool with a designator of the component in the file name of the configure tool. For example, the PKCS #11 library for Client SDK 5 includes a configure tool named `configure-pkcs11` on Linux or `configure-pkcs11.exe` on Windows.

Topics

AWS CloudHSM Client SDK 5 configuration syntax

The following table illustrates the syntax for AWS CloudHSM configuration files for Client SDK 5.

PKCS #11

```
configure-pkcs11[ .exe ]
             -a <ENI IP address>
             [--hsm-ca-cert <customerCA certificate file path>]
             [--cluster-id <cluster ID>]
             [--endpoint <endpoint>]
             [--region <region>]
             [--client-cert-hsm-tls-file <client certificate hsm tls path>]
             [--client-key-hsm-tls-file <client key hsm tls path>]
             [--log-level <error | warn | info | debug | trace>]
                  Default is <info>
             [--log-rotation <daily | weekly>]
                  Default is <daily>
             [--log-file <file name with path>]
```

```
                    Default is </opt/cloudhsm/run/cloudhsm-pkcs11.log>
                    Default for Windows is <C:\\Program Files\\Amazon\\CloudHSM\
\cloudhsm-pkcs11.log>
               [--log-type <file | term>]
                    Default is <file>
               [-h | --help]
               [-V | --version]
               [--disable-key-availability-check]
               [--enable-key-availability-check]
               [--disable-validate-key-at-init]
               [--enable-validate-key-at-init]
                    This is the default for PKCS #11
```

OpenSSL

```
configure-dyn[ .exe ]
               -a <ENI IP address>
               [--hsm-ca-cert <customerCA certificate file path>]
               [--cluster-id <cluster ID>]
               [--endpoint <endpoint>]
               [--region <region>]
               [--client-cert-hsm-tls-file <client certificate hsm tls path>]
               [--client-key-hsm-tls-file <client key hsm tls path>]
               [--log-level <error | warn | info | debug | trace>]
                    Default is <error>
               [--log-type <file | term>]
                    Default is <term>
               [-h | --help]
               [-V | --version]
               [--disable-key-availability-check]
               [--enable-key-availability-check]
               [--disable-validate-key-at-init]
                    This is the default for OpenSSL
               [--enable-validate-key-at-init]
```

KSP

```
configure-ksp[ .exe ]
               -a <ENI IP address>
               [--hsm-ca-cert <customerCA certificate file path>]
               [--cluster-id <cluster ID>]
               [--endpoint <endpoint>]
               [--region <region>]
```

```
            [--client-cert-hsm-tls-file <client certificate hsm tls path>]
            [--client-key-hsm-tls-file <client key hsm tls path>]
            [--log-level <error | warn | info | debug | trace>]
                 Default is <info>
            [--log-rotation <daily | weekly>]
                 Default is <daily>
            [--log-file <file name with path>]
                 Default is <C:\\Program Files\\Amazon\\CloudHSM\\cloudhsm-ksp.log>
            [--log-type <file | term>]
                 Default is <file>
            [-h | --help]
            [-V | --version]
            [--disable-key-availability-check]
            [--enable-key-availability-check]
            [--disable-validate-key-at-init]
                 This is the default for KSP
            [--enable-validate-key-at-init]
```

JCE

```
configure-jce[ .exe ]
            -a <ENI IP address>
            [--hsm-ca-cert <customerCA certificate file path>]
            [--cluster-id <cluster ID>]
            [--endpoint <endpoint>]
            [--region <region>]
            [--client-cert-hsm-tls-file <client certificate hsm tls path>]
            [--client-key-hsm-tls-file <client key hsm tls path>]
            [--log-level <error | warn | info | debug | trace>]
                 Default is <info>
            [--log-rotation <daily | weekly>]
                 Default is <daily>
            [--log-file <file name with path>]
                 Default is </opt/cloudhsm/run/cloudhsm-jce.log>
                 Default for Windows is <C:\\Program Files\\Amazon\\CloudHSM\
\cloudhsm-jce.log>
            [--log-type <file | term>]
                 Default is <file>
            [-h | --help]
            [-V | --version]
            [--disable-key-availability-check]
            [--enable-key-availability-check]
            [--disable-validate-key-at-init]
```

```
                This is the default for JCE
            [--enable-validate-key-at-init]
```

CloudHSM CLI

```
configure-cli[ .exe ]
             -a <ENI IP address>
             [--hsm-ca-cert <customerCA certificate file path>]
             [--cluster-id <cluster ID>]
             [--endpoint <endpoint>]
             [--region <region>]
             [--client-cert-hsm-tls-file <client certificate hsm tls path>]
             [--client-key-hsm-tls-file <client key hsm tls path>]
             [--log-level <error | warn | info | debug | trace>]
                  Default is <info>
             [--log-rotation <daily | weekly>]
                  Default is <daily>
             [--log-file <file name with path>]
                  Default for Linux is </opt/cloudhsm/run/cloudhsm-cli.log>
                  Default for Windows is <C:\\Program Files\\Amazon\\CloudHSM\
\cloudhsm-cli.log>
             [--log-type <file | term>]
                  Default setting is <file>
             [-h | --help]
             [-V | --version]
             [--disable-key-availability-check]
             [--enable-key-availability-check]
             [--disable-validate-key-at-init]
             [--enable-validate-key-at-init]
                  This is the default for CloudHSM CLI
```

AWS CloudHSM Client SDK 5 configuration parameters

The following is a list of parameters to configure AWS CloudHSM Client SDK 5.

-a *<ENI IP address>*

Adds the specified IP address to Client SDK 5 configuration files. Enter any ENI IP address of an HSM from the cluster. For more information about how to use this option, see Bootstrap Client SDK 5.

Required: Yes

--hsm-ca-cert ***<customerCA certificate file path>***

Path to the directory storing the certificate authority (CA) certificate use to connect EC2 client instances to the cluster. You create this file when you initialize the cluster. By default, the system looks for this file in the following location:

Linux

```
/opt/cloudhsm/etc/customerCA.crt
```

Windows

```
C:\ProgramData\Amazon\CloudHSM\customerCA.crt
```

For more information about initializing the cluster or placing the certificate, see ??? and ???.

Required: No

--cluster-id ***<cluster ID>***

Makes a `DescribeClusters` call to find all of the HSM elastic network interface (ENI) IP addresses in the cluster associated with the cluster ID. The system adds the ENI IP addresses to the AWS CloudHSM configuration files.

> **Note**
>
> If you use the `--cluster-id` parameter from an EC2 instance within a VPC that does not have access to the public internet, then you must create an interface VPC endpoint to connect with AWS CloudHSM. For more information about VPC endpoints, see ???.

Required: No

--endpoint ***<endpoint>***

Specify the AWS CloudHSM API endpoint used for making the `DescribeClusters` call. You must set this option in combination with `--cluster-id`.

Required: No

--region ***<region>***

Specify the region of your cluster. You must set this option in combination with `--cluster-id`.

If you don't supply the `--region` parameter, the system chooses the region by attempting to read the `AWS_DEFAULT_REGION` or `AWS_REGION` environment variables. If those variables aren't set, then the system checks the region associated with your profile in your AWS config file (typically `~/.aws/config`) unless you specified a different file in the `AWS_CONFIG_FILE` environment variable. If none of the above are set, the system defaults to the `us-east-1` region.

Required: No

--client-cert-hsm-tls-file ***<client certificate hsm tls path>***

Path to the client certificate used for TLS client-HSM mutual authentication.

Only use this option if you have registered at least one trust anchor onto HSM with CloudHSM CLI. You must set this option in combination with `--client-key-hsm-tls-file`.

Required: No

--client-key-hsm-tls-file ***<client key hsm tls path>***

Path to the client key used for TLS client-HSM mutual authentication.

Only use this option if you have registered at least one trust anchor onto HSM with CloudHSM CLI. You must set this option in combination with `--client-cert-hsm-tls-file`.

Required: No

--log-level ***<error | warn | info | debug | trace>***

Specifies the minimum logging level the system should write to the log file. Each level includes the previous levels, with error as the minimum level and trace the maximum level. This means that if you specify errors, the system only writes errors to the log. If you specify trace, the system writes errors, warnings, informational (info) and debug messages to the log. For more information, see Client SDK 5 Logging.

Required: No

--log-rotation *<daily | weekly>*

Specifies the frequency with which the system rotates logs. For more information, see Client SDK 5 Logging.

Required: No

--log-file *<file name with path>*

Specifies where the system will write the log file. For more information, see Client SDK 5 Logging.

Required: No

--log-type *<term | file>*

Specifies whether the system will write the log to a file or terminal. For more information, see Client SDK 5 Logging.

Required: No

-h | --help

Displays help.

Required: No

-v | --version

Displays version.

Required: No

--disable-key-availability-check

Flag to disable key availability quorum. Use this flag to indicate AWS CloudHSM should disable key availability quorum and you can use keys that exist on only one HSM in the cluster. For more information about using this flag to set key availability quorum, see ???.

Required: No

--enable-key-availability-check

Flag to enable key availability quorum. Use this flag to indicate AWS CloudHSM should use key availability quorum and not allow you to use keys until those keys exist on two HSMs in the cluster. For more information about using this flag to set key availability quorum, see ???.

Enabled by default.

Required: No

--disable-validate-key-at-init

Improves performance by specifying that you can skip an initialization call to verify permissions on a key for subsequent calls. Use with caution.

Background: Some mechanisms in the PKCS #11 library support multi-part operations where an initialization call verifies if you can use the key for subsequent calls. This requires a verification call to the HSM, which adds latency to the overall operation. This option enables you to disable the subsequent call and potentially improve performance.

Required: No

--enable-validate-key-at-init

Specifies that you should use an initialization call to verify permissions on a key for subsequent calls. This is the default option. Use `enable-validate-key-at-init` to resume these initialization calls after you use `disable-validate-key-at-init` to suspend them.

Required: No

AWS CloudHSM Client SDK 5 configuration examples

These examples show how to use the configure tool for AWS CloudHSM Client SDK 5.

Bootstrap Client SDK 5

Example

This example uses the `-a` parameter to update the HSM data for Client SDK 5. To use the `-a` parameter, you must have the IP address for one of the HSMs in your cluster.

PKCS #11 library

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of an HSM in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 -a <HSM IP addresses>
```

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of an HSM in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" -a <HSM IP
 addresses>
```

OpenSSL Dynamic Engine

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of an HSM in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-dyn -a <HSM IP addresses>
```

Key Storage Provider (KSP)

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of an HSM in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" -a <HSM IP
 addresses>
```

JCE provider

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of an HSM in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-jce -a <HSM IP addresses>
```

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of an HSM in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" -a <HSM IP
 addresses>
```

CloudHSM CLI

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of the HSM(s) in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-cli -a <The ENI IPv4 / IPv6 addresses of the
 HSMs>
```

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of the HSM(s) in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" -a <The ENI
 IPv4 / IPv6 addresses of the HSMs>
```

Note

you can use the `--cluster-id` parameter in place of `-a <HSM_IP_ADDRESSES>`. To see requirements for using `--cluster-id`, see AWS CloudHSM Client SDK 5 configure tool.

For more information about the `-a` parameter, see the section called “Parameters”.

Specify cluster, region, and endpoint for Client SDK 5

Example

This example uses the `cluster-id` parameter to bootstrap Client SDK 5 by making a `DescribeClusters` call.

PKCS #11 library

To bootstrap a Linux EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of an HSM in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --cluster-id <cluster-1234567>
```

To bootstrap a Windows EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of an HSM in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" --cluster-id <cluster-1234567>
```

OpenSSL Dynamic Engine

To bootstrap a Linux EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of an HSM in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --cluster-id <cluster-1234567>
```

Key Storage Provider (KSP)

To bootstrap a Windows EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of an HSM in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" --cluster-id <cluster-1234567>
```

JCE provider

To bootstrap a Linux EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of an HSM in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-jce --cluster-id <cluster-1234567>
```

To bootstrap a Windows EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of an HSM in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" --cluster-
id <cluster-1234567>
```

CloudHSM CLI

To bootstrap a Linux EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of an HSM in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-cli --cluster-id <cluster-1234567>
```

To bootstrap a Windows EC2 instance for Client SDK 5 with `cluster-id`

- Use the cluster ID `cluster-1234567` to specify the IP address of an HSM in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" --cluster-
id <cluster-1234567>
```

You can use the `--region` and `--endpoint` parameters in combination with the `cluster-id` parameter to specify how the system makes the `DescribeClusters` call. For instance, if the region of the cluster is different than the one configured as your AWS CLI default, you should use the `--region` parameter to use that region. Additionally, you have the ability to specify the AWS CloudHSM API endpoint to use for the call, which might be necessary for various network setups, such as using VPC interface endpoints that don't use the default DNS hostname for AWS CloudHSM.

PKCS #11 library

To bootstrap a Linux EC2 instance with a custom endpoint and region

- Use the configure tool to specify the IP address of an HSM in your cluster with a custom region and endpoint.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --cluster-id <cluster-1234567> --
region <us-east-1> --endpoint <https://cloudhsmv2.us-east-1.amazonaws.com>
```

To bootstrap a Windows EC2 instance with a endpoint and region

- Use the configure tool to specify the IP address of an HSM in your cluster with a custom region and endpoint.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" --cluster-
id <cluster-1234567>--region <us-east-1> --endpoint <https://cloudhsmv2.us-
east-1.amazonaws.com>
```

OpenSSL Dynamic Engine

To bootstrap a Linux EC2 instance with a custom endpoint and region

- Use the configure tool to specify the IP address of an HSM in your cluster with a custom region and endpoint.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --cluster-id <cluster-1234567> --
region <us-east-1> --endpoint <https://cloudhsmv2.us-east-1.amazonaws.com>
```

Key Storage Provider (KSP)

To bootstrap a Windows EC2 instance with a endpoint and region

- Use the configure tool to specify the IP address of an HSM in your cluster with a custom region and endpoint.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" --cluster-id <cluster-1234567> --region <us-east-1> --endpoint <https://cloudhsmv2.us-east-1.amazonaws.com>
```

JCE provider

To bootstrap a Linux EC2 instance with a custom endpoint and region

- Use the configure tool to specify the IP address of an HSM in your cluster with a custom region and endpoint.

```
$ sudo /opt/cloudhsm/bin/configure-jce --cluster-id <cluster-1234567> --region <us-east-1> --endpoint <https://cloudhsmv2.us-east-1.amazonaws.com>
```

To bootstrap a Windows EC2 instance with a endpoint and region

- Use the configure tool to specify the IP address of an HSM in your cluster with a custom region and endpoint.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" --cluster-id <cluster-1234567> --region <us-east-1> --endpoint <https://cloudhsmv2.us-east-1.amazonaws.com>
```

CloudHSM CLI

To bootstrap a Linux EC2 instance with a custom endpoint and region

- Use the configure tool to specify the IP address of an HSM in your cluster with a custom region and endpoint.

```
$ sudo /opt/cloudhsm/bin/configure-cli --cluster-id <cluster-1234567> --region <us-east-1> --endpoint <https://cloudhsmv2.us-east-1.amazonaws.com>
```

To bootstrap a Windows EC2 instance with a endpoint and region

- Use the configure tool to specify the IP address of an HSM in your cluster with a custom region and endpoint.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" --cluster-id <cluster-1234567> --region <us-east-1> --endpoint <https://cloudhsmv2.us-east-1.amazonaws.com>
```

For more information about the `--cluster-id`, `--region`, and `--endpoint` parameters, see the section called “Parameters”.

Update client certificate and key for TLS client-HSM mutual authentication

Example

This examples shows how to use the `--client-cert-hsm-tls-file` and `--client-key-hsm-tls-file` parameters to reconfigure SSL by specifying a custom key and SSL certificate for AWS CloudHSM

PKCS #11 library

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

```
$ sudo cp ssl-client.pem </opt/cloudhsm/etc>
$ sudo cp ssl-client.key </opt/cloudhsm/etc>
```

2. Use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 \
            --client-cert-hsm-tls-file </opt/cloudhsm/etc/ssl-client.pem> \
            --client-key-hsm-tls-file </opt/cloudhsm/etc/ssl-client.key>
```

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Windows

1. Copy your key and certificate to the appropriate directory.

```
cp ssl-client.pem <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem>
cp ssl-client.key <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

2. With a PowerShell interpreter, use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" `
            --client-cert-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem> `
            --client-key-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

OpenSSL Dynamic Engine

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

```
$ sudo cp ssl-client.pem </opt/cloudhsm/etc>
sudo cp ssl-client.key </opt/cloudhsm/etc>
```

2. Use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
$ sudo /opt/cloudhsm/bin/configure-dyn \
            --client-cert-hsm-tls-file </opt/cloudhsm/etc/ssl-client.pem> \
            --client-key-hsm-tls-file </opt/cloudhsm/etc/ssl-client.key>
```

Key Storage Provider (KSP)

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Windows

1. Copy your key and certificate to the appropriate directory.

```
cp ssl-client.pem <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem>
cp ssl-client.key <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

2. With a PowerShell interpreter, use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" `
            --client-cert-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem> `
            --client-key-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

JCE provider

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

```
$ sudo cp ssl-client.pem </opt/cloudhsm/etc>
sudo cp ssl-client.key </opt/cloudhsm/etc>
```

2. Use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
$ sudo /opt/cloudhsm/bin/configure-jce \
            --client-cert-hsm-tls-file </opt/cloudhsm/etc/ssl-client.pem> \
            --client-key-hsm-tls-file </opt/cloudhsm/etc/ssl-client.key>
```

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Windows

1. Copy your key and certificate to the appropriate directory.

```
cp ssl-client.pem <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem>
cp ssl-client.key <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

2. With a PowerShell interpreter, use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" `
            --client-cert-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem> `
```

```
            --client-key-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

CloudHSM CLI

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Linux

1. Copy your key and certificate to the appropriate directory.

```
$ sudo cp ssl-client.pem </opt/cloudhsm/etc>
sudo cp ssl-client.key </opt/cloudhsm/etc>
```

2. Use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
$ sudo /opt/cloudhsm/bin/configure-cli \
            --client-cert-hsm-tls-file </opt/cloudhsm/etc/ssl-client.pem> \
            --client-key-hsm-tls-file </opt/cloudhsm/etc/ssl-client.key>
```

To use a custom certificate and key for TLS client-HSM mutual authentication with Client SDK 5 on Windows

1. Copy your key and certificate to the appropriate directory.

```
cp ssl-client.pem <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem>
cp ssl-client.key <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

2. With a PowerShell interpreter, use the configure tool to specify `ssl-client.pem` and `ssl-client.key`.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" `
            --client-cert-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-client.pem> `
            --client-key-hsm-tls-file <C:\ProgramData\Amazon\CloudHSM\ssl-client.key>
```

For more information about the `--client-cert-hsm-tls-file` and `--client-key-hsm-tls-file` parameters, see the section called “Parameters”.

Disable client key durability settings

Example

This example uses the `--disable-key-availability-check` parameter to disable client key durability settings. To run a cluster with a single HSM, you must disable client key durability settings.

PKCS #11 library

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --disable-key-availability-check
```

To disable client key durability for Client SDK 5 on Windows

- Use the configure tool to disable client key durability settings.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" --disable-
key-availability-check
```

OpenSSL Dynamic Engine

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --disable-key-availability-check
```

Key Storage Provider (KSP)

To disable client key durability for Client SDK 5 on Windows

- Use the configure tool to disable client key durability settings.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" --disable-key-availability-check
```

JCE provider

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
$ sudo /opt/cloudhsm/bin/configure-jce --disable-key-availability-check
```

To disable client key durability for Client SDK 5 on Windows

- Use the configure tool to disable client key durability settings.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" --disable-key-availability-check
```

CloudHSM CLI

To disable client key durability for Client SDK 5 on Linux

- Use the configure tool to disable client key durability settings.

```
$ sudo /opt/cloudhsm/bin/configure-cli --disable-key-availability-check
```

To disable client key durability for Client SDK 5 on Windows

- Use the configure tool to disable client key durability settings.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" --disable-key-availability-check
```

For more information about the `--disable-key-availability-check` parameter, see the section called "Parameters".

Manage logging options

Example

Client SDK 5 uses the `log-file`, `log-level`, `log-rotation`, and `log-type` parameters to manage logging.

> **Note**
>
> To configure your SDK for serverless environments such as AWS Fargate or AWS Lambda, we recommend you configure your AWS CloudHSM log type to `term`. The client logs will be output to `stderr` and captured in the CloudWatch Logs log group configured for that environment.

PKCS #11 library

Default logging location

- If you do not specify a location for the file, the system writes logs to the following default location:

  Linux

  ```
  /opt/cloudhsm/run/cloudhsm-pkcs11.log
  ```

  Windows

  ```
  C:\Program Files\Amazon\CloudHSM\cloudhsm-pkcs11.log
  ```

To configure the logging level and leave other logging options set to default

- ```
  $ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-level info
  ```
  ```

To configure file logging options

- ```
  $ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-type file --log-file <file name with path> --log-rotation daily --log-level info
  ```

To configure terminal logging options

- ```
  $ sudo /opt/cloudhsm/bin/configure-pkcs11 --log-type term --log-level info
  ```

OpenSSL Dynamic Engine

Default logging location

- If you do not specify a location for the file, the system writes logs to the following default location:

  Linux

  ```
  stderr
  ```

To configure the logging level and leave other logging options set to default

- ```
  $ sudo /opt/cloudhsm/bin/configure-dyn --log-level info
  ```

To configure file logging options

- ```
  $ sudo /opt/cloudhsm/bin/configure-dyn --log-type <file name> --log-file file --log-rotation daily --log-level info
  ```

To configure terminal logging options

- ```
  $ sudo /opt/cloudhsm/bin/configure-dyn --log-type term --log-level info
  ```
  ```

Key Storage Provider (KSP)

Default logging location

- If you do not specify a location for the file, the system writes logs to the following default location:

  Windows

  ```
  C:\Program Files\Amazon\CloudHSM\cloudhsm-ksp.log
  ```

To configure the logging level and leave other logging options set to default

- ```
  PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" --log-level
   info
  ```

To configure file logging options

- ```
  PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" --log-type
   file --log-file <file name> --log-rotation daily --log-level info
  ```

To configure terminal logging options

- ```
  PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" --log-type
   term --log-level info
  ```

JCE provider

Default logging location

- If you do not specify a location for the file, the system writes logs to the following default location:

  Linux

  ```
  /opt/cloudhsm/run/cloudhsm-jce.log
  ```
  ```

Windows

```
C:\Program Files\Amazon\CloudHSM\cloudhsm-jce.log
```

To configure the logging level and leave other logging options set to default

- ```
  $ sudo /opt/cloudhsm/bin/configure-jce --log-level info
  ```

To configure file logging options

- ```
  $ sudo /opt/cloudhsm/bin/configure-jce --log-type file --log-file <file name> --
  log-rotation daily --log-level info
  ```

To configure terminal logging options

- ```
  $ sudo /opt/cloudhsm/bin/configure-jce --log-type term --log-level info
  ```

CloudHSM CLI

Default logging location

- If you do not specify a location for the file, the system writes logs to the following default location:

  Linux

  ```
  /opt/cloudhsm/run/cloudhsm-cli.log
  ```

  Windows

  ```
  C:\Program Files\Amazon\CloudHSM\cloudhsm-cli.log
  ```
  ```

To configure the logging level and leave other logging options set to default

- ```
  $ sudo /opt/cloudhsm/bin/configure-cli --log-level info
  ```

To configure file logging options

- ```
  $ sudo /opt/cloudhsm/bin/configure-cli --log-type file --log-file <file name> --
  log-rotation daily --log-level info
  ```

To configure terminal logging options

- ```
  $ sudo /opt/cloudhsm/bin/configure-cli --log-type term --log-level info
  ```

For more information about the `log-file`, `log-level`, `log-rotation`,and `log-type` parameters, see the section called “Parameters”.

Place the issuing certificate for Client SDK 5

Example

This example uses the `--hsm-ca-cert` parameter to update the location of the issuing certificate for Client SDK 5.

PKCS #11 library

To place the issuing certificate on Linux for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

  ```
  $ sudo /opt/cloudhsm/bin/configure-pkcs11 --hsm-ca-cert <customerCA certificate
   file>
  ```

To place the issuing certificate on Windows for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.
  ```

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" --hsm-ca-cert <customerCA certificate file>
```

OpenSSL Dynamic Engine

To place the issuing certificate on Linux for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
$ sudo /opt/cloudhsm/bin/configure-dyn --hsm-ca-cert <customerCA certificate file>
```

Key Storage Provider (KSP)

To place the issuing certificate on Windows for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" --hsm-ca-cert <customerCA certificate file>
```

JCE provider

To place the issuing certificate on Linux for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
$ sudo /opt/cloudhsm/bin/configure-jce --hsm-ca-cert <customerCA certificate file>
```

To place the issuing certificate on Windows for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" --hsm-ca-cert <customerCA certificate file>
```

CloudHSM CLI

To place the issuing certificate on Linux for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
$ sudo /opt/cloudhsm/bin/configure-cli --hsm-ca-cert <customerCA certificate
 file>
```

To place the issuing certificate on Windows for Client SDK 5

- Use the configure tool to specify a location for the issuing certificate.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" --hsm-ca-
cert <customerCA certificate file>
```

For more information about the `--hsm-ca-cert` parameter, see the section called “Parameters”.

Advanced configurations for the Client SDK 5 configure tool

The AWS CloudHSM Client SDK 5 configure tool includes advanced configurations that are not part of the general features most customers utilize. Advanced configurations provide additional capabilities.

> **Important**
>
> After making any changes to your configuration, you need to restart your application for the changes to take effect.

- Advanced configurations for PKCS #11
  - Multiple slot configuration with PKCS #11 library for AWS CloudHSM
  - Retry commands for PKCS #11 library for AWS CloudHSM
- Advanced configurations for OpenSSL
  - Retry commands for OpenSSL for AWS CloudHSM
- Advanced configurations for KSP
  - SDK3 compatibility mode for Key Storage Provider (KSP) for AWS CloudHSM

- Advanced configurations for JCE
  - Connecting to multiple AWS CloudHSM clusters with the JCE provider
  - Retry commands for JCE for AWS CloudHSM
  - Key extraction using JCE for AWS CloudHSM
- Advanced configurations for AWS CloudHSM Command Line Interface (CLI)
  - Connecting to multiple clusters with CloudHSM CLI

AWS CloudHSM Client SDK 5 related topics

See the following related topics to learn more about the AWS CloudHSM Client SDK 5.

- DescribeClusters API operation
- describe-clusters AWS CLI
- Get-HSM2Cluster PowerShell cmdlet
- Bootstrap Client SDK 5
- AWS CloudHSM VPC endpoints
- Managing Client SDK 5 Key Durability Settings
- Client SDK 5 Logging
- Setup mTLS (recommended)

AWS CloudHSM Client SDK 3 configure tool

Use the AWS CloudHSM Client SDK 3 configure tool to bootstrap the client daemon and configure CloudHSM Management Utility (CMU).

Topics

- AWS CloudHSM Client SDK 3 configuration syntax
- AWS CloudHSM Client SDK 3 configuration parameters
- AWS CloudHSM Client SDK 3 configuration examples
- AWS CloudHSM Client SDK 3 configuration related topics

AWS CloudHSM Client SDK 3 configuration syntax

The following table illustrates the syntax for AWS CloudHSM configuration files for Client SDK 3.

```
configure -h | --help
          -a <ENI IP address>
          -m [-i <daemon_id>]
          --ssl --pkey <private key file> --cert <certificate file>
          --cmu <ENI IP address>
```

AWS CloudHSM Client SDK 3 configuration parameters

The following is a list of parameters to configure AWS CloudHSM Client SDK 3.

-h | --help

Displays command syntax.

Required: Yes

-a *<ENI IP address>*

Adds the specified HSM elastic network interface (ENI) IP address to AWS CloudHSM configuration files. Enter the ENI IP address of any one of the HSMs in the cluster. It does not matter which one you select.

To get the ENI IP addresses of the HSMs in your cluster, use the DescribeClusters operation, the describe-clusters AWS CLI command, or the Get-HSM2Cluster PowerShell cmdlet.

> **Note**
>
> Before running the `-a` **configure** command, stop the AWS CloudHSM client. Then, when the `-a` command completes, restart the AWS CloudHSM client. For details, see the examples.

This parameter edits the following configuration files:

- `/opt/cloudhsm/etc/cloudhsm_client.cfg`: Used by AWS CloudHSM client and key_mgmt_util.
- `/opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg`: Used by cloudhsm_mgmt_util.

When the AWS CloudHSM client starts, it uses the ENI IP address in its configuration file to query the cluster and update the `cluster.info` file (`/opt/cloudhsm/daemon/1/cluster.info`) with the correct ENI IP addresses for all HSMs in the cluster.

Required: Yes

-m

Updates the HSM ENI IP addresses in the configuration file that CMU uses.

> **Note**
>
> The `-m` parameter is for use with CMU from Client SDK 3.2.1 and earlier. For CMU from Client SDK 3.3.0 and later, see `--cmu` parameter, which simplifies the process of updating HSM data for CMU.

When you update the `-a` parameter of **configure** and then start the AWS CloudHSM client, the client daemon queries the cluster and updates the `cluster.info` files with the correct HSM IP addresses for all HSMs in the cluster. Running the `-m` **configure** command completes the update by copying the HSM IP addresses from the `cluster.info` to the `cloudhsm_mgmt_util.cfg` configuration file that cloudhsm_mgmt_util uses.

Be sure to run `-a` **configure** command and restart the AWS CloudHSM client before running the `-m` command. This ensures that the data copied into `cloudhsm_mgmt_util.cfg` from `cluster.info` is complete and accurate.

Required: Yes

-i

Specifies an alternate client daemon. The default value represents the AWS CloudHSM client.

Default: 1

Required: No

--ssl

Replaces the SSL key and certificate for the cluster with the specified private key and certificate. When you use this parameter, the `--pkey` and `--cert` parameters are required.

Required: No

--pkey

Specifies the new private key. Enter the path and file name of the file that contains the private key.

Required: Yes if **--ssl** is specified. Otherwise, this should not be used.

--cert

Specifies the new certificate. Enter the path and file name of the file that contains the certificate. The certificate should chain up to the `customerCA.crt` certificate, the self-signed certificate used to initialize the cluster. For more information, see Initialize the Cluster.

Required: Yes if **--ssl** is specified. Otherwise, this should not be used.

--cmu ***<ENI IP address>***

Combines the `-a` and `-m` parameters into one parameter. Adds the specified HSM elastic network interface (ENI) IP address to AWS CloudHSM configuration files and then updates the CMU configuration file. Enter an IP address from any HSM in the cluster. For Client SDK 3.2.1 and earlier, see Using CMU with Client SDK 3.2.1 and Earlier.

Required: Yes

AWS CloudHSM Client SDK 3 configuration examples

These examples show how to use the **configure** tool for AWS CloudHSM Client SDK 3.

Example : Update the HSM data for the AWS CloudHSM client and key_mgmt_util

This example uses the `-a` parameter of **configure** to update the HSM data for the AWS CloudHSM client and key_mgmt_util. To use the `-a` parameter, you must have the IP address for one of the HSMs in your cluster. Use either the console or the AWS CLI to get the IP address.

To get an IP address for an HSM (console)

1. Open the AWS CloudHSM console at https://console.aws.amazon.com/cloudhsm/home.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To open the cluster detail page, in the cluster table, choose the cluster ID.
4. To get the IP address, go to the HSMs tab. For IPv4 clusters, choose an address listed under **ENI IPv4 address**. For dual-stack clusters use either the ENI IPv4 or the **ENI IPv6 address**.

To get an IP address for an HSM (AWS CLI)

- Get the IP address of an HSM by using the [describe-clusters](#) command from the AWS CLI. In the output from the command, the IP address of the HSMs are the values of `EniIp` and `EniIpV6` (if it is a dual-stack cluster).

```
$ aws cloudhsmv2 describe-clusters
{
    "Clusters": [
        { ... }
            "Hsms": [
                {
...
                    "EniIp": "10.0.0.9",
...
                },
                {
...
                    "EniIp": "10.0.1.6",
                    "EniIpV6": "2600:113f:404:be09:310e:ed34:3412:f733",
...
```

To update the HSM data

1. Before updating the `-a` parameter, stop the AWS CloudHSM client. This prevents conflicts that might occur while **configure** edits the client's configuration file. If the client is already stopped, this command has no effect, so you can use it in a script.

   Amazon Linux

   ```
   $ sudo stop cloudhsm-client
   ```

   Amazon Linux 2

   ```
   $ sudo service cloudhsm-client stop
   ```

   CentOS 7

   ```
   $ sudo service cloudhsm-client stop
   ```

CentOS 8

```
$ sudo service cloudhsm-client stop
```

RHEL 7

```
$ sudo service cloudhsm-client stop
```

RHEL 8

```
$ sudo service cloudhsm-client stop
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client stop
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client stop
```

Windows

- For Windows client 1.1.2+:

  ```
  C:\Program Files\Amazon\CloudHSM>net.exe stop AWSCloudHSMClient
  ```

- For Windows clients 1.1.1 and older:

  Use **Ctrl**+**C** in the command window where you started the AWS CloudHSM client.

2. This step uses the `-a` parameter of **configure** to add the `10.0.0.9` ENI IP address to the configurations files.

   Amazon Linux

   ```
   $ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
   ```

Amazon Linux 2

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

CentOS 7

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

CentOS 8

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

RHEL 7

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

RHEL 8

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

Ubuntu 16.04 LTS

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

Ubuntu 18.04 LTS

```
$ sudo /opt/cloudhsm/bin/configure -a 10.0.0.9
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" -a 10.0.0.9
```

3. Next, restart the AWS CloudHSM client. When the client starts, it uses the ENI IP address in its configuration file to query the cluster. Then, it writes the ENI IP addresses of all HSMs in the cluster to the `cluster.info` file.

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- For Windows client 1.1.2+:

```
C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
```

- For Windows clients 1.1.1 and older:

```
C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe
 C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

When the command completes, the HSM data that the AWS CloudHSM client and key_mgmt_util use is complete and accurate.

Example : Update the HSM Data for CMU from client SDK 3.2.1 and earlier

This example uses the `-m` **configure** command to copy the updated HSM data from the `cluster.info` file to the `cloudhsm_mgmt_util.cfg` file that cloudhsm_mgmt_util uses. Use this with CMU that ships with Client SDK 3.2.1 and earlier.

- Before running the `-m`, stop the AWS CloudHSM client, run the `-a` command, and then restart the AWS CloudHSM client, as shown in the previous example. This ensures that the data copied into the `cloudhsm_mgmt_util.cfg` file from the `cluster.info` file is complete and accurate.

  Linux

  ```
  $ sudo /opt/cloudhsm/bin/configure -m
  ```

  Windows

  ```
  PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" -m
  ```

Example : Update the HSM Data for CMU from client SDK 3.3.0 and later

This example uses the `--cmu` parameter of the **configure** command to update HSM data for CMU. Use this with CMU that ships with Client SDK 3.3.0 and later. For more information about using CMU, see Using CloudHSM Management Utility (CMU) to Manage Users and Using CMU with Client SDK 3.2.1 and Earlier.

- Use the `--cmu` parameter to pass the IP address of an HSM in your cluster.

Linux

```
$ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" --cmu <IP address>
```

AWS CloudHSM Client SDK 3 configuration related topics

See the following related topics to learn more about the AWS CloudHSM Client SDK 3.

- Set up AWS CloudHSM key_mgmt_util

AWS CloudHSM Command Line Interface (CLI)

CloudHSM CLI helps admins manage users and crypto users manage keys in their cluster in AWS CloudHSM. The CLI includes tools that can be used to create, delete and list users, change user passwords, update user multi-factor authentication (MFA). It also includes commands that generate, delete, import, and export keys, get and set attributes, find keys, and perform cryptographic operations.

For defined list of CloudHSM CLI users, see HSM user management with CloudHSM CLI . For a defined list of key attributes for CloudHSM CLI, see Key attributes for CloudHSM CLI. For information on how to use CloudHSM CLI to manage keys, see Key management with CloudHSM CLI.

For a quick start, see Getting started with AWS CloudHSM Command Line Interface (CLI). For detailed information about the CloudHSM CLI commands and examples of using the commands, see Reference for CloudHSM CLI commands.

Topics

- AWS CloudHSM Command Line Interface (CLI) supported platforms
- Migrate from AWS CloudHSM Client SDK 3 CMU and KMU to Client SDK 5 CloudHSM CLI
- Getting started with AWS CloudHSM Command Line Interface (CLI)
- Command modes in CloudHSM CLI

- Key attributes for CloudHSM CLI
- Advanced configurations for CloudHSM CLI
- Reference for CloudHSM CLI commands

AWS CloudHSM Command Line Interface (CLI) supported platforms

This topic describes the Linux and Windows platforms that the AWS CloudHSM CLI supports.

Linux support

Supported platforms	X86_64 Architecture	ARM architecture
Amazon Linux 2	Yes	Yes
Amazon Linux 2023	Yes	Yes
Red Hat Enterprise Linux 8 (8.3+)	Yes	No
Red Hat Enterprise Linux 9 (9.2+)	Yes	Yes
Ubuntu 22.04 LTS	Yes	Yes
Ubuntu 24.04 LTS	Yes	Yes

- SDK 5.16 was the last release to provide Ubuntu 20.04 LTS platform support. For more information, see the Ubuntu website.
- SDK 5.12 was the last release to provide CentOS 7 (7.8+) platform support. For more information, see the CentOS website.
- SDK 5.12 was the last release to provide Red Hat Enterprise Linux 7 (7.8+) platform support. For more information, see the Red Hat website.
- SDK 5.4.2 was the last release to provide CentOS 8 platform support. For more information, see the CentOS website.

Windows support

- Microsoft Windows Server 2016
- Microsoft Windows Server 2019

Migrate from AWS CloudHSM Client SDK 3 CMU and KMU to Client SDK 5 CloudHSM CLI

Use this topic to migrate workflows that use the AWS CloudHSM Client SDK 3 command line tools, the CloudHSM Management Utility (CMU) and the Key Management Utility (KMU), to instead use the Client SDK 5 command line tool, CloudHSM CLI.

In AWS CloudHSM, customer applications perform cryptographic operations using the AWS CloudHSM Client Software Development Kit (SDK). Client SDK 5 is the primary SDK that continues to have new features and platform support added to it. This topic provides details specific to migrating from Client SDK 3 to Client SDK 5 for command line tools.

Client SDK 3 includes two separate command line tools: the CMU for managing users and the KMU for managing keys and performing operations with keys. Client SDK 5 consolidates the functions of the CMU and KMU (tools that were offered with Client SDK 3) into a single tool, the AWS CloudHSM Command Line Interface (CLI). User management operations can be found under the subcommands The user category in CloudHSM CLI and The quorum category in CloudHSM CLI. Key management operations can be found under the key subcommand, and cryptographic operations can be found under the crypto subcommand. See Reference for CloudHSM CLI commands for a complete list of commands.

For instructions on migrating to Client SDK 5, see Migrating from AWS CloudHSM Client SDK 3 to Client SDK 5. For benefits on migrating, see Benefits of AWS CloudHSM Client SDK 5.

Getting started with AWS CloudHSM Command Line Interface (CLI)

With the CloudHSM CLI Command Line Interface (CLI), you can manage users in your AWS CloudHSM cluster. Use this topic to get started with basic hardware security module (HSM) user management tasks, such as creating users, listing users, and connecting CloudHSM CLI to the cluster.

Topics

- Install the CloudHSM CLI
- Use the CloudHSM CLI

Install the CloudHSM CLI

Use the following commands to download and install the CloudHSM CLI for AWS CloudHSM.

Amazon Linux 2023

Amazon Linux 2023 on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/
cloudhsm-cli-latest.amzn2023.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.amzn2023.x86_64.rpm
```

Amazon Linux 2023 on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/
cloudhsm-cli-latest.amzn2023.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.amzn2023.aarch64.rpm
```

Amazon Linux 2

Amazon Linux 2 on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-cli-
latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el7.x86_64.rpm
```

Amazon Linux 2 on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-cli-
latest.el7.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el7.aarch64.rpm
```

RHEL 9 (9.2+)

RHEL 9 on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-cli-latest.el9.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el9.x86_64.rpm
```

RHEL 9 on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-cli-latest.el9.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el9.aarch64.rpm
```

RHEL 8 (8.3+)

RHEL 8 on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-cli-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-cli-latest.el8.x86_64.rpm
```

Ubuntu 24.04 LTS

Ubuntu 24.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Noble/cloudhsm-cli_latest_u24.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u24.04_amd64.deb
```

Ubuntu 24.04 LTS on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Noble/cloudhsm-cli_latest_u24.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u24.04_arm64.deb
```

Ubuntu 22.04 LTS

Ubuntu 22.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/cloudhsm-cli_latest_u22.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u22.04_amd64.deb
```

Ubuntu 22.04 LTS on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/cloudhsm-cli_latest_u22.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u22.04_arm64.deb
```

Ubuntu 20.04 LTS

Ubuntu 20.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Focal/cloudhsm-cli_latest_u20.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-cli_latest_u20.04_amd64.deb
```

Windows Server 2022

For Windows Server 2022 on x86_64 architecture, open PowerShell as an administrator and run the following command:

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Windows/AWSCloudHSMCLI-latest.msi -Outfile C:\AWSCloudHSMCLI-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMCLI-latest.msi /
quiet /norestart /log C:\client-install.txt' -Wait
```

Windows Server 2019

For Windows Server 2019 on x86_64 architecture, open PowerShell as an administrator and run the following command:

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Windows/
AWSCloudHSMCLI-latest.msi -Outfile C:\AWSCloudHSMCLI-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMCLI-latest.msi /
quiet /norestart /log C:\client-install.txt' -Wait
```

Windows Server 2016

For Windows Server 2016 on x86_64 architecture, open PowerShell as an administrator and run the following command:

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Windows/
AWSCloudHSMCLI-latest.msi -Outfile C:\AWSCloudHSMCLI-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMCLI-latest.msi /
quiet /norestart /log C:\client-install.txt' -Wait
```

Use the following commands to configure CloudHSM CLI.

To bootstrap a Linux EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of the HSM(s) in your cluster.

```
$ sudo /opt/cloudhsm/bin/configure-cli -a <The ENI IPv4 / IPv6 addresses of the
 HSMs>
```

To bootstrap a Windows EC2 instance for Client SDK 5

- Use the configure tool to specify the IP address of the HSM(s) in your cluster.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" -a <The ENI
 IPv4 / IPv6 addresses of the HSMs>
```

Use the CloudHSM CLI

Use the following commands to start and use the CloudHSM CLI.

1. Use the following command to start CloudHSM CLI.

   Linux

   ```
   $ /opt/cloudhsm/bin/cloudhsm-cli interactive
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
   ```

2. Use the **login** command to log in to the cluster. All users can use this command.

   The command in the following example logs in *admin*, which is the default admin account. You set this user's password when you activated the cluster.

   ```
   aws-cloudhsm > login --username admin --role admin
   ```

   The system prompts you for your password. You enter the password, and the output shows that the command was successful.

   ```
   Enter password:
   {
     "error_code": 0,
     "data": {
       "username": "admin",
       "role": "admin"
     }
   }
   ```

3. Run the **user list** command to list all the users on the cluster.

   ```
   aws-cloudhsm > user list
   ```

```
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      },
      {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

4. Use **user create** to create a CU user named **`example_user`**.

   You can create CUs because in a previous step you logged in as an admin user. Only admin users can perform user management tasks, such as creating and deleting users and changing the passwords of other users.

```
aws-cloudhsm > user create --username example_user --role crypto-user
Enter password:
Confirm password:
{
 "error_code": 0,
 "data": {
   "username": "example_user",
   "role": "crypto-user"
 }
}
```

5. Use **user list** to list all the users on the cluster.

```
aws-cloudhsm > user list
```

```
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      },
      {
        "username": "example_user",
        "role": "crypto_user",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      },
      {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```


6. Use the **logout** command to log out of AWS CloudHSM cluster.

```
aws-cloudhsm > logout
{
  "error_code": 0,
  "data": "Logout successful"
}
```

7. Use the **quit** command to stop the CLI.

```
aws-cloudhsm > quit
```

Command modes in CloudHSM CLI

In CloudHSM CLI, you can run commands two different ways: in single command mode and interactive mode. Interactive mode is designed for users, and single command mode is designed for scripts.

> **Note**
>
> All commands work in interactive mode and single command mode.

Interactive mode

Use the following commands to start CloudHSM CLI interactive mode

Linux

```
$ /opt/cloudhsm/bin/cloudhsm-cli interactive
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive
```

When using the CLI in Interactive Mode, you can log in to a user account using the **login** command.

```
aws-cloudhsm > login --username <USERNAME> --role ROLE>
```

To list all CloudHSM CLI commands, run the following command:

```
aws-cloudhsm > help
```

To get the syntax for a CloudHSM CLI command, run the following command:

```
aws-cloudhsm >  help <command-name>
```

To get a list of users on the HSMs, enter **user list**.

```
aws-cloudhsm > user list
```

To end your CloudHSM CLI session, run the following command:

```
aws-cloudhsm > quit
```

Single Command mode

If you run CloudHSM CLI using Single Command Mode, you need to set two environment variables to provide credentials: CLOUDHSM_PIN and CLOUDHSM_ROLE:

```
$ export CLOUDHSM_ROLE=admin
```

```
$ export CLOUDHSM_PIN=admin_username:admin_password
```

After doing this, you can execute commands using the credentials stored in your environment.

```
$ cloudhsm-cli user change-password --username alice --role crypto-user
Enter password:
Confirm password:
{
    "error_code": 0,
    "data": {
      "username": "alice",
      "role": "crypto-user"
    }
}
```

Key attributes for CloudHSM CLI

This topic describes how to use CloudHSM CLI to set key attributes. A key attribute in CloudHSM CLI can define a key's type, how a key can function, or how a key is labeled. Some attributes define unique characteristics (a key's type, for example). Other attributes can be set to true or false—changing them either activates or deactivates a part of the key's functionality.

For examples showing how to use key attributes, see the commands listed under the parent command The key category in CloudHSM CLI.

The following topics provide additional detail about key attributes in CloudHSM CLI.

Topics

- Check value in CloudHSM CLI
- Related topics for CloudHSM CLI

Supported attributes for CloudHSM CLI

As a best practice, only set values for attributes you wish to make restrictive. If you don't specify a value, CloudHSM CLI uses the default value specified in the table below.

The following table lists the key attributes, possible values, defaults, and related notes for CloudHSM CLI. An empty cell in the **Value** column indicates that there is no specific default value assigned to the attribute.

CloudHSM CLI attribute	Value	Modifiable with key set-attribute	Settable at key creation
`always-sensitive`	The value is `True` if `sensitive` has always been set to `True` and has never changed.	No	No
`check-value`	The check value of the key. For more information, see Additional Details.	No	No
`class`	Possible values: `secret-key` , `public-key` , and `private-key` .	No	Yes
`curve`	Elliptic curve used to generate the EC key pair. Valid Values: `secp224r1` , `secp256r1` ,	No	Settable with EC, not settable with RSA

CloudHSM CLI attribute	Value	Modifiable with [key set-attribute](#)	Settable at key creation
	`prime256v1` , `secp384r1` , `secp256k1` , and `secp521r1`		
`decrypt`	Default: `False`	Yes	Yes
`derive`	Default: `False`	Derive can be set on hsm2m.medium instances. It cannot be set for RSA keys on hsm1.medium instances.	Yes
`destroyable`	Default: `True`	Yes	Yes
`ec-point`	For EC keys, DER-encoding of ANSI X9.62 ECPoint value "Q" in a hexadecimal format. For other key types, this attribute does not exist.	No	No
`encrypt`	Default: `False`	Yes	Yes
`extractable`	Default: `True`	No	Yes
`id`	Default: Empty	id can be set on hsm2m.medium instances. It cannot be set on hsm1.medi um instances.	Yes

CloudHSM CLI attribute	Value	Modifiable with [key set-attribute](#)	Settable at key creation
`key-length-bytes`	Required for generating an AES key. Valid values: 16, 24, and 32 bytes.	No	No
`key-type`	Possible values: `aes`, `rsa`, and `ec`	No	Yes
`label`	Default: Empty	Yes	Yes
`local`	Default: `True` for keys generated in the HSM, `False` for keys imported into the HSM.	No	No
`modifiable`	Default: `True`	Can be changed from true to false, but not from false to true.	Yes
`modulus`	The modulus that was used to generate an RSA key pair. For other key types, this attribute does not exist.	No	No
`modulus-size-bits`	Required for generating an RSA key pair. Minimum value is `2048`.	No	Settable with RSA, not settable with EC

CloudHSM CLI attribute	Value	Modifiable with [key set-attribute](#)	Settable at key creation
`never-extractable`	The value is `True` if extractable has never been set to `False`. The value is `False` if extractable has ever been set to `True`.	No	No
`private`	Default: `True`	No	Yes
`public-exponent`	Required for generating an RSA key pair. Valid values: The value must be an odd number greater than or equal to `65537`.	No	Settable with RSA, not settable with EC
`sensitive`	Default: • The value is `True` for AES keys and EC and RSA private keys. • The value is `False` for EC and RSA public keys.	No	Settable with private keys, not settable with public keys.

CloudHSM CLI attribute	Value	Modifiable with [key set-attribute]()	Settable at key creation
`sign`	Default: • The value is `True` for AES keys. • The value is `False` for RSA and EC keys.	Yes	Yes
`token`	Default: `True`	Can be changed from false to true, but not from true to false.	Yes
`trusted`	Default: `False`	Only admin users can set this parameter.	No
`unwrap`	Default: `False`	Yes	Yes, except for public keys.
`unwrap-template`	Values should use the attribute template applied to any key unwrapped using this wrapping key.	Yes	No
`verify`	Default: • The value is `True` for AES keys. • The value is `False` for RSA and EC keys.	Yes	Yes
`wrap`	Default: `False`	Yes	Yes, except for private keys.

CloudHSM CLI attribute	Value	Modifiable with key set-attribute	Settable at key creation
`wrap-template`	Values should use the attribute template to match the key wrapped using this wrapping key.	Yes	No
`wrap-with-trusted`	Default: `False`	Yes	Yes

Check value in CloudHSM CLI

The *check value* in CloudHSM CLI is a 3-byte hash or checksum of a key that is generated when the HSM imports or generates a key. You can also calculate a check value outside of the HSM, such as after you export a key. You can then compare the check value values to confirm the identity and integrity of the key. To get the check value of a key, use key list with the verbose flag.

AWS CloudHSM uses the following standard methods to generate a check value:

- **Symmetric keys**: First 3 bytes of the result of encrypting a zero-block with the key.
- **Asymmetric key pairs**: First 3 bytes of the SHA-1 hash of the public key.
- **HMAC keys**: KCV for HMAC keys is not supported at this time.

Related topics for CloudHSM CLI

See the following topics for more information about CloudHSM CLI.

- The key category in CloudHSM CLI
- Reference for CloudHSM CLI commands

Advanced configurations for CloudHSM CLI

The AWS CloudHSM Command Line Interface (CLI) includes the following advanced configuration, which is not part of the general configurations most customers utilize. These configurations provide additional capabilities.

- Connecting to multiple clusters

Connecting to multiple clusters with CloudHSM CLI

With AWS CloudHSM Client SDK 5, you can configure CloudHSM CLI to allow connections to multiple CloudHSM clusters from a single CLI instance.

The following topics describe how to use the CloudHSM CLI multi-cluster functionality to connect with multiple clusters.

Topics

Multi-cluster prerequisites for AWS CloudHSM

Before configuring your cluster in AWS CloudHSM to connect to multiple clusters, you must meet the following prerequisites:

- Two or more AWS CloudHSM clusters to which you'd like to connect to, along with their cluster certificates.
- An EC2 instance with Security Groups correctly configured to connect to all of the clusters above. For more information about how to set up a cluster and the client instance, refer to Getting started with AWS CloudHSM.
- To set up multi-cluster functionality, you must have already downloaded and installed the CloudHSM CLI. If you have not already done this, refer to the instructions in ???.
- You will not be able to access a cluster configured with `./configure-cli[.exe] -a` since it will not be associated with a `cluster-id`. You can reconfigure it by following `config-cli add-cluster` as described in this guide.

Configure the CloudHSM CLI for multi-cluster functionality

To configure your CloudHSM CLI for multi-cluster functionality, follow these steps:

1. Identify the clusters you want to connect to.
2. Add these clusters to your CloudHSM CLI configuration using the configure-cli subcommand `add-cluster` as described below.
3. Restart any CloudHSM CLI processes in order for the new configuration to take effect.

Add a cluster to your AWS CloudHSM configuration

When connecting to multiple clusters, use the `configure-cli add-cluster` command to add a cluster to your configuration.

Syntax

```
configure-cli add-cluster [OPTIONS]
        --cluster-id <CLUSTER ID>
        [--region <REGION>]
        [--endpoint <ENDPOINT>]
        [--hsm-ca-cert <HSM CA CERTIFICATE FILE>]
        [--client-cert-hsm-tls-file <CLIENT CERTIFICATE FILE>]
        [--client-key-hsm-tls-file <CLIENT KEY FILE>]
        [-h, --help]
```

Examples

Add a cluster using the `cluster-id` parameter

Example

Use the `configure-cli add-cluster` along with the `cluster-id` parameter to add a cluster (with the ID of `cluster-1234567`) to your configuration.

Linux

```
$ sudo /opt/cloudhsm/bin/configure-cli add-cluster --cluster-id <cluster-1234567>
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" add-cluster --
cluster-id <cluster-1234567>
```

> **Tip**
>
> If using `configure-cli add-cluster` with the `cluster-id` parameter doesn't result in the cluster being added, refer to the following example for a longer version of this command that also requires `--region` and `--endpoint` parameters to identify the cluster being added. If, for example, the region of the cluster is different than the one configured as your AWS CLI default, you should use the `--region` parameter to use the correct region. Additionally, you have the ability to specify the AWS CloudHSM API endpoint to use for the call, which may be necessary for various network setups, such as using VPC interface endpoints that don't use the default DNS hostname for AWS CloudHSM.

Add a cluster using `cluster-id`, `endpoint`, and `region` parameters

Example

Use the `configure-cli add-cluster` along with the `cluster-id`, `endpoint`, and `region` parameters to add a cluster (with the ID of `cluster-1234567`) to your configuration.

Linux

```
$ sudo /opt/cloudhsm/bin/configure-cli add-cluster --cluster-id <cluster-1234567> --
region <us-east-1> --endpoint <https://cloudhsmv2.us-east-1.amazonaws.com>
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" add-cluster --
cluster-id <cluster-1234567> --region <us-east-1> --endpoint <https://cloudhsmv2.us-
east-1.amazonaws.com>
```

For more information about the `--cluster-id`, `--region`, and `--endpoint` parameters, see the section called “Parameters”.

Parameters

--cluster-id ***<Cluster ID>***

Makes a `DescribeClusters` call to find all of the HSM elastic network interface (ENI) IP addresses in the cluster associated with the cluster ID. The system adds the ENI IP addresses to the AWS CloudHSM configuration files.

> **Note**
>
> If you use the `--cluster-id` parameter from an EC2 instance within a VPC that does not have access to the public internet, then you must create an interface VPC endpoint to connect with AWS CloudHSM. For more information about VPC endpoints, see ???.

Required: Yes

--endpoint ***<Endpoint>***

Specify the AWS CloudHSM API endpoint used for making the `DescribeClusters` call. You must set this option in combination with `--cluster-id`.

Required: No

--hsm-ca-cert ***<HsmCA Certificate Filepath>***

Specifies the filepath to the HSM CA certificate.

Required: No

--region ***<Region>***

Specify the region of your cluster. You must set this option in combination with `--cluster-id`.

If you don't supply the `--region` parameter, the system chooses the region by attempting to read the `AWS_DEFAULT_REGION` or `AWS_REGION` environment variables. If those variables aren't set, then the system checks the region associated with your profile in your AWS config file (typically `~/.aws/config`) unless you specified a different file in the `AWS_CONFIG_FILE` environment variable. If none of the above are set, the system defaults to the `us-east-1` region.

Required: No

--client-cert-hsm-tls-file ***<client certificate hsm tls path>***

Path to the client certificate used for TLS client-HSM mutual authentication.

Only use this option if you have registered at least one trust anchor onto HSM with CloudHSM CLI. You must set this option in combination with `--client-key-hsm-tls-file`.

Required: No

--client-key-hsm-tls-file ***<client key hsm tls path>***

Path to the client key used for TLS client-HSM mutual authentication.

Only use this option if you have registered at least one trust anchor onto HSM with CloudHSM CLI. You must set this option in combination with `--client-cert-hsm-tls-file`.

Required: No

Remove a cluster from your AWS CloudHSM configuration

When connecting to multiple clusters with CloudHSM CLI, use the `configure-cli remove-cluster` command to remove a cluster from your configuration.

Syntax

```
configure-cli remove-cluster [OPTIONS]
        --cluster-id <CLUSTER ID>
        [-h, --help]
```

Examples

Remove a cluster using the `cluster-id` parameter

Example

Use the `configure-cli remove-cluster` along with the `cluster-id` parameter to remove a cluster (with the ID of `cluster-1234567`) from your configuration.

Linux

```
$ sudo /opt/cloudhsm/bin/configure-cli remove-cluster --cluster-id <cluster-1234567>
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-cli.exe" remove-cluster --
cluster-id <cluster-1234567>
```

For more information about the `--cluster-id` parameter, see the section called “Parameters”.

Parameter

--cluster-id *<Cluster ID>*

The ID of the cluster to remove from the configuration.

Required: Yes

Interact with multiple clusters in AWS CloudHSM

After configuring multiple clusters with CloudHSM CLI, use the `cloudhsm-cli` command to interact with them.

Examples

Setting a default `cluster-id` when using interactive mode

Example

Use the ??? along with the `cluster-id` parameter to set a default cluster (with the ID of `cluster-1234567`) from your configuration.

Linux

```
$ cloudhsm-cli interactive --cluster-id <cluster-1234567>
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" interactive --
cluster-id <cluster-1234567>
```

Setting the `cluster-id` when running a single command

Example

Use the `cluster-id` parameter to set the cluster (with the ID of `cluster-1234567`) to get ??? from.

Linux

```
$ cloudhsm-cli cluster hsm-info --cluster-id <cluster-1234567>
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" cluster hsm-info
 --cluster-id <cluster-1234567>
```

Reference for CloudHSM CLI commands

CloudHSM CLI helps admins manage users in their AWS CloudHSM cluster. CloudHSM CLI can be run in two modes: Interactive Mode and Single Command Mode. For a quick start, see Getting started with AWS CloudHSM Command Line Interface (CLI).

To run most CloudHSM CLI commands, you must start the CloudHSM CLI and log in to the HSM. If you add or delete HSMs, update the configuration files for CloudHSM CLI. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

The following topics describe commands in CloudHSM CLI:

Command	Description	User Type
activate	Activates an CloudHSM cluster and provides confirmation the cluster is new. This must be done before any other operations can be performed.	Unactivated admin

Command	Description	User Type
hsm-info	List the HSMs in your cluster.	All [1], including unauthenticated users. Login is not required.
ECDSA	Generates a signature using an EC private key and the ECDSA signing mechanism.	Crypto users (CU)
rsa-pkcs	Generates a signature using an RSA private key and the RSA-PKCS signing mechanism.	CU
rsa-pkcs-pss	Generates a signature using an RSA private key and the RSA-PKCS-PSS signing mechanism.	CU
ecdsa	Confirms a file has been signed in the HSM by a given public key. Verifies the signature was generated using the ECDSA signing mechanism. Compares a signed file against a source file and determine whether the two are cryptographically related based on a given ecdsa public key and signing mechanism.	CU

Command	Description	User Type
rsa-pkcs	Confirms a file has been signed in the HSM by a given public key. Verifies the signature was generated using the RSA-PKCS signing mechanism. Compares a signed file against a source file and determines whether the two are cryptographically related based on a given rsa public key and signing mechanism.	CU
rsa-pkcs-pss	Confirms a file has been signed in the HSM by a given public key. Verifies the signature was generated using the RSA-PKCS-PSS signing mechanism . Compares a signed file against a source file and determines whether the two are cryptographically related based on a given rsa public key and signing mechanism.	CU
key delete	Deletes a key from your AWS CloudHSM cluster.	CU
key generate-file	Generates a key file in your AWS CloudHSM cluster.	CU
key generate-asymmetric-pair rsa	Generates an asymmetric RSA key pair in your AWS CloudHSM cluster.	CU

Command	Description	User Type
key generate-asymmetric-pair ec	Generates an asymmetric Elliptic-curve (EC) key pair in your AWS CloudHSM cluster.	CU
key generate-symmetric aes	Generates a symmetric AES key in your AWS CloudHSM cluster.	CU
key generate-symmetric generic-secret	Generates a symmetric Generic Secret key in your AWS CloudHSM cluster.	CU
key import pem	Imports a PEM format key into an HSM. You can use it to import public keys that were generated outside of the HSM.	CU
key list	Finds all keys for the current user present in your AWS CloudHSM cluster.	CU
key replicate	Replicate a key from a source cluster to a cloned destinati on cluster.	CU
key set-attribute	Sets the attributes of keys in your AWS CloudHSM cluster.	CUs can run this command, admins can set the trusted attribute.
key share	Shares a key with other CUs in your AWS CloudHSM cluster.	CU
key unshare	Unshares a key with other CUs in your AWS CloudHSM cluster.	CU

Command	Description	User Type
[aes-gcm]()	Unwraps a payload key into the cluster using the AES wrapping key and the AES-GCM unwrapping mechanism.	CU
[aes-no-pad]()	Unwraps a payload key into the cluster using the AES wrapping key and the AES-NO-PAD unwrapping mechanism.	CU
[aes-pkcs5-pad]()	Unwraps a payload key using the AES wrapping key and the AES-PKCS5-PAD unwrapping mechanism.	CU
[aes-zero-pad]()	Unwraps a payload key into the cluster using the AES wrapping key and the AES-ZERO-PAD unwrapping mechanism.	CU
[cloudhsm-aes-gcm]()	Unwraps a payload key into the cluster using the AES wrapping key and the CLOUDHSM-AES-GCM unwrapping mechanism.	CU
[rsa-aes]()	Unwraps a payload key using an RSA private key and the RSA-AES unwrapping mechanism.	CU

Command	Description	User Type
rsa-oaep	Unwraps a payload key using the RSA private key and the RSA-OAEP unwrapping mechanism.	CU
rsa-pkcs	Unwraps a payload key using the RSA private key and the RSA-PKCS unwrapping mechanism.	CU
aes-gcm	Wraps a payload key using an AES key on the HSM and the AES-GCM wrapping mechanism.	CU
aes-no-pad	Wraps a payload key using an AES key on the HSM and the AES-NO-PAD wrapping mechanism.	CU
aes-pkcs5-pad	Wraps a payload key using an AES key on the HSM and the AES-PKCS5-PAD wrapping mechanism.	CU
aes-zero-pad	Wraps a payload key using an AES key on the HSM and the AES-ZERO-PAD wrapping mechanism.	CU
cloudhsm-aes-gcm	Wraps a payload key using an AES key on the HSM and the CLOUDHSM-AES-GCM wrapping mechanism.	CUs

Command	Description	User Type
rsa-aes	Wraps a payload key using an RSA public key on the HSM and the RSA-AES wrapping mechanism.	CU
rsa-oaep	Wraps a payload key using an RSA public key on the HSM and the RSA-OAEP wrapping mechanism.	CU

Command	Description	User Type
Use the **key wrap rsa-pkcs** command in CloudHSM CLI to wrap a payload key using an RSA public key on the hardware security module (HSM) and the `RSA-PKCS` wrapping mechanism. The payload key's `extractable` attribute must be set to `true`. Only the owner of a key, that is the crypto user (CU) who created the key, can wrap the key. Users who share the key can use the key in cryptographic operations. To use the **key wrap rsa-pkcs** command, you must first have an RSA key in your AWS CloudHSM cluster. You can generate an RSA key pair using the The generate-asymmetric-pair category in CloudHSM CLI command and the `wrap` attribute set to `true`. **User type** The following types of users can run this command. • Crypto users (CUs) **Requirements** • To run this command, you must be logged in as a CU. **Syntax**	Wraps a payload key using an RSA public key on the HSM and the RSA-PKCS wrapping mechanism.	CU

Command	Description	User Type
login	Log in to your AWS CloudHSM cluster.	Admin, crypto user (CU), and appliance user (AU)
logout	Log out of your AWS CloudHSM cluster.	Admin, CU, and appliance user (AU)
quorum token-sign delete	Deletes one or more tokens for a quorum authorized service.	Admin
quorum token-sign generate	Generates a token for a quorum authorized service.	Admin
quorum token-sign list	Lists all token-sign quorum tokens present in your CloudHSM cluster.	All [1], including unauthent icated users. Login is not required.
quorum token-sign list-quor um-values	Lists the quorum values set in your CloudHSM cluster.	All [1], including unauthent icated users. Login is not required.
quorum token-sign set-quoru m-value	Sets a new quorum value for a quorum authorized service.	Admin
user change-mfa	Changes a user's multi-factor authentication (MFA) strategy.	Admin, CU
user change-password	Changes the passwords of users on the HSMs. Any user can change their own password. Admins can change anyone's password.	Admin, CU
user create	Creates a user in your AWS CloudHSM cluster.	Admin

Command	Description	User Type
user delete	Deletes a user in your AWS CloudHSM cluster.	Admin
user list	Lists the users in your AWS CloudHSM cluster.	All [1], including unauthent icated users. Login is not required.
user change-quorum token-sign register	Registers the quorum token-sign quorum strategy for a user.	Admin

Annotations

- [1] All users includes all listed roles and users not logged in.

The cluster category in CloudHSM CLI

In the CloudHSM CLI, **cluster** is a parent category for a group of commands that, when combined with the parent category, create a command specific to clusters. Currently, the cluster category consists of the following commands:

Topics

Activate a cluster with CloudHSM CLI

Use the **cluster activate** command in CloudHSM CLI to activate a new cluster in AWS CloudHSM. This command must be run before the cluster can be used to perform cryptographic operations.

User type

The following types of users can run this command.

- Unactivated admin

Syntax

This command has no parameters.

```
aws-cloudhsm > help cluster activate
Activate a cluster

This command will set the initial Admin password. This process will cause your CloudHSM
 cluster to
move into the ACTIVE state.

USAGE:
    cloudhsm-cli cluster activate [OPTIONS] [--password <PASSWORD>]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --password <PASSWORD>
          Optional: Plaintext activation password If you do not include this argument
 you will be prompted for it

  -h, --help
          Print help (see a summary with '-h')
```

Example

This command activates your cluster by setting the initial password for you admin user.

```
aws-cloudhsm > cluster activate
Enter password:
Confirm password:
{
  "error_code": 0,
  "data": "Cluster activation successful"
}
```

Related topics

- user create
- user delete

- user change-password

List HSMs with CloudHSM CLI

Use the **cluster hsm-info** command in CloudHSM CLI to list the hardware security modules (HSMs) in your AWS CloudHSM cluster. You do not need to be logged in to CloudHSM CLI to run this command.

> **Note**
>
> If you add or delete HSMs, update the configuration files that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective on all HSMs in the cluster.

User type

The following types of users can run this command.

- All users. You do not need to be logged in to run this command.

Syntax

```
aws-cloudhsm > help cluster hsm-info
List info about each HSM in the cluster

Usage: cloudhsm-cli cluster hsm-info [OPTIONS]

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
  -h, --help                     Print help
```

Example

This command lists the HSMs present in your AWS CloudHSM cluster.

```
aws-cloudhsm > cluster hsm-info
```

```
{
  "error_code": 0,
  "data": {
    "hsms": [
      {
        "vendor": "Marvell Semiconductors, Inc.",
        "model": "NITROX-III CNN35XX-NFBE",
        "serial-number": "5.3G1941-ICM000590",
        "hardware-version-major": "5",
        "hardware-version-minor": "3",
        "firmware-version-major": "2",
        "firmware-version-minor": "6",
        "firmware-build-number": "16",
        "firmware-id": "CNN35XX-NFBE-FW-2.06-16"
        "fips-state": "2 [FIPS mode with single factor authentication]"
      },
      {
        "vendor": "Marvell Semiconductors, Inc.",
        "model": "NITROX-III CNN35XX-NFBE",
        "serial-number": "5.3G1941-ICM000625",
        "hardware-version-major": "5",
        "hardware-version-minor": "3",
        "firmware-version-major": "2",
        "firmware-version-minor": "6",
        "firmware-build-number": "16",
        "firmware-id": "CNN35XX-NFBE-FW-2.06-16"
        "fips-state": "2 [FIPS mode with single factor authentication]"
      },
      {
        "vendor": "Marvell Semiconductors, Inc.",
        "model": "NITROX-III CNN35XX-NFBE",
        "serial-number": "5.3G1941-ICM000663",
        "hardware-version-major": "5",
        "hardware-version-minor": "3",
        "firmware-version-major": "2",
        "firmware-version-minor": "6",
        "firmware-build-number": "16",
        "firmware-id": "CNN35XX-NFBE-FW-2.06-16"
        "fips-state": "2 [FIPS mode with single factor authentication]"
      }
    ]
  }
}
```

The output has the following attributes:

- **Vendor**: The vendor name of the HSM.
- **Model**: The model number of the HSM.
- **Serial-number**: The serial number of the HSM. This may change due to replacements.
- **Hardware-version-major**: The major hardware version.
- **Hardware-version-minor**: The minor hardware version.
- **Firmware-version-major**: The major firmware version.
- **Firmware-version-minor**: The minor firmware version.
- **Firmware-build-number**: The firmware build number.
- **Firmware-id**: The firmware ID, which includes the major and minor versions along with the build.
- **FIPS-state**: The FIPS mode the cluster and the HSMs in it. If in FIPS mode, the output is "2 [FIPS mode with single factor authentication]." If in non-FIPS mode, the output is "0 [non-FIPS mode with single factor authentication]".

Related topics

- Activate a cluster with CloudHSM CLI

The cluster mtls category in CloudHSM CLI

In CloudHSM CLI, **cluster mtls** is a parent category for a group of commands that, when combined with the parent category, create a command specific to AWS CloudHSM clusters. Currently, this category consists of the following commands:

Topics

- Deregister a trust anchor with CloudHSM CLI
- Get the mTLS enforcement level with CloudHSM CLI
- List trust anchors with CloudHSM CLI
- Register a trust anchor with CloudHSM CLI
- Set the mTLS enforcement level with CloudHSM CLI

Deregister a trust anchor with CloudHSM CLI

Use the **cluster mtls deregister-trust-anchor** command in CloudHSM CLI to deregister a trust anchor for mutual TLS between client and AWS CloudHSM.

User type

The following users can run this command.

- Admin

Requirements

- To run this command, you must be logged in as a admin user.

Syntax

```
aws-cloudhsm > help cluster mtls deregister-trust-anchor

Deregister a trust anchor for mtls

Usage: cluster mtls deregister-trust-anchor [OPTIONS] --certificate-reference
 [<CERTIFICATE_REFERENCE>...]

Options:
      --certificate-reference <CERTIFICATE_REFERENCE>  A hexadecimal or decimal
 certificate reference
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
      --approval <APPROVAL>  Filepath of signed quorum token file to approve operation
  -h, --help                     Print help
```

Example

Example

In the following example, this command removes a trust anchor from the HSM.

```
aws-cloudhsm > cluster mtls deregister-trust-anchor --certificate-reference 0x01
```

```
{
  "error_code": 0,
  "data": {
    "message": "Trust anchor with reference 0x01 deregistered successfully"
  }
}
```

You can then run the **list-trust-anchors** command to confirm that trust anchor has been deregistered from the AWS CloudHSM:

```
aws-cloudhsm > cluster mtls list-trust-anchors

{
  "error_code": 0,
  "data": {
    "trust_anchors": []
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<CERTIFICATE_REFERENCE>

A hexadecimal or decimal certificate reference.

Required: Yes

> **Warning**
>
> After you deregister a trust anchor in the cluster, all existing mTLS connections using the client certificate signed by that trust anchor will be dropped.

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if quorum cluster service quorum value is greater than 1.

Related topics

- cluster mtls reregister-trust-anchor
- cluster mtls list-trust-anchors
- Setup mTLS (recommended)

Get the mTLS enforcement level with CloudHSM CLI

Use the **cluster mtls get-enforcement** command in CloudHSM CLI to get the enforcement level of the usage of mutual TLS between client and AWS CloudHSM.

User type

The following users can run this command.

- Admin
- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a admin user or crypto user (CUs).

Syntax

```
aws-cloudhsm > help cluster mtls get-enforcement

Get the status of mtls enforcement in the cluster

Usage: cluster mtls get-enforcement [OPTIONS]

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
```

```
  -h, --help                          Print help
```

Example

Example

In the following example, this command lists the mtls enforcement level of the AWS CloudHSM.

```
aws-cloudhsm > cluster mtls get-enforcement

{
  "error_code": 0,
  "data": {
    "mtls-enforcement-level": "none"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

Related topics

- cluster mtls set-enforcement
- Setup mTLS (recommended)

List trust anchors with CloudHSM CLI

Use the **cluster mtls list-trust-anchors** command in CloudHSM CLI to list all the trust anchors which can be used for mutual TLS between client and AWS CloudHSM.

User type

The following users can run this command.

- All users. You do not need to be logged in to run this command.

Syntax

```
aws-cloudhsm > help cluster mtls list-trust-anchors

List all trust anchors for mtls

Usage: cluster mtls list-trust-anchors [OPTIONS]

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
  -h, --help                     Print help
```

Example

Example

In the following example, this command lists all the registered trust anchors from the AWS CloudHSM.

```
aws-cloudhsm > cluster mtls list-trust-anchors

{
  "error_code": 0,
  "data": {
    "trust_anchors": [
      {
        "certificate-reference": "0x01",
        "certificate": "<PEM Encoded Certificate 1>",
        "cluster-coverage": "full"
      },
      {
        "certificate-reference": "0x02",
        "certificate": "<PEM Encoded Certificate 2>",
        "cluster-coverage": "full"
      }
    ]
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

Related topics

- cluster mtls reregister-trust-anchor
- cluster mtls deregister-trust-anchor
- Setup mTLS (recommended)

Register a trust anchor with CloudHSM CLI

Use the **cluster mtls register-trust-anchor** command in CloudHSM CLI to register a trust anchor for mutual TLS between client and AWS CloudHSM.

User type

The following users can run this command.

- Admin

Requirements

The AWS CloudHSM accepts trust anchors with the following key types:

Key Type	Description
EC	secp256r1 (P-256), secp384r1 (P-384), and secp521r1 (P-521) curves.
RSA	2048-bit, 3072-bit, and 4096-bit RSA keys.

Syntax

```
aws-cloudhsm > help cluster mtls register-trust-anchor
```

```
Register a trust anchor for mtls

Usage: cluster mtls register-trust-anchor [OPTIONS] --path [<PATH>...]

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
      --path <PATH>  Filepath of the trust anchor to register
      --approval <APPROVAL>  Filepath of signed quorum token file to approve operation
  -h, --help                     Print help
```

Example

Example

In the following example, this command registers a trust anchor onto the HSM. The maximum number of trust anchors can be registered is two (2).

```
aws-cloudhsm > cluster mtls register-trust-anchor --path /home/rootCA

{
  "error_code": 0,
  "data": {
    "trust_anchor": {
      "certificate-reference": "0x01",
      "certificate": "<PEM Encoded Certificate>",
      "cluster-coverage": "full"
    }
  }
}
```

You can then run the **list-trust-anchors** command to confirm that trust anchor has been registered onto the AWS CloudHSM:

```
aws-cloudhsm > cluster mtls list-trust-anchors

{
  "error_code": 0,
  "data": {
```

```
    "trust_anchors": [
      {
        "certificate-reference": "0x01",
        "certificate": "<PEM Encoded Certificate>",
        "cluster-coverage": "full"
      }
    ]
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<PATH>

Filepath of the trust anchor to register.

Required: Yes

> **Note**
>
> AWS CloudHSM supports registering intermediate certificates as trust anchor. In such cases, the entire PEM-encoded certificate chain file needs to be registered onto the HSM, with the certificates in hierarchical order.
> AWS CloudHSM supports a certificate chain of 6980 bytes.

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if quorum cluster service quorum value is greater than 1.

Related topics

- cluster mtls deregister-trust-anchor
- cluster mtls list-trust-anchors

- Setup mTLS (recommended)

Set the mTLS enforcement level with CloudHSM CLI

Use the **cluster mtls set-enforcement** command in CloudHSM CLI to set the enforcement level of the usage of mutual TLS between client and AWS CloudHSM.

User type

The following users can run this command.

- Admin with username as admin

Requirements

To run this command:

- At least one trust anchor has been successfully registered onto the AWS CloudHSM.
- Configure the CloudHSM CLI with the right private key and client certificate, and start CloudHSM CLI under a mutual TLS connection.
- You must be logged in as the default admin with username "admin". Any other admin user will not be able to run this command.

Syntax

```
aws-cloudhsm > help cluster mtls set-enforcement

Set mtls enforcement policy in the cluster

Usage: cluster mtls set-enforcement [OPTIONS] --level [<LEVEL>...]

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
      --level <LEVEL>  Level to be set for mtls in the cluster [possible values: none,
 cluster]
      --approval <APPROVAL>  Filepath of signed quorum token file to approve operation
  -h, --help                     Print help
```

Example

Example

In the following example, this command set the mtls enforcement level of the AWS CloudHSM to be cluster. The set-enforcement command can only be performed in a mutual TLS connection and logged in as the admin user with username as admin, see [set the mTLS enforcement for AWS CloudHSM](#).

```
aws-cloudhsm > cluster mtls set-enforcement --level cluster

{
  "error_code": 0,
  "data": {
    "message": "Mtls enforcement level set to Cluster successfully"
  }
}
```

You can then run the **get-enforcement** command to confirm that enforcement level has been set to cluster:

```
aws-cloudhsm > cluster mtls get-enforcement

{
  "error_code": 0,
  "data": {
    "mtls-enforcement-level": "cluster"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been [configured.](#)

<LEVEL>

Level to be set for mtls in the cluster.

Valid values

- **cluster**: Enforce the usage of mutual TLS between client and AWS CloudHSM in the cluster.
- **none**: Do not enforce the usage of mutual TLS between client and AWS CloudHSM in the cluster.

Required: Yes

> **Warning**
>
> After you enforce mTLS usage in the cluster, all existing non-mTLS connections will be dropped and you can only connect to the cluster with mTLS certificates.

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if quorum cluster service quorum value is greater than 1.

Related topics

- cluster mtls get-enforcement
- Setup mTLS (recommended)

The crypto category in CloudHSM CLI

In the CloudHSM CLI, **crypto** is a parent category for a group of commands that, when combined with the parent category, create a command specific to cryptographic operations. Currently, this category consists of the following commands:

- sign
  - ECDSA
  - rsa-pkcs
  - rsa-pkcs-pss
- verify
  - ecdsa
  - rsa-pkcs

- rsa-pkcs-pss

The crypto sign category in CloudHSM CLI

In the CloudHSM CLI, **crypto sign** is a parent category for a group of commands that, when combined with the parent category, uses a chosen private key in your AWS CloudHSM cluster to generate a signature. **crypto sign** has the following subcommands:

- Generate a signature with the ECDSA mechanism in CloudHSM CLI
- Generate a signature with the RSA-PKCS mechanism in CloudHSM CLI
- Generate a signature with the RSA-PKCS-PSS mechanism in CloudHSM CLI

To use **crypto sign**, you must have a private key in your HSM. You can generate a private key with the following commands:

- key generate-asymmetric-pair ec
- key generate-asymmetric-pair rsa

Generate a signature with the ECDSA mechanism in CloudHSM CLI

Use the **crypto sign ecdsa** command in CloudHSM CLI to generate a signature using an EC private key and the ECDSA signing mechanism.

To use the **crypto sign ecdsa** command, you must first have an EC private key in your AWS CloudHSM cluster. You can generate an EC private key using the Generate an asymmetric EC key pair with CloudHSM CLI command with the `sign` attribute set to `true`.

> **Note**
>
> Signatures can be verified in AWS CloudHSM with The crypto verify category in CloudHSM CLI subcommands.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help crypto sign ecdsa
Sign with the ECDSA mechanism

Usage: crypto sign ecdsa --key-filter [<KEY_FILTER>>...] --hash-
function <HASH_FUNCTION> <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --key-filter [<KEY_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key
      --hash-function <HASH_FUNCTION>
          [possible values: sha1, sha224, sha256, sha384, sha512]
      --data-path <DATA_PATH>
          The path to the file containing the data to be signed
      --data <DATA>
          Base64 Encoded data to be signed
      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation
      --data-type <DATA_TYPE>
          The type of data passed in, either raw or digest [possible values: raw,
 digest]
  -h, --help
          Print help
```

Example

These examples show how to use **crypto sign ecdsa** to generate a signature using the ECDSA signing mechanism and SHA256 hash function. This command uses a private key in the HSM.

Example Example: Generate a signature for base 64 encoded data

```
aws-cloudhsm > crypto sign ecdsa --key-filter attr.label=ec-private --hash-function
 sha256 --data YWJjMTIz
{
  "error_code": 0,
  "data": {
    "key-reference": "0x00000000007808dd",
    "signature": "4zki+FzjhP7Z/KqoQvh4ueMAxQQVp7FQguZ2wOS3Q5bzk
+Hc5irV5iTkuxQbropPttVFZ8V6FgR2fz+sPegwCw=="
  }
}
```

Example Example: Generate a signature for a data file

```
aws-cloudhsm > crypto sign ecdsa --key-filter attr.label=ec-private --hash-function
 sha256 --data-path data.txt
{
  "error_code": 0,
  "data": {
    "key-reference": "0x00000000007808dd",
    "signature": "4zki+FzjhP7Z/KqoQvh4ueMAxQQVp7FQguZ2wOS3Q5bzk
+Hc5irV5iTkuxQbropPttVFZ8V6FgR2fz+sPegwCw=="
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<DATA>

Base64 encoded data to be signed.

Required: Yes (unless provided through data path)

<DATA_PATH>

Specifies the location of the data to be signed.

Required: Yes (unless provided through data path)

<HASH_FUNCTION>

Specifies the hash function.

Valid values:

- sha1
- sha224
- sha256
- sha384
- sha512

Required: Yes

<KEY_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a matching key.

For a listing of supported CloudHSM CLI key attributes, see Key attributes for CloudHSM CLI.

Required: Yes

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key usage service quorum value of the private key is greater than 1.

<DATA_TYPE>

Specifies whether the value of the data parameter should be hashed as part of the signing algorithm. Use `raw` for unhashed data; use `digest` for digests, which are already hashed.

Valid values:

- raw
- digest

Related topics

- [The crypto sign category in CloudHSM CLI]
- [The crypto verify category in CloudHSM CLI]

Generate a signature with the RSA-PKCS mechanism in CloudHSM CLI

Use the **crypto sign rsa-pkcs** command in CloudHSM CLI to generate a signature using an RSA private key and the RSA-PKCS signing mechanism.

To use the **crypto sign rsa-pkcs** command, you must first have a RSA private key in your AWS CloudHSM cluster. You can generate an RSA private key using the Generate an asymmetric RSA key pair with CloudHSM CLI command with the `sign` attribute set to `true`.

> **Note**
>
> Signatures can be verified in AWS CloudHSM with The crypto verify category in CloudHSM CLI subcommands.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help crypto sign rsa-pkcs
Sign with the RSA-PKCS mechanism

Usage: crypto sign rsa-pkcs --key-filter [<KEY_FILTER>>...] --hash-
function <HASH_FUNCTION> <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --key-filter [<KEY_FILTER>...]
```

```
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key
      --hash-function <HASH_FUNCTION>
          [possible values: sha1, sha224, sha256, sha384, sha512]
      --data-path <DATA_PATH>
          The path to the file containing the data to be signed
      --data <DATA>
          Base64 Encoded data to be signed
      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation
      --data-type <DATA_TYPE>
          The type of data passed in, either raw or digest [possible values: raw,
 digest]
  -h, --help
          Print help
```

Example

These examples show how to use **crypto sign rsa-pkcs** to generate a signature using the RSA-PKCS signing mechanism and SHA256 hash function. This command uses a private key in the HSM.

Example Example: Generate a signature for base 64 encoded data

```
aws-cloudhsm > crypto sign rsa-pkcs --key-filter attr.label=rsa-private --hash-function
 sha256 --data YWJjMTIz
{
  "error_code": 0,
  "data": {
    "key-reference": "0x00000000007008db",
    "signature": "XJ7mRyHnDRYrDWTQuuNb
+5mhoXx7VTsPMjgOQW4iMN7E42eNHj2Q0oovMmBdHUEH0F4HYG8FBJOBhvGuM8J/
z6y41GbowVpUT6WzjnIQs79K9i7i6oR1TYjLnIS3r/zkimuXcS8/ZxyDzru+GO9BUT9FFU/
of9cvu4Oyn6a5+IXuCbKNQs19uASuFARUTZ0a0Ny1CB1MulxUpqGTmI91J6evlP7k/2khwDmJ5E8FEar5/
Cvbn9t21p3Uj561ngTXrYbIZ2KHpef9jQh/cEIvFLG61sexJjQi8EdTxeDA
+I3ITO0qrvvESvA9+Sj7kdG2ceIicFS8/8LwyxiIC31UHQ=="
  }
}
```

Example Example: Generate a signature for a data file

```
aws-cloudhsm > crypto sign rsa-pkcs --key-filter attr.label=rsa-private --hash-function
 sha256 --data-path data.txt
```

```
{
  "error_code": 0,
  "data": {
    "key-reference": "0x00000000007008db",
    "signature": "XJ7mRyHnDRYrDWTQuuNb
+5mhoXx7VTsPMjgOQW4iMN7E42eNHj2Q0oovMmBdHUEH0F4HYG8FBJOBhvGuM8J/
z6y41GbowVpUT6WzjnIQs79K9i7i6oR1TYjLnIS3r/zkimuXcS8/ZxyDzru+GO9BUT9FFU/
of9cvu4Oyn6a5+IXuCbKNQs19uASuFARUTZ0a0Ny1CB1MulxUpqGTmI91J6evlP7k/2khwDmJ5E8FEar5/
Cvbn9t21p3Uj561ngTXrYbIZ2KHpef9jQh/cEIvFLG61sexJjQi8EdTxeDA
+I3ITO0qrvvESvA9+Sj7kdG2ceIicFS8/8LwyxiIC31UHQ=="
  }
}
```


Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<DATA>

Base64 encoded data to be signed.

Required: Yes (unless provided through data path)

<DATA_PATH>

Specifies the location of the data to be signed.

Required: Yes (unless provided through data)

<HASH_FUNCTION>

Specifies the hash function.

Valid values:

- sha1
- sha224
- sha256
- sha384

- sha512

Required: Yes

<KEY_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a matching key.

For a listing of supported CloudHSM CLI key attributes, see Key attributes for CloudHSM CLI.

Required: Yes

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key usage service quorum value of the private key is greater than 1.

<DATA_TYPE>

Specifies whether the value of the data parameter should be hashed as part of the signing algorithm. Use `raw` for unhashed data; use `digest` for digests, which are already hashed.

For RSA-PKCS, the data must be passed in DER encoded format as specified in [RFC 8017, Section 9.2](#)

Valid values:

- raw
- digest

Related topics

- [The crypto sign category in CloudHSM CLI](#)
- [The crypto verify category in CloudHSM CLI](#)

Generate a signature with the RSA-PKCS-PSS mechanism in CloudHSM CLI

Use the **crypto sign rsa-pkcs-pss** command in CloudHSM CLI to generate a signature using an RSA private key and the `RSA-PKCS-PSS` signing mechanism.

To use the **crypto sign rsa-pkcs-pss** command, you must first have a RSA private key in your AWS CloudHSM cluster. You can generate an RSA private key using the Generate an asymmetric RSA key pair with CloudHSM CLI command with the `sign` attribute set to `true`.

> **Note**
>
> Signatures can be verified in AWS CloudHSM with The crypto verify category in CloudHSM CLI subcommands.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help crypto sign rsa-pkcs-pss
Sign with the RSA-PKCS-PSS mechanism

Usage: crypto sign rsa-pkcs-pss [OPTIONS] --key-filter [<KEY_FILTER>...] --
hash-function <HASH_FUNCTION> --mgf <MGF> --salt-length <SALT_LENGTH> <--data-
path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>        Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
      --key-filter [<KEY_FILTER>...]   Key reference (e.g. key-
reference=0xabc) or space separated list of key attributes in the form of
 attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a matching key
      --hash-function <HASH_FUNCTION>  [possible values: sha1, sha224, sha256, sha384,
 sha512]
      --data-path <DATA_PATH>          The path to the file containing the data to be
 signed
      --data <DATA>                    Base64 Encoded data to be signed
```

```
      --mgf <MGF>                        The mask generation function [possible values:
 mgf1-sha1, mgf1-sha224, mgf1-sha256, mgf1-sha384, mgf1-sha512]
      --salt-length <SALT_LENGTH>        The salt length
      --approval <APPROVAL>              Filepath of signed quorum token file to approve
 operation
      --data-type <DATA_TYPE>            The type of data passed in, either raw or digest
 [possible values: raw, digest]
  -h, --help                             Print help
```

Example

These examples show how to use **crypto sign rsa-pkcs-pss** to generate a signature using the RSA-PKCS-PSS signing mechanism and SHA256 hash function. This command uses a private key in the HSM.

Example Example: Generate a signature for base 64 encoded data

```
aws-cloudhsm > crypto sign rsa-pkcs-pss --key-filter attr.label=rsa-private --hash-function sha256 --data YWJjMTIz --salt-length 10 --mgf mgf1-sha256
{
  "error_code": 0,
  "data": {
    "key-reference": "0x00000000007008db",
    "signature": "H/z1rYVMzNAa31K4amE5MTiwGxDdCTgQXCJXRBKVOVm7ZuyI0fGE4sT/BUN
+977mQEV2TqtWpTsiF2IpwGM1VfSBRt7h/g4o6YERm1tTQLl7q+AJ7uGGK37zCsWQrAo7Vy8NzPShxekePo/
ZegrB1aHWN1fE8H3IPUKqLuMDI9o1Jq6kM986ExS7YmeOIclcZkyykTWqHLQVL2C3+A2bHJZBqRcM5XoIpk8HkPypjpN
+m4FNUds30GAemoOMl6asSrEJSthaZWV53OBsDOqzA8Rt8JdhXS+GZp3vNLdL1OTBELDPweXVgAu4dBX0FOvpw/
gg6sNvuaDK4YOBv2fqKg=="
  }
}
```

Example Example: Generate a signature for a data file

```
aws-cloudhsm > crypto sign rsa-pkcs-pss --key-filter attr.label=rsa-private --hash-function sha256 --data-path data.txt --salt-length 10 --mgf mgf1-sha256
{
  "error_code": 0,
  "data": {
    "key-reference": "0x00000000007008db",
    "signature": "H/z1rYVMzNAa31K4amE5MTiwGxDdCTgQXCJXRBKVOVm7ZuyI0fGE4sT/BUN
+977mQEV2TqtWpTsiF2IpwGM1VfSBRt7h/g4o6YERm1tTQLl7q+AJ7uGGK37zCsWQrAo7Vy8NzPShxekePo/
ZegrB1aHWN1fE8H3IPUKqLuMDI9o1Jq6kM986ExS7YmeOIclcZkyykTWqHLQVL2C3+A2bHJZBqRcM5XoIpk8HkPypjpN
```

```
+m4FNUds30GAemoOMl6asSrEJSthaZWV53OBsDOqzA8Rt8JdhXS+GZp3vNLdL1OTBELDPweXVgAu4dBX0FOvpw/
gg6sNvuaDK4YOBv2fqKg=="
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<DATA>

Base64 encoded data to be signed.

Required: Yes (unless provided through data path)

<DATA_PATH>

Specifies the location of the data to be signed.

Required: Yes (unless provided through data)

<HASH_FUNCTION>

Specifies the hash function.

Valid values:

- sha1
- sha224
- sha256
- sha384
- sha512

Required: Yes

<KEY_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a matching key.

For a listing of supported CloudHSM CLI key attributes, see Key attributes for CloudHSM CLI.

Required: Yes

<MGF>

Specifies the mask generation function.

> **Note**
>
> The mask generation function hash function must match the signing mechanism hash function.

Valid values:

- mgf1-sha1
- mgf1-sha224
- mgf1-sha256
- mgf1-sha384
- mgf1-sha512

Required: Yes

<SALT_LENGTH>

Specifies the salt length.

Required: Yes

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key usage service quorum value of the private key is greater than 1.

<DATA_TYPE>

Specifies whether the value of the data parameter should be hashed as part of the signing algorithm. Use `raw` for unhashed data; use `digest` for digests, which are already hashed.

Valid values:

- raw
- digest

Related topics

- The crypto sign category in CloudHSM CLI
- The crypto verify category in CloudHSM CLI

Related topics

- The crypto verify category in CloudHSM CLI

The crypto verify category in CloudHSM CLI

In the CloudHSM CLI, **crypto verify** is a parent category for a group of commands that, when combined with the parent category, confirms whether a file has been signed by a given key. **crypto verify** has the following subcommands:

- crypto verify ecdsa
- crypto verify rsa-pkcs
- crypto verify rsa-pkcs-pss

The **crypto verify** command compares a signed file against a source file and analyzes whether they are cryptographically related based on a given public key and signing mechanism.

> **Note**
>
> Files can be signed in AWS CloudHSM with the The crypto sign category in CloudHSM CLI operation.

Verify a signature signed with the ECDSA mechanism in CloudHSM CLI

Use the **crypto verify ecdsa** command in CloudHSM CLI to complete the following operations:

- Confirm a file has been signed in the HSM by a given public key.
- Verify the signature was generated using the ECDSA signing mechanism.
- Compare a signed file against a source file and determine whether the two are cryptographically related based on a given ecdsa public key and signing mechanism.

To use the **crypto verify ecdsa** command, you must first have an EC public key in your AWS CloudHSM cluster. You can import an EC public key using the Import a PEM format key with CloudHSM CLI command with the `verify` attribute set to `true`.

> **Note**
>
> You can generate a signature in CloudHSM CLI with The crypto sign category in CloudHSM CLI subcommands.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help crypto verify ecdsa
Verify with the ECDSA mechanism

Usage: crypto verify ecdsa --key-filter [<KEY_FILTER>...] --hash-
function <HASH_FUNCTION> <--data-path <DATA_PATH>|--data <DATA>> <--signature-
path <SIGNATURE_PATH>|--signature <SIGNATURE>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --key-filter [<KEY_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key
      --hash-function <HASH_FUNCTION>
          [possible values: sha1, sha224, sha256, sha384, sha512]
      --data-path <DATA_PATH>
```

```
            The path to the file containing the data to be verified
        --data <DATA>
            Base64 encoded data to be verified
        --signature-path <SIGNATURE_PATH>
            The path to where the signature is located
        --signature <SIGNATURE>
            Base64 encoded signature to be verified
        --data-type <DATA_TYPE>
            The type of data passed in, either raw or digest [possible values: raw,
 digest]
  -h, --help
            Print help
```

Example

These examples show how to use **crypto verify ecdsa** to verify a signature that was generated using the ECDSA signing mechanism and SHA256 hash function. This command uses a public key in the HSM.

Example Example: Verify a Base64 encoded signature with Base64 encoded data

```
aws-cloudhsm > crypto verify ecdsa --hash-function sha256 --key-filter attr.label=ec-
public --data YWJjMTIz --signature 4zki+FzjhP7Z/KqoQvh4ueMAxQQVp7FQguZ2wOS3Q5bzk
+Hc5irV5iTkuxQbropPttVFZ8V6FgR2fz+sPegwCw==
{
  "error_code": 0,
  "data": {
    "message": "Signature verified successfully"
  }
}
```

Example Example: Verify a signature file with a data file

```
aws-cloudhsm > crypto verify ecdsa --hash-function sha256 --key-filter attr.label=ec-
public --data-path data.txt --signature-path signature-file
{
   "error_code": 0,
  "data": {
    "message": "Signature verified successfully"
  }
}
```

Example Example: Prove false signing relationship

This command verifies whether the data located at /home/data was signed by a public key with the label ecdsa-public using the ECDSA signing mechanism to produce the signature located in /home/signature. Because the given arguments do not make up a true signing relationship, the command returns an error message.

```
aws-cloudhsm > crypto verify ecdsa --hash-function sha256 --
key-filter attr.label=ec-public --data aW52YWxpZA== --signature
 +ogk7M7S3iTqFg3SndJfd91dZFr5Qo6YixJl8JwcvqqVgsVuO6o+VKvTRjz0/V05kf3JJbBLr87Q
+wLWcMAJfA==
{
  "error_code": 1,
  "data": "Signature verification failed"
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<DATA>

Base64 encoded data to be signed.

Required: Yes (unless provided through data path)

<DATA_PATH>

Specifies the location of the data to be signed.

Required: Yes (unless provided through data path)

<HASH_FUNCTION>

Specifies the hash function.

Valid values:

- sha1

- sha224
- sha256
- sha384
- sha512

Required: Yes

<KEY_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a matching key.

For a listing of supported CloudHSM CLI key attributes, see Key attributes for CloudHSM CLI.

Required: Yes

<SIGNATURE>

Base64 encoded signature.

Required: Yes (unless provided through signature path)

<SIGNATURE_PATH>

Specifies the location of the signature.

Required: Yes (unless provided through signature path)

<DATA_TYPE>

Specifies whether the value of the data parameter should be hashed as part of the signing algorithm. Use `raw` for unhashed data; use `digest` for digests, which are already hashed.

Valid values:

- raw
- digest

Related topics

- The crypto sign category in CloudHSM CLI

- The crypto verify category in CloudHSM CLI

Verify a signature signed with the RSA-PKCS mechanism in CloudHSM CLI

Use the **crypto verify rsa-pkcs** command in CloudHSM CLI complete the following operations:

- Confirm a file has been signed in the HSM by a given public key.
- Verify the signature was generated using the RSA-PKCS signing mechanism.
- Compare a signed file against a source file and determines whether the two are cryptographically related based on a given rsa public key and signing mechanism.

To use the **crypto verify rsa-pkcs** command, you must first have an RSA public key in your AWS CloudHSM cluster.

> **Note**
>
> You can generate a signature using the CloudHSM CLI with the The crypto sign category in CloudHSM CLI subcommands.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help crypto verify rsa-pkcs
Verify with the RSA-PKCS mechanism

Usage: crypto verify rsa-pkcs --key-filter [<KEY_FILTER>...] --hash-
function <HASH_FUNCTION> <--data-path <DATA_PATH>|--data <DATA>> <--signature-
path <SIGNATURE_PATH>|--signature <SIGNATURE>>
```

```
Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --key-filter [<KEY_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key
      --hash-function <HASH_FUNCTION>
          [possible values: sha1, sha224, sha256, sha384, sha512]
      --data-path <DATA_PATH>
          The path to the file containing the data to be verified
      --data <DATA>
          Base64 encoded data to be verified
      --signature-path <SIGNATURE_PATH>
          The path to where the signature is located
      --signature <SIGNATURE>
          Base64 encoded signature to be verified
      --data-type <DATA_TYPE>
          The type of data passed in, either raw or digest [possible values: raw,
 digest]
  -h, --help
          Print help
```

Example

These examples show how to use **crypto verify rsa-pkcs** to verify a signature that was generated using the RSA-PKCS signing mechanism and SHA256 hash function. This command uses a public key in the HSM.

Example Example: Verify a Base64 encoded signature with Base64 encoded data

```
aws-cloudhsm > crypto verify rsa-pkcs --hash-function sha256 --key-filter
 attr.label=rsa-public --data YWJjMTIz --signature XJ7mRyHnDRYrDWTQuuNb
+5mhoXx7VTsPMjgOQW4iMN7E42eNHj2Q0oovMmBdHUEH0F4HYG8FBJOBhvGuM8J/
z6y41GbowVpUT6WzjnIQs79K9i7i6oR1TYjLnIS3r/zkimuXcS8/ZxyDzru+GO9BUT9FFU/
of9cvu4Oyn6a5+IXuCbKNQs19uASuFARUTZ0a0Ny1CB1MulxUpqGTmI91J6evlP7k/2khwDmJ5E8FEar5/
Cvbn9t21p3Uj561ngTXrYbIZ2KHpef9jQh/cEIvFLG61sexJjQi8EdTxeDA
+I3ITO0qrvvESvA9+Sj7kdG2ceIicFS8/8LwyxiIC31UHQ==
{
  "error_code": 0,
```

```
  "data": {
    "message": "Signature verified successfully"
  }
}
```

Example Example: Verify a signature file with a data file

```
aws-cloudhsm > crypto verify rsa-pkcs --hash-function sha256 --key-filter
 attr.label=rsa-public --data-path data.txt --signature-path signature-file
{
  "error_code": 0,
  "data": {
    "message": "Signature verified successfully"
  }
}
```

Example Example: Prove false signing relationship

This command verifies whether the invalid data was signed by a public key with the label `rsa-public` using the RSAPKCS signing mechanism to produce the signature located in `/home/signature`. Because the given arguments do not make up a true signing relationship, the command returns an error message.

```
aws-cloudhsm > crypto verify rsa-pkcs --hash-function sha256 --key-filter
 attr.label=rsa-public --data aW52YWxpZA== --signature XJ7mRyHnDRYrDWTQuuNb
+5mhoXx7VTsPMjgOQW4iMN7E42eNHj2Q0oovMmBdHUEH0F4HYG8FBJOBhvGuM8J/
z6y41GbowVpUT6WzjnIQs79K9i7i6oR1TYjLnIS3r/zkimuXcS8/ZxyDzru+GO9BUT9FFU/
of9cvu4Oyn6a5+IXuCbKNQs19uASuFARUTZ0a0Ny1CB1MulxUpqGTmI91J6evlP7k/2khwDmJ5E8FEar5/
Cvbn9t21p3Uj561ngTXrYbIZ2KHpef9jQh/cEIvFLG61sexJjQi8EdTxeDA
+I3ITO0qrvvESvA9+Sj7kdG2ceIicFS8/8LwyxiIC31UHQ==
{
  "error_code": 1,
  "data": "Signature verification failed"
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been [configured.](#)

<DATA>

Base64 encoded data to be signed.

Required: Yes (unless provided through data path)

<DATA_PATH>

Specifies the location of the data to be signed.

Required: Yes (unless provided through data path)

<HASH_FUNCTION>

Specifies the hash function.

Valid values:

- sha1
- sha224
- sha256
- sha384
- sha512

Required: Yes

<KEY_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a matching key.

For a listing of supported CloudHSM CLI key attributes, see Key attributes for CloudHSM CLI.

Required: Yes

<SIGNATURE>

Base64 encoded signature.

Required: Yes (unless provided through signature path)

<SIGNATURE_PATH>

Specifies the location of the signature.

Required: Yes (unless provided through signature path)

<DATA_TYPE>

Specifies whether the value of the data parameter should be hashed as part of the signing algorithm. Use `raw` for unhashed data; use `digest` for digests, which are already hashed.

For RSA-PKCS, the data must be passed in DER encoded format as specified in [RFC 8017, Section 9.2](#)

Valid values:

- raw
- digest

Related topics

- The crypto sign category in CloudHSM CLI
- The crypto verify category in CloudHSM CLI

Verify a signature signed with the RSA-PKCS-PSS mechanism in CloudHSM CLI

Use the **crypto sign rsa-pkcs-pss** command in CloudHSM CLI to complete the following operations.

- Confirm a file has been signed in the HSM by a given public key.
- Verify the signature was generated using the RSA-PKCS-PSS signing mechanism.
- Compare a signed file against a source file and determines whether the two are cryptographically related based on a given rsa public key and signing mechanism.

To use the **crypto verify rsa-pkcs-pss** command, you must first have an RSA public key in your AWS CloudHSM cluster. You can import an RSA public key using the key import pem command ADD UNWRAP LINK HERE) with the `verify` attribute set to `true`.

> **Note**
>
> You can generate a signature using the CloudHSM CLI with the The crypto sign category in CloudHSM CLI subcommands.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help crypto verify rsa-pkcs-pss
Verify with the RSA-PKCS-PSS mechanism

Usage: crypto verify rsa-pkcs-pss --key-filter [<KEY_FILTER>...] --hash-
function <HASH_FUNCTION> --mgf <MGF> --salt-length >SALT_LENGTH< <--data-
path <DATA_PATH>|--data <DATA> <--signature-path <SIGNATURE_PATH>|--
signature <SIGNATURE>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --key-filter [<KEY_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key
      --hash-function <HASH_FUNCTION>
          [possible values: sha1, sha224, sha256, sha384, sha512]
      --data-path <DATA_PATH>
          The path to the file containing the data to be verified
      --data <DATA>
          Base64 encoded data to be verified
      --signature-path <SIGNATURE_PATH>
          The path to where the signature is located
      --signature <SIGNATURE>
          Base64 encoded signature to be verified
      --data-type <DATA_TYPE>
          The type of data passed in, either raw or digest [possible values: raw,
 digest]
```

```
      --mgf <MGF>
          The mask generation function [possible values: mgf1-sha1, mgf1-sha224, mgf1-sha256, mgf1-sha384, mgf1-sha512]
      --salt-length <SALT_LENGTH>
          The salt length
  -h, --help
          Print help
```

Example

These examples show how to use **crypto verify rsa-pkcs-pss** to verify a signature that was generated using the RSA-PKCS-PSS signing mechanism and SHA256 hash function. This command uses a public key in the HSM.

Example Example: Verify a Base64 encoded signature with Base64 encoded data

```
aws-cloudhsm > crypto verify rsa-pkcs-pss --key-filter attr.label=rsa-public
 --hash-function sha256 --data YWJjMTIz --salt-length 10 --mgf mgf1-sha256
 --signature H/z1rYVMzNAa31K4amE5MTiwGxDdCTgQXCJXRBKVOVm7ZuyI0fGE4sT/BUN
+977mQEV2TqtWpTsiF2IpwGM1VfSBRt7h/g4o6YERm1tTQLl7q+AJ7uGGK37zCsWQrAo7Vy8NzPShxekePo/
ZegrB1aHWN1fE8H3IPUKqLuMDI9o1Jq6kM986ExS7YmeOIclcZkyykTWqHLQVL2C3+A2bHJZBqRcM5XoIpk8HkPypjpN
+m4FNUds30GAemoOMl6asSrEJSthaZWV53OBsDOqzA8Rt8JdhXS+GZp3vNLdL1OTBELDPweXVgAu4dBX0FOvpw/
gg6sNvuaDK4YOBv2fqKg==
{
  "error_code": 0,
  "data": {
    "message": "Signature verified successfully"
  }
}
```

Example Example: Verify a signature file with a data file

```
aws-cloudhsm > crypto verify rsa-pkcs-pss --key-filter attr.label=rsa-public --hash-function sha256 --data-path data.txt --salt-length 10 --mgf mgf1-sha256 --signature
 signature-file
{
  "error_code": 0,
  "data": {
    "message": "Signature verified successfully"
  }
}
```

Example Example: Prove false signing relationship

This command verifies whether the invalid data was signed by a public key with the label `rsa-public` using the RSAPKCSPSS signing mechanism to produce the signature located in `/home/signature`. Because the given arguments do not make up a true signing relationship, the command returns an error message.

```
aws-cloudhsm > crypto verify rsa-pkcs-pss --key-filter attr.label=rsa-public
 --hash-function sha256 --data aW52YWxpZA== --salt-length 10 --mgf mgf1-sha256
 --signature H/z1rYVMzNAa31K4amE5MTiwGxDdCTgQXCJXRBKVOVm7ZuyI0fGE4sT/BUN
+977mQEV2TqtWpTsiF2IpwGM1VfSBRt7h/g4o6YERm1tTQLl7q+AJ7uGGK37zCsWQrAo7Vy8NzPShxekePo/
ZegrB1aHWN1fE8H3IPUKqLuMDI9o1Jq6kM986ExS7YmeOIclcZkyykTWqHLQVL2C3+A2bHJZBqRcM5XoIpk8HkPypjpN
+m4FNUds30GAemoOMl6asSrEJSthaZWV53OBsDOqzA8Rt8JdhXS+GZp3vNLdL1OTBELDPweXVgAu4dBX0FOvpw/
gg6sNvuaDK4YOBv2fqKg==
{
  "error_code": 1,
  "data": "Signature verification failed"
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been [configured.]()

<DATA>

Base64 encoded data to be signed.

Required: Yes (unless provided through data path)

<DATA_PATH>

Specifies the location of the data to be signed.

Required: Yes (unless provided through data path)

<HASH_FUNCTION>

Specifies the hash function.

Valid values:

- sha1
- sha224
- sha256
- sha384
- sha512

Required: Yes

<KEY_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a matching key.

For a listing of supported CloudHSM CLI key attributes, see Key attributes for CloudHSM CLI.

Required: Yes

<MFG>

Specifies the mask generation function.

> **Note**
>
> The mask generation function hash function must match the signing mechanism hash function.

Valid values:

- mgf1-sha1
- mgf1-sha224
- mgf1-sha256
- mgf1-sha384
- mgf1-sha512

Required: Yes

<SIGNATURE>

Base64 encoded signature.

Required: Yes (unless provided through signature path)

<SIGNATURE_PATH>

Specifies the location of the signature.

Required: Yes (unless provided through signature path)

<DATA_TYPE>

Specifies whether the value of the data parameter should be hashed as part of the signing algorithm. Use `raw` for unhashed data; use `digest` for digests, which are already hashed.

Valid values:

- raw
- digest

Related topics

- The crypto sign category in CloudHSM CLI
- The crypto verify category in CloudHSM CLI

The key category in CloudHSM CLI

In the CloudHSM CLI, **key** is a parent category for a group of commands that, when combined with the parent category, create a command specific to keys. Currently, this category consists of the following commands:

- delete
- generate-file
- key generate-asymmetric-pair
  - key generate-asymmetric-pair rsa
  - key generate-asymmetric-pair ec
- key generate-symmetric
  - key generate-symmetric aes
  - key generate-symmetric generic-secret
- import pem

- list
- replicate
- set-attribute
- share
- unshare
- unwrap
- wrap

Delete a key with CloudHSM CLI

Use the **key delete** command in CloudHSM CLI to delete a key from an AWS CloudHSM cluster. You can only delete one key at a time. Deleting one key in a key pair has no effect on the other key in the pair.

Only the CU who created the key and consequently owns it can delete the key. Users who share the key, but do not own it, can use the key in cryptographic operations, but can not delete it.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key delete
Delete a key in the HSM cluster

Usage: key delete [OPTIONS] --filter [<FILTER>...]

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
```

```
      --filter [<FILTER>...]      Key reference (e.g. key-reference=0xabc)
 or space separated list of key attributes in the form of
 attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a matching key for deletion
  -h, --help                      Print help
```

Example

```
aws-cloudhsm > key delete --filter attr.label="ec-test-public-key"
{
  "error_code": 0,
  "data": {
    "message": "Key deleted successfully"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a matching key for deletion.

For a list of supported CloudHSM CLI key attributes, see Key attributes for CloudHSM CLI

Required: Yes

Related topics

- List keys for a user with CloudHSM CLI
- Export an asymmetric key with CloudHSM CLI
- Unshare a key using CloudHSM CLI
- Key attributes for CloudHSM CLI
- Filter keys using CloudHSM CLI

Export an asymmetric key with CloudHSM CLI

Use the **key generate-file** command in CloudHSM CLI to export an asymmetric key from the hardware security module (HSM). If the target is a private key, then the reference to the private key will be exported in fake PEM format. If the target is a public key, then the public key bytes will be exported in PEM format.

The fake PEM file, which does not contain the actual private key material but instead references the private key in the HSM, can be used to establish SSL/TLS offloading from your web server to AWS CloudHSM. For more information, see SSL/TLS offloading.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key generate-file
Generate a key file from a key in the HSM cluster. This command does not export any
 private key data from the HSM

Usage: key generate-file --encoding <ENCODING> --path <PATH> --filter [<FILTER>...]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --encoding <ENCODING>
          Encoding format for the key file

          Possible values:
          - reference-pem: PEM formatted key reference (supports private keys)
          - pem:           PEM format (supports public keys)

      --path <PATH>
```

```
            Filepath where the key file will be written

        --filter [<FILTER>...]
            Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key for file generation

  -h, --help
            Print help (see a summary with '-h')
```

Example

This example shows how to use **key generate-file** to generate a key file in your AWS CloudHSM cluster.

Example

```
aws-cloudhsm > key generate-file --encoding reference-pem --path /tmp/ec-private-key.pem --filter attr.label="ec-test-private-key"
{
  "error_code": 0,
  "data": {
    "message": "Successfully generated key file"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a matching key for deletion.

For a listing of supported CloudHSM CLI key attributes, see Key attributes for CloudHSM CLI

Required: No

<ENCODING>

Specifies the encoding format for the key file

Required: Yes

<PATH>

Specifies the file path where the key file will be written

Required: Yes

Generating KSP key references (Windows)

> **Note**
>
> This feature is only in SDK version 5.16.0 and later.

Prerequisites

- You can generate KSP key references only on Windows platforms.
- You must sign in as a crypto user (CU).

File location

By default, AWS CloudHSM stores generated files in: `C:\Users\Default\AppData\Roaming\Microsoft\Crypto\CaviumKSP\GlobalPartition`

To specify a different location, use the `--path` parameter.

Syntax

```
aws-cloudhsm > help key generate-file --encoding ksp-key-reference
Generate a key file from a key in the HSM cluster. This command does not export any
 private key data from the HSM

Usage: key generate-file --encoding <ENCODING> --path <PATH> --filter [<FILTER>...]

Options:
      --encoding <ENCODING>
```

```
        Encoding format for the key file

        Possible values:
        - reference-pem:     PEM formatted key reference (supports private keys)
        - pem:               PEM format (supports public keys)
        - ksp-key-reference: KSP key reference format

      --cluster-id <CLUSTER_ID>
        Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided with multiple clusters configured, will error

      --path <PATH>
        Directory path where the key file will be written

      --filter [<FILTER>...]
        Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key for file generation

      --all
        Generate ksp key reference for all available key pairs in HSM

  -h, --help
        Print help (see a summary with '-h')
```

Example – Generate a KSP key reference using an attribute filter of a private key

The following example generates a KSP key reference for a private key with a specific label.

Example

```
aws-cloudhsm > key generate-file --encoding ksp-key-reference --path  --filter
 attr.label="ec-test-private-key"
{
  "error_code": 0,
  "data": {
    "message": "Successfully generated key file"
  }
}
```

Example – Generate KSP key references for all key pairs

The following example generates KSP key references for all key pairs in your cluster.

Example

```
aws-cloudhsm > key generate-file --encoding ksp-key-reference --all
{
  "error_code": 0,
  "data": {
    "message": "Successfully generated key file"
  }
}
```

Related topics

- Key attributes for CloudHSM CLI
- Filter keys using CloudHSM CLI
- The generate-asymmetric-pair category in CloudHSM CLI
- The generate-symmetric category in CloudHSM CLI

The generate-asymmetric-pair category in CloudHSM CLI

In the CloudHSM CLI, **key generate-asymmetric-pair** is a parent category for a group of commands that, when combined with the parent category, create a command that generates asymmetric key pairs. Currently, this category consists of the following commands:

- key generate-asymmetric-pair ec
- key generate-asymmetric-pair rsa

Generate an asymmetric EC key pair with CloudHSM CLI

Use the **key asymmetric-pair ec** command in CloudHSM CLI to generate an asymmetric Elliptic-curve (EC) key pair in your AWS CloudHSM cluster.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key generate-asymmetric-pair ec
Generate an Elliptic-Curve Cryptography (ECC) key pair

Usage: key generate-asymmetric-pair ec [OPTIONS] --public-label <PUBLIC_LABEL> --
private-label <PRIVATE_LABEL> --curve <CURVE>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --public-label <PUBLIC_LABEL>
          Label for the public key
      --private-label <PRIVATE_LABEL>
          Label for the private key
      --session
          Creates a session key pair that exists only in the current session. The key
 cannot be recovered after the session ends
      --curve <CURVE>
          Elliptic curve used to generate the key pair [possible values: prime256v1,
 secp256r1, secp224r1, secp384r1, secp256k1, secp521r1]
      --public-attributes [<PUBLIC_KEY_ATTRIBUTES>...]
          Space separated list of key attributes to set for the generated EC public key
 in the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE
      --private-attributes [<PRIVATE_KEY_ATTRIBUTES>...]
          Space separated list of key attributes to set for the generated EC private
 key in the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE
      --share-crypto-users [<SHARE_CRYPTO_USERS>...]
          Space separated list of Crypto User usernames to share the EC private key
 with
      --manage-private-key-quorum-value <MANAGE_PRIVATE_KEY_QUORUM_VALUE>
          The quorum value for key management operations for the private key
      --use-private-key-quorum-value <USE_PRIVATE_KEY_QUORUM_VALUE>
          The quorum value for key usage operations for the private key
  -h, --help
          Print help
```

Examples

These examples show how to use the **key generate-asymmetric-pair ec** command to create an EC key pair.

Example Example: Create an EC key pair

```
aws-cloudhsm > key generate-asymmetric-pair ec \
    --curve secp224r1 \
    --public-label ec-public-key-example \
    --private-label ec-private-key-example
{
  "error_code": 0,
  "data": {
    "public_key": {
      "key-reference": "0x000000000012000b",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "ec",
        "label": "ec-public-key-example",
        "id": "",
        "check-value": "0xd7c1a7",
        "class": "public-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
```

```
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 57,
        "ec-point":
 "0x047096513df542250a6b228fd9cb67fd0c903abc93488467681974d6f371083fce1d79da8ad1e9ede745fb9f38a
        "curve": "secp224r1"
      }
    },
"private_key": {
      "key-reference": "0x000000000012000c",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "ec",
        "label": "ec-private-key-example",
        "id": "",
        "check-value": "0xd7c1a7",
        "class": "private-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
```

```
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 122,
        "ec-point":
 "0x047096513df542250a6b228fd9cb67fd0c903abc93488467681974d6f371083fce1d79da8ad1e9ede745fb9f38a
        "curve": "secp224r1"
      }
    }
  }
}
```

Example Example: Create an EC key pair with optional attributes

```
aws-cloudhsm > key generate-asymmetric-pair ec \
    --curve secp224r1 \
    --public-label ec-public-key-example \
    --private-label ec-private-key-example \
    --public-attributes encrypt=true \
    --private-attributes decrypt=true
{
  "error_code": 0,
  "data": {
    "public_key": {
      "key-reference": "0x00000000002806eb",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
```

```
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "ec",
        "label": "ec-public-key-example",
        "id": "",
        "check-value": "0xedef86",
        "class": "public-key",
        "encrypt": true,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 57,
        "ec-point":
 "0x0487af31882189ec29eddf17a48e8b9cebb075b7b5afc5522fe9c83a029a450cc68592889a1ebf45f32240da514
        "curve": "secp224r1"
      }
    },
    "private_key": {
      "key-reference": "0x0000000000280c82",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
```

```
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "ec",
        "label": "ec-private-key-example",
        "id": "",
        "check-value": "0xedef86",
        "class": "private-key",
        "encrypt": false,
        "decrypt": true,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 122,
        "ec-point":
 "0x0487af31882189ec29eddf17a48e8b9cebb075b7b5afc5522fe9c83a029a450cc68592889a1ebf45f32240da514
        "curve": "secp224r1"
      }
    }
  }
}
```

Example Example: Create an EC key pair with quorum values

When generating a key with quorum controls, the key must be associated with a minimum number of users equal to the largest key quorum value. Associated users include the key owner and Crypto Users with whom the key is shared with. To determine the number of minimum users to share the key with, get the largest quorum value between the key usage quorum value and the key management quorum value and subtract 1 to account for the key owner, who is by default associated with the key. To share the key with more users, use the **Share a key using CloudHSM CLI** command.

```
aws-cloudhsm > key generate-asymmetric-pair ec \
    --curve secp224r1 \
    --public-label ec-public-key-example \
    --private-label ec-private-key-example \
    --public-attributes verify=true \
    --private-attributes sign=true
    --share-crypto-users cu2 cu3 cu4 \
    --manage-private-key-quorum-value 4 \
    --use-private-key-quorum-value 2
{
  "error_code": 0,
  "data": {
    "public_key": {
      "key-reference": "0x00000000002806eb",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "ec",
        "label": "ec-public-key-example",
        "id": "",
        "check-value": "0xedef86",
```

```
        "class": "public-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 57,
        "ec-point":
 "0x0487af31882189ec29eddf17a48e8b9cebb075b7b5afc5522fe9c83a029a450cc68592889a1ebf45f32240da514
        "curve": "secp224r1"
      }
    },
    "private_key": {
      "key-reference": "0x0000000000280c82",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [
          {
            "username": "cu2",
            "key-coverage": "full"
          },
          {
            "username": "cu3",
            "key-coverage": "full"
          },
          {
```

```
            "username": "cu4",
            "key-coverage": "full"
          },
        ],
        "key-quorum-values": {
          "manage-key-quorum-value": 4,
          "use-key-quorum-value": 2
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "ec",
        "label": "ec-private-key-example",
        "id": "",
        "check-value": "0xedef86",
        "class": "private-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 122,
        "ec-point":
 "0x0487af31882189ec29eddf17a48e8b9cebb075b7b5afc5522fe9c83a029a450cc68592889a1ebf45f32240da514
        "curve": "secp224r1"
      }
    }
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<CURVE>

Specifies the identifier for the elliptic curve.

- prime256v1
- secp256r1
- secp224r1
- secp384r1
- secp256k1
- secp521r1

Required: Yes

<PUBLIC_KEY_ATTRIBUTES>

Specifies a space separated list of key attributes to set for the generated EC public key in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (for example, `verify=true`)

For a list of supported key attributes, see Key attributes for CloudHSM CLI.

Required: No

<PUBLIC_LABEL>

Specifies a user defined label for the public-key. The maximum size allowable for `label` is 127 characters for Client SDK 5.11 and after. Client SDK 5.10 and before has a limit of 126 characters.

Required: Yes

<PRIVATE_KEY_ATTRIBUTES>

Specifies a space separated list of key attributes to set for the generated EC private key in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (for example, `sign=true`)

For a list of supported key attributes, see Key attributes for CloudHSM CLI.

Required: No

<PRIVATE_LABEL>

Specifies a user defined label for the private-key. The maximum size allowable for `label` is 127 characters for Client SDK 5.11 and after. Client SDK 5.10 and before has a limit of 126 characters.

Required: Yes

<SESSION>

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

By default, keys that are generated are persistent (token) keys. Passing in <SESSION> changes this, ensuring a key generated with this argument is a session (ephemeral) key.

Required: No

<SHARE_CRYPTO_USERS>

Specifies a space separated list of Crypto User usernames to share the EC private key with

Required: No

<MANAGE_PRIVATE_KEY_QUORUM_VALUE>

The quorum value for the private key's key management operations. This value must be less than or equal to the number of users that the key is associated with. This includes users with whom the key is shared with and the key owner. Max value of 8.

Required: No

<USE_PRIVATE_KEY_QUORUM_VALUE>

The quorum value for private key's key usage operations. This value must be less than or equal to the number of users that the key is associated with. This includes users with whom the key is shared with and the key owner. Max value of 8.

Required: No

Related topics

- [Key attributes for CloudHSM CLI]
- [Filter keys using CloudHSM CLI]

Generate an asymmetric RSA key pair with CloudHSM CLI

Use the **key generate-asymmetric-pair rsa** command in CloudHSM CLI to generate an asymmetric RSA key pair in your AWS CloudHSM cluster.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key generate-asymmetric-pair rsa
Generate an RSA key pair

Usage: key generate-asymmetric-pair rsa [OPTIONS] --public-label <PUBLIC_LABEL>
 --private-label <PRIVATE_LABEL> --modulus-size-bits <MODULUS_SIZE_BITS> --public-
exponent <PUBLIC_EXPONENT>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --public-label <PUBLIC_LABEL>
          Label for the public key
      --private-label <PRIVATE_LABEL>
          Label for the private key
      --session
          Creates a session key pair that exists only in the current session. The key
 cannot be recovered after the session ends
      --modulus-size-bits <MODULUS_SIZE_BITS>
```

```
            Modulus size in bits used to generate the RSA key pair
        --public-exponent <PUBLIC_EXPONENT>
            Public exponent used to generate the RSA key pair
        --public-attributes [<PUBLIC_KEY_ATTRIBUTES>...]
            Space separated list of key attributes to set for the generated RSA public
 key in the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE
        --private-attributes [<PRIVATE_KEY_ATTRIBUTES>...]
            Space separated list of key attributes to set for the generated RSA private
 key in the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE
        --share-crypto-users [<SHARE_CRYPTO_USERS>...]
            Space separated list of Crypto User usernames to share the RSA key with
        --manage-private-key-quorum-value <MANAGE_PRIVATE_KEY_QUORUM_VALUE>
            The quorum value for key management operations for the private key
        --use-private-key-quorum-value <USE_PRIVATE_KEY_QUORUM_VALUE>
            The quorum value for key usage operations for the private key
    -h, --help
            Print help
```

Examples

These examples show how to use `key generate-asymmetric-pair rsa` to create a RSA key pair.

Example Example: Create an RSA key pair

```
aws-cloudhsm > key generate-asymmetric-pair rsa \
--public-exponent 65537 \
--modulus-size-bits 2048 \
--public-label rsa-public-key-example \
--private-label rsa-private-key-example
{
  "error_code": 0,
  "data": {
    "public_key": {
      "key-reference": "0x0000000000160010",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
```

```
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa-public-key-example",
        "id": "",
        "check-value": "0x498e1f",
        "class": "public-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 512,
        "public-exponent": "0x010001",
        "modulus":
 "0xdfca0669dc8288ed3bad99509bd21c7e6192661407021b3f4cdf4a593d939dd24f4d641af8e4e73b04c847731c6
e89a065e7d1a46ced96b46b909db2ab6be871ee700fd0a448b6e975bb64cae77c49008749212463e37a577baa57ce3e
bcebb7d20bd6df1948ae336ae23b52d73b7f3b6acc2543edb6358e08d326d280ce489571f4d34e316a2ea1904d513ca
        "modulus-size-bits": 2048
      }
    },
"private_key": {
      "key-reference": "0x0000000000160011",
      "key-info": {
        "key-owners": [
          {
```

```
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa-private-key-example",
        "id": "",
        "check-value": "0x498e1f",
        "class": "private-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 1217,
        "public-exponent": "0x010001",
        "modulus":
 "0xdfca0669dc8288ed3bad99509bd21c7e6192661407021b3f4cdf4a593d939dd24f4d641af8e4e73b04c847731c6
        "modulus-size-bits": 2048
      }
    }
  }
```

```
}
```

Example Example: Create an RSA key pair with optional attributes

```
aws-cloudhsm > key generate-asymmetric-pair rsa \
--public-exponent 65537 \
--modulus-size-bits 2048 \
--public-label rsa-public-key-example \
--private-label rsa-private-key-example \
--public-attributes encrypt=true \
--private-attributes decrypt=true
{
  "error_code": 0,
  "data": {
    "public_key": {
      "key-reference": "0x0000000000280cc8",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa-public-key-example",
        "id": "",
        "check-value": "0x01fe6e",
        "class": "public-key",
        "encrypt": true,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
```

```
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 512,
        "public-exponent": "0x010001",
        "modulus":
 "0xb1d27e857a876f4e9fd5de748a763c539b359f937eb4b4260e30d1435485a732c878cdad9c72538e2215351b1d4
73a80fdb457aa7b20cd61e486c326e2cfd5e124a7f6a996437437812b542e3caf85928aa866f0298580f7967ee6aa01
f6e6296d6c116d5744c6d60d14d3bf3cb978fe6b75ac67b7089bafd50d8687213b31abc7dc1bad422780d29c851d510
133022653225bd129f8491101725e9ea33e1ded83fb57af35f847e532eb30cd7e726f23910d2671c6364092e834697e
ac3160f0ca9725d38318b7",
        "modulus-size-bits": 2048
      }
    },
    "private_key": {
      "key-reference": "0x0000000000280cc7",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa-private-key-example",
        "id": "",
        "check-value": "0x01fe6e",
        "class": "private-key",
```

```
        "encrypt": false,
        "decrypt": true,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 1217,
        "public-exponent": "0x010001",
        "modulus":
 "0xb1d27e857a876f4e9fd5de748a763c539b359f937eb4b4260e30d1435485a732c878cdad9c72538e2215351b1d4
        "modulus-size-bits": 2048
      }
    }
  }
}
```

Example Example: Create an RSA key pair with quorum values

When generating a key with quorum controls, the key must be associated with a minimum number of users equal to the largest key quorum value. Associated users include the key owner and Crypto Users with whom the key is shared with. To determine the number of minimum users to share the key with, get the largest quorum value between the key usage quorum value and the key management quorum value and subtract 1 to account for the key owner, who is by default associated with the key. To share the key with more users, use the **Share a key using CloudHSM CLI** command.

```
aws-cloudhsm > key generate-asymmetric-pair rsa \
--public-exponent 65537 \
--modulus-size-bits 2048 \
--public-label rsa-public-key-example \
```

```
--private-label rsa-private-key-example \
--public-attributes verify=true \
--private-attributes sign=true
--share-crypto-users cu2 cu3 cu4 \
--manage-private-key-quorum-value 4 \
--use-private-key-quorum-value 2
{
  "error_code": 0,
  "data": {
    "public_key": {
      "key-reference": "0x0000000000280cc8",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa-public-key-example",
        "id": "",
        "check-value": "0x01fe6e",
        "class": "public-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": true,
```

```
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 512,
        "public-exponent": "0x010001",
        "modulus":
 "0xb1d27e857a876f4e9fd5de748a763c539b359f937eb4b4260e30d1435485a732c878cdad9c72538e2215351b1d4
73a80fdb457aa7b20cd61e486c326e2cfd5e124a7f6a996437437812b542e3caf85928aa866f0298580f7967ee6aa01
f6e6296d6c116d5744c6d60d14d3bf3cb978fe6b75ac67b7089bafd50d8687213b31abc7dc1bad422780d29c851d510
133022653225bd129f8491101725e9ea33e1ded83fb57af35f847e532eb30cd7e726f23910d2671c6364092e834697e
ac3160f0ca9725d38318b7",
        "modulus-size-bits": 2048
      }
    },
    "private_key": {
      "key-reference": "0x0000000000280cc7",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [
          {
            "username": "cu2",
            "key-coverage": "full"
          },
          {
            "username": "cu3",
            "key-coverage": "full"
          },
          {
            "username": "cu4",
            "key-coverage": "full"
          },
        ],
        "key-quorum-values": {
          "manage-key-quorum-value": 4,
          "use-key-quorum-value": 2
        },
        "cluster-coverage": "full"
```

```
      },
      "attributes": {
        "key-type": "rsa",
        "label": "rsa-private-key-example",
        "id": "",
        "check-value": "0x01fe6e",
        "class": "private-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 1217,
        "public-exponent": "0x010001",
        "modulus":
 "0xb1d27e857a876f4e9fd5de748a763c539b359f937eb4b4260e30d1435485a732c878cdad9c72538e2215351b1d4
        "modulus-size-bits": 2048
      }
    }
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<MODULUS_SIZE_BITS>

Specifies the length of the modulus in bits. The minimum value is 2048.

Required: Yes

<PRIVATE_KEY_ATTRIBUTES>

Specifies a space separated list of key attributes to set for the generated RSA private key in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (for example, `sign=true`)

For a list of supported key attributes, see [Key attributes for CloudHSM CLI](#).

Required: No

<PRIVATE_LABEL>

Specifies a user defined label for the private-key. The maximum size allowable for `label` is 127 characters for Client SDK 5.11 and after. Client SDK 5.10 and before has a limit of 126 characters.

Required: Yes

<PUBLIC_EXPONENT>

Specifies the public exponent. The value must be an odd number greater than or equal to 65537.

Required: Yes

<PUBLIC_KEY_ATTRIBUTES>

Specifies a space separated list of key attributes to set for the generated RSA public key in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (for example, `verify=true`)

For a list of supported key attributes, see [Key attributes for CloudHSM CLI](#).

Required: No

<PUBLIC_LABEL>

Specifies a user defined label for the public-key. The maximum size allowable for `label` is 127 characters for Client SDK 5.11 and after. Client SDK 5.10 and before has a limit of 126 characters.

Required: Yes

<SESSION>

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

By default, keys that are generated are persistent (token) keys. Passing in <SESSION> changes this, ensuring a key generated with this argument is a session (ephemeral) key.

Required: No

<SHARE_CRYPTO_USERS>

Specifies a space separated list of Crypto User usernames to share the RSA private key with

Required: No

<MANAGE_PRIVATE_KEY_QUORUM_VALUE>

The quorum value for the private key's key management operations. This value must be less than or equal to the number of users that the key is associated with. This includes users with whom the key is shared with and the key owner. Max value of 8.

Required: No

<USE_PRIVATE_KEY_QUORUM_VALUE>

The quorum value for private key's key usage operations. This value must be less than or equal to the number of users that the key is associated with. This includes users with whom the key is shared with and the key owner. Max value of 8.

Required: No

Related topics

- [Key attributes for CloudHSM CLI]
- [Filter keys using CloudHSM CLI]

The generate-symmetric category in CloudHSM CLI

In the CloudHSM CLI, **key generate-symmetric** is a parent category for a group of commands that, when combined with the parent category, create a command that generates symmetric keys. Currently, this category consists of the following commands:

- key generate-symmetric aes
- key generate-symmetric generic-secret

Generate a symmetric AES key with CloudHSM CLI

Use the **key generate-symmetric aes** command in CloudHSM CLI to generate a symmetric AES key in your AWS CloudHSM cluster.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key generate-symmetric aes
Generate an AES key

Usage: key generate-symmetric aes [OPTIONS] --label <LABEL> --key-length-
bytes <KEY_LENGTH_BYTES>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --label <LABEL>
          Label for the key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
```

```
      --key-length-bytes <KEY_LENGTH_BYTES>
          Key length in bytes
      --attributes [<KEY_ATTRIBUTES>...]
          Space separated list of key attributes to set for the generated AES key in
 the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE
      --share-crypto-users [<SHARE_CRYPTO_USERS>...]
          Space separated list of Crypto User usernames to share the AES key with
      --manage-key-quorum-value <MANAGE_KEY_QUORUM_VALUE>
          The quorum value for key management operations
      --use-key-quorum-value <USE_KEY_QUORUM_VALUE>
          The quorum value for key usage operations
  -h, --help
          Print help
```

Examples

These examples show how to use the **key generate-symmetric aes** command to create an AES key.

Example Example: Create an AES key

```
aws-cloudhsm > key generate-symmetric aes \
--label example-aes \
--key-length-bytes 24
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000002e06bf",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
```

```
        "label": "example-aes",
        "id": "",
        "check-value": "0x9b94bd",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 24
      }
    }
  }
}
```

Example Example: Create an AES key with optional attributes

```
aws-cloudhsm > key generate-symmetric aes \
--label example-aes \
--key-length-bytes 24 \
--attributes decrypt=true encrypt=true
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000002e06bf",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
```

```
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
          "key-quorum-values": {
            "manage-key-quorum-value": 0,
            "use-key-quorum-value": 0
          },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "aes",
          "label": "example-aes",
          "id": "",
          "check-value": "0x9b94bd",
          "class": "secret-key",
          "encrypt": true,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": false,
          "verify": true,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 24
        }
      }
    }
}
```

Example Example: Create an AES key with quorum values

When generating a key with quorum controls, the key must be associated with a minimum number of users equal to the largest key quorum value. Associated users include the key owner and Crypto Users with whom the key is shared with. To determine the number of minimum users to share the key with, get the largest quorum value between the key usage quorum value and the key management quorum value and subtract 1 to account for the key owner, who is by default associated with the key. To share the key with more users, use the **Share a key using CloudHSM CLI** command.

```
aws-cloudhsm > key generate-symmetric aes \
--label example-aes \
--key-length-bytes 24 \
--attributes decrypt=true encrypt=true
--share-crypto-users cu2 cu3 cu4 \
--manage-key-quorum-value 4 \
--use-key-quorum-value 2
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000002e06bf",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [
          {
            "username": "cu2",
            "key-coverage": "full"
          },
          {
            "username": "cu3",
            "key-coverage": "full"
          },
          {
            "username": "cu4",
            "key-coverage": "full"
          },
        ],
```

```
          "key-quorum-values": {
            "manage-key-quorum-value": 4,
            "use-key-quorum-value": 2
          },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "aes",
          "label": "example-aes",
          "id": "",
          "check-value": "0x9b94bd",
          "class": "secret-key",
          "encrypt": true,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": false,
          "verify": true,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 24
        }
      }
    }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<KEY_ATTRIBUTES>

Specifies a space separated list of key attributes to set for the generated AES key in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (for example, `sign=true`).

For a list of supported key attributes, see Key attributes for CloudHSM CLI.

Required: No

<KEY-LENGTH-BYTES>

Specifies the key length in bytes.

Valid values:

- 16, 24, and 32

Required: Yes

<LABEL>

Specifies a user defined label for the AES key. The maximum size allowable for `label` is 127 characters for Client SDK 5.11 and after. Client SDK 5.10 and before has a limit of 126 characters.

Required: Yes

<SESSION>

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

By default, keys that are generated are persistent (token) keys. Passing in <SESSION> changes this, ensuring a key generated with this argument is a session (ephemeral) key.

Required: No

<SHARE_CRYPTO_USERS>

Specifies a space separated list of Crypto User usernames to share the AES key with

Required: No

<MANAGE_KEY_QUORUM_VALUE>

The quorum value for key management operations. This value must be less than or equal to the number of users that the key is associated with. This includes users with whom the key is shared with and the key owner. Max value of 8.

Required: No

<USE_KEY_QUORUM_VALUE>

The quorum value for key usage operations. This value must be less than or equal to the number of users that the key is associated with. This includes users with whom the key is shared with and the key owner. Max value of 8.

Required: No

Related topics

- Key attributes for CloudHSM CLI
- Filter keys using CloudHSM CLI

Generate a symmetric Generic Secret key with CloudHSM CLI

Use the **key generate-asymmetric-pair** command in CloudHSM CLI to generate a symmetric Generic Secret key in your AWS CloudHSM cluster.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > key help generate-symmetric generic-secret
Generate a generic secret key
```

```
Usage: key generate-symmetric generic-secret [OPTIONS] --label <LABEL> --key-length-bytes <KEY_LENGTH_BYTES>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --label <LABEL>
          Label for the key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
      --key-length-bytes <KEY_LENGTH_BYTES>
          Key length in bytes
      --attributes [<KEY_ATTRIBUTES>...]
          Space separated list of key attributes to set for the generated generic
 secret key in the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE
      --share-crypto-users [<SHARE_CRYPTO_USERS>...]
          Space separated list of Crypto User usernames to share the generic secret key
 with
      --manage-key-quorum-value <MANAGE_KEY_QUORUM_VALUE>
          The quorum value for key management operations
      --use-key-quorum-value <USE_KEY_QUORUM_VALUE>
          The quorum value for key usage operations
  -h, --help
          Print help
```

Examples

These examples show how to use the **key generate-symmetric generic-secret** command to create a generic secret key.

Example Example: Create a generic secret key

```
aws-cloudhsm > key generate-symmetric generic-secret \
--label example-generic-secret \
--key-length-bytes 256
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000002e08fd",
      "key-info": {
```

```
          "key-owners": [
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
          "key-quorum-values": {
            "manage-key-quorum-value": 0,
            "use-key-quorum-value": 0
          },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "generic-secret",
          "label": "example-generic-secret",
          "id": "",
          "class": "secret-key",
          "encrypt": false,
          "decrypt": false,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": false,
          "verify": true,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 256
        }
      }
    }
}
```

Example Example: Create a generic secret key with optional attributes

```
aws-cloudhsm > key generate-symmetric generic-secret \
--label example-generic-secret \
--key-length-bytes 256 \
--attributes encrypt=true
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000002e08fd",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "generic-secret",
        "label": "example-generic-secret",
        "id": "",
        "class": "secret-key",
        "encrypt": true,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
```

```
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 256
      }
    }
  }
}
```

Example Example: Create a generic secret key with quorum values

When generating a key with quorum controls, the key must be associated with a minimum number of users equal to the largest key quorum value. Associated users include the key owner and Crypto Users with whom the key is shared with. To determine the number of minimum users to share the key with, get the largest quorum value between the key usage quorum value and the key management quorum value and subtract 1 to account for the key owner, who is by default associated with the key. To share the key with more users, use the **Share a key using CloudHSM CLI** command.

```
aws-cloudhsm > key generate-symmetric generic-secret \
--label example-generic-secret \
--key-length-bytes 256 \
--attributes encrypt=true
--share-crypto-users cu2 cu3 cu4 \
--manage-key-quorum-value 4 \
--use-key-quorum-value 2
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000002e08fd",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [
          {
            "username": "cu2",
```

```
            "key-coverage": "full"
          },
          {
            "username": "cu3",
            "key-coverage": "full"
          },
          {
            "username": "cu4",
            "key-coverage": "full"
          },
        ],
        "key-quorum-values": {
          "manage-key-quorum-value": 4,
          "use-key-quorum-value": 2
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "generic-secret",
        "label": "example-generic-secret",
        "id": "",
        "class": "secret-key",
        "encrypt": true,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 256
      }
    }
  }
```

```
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<KEY_ATTRIBUTES>

Specifies a space separated list of key attributes to set for the generated AES key in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (for example, `sign=true`).

For a list of supported key attributes, see Key attributes for CloudHSM CLI.

Required: No

<KEY-LENGTH-BYTES>

Specifies the key length in bytes.

Valid values:

- 1 to 800

Required: Yes

<LABEL>

Specifies a user defined label for the generic secret key. The maximum size allowable for `label` is 127 characters for Client SDK 5.11 and after. Client SDK 5.10 and before has a limit of 126 characters.

Required: Yes

<SESSION>

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

By default, keys that are generated are persistent (token) keys. Passing in <SESSION> changes this, ensuring a key generated with this argument is a session (ephemeral) key.

Required: No

<SHARE_CRYPTO_USERS>

Space separated list of Crypto User usernames to share the generic secret key with

Required: No

<MANAGE_KEY_QUORUM_VALUE>

The quorum value for key management operations. This value must be less than or equal to the number of users that the key is associated with. This includes users with whom the key is shared with and the key owner. Max value of 8.

Required: No

<USE_KEY_QUORUM_VALUE>

The quorum value for key usage operations. This value must be less than or equal to the number of users that the key is associated with. This includes users with whom the key is shared with and the key owner. Max value of 8.

Required: No

Related topics

- Key attributes for CloudHSM CLI
- Filter keys using CloudHSM CLI

Import a PEM format key with CloudHSM CLI

Use the **key import pem** command in AWS CloudHSM to import a PEM format key into a hardware security module (HSM). You can use it to import public keys that were generated outside of the HSM.

> **Note**
>
> Use the Export an asymmetric key with CloudHSM CLI command to create a standard PEM file from a public key or to create a reference PEM file from a private key.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key import pem
Import key from a PEM file

Usage: key import pem [OPTIONS] --path <PATH> --label <LABEL> --key-type-
class <KEY_TYPE_CLASS>
Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --path <PATH>
          Path where the key is located in PEM format
      --label <LABEL>
          Label for the imported key
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of the imported key [possible values: ec-public, rsa-
public]
      --attributes [<IMPORT_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the imported key
  -h, --help
          Print help
```

Examples

These example shows how to use the **key import pem** command to import an RSA public key from a file in PEM format.

Example Example: Import an RSA public key

```
aws-cloudhsm > key import pem --path /home/example --label example-imported-key --key-
type-class rsa-public
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001e08e3",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "example-imported-key",
        "id": "0x",
        "check-value": "0x99fe93",
        "class": "public-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
```

```
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 512,
        "public-exponent": "0x010001",
        "modulus":
 "0x8e9c172c37aa22ed1ce25f7c3a7c936dadc532201400128b044ebb4b96#··3e4930ab910df5a2896eaeb8853cfe
        "modulus-size-bits": 2048
      }
    },
    "message": "Successfully imported key"
  }
}
```

Example Example: Import an RSA public key with optional attributes

```
aws-cloudhsm > key import pem --path /home/example --label example-imported-key-with-attributes --key-type-class rsa-public --attributes verify=true
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001e08e3",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "example-imported-key-with-attributes",
        "id": "0x",
        "check-value": "0x99fe93",
        "class": "public-key",
```

```
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 512,
        "public-exponent": "0x010001",
        "modulus":
 "0x8e9c172c37aa22ed1ce25f7c3a7c936dadc532201400128b044ebb4b96#··3e4930ab910df5a2896eaeb8853cfe
        "modulus-size-bits": 2048
      }
    },
    "message": "Successfully imported key"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<PATH>

Specifies the file path where the key file is located.

Required: Yes

<LABEL>

Specifies a user defined label for the imported key. The maximum size allowable for `label` is 126 characters.

Required: Yes

<KEY_TYPE_CLASS>

Key type and class of wrapped key.

Possible values:

- ec-public
- rsa-public

Required: Yes

<IMPORT_KEY_ATTRIBUTES>

Specifies a space separated list of key attributes to set for the imported key in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` (for example, `sign=true`). For a list of supported key attributes, see Key attributes for CloudHSM CLI.

Required: No

Related topics

- The crypto sign category in CloudHSM CLI
- The crypto verify category in CloudHSM CLI

List keys for a user with CloudHSM CLI

Use the **key list** command in CloudHSM CLI to find all keys for the current user present in your AWS CloudHSM cluster. The output includes keys that the user owns and shares, as well as all public keys in the CloudHSM cluster.

User type

The following types of users can run this command.

- Admins (COs)
- Crypto users (CUs)

Syntax

```
aws-cloudhsm > help key list
List the keys the current user owns, shares, and all public keys in the HSM cluster

Usage: key list [OPTIONS]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select
 matching key(s) to list
      --max-items <MAX_ITEMS>
          The total number of items to return in the command's output. If the total
 number of items available is more than the value specified, a next-token is provided
 in the command's output. To resume pagination, provide the next-token value in the
 starting-token argument of a subsequent command [default: 10]
      --starting-token <STARTING_TOKEN>
          A token to specify where to start paginating. This is the next-token from a
 previously truncated response
  -v, --verbose
          If included, prints all attributes and key information for each matched key.
 By default each matched key only displays its key-reference and label attribute. This
 flag when used by Admins has no effect
  -h, --help
          Print help
```

Examples

The following examples show the different ways you run the **key list** command. The following examples show the outputs as a crypto user.

Example Example: Find all keys – default

This command lists the keys of the logged in user present in the AWS CloudHSM cluster.

> **Note**
>
> By default, only 10 keys of the currently logged in user are displayed, and only the `key-reference` and `label` are displayed as output. Use the appropriate pagination options to display more or less keys as output.

```
aws-cloudhsm > key list
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000000003d5",
        "attributes": {
          "label": "test_label_1"
        }
      },
      {
        "key-reference": "0x0000000000000626",
        "attributes": {
          "label": "test_label_2"
        }
      },.
      ...8 keys later...
    ],
    "total_key_count": 56,
    "returned_key_count": 10,
    "next_token": "10"
  }
}
```

Example Example: Find all keys – verbose

The output includes keys that the user owns and shares, as well as all public keys in the HSMs.

> **Note**
>
> Note: By default, only 10 keys of the currently logged in user are displayed. Use the appropriate pagination options to display more or less keys as output.

```
aws-cloudhsm > key list --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x000000000012000c",
        "key-info": {
          "key-owners": [
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "ec",
          "label": "ec-test-private-key",
          "id": "",
          "check-value": "0x2a737d",
          "class": "private-key",
          "encrypt": false,
          "decrypt": false,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": false,
          "trusted": false,
          "unwrap": false,
          "verify": false,
```

```
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 122,
          "ec-point":
  "0x0442d53274a6c0ec1a23c165dcb9ccdd72c64e98ae1a9594bb5284e752c746280667e11f1e983493c1c605e0a80
          "curve": "secp224r1"
        }
      },
      {
        "key-reference": "0x000000000012000d",
        "key-info": {
          "key-owners": [
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "ec",
          "label": "ec-test-public-key",
          "id": "",
          "check-value": "0x2a737d",
          "class": "public-key",
          "encrypt": false,
          "decrypt": false,
          "token": true,
          "always-sensitive": false,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": false,
          "sign": false,
          "trusted": false,
```

```
          "unwrap": false,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 57,
          "ec-point":
 "0x0442d53274a6c0ec1a23c165dcb9ccdd72c64e98ae1a9594bb5284e752c746280667e11f1e983493c1c605e0a80
          "curve": "secp224r1"
        }
      }
    ],
      ...8 keys later...
    "total_key_count": 1580,
    "returned_key_count": 10
  }
}
```

Example Example: Paginated return

The following example displays a paginated subset of the keys which shows only two keys. The example then provides a subsequent call to display the next two keys.

```
aws-cloudhsm > key list --verbose --max-items 2
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x0000000000000030",
        "key-info": {
          "key-owners": [
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
          "cluster-coverage": "full"
        },
```

```
      "attributes": {
        "key-type": "aes",
        "label": "98a6688d1d964ed7b45b9cec5c4b1909",
        "id": "",
        "check-value": "0xb28a46",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 32
      }
    },
    {
      "key-reference": "0x0000000000000042",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
      "key-quorum-values": {
        "manage-key-quorum-value": 0,
        "use-key-quorum-value": 0
      },
        "cluster-coverage": "full"
      },
      "attributes": {
```

```
          "key-type": "aes",
          "label": "4ad6cdcbc02044e09fa954143efde233",
          "id": "",
          "check-value": "0xc98104",
          "class": "secret-key",
          "encrypt": true,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": true,
          "wrap": true,
          "wrap-with-trusted": false,
          "key-length-bytes": 16
        }
      }
    ],
    "total_key_count": 1580,
    "returned_key_count": 2,
    "next_token": "2"
  }
}
```

To display the next 2 keys, a subsequent call can be made:

```
aws-cloudhsm > key list --verbose --max-items 2 --starting-token 2
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x0000000000000081",
        "key-info": {
```

```
          "key-owners": [
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "aes",
          "label": "6793b8439d044046982e5b895791e47f",
          "id": "",
          "check-value": "0x3f986f",
          "class": "secret-key",
          "encrypt": false,
          "decrypt": false,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": false,
          "verify": true,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 32
        }
      },
      {
        "key-reference": "0x0000000000000089",
        "key-info": {
          "key-owners": [
```

```
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "aes",
          "label": "56b30fa05c6741faab8f606d3b7fe105",
          "id": "",
          "check-value": "0xe9201a",
          "class": "secret-key",
          "encrypt": false,
          "decrypt": false,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": false,
          "verify": true,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 32
        }
      }
    ],
    "total_key_count": 1580,
    "returned_key_count": 2,
    "next_token": "4"
  }
```

```
}
```

For more examples that demonstrate how the key filtration mechanism works in the CloudHSM CLI, see Filter keys using CloudHSM CLI.

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select matching key(s) to list.

For a listing of supported CloudHSM CLI key attributes, see Key attributes for CloudHSM CLI

Required: No

<MAX_ITEMS>

The total number of items to return in the command's output. If the total number of items available is more than the value specified, a next-token is provided in the command's output. To resume pagination, provide the next-token value in the starting-token argument of a subsequent command.

Required: No

<STARTING_TOKEN>

A token to specify where to start paginating. This is the next-token from a previously truncated response.

Required: No

<VERBOSE>

If included, prints all attributes and key information for each matched key. By default each matched key only displays its key-reference and label attribute. This flag when used by Admins has no effect.

Required: No

Related topics

- Delete a key with CloudHSM CLI
- Export an asymmetric key with CloudHSM CLI
- Unshare a key using CloudHSM CLI
- Key attributes for CloudHSM CLI
- Filter keys using CloudHSM CLI

Replicate a key with CloudHSM CLI

Use the **key replicate** command in CloudHSM CLI to replicate a key from a source AWS CloudHSM cluster to a destination AWS CloudHSM cluster.

User type

The following types of users can run this command.

- Admins (COs)
- Crypto users (CUs)

> **Note**
>
> Crypto Users must own the key to use this command.

Requirements

- The source and destination clusters must be clones. This means one was created from a backup of the other, or they were both created from a common backup. See Creating clusters from backups for more information.
- The owner of the key must exist on the destination cluster. Additionally, if the key is shared with any users, those users must also exist on the destination cluster.
- To run this command, you must be logged in as a crypto user or an admin on both the source and destination clusters.
  - In single command mode, the command will use the CLOUDHSM_PIN and CLOUDHSM_ROLE environmental variables to authenticate on the source cluster. See Single Command mode for more information. To provide credentials for the destination cluster, you need

to set two additional environmental variables: DESTINATION_CLOUDHSM_PIN and DESTINATION_CLOUDHSM_ROLE:

```
$ export DESTINATION_CLOUDHSM_ROLE=<role>
```

```
$ export DESTINATION_CLOUDHSM_PIN=<username:password>
```

- In interactive mode, users will need to explicitly log into both the source and destination clusters.

Syntax

```
aws-cloudhsm > help key replicate
Replicate a key from a source to a destination cluster

Usage: key replicate --filter [<FILTER>...] --source-cluster-id <SOURCE_CLUSTER_ID> --
destination-cluster-id <DESTINATION_CLUSTER_ID>

Options:
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select
 matching key on the source cluster
      --source-cluster-id <SOURCE_CLUSTER_ID>
          Source cluster ID
      --destination-cluster-id <DESTINATION_CLUSTER_ID>
          Destination cluster ID
  -h, --help
          Print help
```

Examples

Example Example: Replicate key

This command replicates a key from a source cluster with to a cloned destination cluster. The example below demonstrates the output when logged in as a crypto user on both clusters.

```
crypto-user-1@cluster-1234abcdefg > key replicate \
      --filter attr.label=example-key \
      --source-cluster-id cluster-1234abcdefg \
      --destination-cluster-id cluster-2345bcdefgh
```

```
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x0000000000300006",
      "key-info": {
        "key-owners": [
          {
            "username": "crypto-user-1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "example-key",
        "id": "0x",
        "check-value": "0x5e118e",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": true,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
```

```
      }
    },
    "message": "Successfully replicated key"
  }
}
```

Arguments

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a matching key on the source cluster.

For a listing of supported CloudHSM CLI key attributes, see Key attributes for CloudHSM CLI

Required: Yes

<SOURCE_CLUSTER_ID>

The source cluster ID.

Required: Yes

<DESTINATION_CLUSTER_ID>

The destination cluster ID.

Required: Yes

Related topics

- Connecting to multiple clusters with CloudHSM CLI

Set the attributes of keys with CloudHSM CLI

Use the **key set-attribute** command in CloudHSM CLI to set the attributes of keys in your AWS CloudHSM cluster. Only the CU who created the key and consequently owns it can change the key's attributes.

For a list of key attributes that can be used in CloudHSM CLI, see Key attributes for CloudHSM CLI.

User type

The following types of users can run this command.

- Crypto users (CUs) can run this command.
- Admins can set the trusted attribute.

Requirements

To run this command, you must be logged in as a CU. To set the trusted attribute, you must be logged in as an admin user.

Syntax

```
aws-cloudhsm > help key set-attribute
Set an attribute for a key in the HSM cluster

Usage: cloudhsm-cli key set-attribute [OPTIONS] --filter [<FILTER>...] --
name <KEY_ATTRIBUTE> --value <KEY_ATTRIBUTE_VALUE>

Options:
      --cluster-id <CLUSTER_ID>         Unique Id to choose which of the clusters in
 the config file to run the operation against. If not provided, will fall back to the
 value provided when interactive mode was started, or error
      --filter [<FILTER>...]            Key reference (e.g. key-
reference=0xabc) or space separated list of key attributes in the form of
 attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a matching key to modify
      --name <KEY_ATTRIBUTE>            Name of attribute to be set
      --value <KEY_ATTRIBUTE_VALUE>...  Attribute value to be set
      --approval <APPROVAL>            Filepath of signed quorum token file to approve
 operation
  -h, --help                            Print help
```

Example: Setting a key attribute

The following example shows how to use the **key set-attribute** command to set the label.

Example

1. Use the key with the label `my_key`, as shown here:

```
aws-cloudhsm > key set-attribute --filter attr.label=my_key --name encrypt --value
 false
{
  "error_code": 0,
  "data": {
    "message": "Attribute set successfully"
  }
}
```

2. Use the **key list** command to confirm the `encrypt` attribute has changed:

```
aws-cloudhsm > key list --filter attr.label=my_key --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000006400ec",
        "key-info": {
          "key-owners": [
            {
              "username": "bob",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "aes",
          "label": "my_key",
          "id": "",
          "check-value": "0x6bd9f7",
          "class": "secret-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
```

```
            "derive": true,
            "destroyable": true,
            "extractable": true,
            "local": true,
            "modifiable": true,
            "never-extractable": false,
            "private": true,
            "sensitive": true,
            "sign": true,
            "trusted": true,
            "unwrap": true,
            "verify": true,
            "wrap": true,
            "wrap-with-trusted": false,
            "key-length-bytes": 32
          }
        }
      ],
      "total_key_count": 1,
      "returned_key_count": 1
    }
  }
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<KEY_ATTRIBUTE>

Specifies the name of the key's attribute.

Required: Yes

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a matching key for deletion.

For a listing of supported CloudHSM CLI key attributes, see Key attributes for CloudHSM CLI

Required: No

<KEY_ATTRIBUTE_VALUE>

Specifies the value of the key's attribute.

Required: Yes

<KEY_REFERENCE>

A hexadecimal or decimal representation of the key. (such as a key handle).

Required: No

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key management service quorum value of the key is greater than 1.

Related topics

- Filter keys using CloudHSM CLI
- Key attributes for CloudHSM CLI

Share a key using CloudHSM CLI

Use the **key share** command in CloudHSM CLI to share a key with other CUs in your AWS CloudHSM cluster.

Only the CU who created the key and consequently owns it can share the key. Users with whom a key is shared can use the key in cryptographic operations, but they cannot delete, export, share, or unshare the key. Additionally, these users cannot change key attributes.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key share
Share a key in the HSM cluster with another user

Usage: key share --filter [<FILTER>...] --username <USERNAME> --role <ROLE>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key for sharing

      --username <USERNAME>
          A username with which the key will be shared

      --role <ROLE>
          Role the user has in the cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation

  -h, --help
          Print help (see a summary with '-h')
```

Example: Share a key with another CU

The following example shows how to use the **key share** command to share a key with the CU `alice`.

Example

1. Run the **key share** command to share the key with `alice`.

```
aws-cloudhsm > key share --filter attr.label="rsa_key_to_share" attr.class=private-key --username alice --role crypto-user
{
  "error_code": 0,
  "data": {
    "message": "Key shared successfully"
  }
}
```

2. Run the **key list** command.

```
aws-cloudhsm > key list --filter attr.label="rsa_key_to_share" attr.class=private-key --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            },
            {
              "username": "cu1",
              "key-coverage": "full"
            },
            {
              "username": "cu4",
              "key-coverage": "full"
            },
            {
              "username": "cu5",
              "key-coverage": "full"
```

```
        },
        {
          "username": "cu6",
          "key-coverage": "full"
        },
        {
          "username": "cu7",
          "key-coverage": "full"
        },
        {
          "username": "alice",
          "key-coverage": "full"
        }
      ],
      "key-quorum-values": {
        "manage-key-quorum-value": 0,
        "use-key-quorum-value": 0
      },
      "cluster-coverage": "full"
    },
    "attributes": {
      "key-type": "rsa",
      "label": "rsa_key_to_share",
      "id": "",
      "check-value": "0xae8ff0",
      "class": "private-key",
      "encrypt": false,
      "decrypt": true,
      "token": true,
      "always-sensitive": true,
      "derive": false,
      "destroyable": true,
      "extractable": true,
      "local": true,
      "modifiable": true,
      "never-extractable": false,
      "private": true,
      "sensitive": true,
      "sign": true,
      "trusted": false,
      "unwrap": true,
      "verify": false,
      "wrap": false,
      "wrap-with-trusted": false,
```

```
          "key-length-bytes": 1219,
          "public-exponent": "0x010001",
          "modulus":
 "0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```

3. In the above list, verify `alice` is in the list of `shared-users`

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a matching key for deletion.

For a list of supported key attributes, see Key attributes for CloudHSM CLI.

Required: Yes

<USERNAME>

Specifies a friendly name for the user. The maximum length is 31 characters. The only special character permitted is an underscore (_). The username is not case sensitive in this command, username is always displayed in lowercase.

Required: Yes

<ROLE>

Specifies the role assigned to this user. This parameter is required. To get the user's role, use the user list command. For detailed information about the user types on an HSM, see [HSM user types for CloudHSM CLI](#).

Required: Yes

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key management service quorum value of the key is greater than 1.

Related topics

- [Filter keys using CloudHSM CLI](#)
- [Key attributes for CloudHSM CLI](#)

Unshare a key using CloudHSM CLI

Use the **key unshare** command in CloudHSM CLI to unshare a key with other CUs in your AWS CloudHSM cluster.

Only the CU who created the key and consequently owns it can unshare the key. Users with whom a key is shared can use the key in cryptographic operations, but they cannot delete, export, share, or unshare the key. Additionally, these users cannot change [key attributes](#).

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key unshare
Unshare a key in the HSM cluster with another user

```

```
Usage: key unshare --filter [<FILTER>...] --username <USERNAME> --role <ROLE>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 matching key for unsharing

      --username <USERNAME>
          A username with which the key will be unshared

      --role <ROLE>
          Role the user has in the cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation

  -h, --help
          Print help (see a summary with '-h')
```

Example: Unshare a key with another CU

The following example shows how to use the **key unshare** command to unshare a key with the CU `alice`.

Example

1. Run the **key list** command and filter by the specific key you want to unshare with `alice`.

```
aws-cloudhsm > key list --filter attr.label="rsa_key_to_share" attr.class=private-
key --verbose
{
  "error_code": 0,
  "data": {
```

```
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            },
            {
              "username": "cu1",
              "key-coverage": "full"
            },
            {
              "username": "cu4",
              "key-coverage": "full"
            },
            {
              "username": "cu5",
              "key-coverage": "full"
            },
            {
              "username": "cu6",
              "key-coverage": "full"
            },
            {
              "username": "cu7",
              "key-coverage": "full"
            },
            {
              "username": "alice",
              "key-coverage": "full"
            }
          ],
          "key-quorum-values": {
            "manage-key-quorum-value": 0,
            "use-key-quorum-value": 0
          },
```

```
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "rsa",
          "label": "rsa_key_to_share",
          "id": "",
          "check-value": "0xae8ff0",
          "class": "private-key",
          "encrypt": false,
          "decrypt": true,
          "token": true,
          "always-sensitive": true,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": true,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": true,
          "verify": false,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 1219,
          "public-exponent": "0x010001",
          "modulus":
 "0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254
          "modulus-size-bits": 2048
        }
      }
    ],
    "total_key_count": 1,
    "returned_key_count": 1
  }
}
```

2. Confirm `alice` is in the `shared-users` output, and run the following **key unshare** command to unshare the key with `alice`.

```
aws-cloudhsm > key unshare --filter attr.label="rsa_key_to_share"
 attr.class=private-key --username alice --role crypto-user
{
  "error_code": 0,
  "data": {
    "message": "Key unshared successfully"
  }
}
```

3. Run the `key list` command again to confirm that the key has been unshared with `alice`.

```
aws-cloudhsm > key list --filter attr.label="rsa_key_to_share" attr.class=private-
key --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x00000000001c0686",
        "key-info": {
          "key-owners": [
            {
              "username": "cu3",
              "key-coverage": "full"
            }
          ],
          "shared-users": [
            {
              "username": "cu2",
              "key-coverage": "full"
            },
            {
              "username": "cu1",
              "key-coverage": "full"
            },
            {
              "username": "cu4",
              "key-coverage": "full"
            },
            {
              "username": "cu5",
              "key-coverage": "full"
```

```
           },
           {
             "username": "cu6",
             "key-coverage": "full"
           },
           {
             "username": "cu7",
             "key-coverage": "full"
           },
         ],
         "key-quorum-values": {
           "manage-key-quorum-value": 0,
           "use-key-quorum-value": 0
         },
         "cluster-coverage": "full"
       },
       "attributes": {
         "key-type": "rsa",
         "label": "rsa_key_to_share",
         "id": "",
         "check-value": "0xae8ff0",
         "class": "private-key",
         "encrypt": false,
         "decrypt": true,
         "token": true,
         "always-sensitive": true,
         "derive": false,
         "destroyable": true,
         "extractable": true,
         "local": true,
         "modifiable": true,
         "never-extractable": false,
         "private": true,
         "sensitive": true,
         "sign": true,
         "trusted": false,
         "unwrap": true,
         "verify": false,
         "wrap": false,
         "wrap-with-trusted": false,
         "key-length-bytes": 1219,
         "public-exponent": "0x010001",
         "modulus":
"0xa8855cba933cec0c21a4df0450ec31675c024f3e65b2b215a53d2bda6dcd191f75729150b59b4d86df58254
```

```
            "modulus-size-bits": 2048
          }
        }
      ],
      "total_key_count": 1,
      "returned_key_count": 1
    }
}
```


Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a matching key for deletion.

For a list of supported key attributes, see Key attributes for CloudHSM CLI.

Required: Yes

<USERNAME>

Specifies a friendly name for the user. The maximum length is 31 characters. The only special character permitted is an underscore (_). The username is not case sensitive in this command, username is always displayed in lowercase.

Required: Yes

<ROLE>

Specifies the role assigned to this user. This parameter is required. To get the user's role, use the user list command. For detailed information about the user types on an HSM, see HSM user types for CloudHSM CLI.

Required: Yes

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key management service quorum value of the key is greater than 1.

Related topics

- Filter keys using CloudHSM CLI
- Key attributes for CloudHSM CLI

The key unwrap command in CloudHSM CLI

The **key unwrap** parent command in CloudHSM CLI imports an encrypted (wrapped) symmetric or asymmetric private key from a file and into the HSM. This command is designed to import encrypted keys that were wrapped by the The key wrap command in CloudHSM CLI command, but it can also be used to unwrap keys that were wrapped with other tools. However, in those situations, we recommend using the PKCS#11 or JCE software libraries to unwrap the key.

- aes-gcm
- aes-no-pad
- aes-pkcs5-pad
- aes-zero-pad
- cloudhsm-aes-gcm
- rsa-aes
- rsa-oaep
- rsa-pkcs

Unwrap a key with AES-GCM using CloudHSM CLI

Use the **key unwrap aes-gcm** command in CloudHSM CLI to unwrap a payload key into the cluster using the AES wrapping key and the `AES-GCM` unwrapping mechanism.

Unwrapped keys can be used in the same ways as the keys generated by AWS CloudHSM. To indicate that they were not generated locally, their `local` attribute is set to `false`.

To use the **key unwrap aes-gcm** command, you must have the AES wrapping key in your AWS CloudHSM cluster, and its `unwrap` attribute must be set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key unwrap aes-gcm
Usage: key unwrap aes-gcm [OPTIONS] --filter [<FILTER>...] --tag-length-
bits <TAG_LENGTH_BITS> --key-type-class <KEY_TYPE_CLASS> --label <LABEL> --iv <IV> <--
data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --share-crypto-users [<SHARE_CRYPTO_USERS;...]
          Space separated list of Crypto User usernames to share the unwrapped key with
      --manage-key-quorum-value <MANAGE_KEY_QUORUM_VALUE;
          The quorum value for key management operations for the unwrapped key
      --use-key-quorum-value <USE_KEY_QUORUM_VALUE;
          The quorum value for key usage operations for the unwrapped key
      --aad <AAD>
          Aes GCM Additional Authenticated Data (AAD) value, in hex
      --tag-length-bits <TAG_LENGTH_BITS>
```

```
          Aes GCM tag length in bits
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
 generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
      --iv <IV>
          Initial value used to wrap the key, in hex
      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation
  -h, --help
          Print help
```

Examples

These examples show how to use the **key unwrap aes-gcm** command using an AES key with the `unwrap` attribute value set to `true`.

Example Example: Unwrap a payload key from Base64 encoded wrapped key data

```
aws-cloudhsm > key unwrap aes-gcm --key-type-class aes --label aes-unwrapped
 --filter attr.label=aes-example --tag-length-bits 64  --aad 0x10 --iv
 0xf90613bb8e337ec0339aad21 --data xvslgrtg8kHzrvekny97tLSIeokpPwV8
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001808e4",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
```

```
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example Example: Unwrap a payload key provided through a data path

```
aws-cloudhsm > key unwrap aes-gcm --key-type-class aes --label aes-unwrapped
 --filter attr.label=aes-example --tag-length-bits 64  --aad 0x10 --iv
 0xf90613bb8e337ec0339aad21 --data-path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001808e4",
      "key-info": {
```

```
          "key-owners": [
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
          "key-quorum-values": {
            "manage-key-quorum-value": 0,
            "use-key-quorum-value": 0
          },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "aes",
          "label": "aes-unwrapped",
          "id": "0x",
          "check-value": "0x8d9099",
          "class": "secret-key",
          "encrypt": false,
          "decrypt": false,
          "token": true,
          "always-sensitive": false,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": false,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": false,
          "verify": true,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 16
        }
      }
    }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a key to unwrap with.

Required: Yes

<DATA_PATH>

Path to the binary file containing the wrapped key data.

Required: Yes (unless provided through Base64 encoded data)

<DATA>

Base64 encoded wrapped key data.

Required: Yes (unless provided through data path)

<ATTRIBUTES>

Space separated list of key attributes in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` for the wrapped key.

Required: No

<AAD>

Aes GCM Additional Authenticated Data (AAD) value, in hex.

Required: No

<TAG_LENGTH_BITS>

Aes GCM tag length in bits.

Required: Yes

<KEY_TYPE_CLASS>

Key type and class of wrapped key [possible values: `aes`, `des3`, `ec-private`, `generic-secret`, `rsa-private`].

Required: Yes

<LABEL>

Label for the unwrapped key.

Required: Yes

<SESSION>

Creates a session key that exists only in the current session. The key cannot be recovered after the session ends.

Required: No

<IV>

Initial value used to wrap the key, in hex.

Required: No

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key management service quorum value of the unwrapping key is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Unwrap a key with AES-NO-PAD using CloudHSM CLI

Use the **key unwrap aes-no-pad** command in CloudHSM CLI to unwrap a payload key into the AWS CloudHSM cluster using the AES wrapping key and the `AES-NO-PAD` unwrapping mechanism.

Unwrapped keys can be used in the same ways as the keys generated by AWS CloudHSM. To indicate that they were not generated locally, their `local` attribute is set to `false`.

To use the **key unwrap aes-no-pad** command, you must have the AES wrapping key in your AWS CloudHSM cluster, and its `unwrap` attribute must be set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key unwrap aes-no-pad
Usage: key unwrap aes-no-pad [OPTIONS] --filter [<FILTER>...] --key-type-
class <KEY_TYPE_CLASS> --label <LABEL> <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --share-crypto-users [<SHARE_CRYPTO_USERS;...]
          Space separated list of Crypto User usernames to share the unwrapped key with
      --manage-key-quorum-value <MANAGE_KEY_QUORUM_VALUE;
          The quorum value for key management operations for the unwrapped key
      --use-key-quorum-value <USE_KEY_QUORUM_VALUE;
          The quorum value for key usage operations for the unwrapped key
      --key-type-class <KEY_TYPE_CLASS>
```

```
         Key type and class of wrapped key [possible values: aes, des3, ec-private,
generic-secret, rsa-private]
     --label <LABEL>
         Label for the unwrapped key
     --session
         Creates a session key that exists only in the current session. The key cannot
be recovered after the session ends
     --approval <APPROVAL>
         Filepath of signed quorum token file to approve operation
 -h, --help
         Print help
```

Examples

These examples show how to use the **key unwrap aes-no-pad** command using an AES key with the `unwrap` attribute value set to `true`.

Example Example: Unwrap a payload key from Base64 encoded wrapped key data

```
aws-cloudhsm > key unwrap aes-no-pad --key-type-class aes --label aes-unwrapped --
filter attr.label=aes-example --data eXK3PMAOnKM9y3YX6brbhtMoC060EOH9
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08ec",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
```

```
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example Example: Unwrap a payload key provided through a data path

```
aws-cloudhsm > key unwrap aes-no-pad --key-type-class aes --label aes-unwrapped --
filter attr.label=aes-example --data-path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08ec",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
```

```
          "shared-users": [],
          "key-quorum-values": {
            "manage-key-quorum-value": 0,
            "use-key-quorum-value": 0
          },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "aes",
          "label": "aes-unwrapped",
          "id": "0x",
          "check-value": "0x8d9099",
          "class": "secret-key",
          "encrypt": false,
          "decrypt": false,
          "token": true,
          "always-sensitive": false,
          "derive": false,
          "destroyable": true,
          "extractable": true,
          "local": false,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": false,
          "verify": true,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 16
        }
      }
    }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a key to unwrap with.

Required: Yes

<DATA_PATH>

Path to the binary file containing the wrapped key data.

Required: Yes (unless provided through Base64 encoded data)

<DATA>

Base64 encoded wrapped key data.

Required: Yes (unless provided through data path)

<ATTRIBUTES>

Space separated list of key attributes in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` for the wrapped key.

Required: No

<KEY_TYPE_CLASS>

Key type and class of wrapped key [possible values: `aes`, `des3`, `ec-private`, `generic-secret`, `rsa-private`].

Required: Yes

<LABEL>

Label for the unwrapped key.

Required: Yes

<SESSION>

Creates a session key that exists only in the current session. The key cannot be recovered after the session ends.

Required: No

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key management service quorum value of the unwrapping key is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Unwrap a key with AES-PKCS5-PAD using CloudHSM CLI

Use the **key unwrap aes-pkcs5-pad** command in CloudHSM CLI to unwrap a payload key using the AES wrapping key and the `AES-PKCS5-PAD` unwrapping mechanism.

Unwrapped keys can be used in the same ways as the keys generated by AWS CloudHSM. To indicate that they were not generated locally, their `local` attribute is set to `false`.

To use the **key unwrap aes-pkcs5-pad** command, you must have the AES wrapping key in your AWS CloudHSM cluster, and its `unwrap` attribute must be set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key unwrap aes-pkcs5-pad
Usage: key unwrap aes-pkcs5-pad [OPTIONS] --filter [<FILTER>...] --key-type-
class <KEY_TYPE_CLASS> --label <LABEL> <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
```

```
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --share-crypto-users [<SHARE_CRYPTO_USERS;...]
          Space separated list of Crypto User usernames to share the unwrapped key with
      --manage-key-quorum-value <MANAGE_KEY_QUORUM_VALUE;
          The quorum value for key management operations for the unwrapped key
      --use-key-quorum-value <USE_KEY_QUORUM_VALUE;
          The quorum value for key usage operations for the unwrapped key
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
 generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation
  -h, --help
          Print help
```

Examples

These examples show how to use the **key unwrap aes-pkcs5-pad** command using an AES key with the `unwrap` attribute value set to `true`.

Example Example: Unwrap a payload key from Base64 encoded wrapped key data

```
aws-cloudhsm > key unwrap aes-pkcs5-pad --key-type-class aes --label aes-unwrapped --
filter attr.label=aes-example --data MbuYNresfOKyGNnxKWen88nSfX+uUE/0qmGofSisicY=
{
```

```
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08e3",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
```

```
    }
  }
}
```

Example Example: Unwrap a payload key provided through a data path

```
aws-cloudhsm > key unwrap aes-pkcs5-pad --key-type-class aes --label aes-unwrapped --
filter attr.label=aes-example --data-path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08e3",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
```

```
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": false,
          "verify": true,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 16
        }
      }
    }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a key to unwrap with.

Required: Yes

<DATA_PATH>

Path to the binary file containing the wrapped key data.

Required: Yes (unless provided through Base64 encoded data)

<DATA>

Base64 encoded wrapped key data.

Required: Yes (unless provided through data path)

<ATTRIBUTES>

Space separated list of key attributes in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` for the wrapped key.

Required: No

<KEY_TYPE_CLASS>

Key type and class of wrapped key [possible values: `aes`, `des3`, `ec-private`, `generic-secret`, `rsa-private`].

Required: Yes

<LABEL>

Label for the unwrapped key.

Required: Yes

<SESSION>

Creates a session key that exists only in the current session. The key cannot be recovered after the session ends.

Required: No

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key management service quorum value of the unwrapping key is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Unwrap a key with AES-ZERO-PAD using CloudHSM CLI

Use the **key unwrap aes-zero-pad** command in CloudHSM CLI to unwrap a payload key into the AWS CloudHSM cluster using the AES wrapping key and the `AES-ZERO-PAD` unwrapping mechanism.

Unwrapped keys can be used in the same ways as the keys generated by AWS CloudHSM. To indicate that they were not generated locally, their `local` attribute is set to `false`.

To use the **key unwrap aes-no-pad** command, you must have the AES wrapping key in your AWS CloudHSM cluster, and its `unwrap` attribute must be set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key unwrap aes-zero-pad
Usage: key unwrap aes-zero-pad [OPTIONS] --filter [<FILTER>...] --key-type-
class <KEY_TYPE_CLASS> --label <LABEL> <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --share-crypto-users [<SHARE_CRYPTO_USERS;...]
          Space separated list of Crypto User usernames to share the unwrapped key with
      --manage-key-quorum-value <MANAGE_KEY_QUORUM_VALUE;
          The quorum value for key management operations for the unwrapped key
      --use-key-quorum-value <USE_KEY_QUORUM_VALUE;
          The quorum value for key usage operations for the unwrapped key
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
 generic-secret, rsa-private]
      --label <LABEL>
```

```
            Label for the unwrapped key
        --session
            Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
        --approval <APPROVAL>
            Filepath of signed quorum token file to approve operation
  -h, --help
            Print help
```

Examples

These examples show how to use the **key unwrap aes-zero-pad** command using an AES key with the `unwrap` attribute value set to `true`.

Example Example: Unwrap a payload key from Base64 encoded wrapped key data

```
aws-cloudhsm > key unwrap aes-zero-pad --key-type-class aes --label aes-unwrapped --
filter attr.label=aes-example --data L1wVlL/YeBNVAw6Mpk3owFJZXBzDLONt
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08e7",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
```

```
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example Example: Unwrap a payload key provided through a data path

```
aws-cloudhsm > key unwrap aes-zero-pad --key-type-class aes --label aes-unwrapped --
filter attr.label=aes-example --data-path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08e7",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
```

```
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a key to unwrap with.

Required: Yes

<DATA_PATH>

Path to the binary file containing the wrapped key data.

Required: Yes (unless provided through Base64 encoded data)

<DATA>

Base64 encoded wrapped key data.

Required: Yes (unless provided through data path)

<ATTRIBUTES>

Space separated list of key attributes in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` for the wrapped key.

Required: No

<KEY_TYPE_CLASS>

Key type and class of wrapped key [possible values: `aes`, `des3`, `ec-private`, `generic-secret`, `rsa-private`].

Required: Yes

<LABEL>

Label for the unwrapped key.

Required: Yes

<SESSION>

Creates a session key that exists only in the current session. The key cannot be recovered after the session ends.

Required: No

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key management service quorum value of the unwrapping key is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Unwrap a key with CLOUDHSM-AES-GCM using CloudHSM CLI

Use the **key unwrap cloudhsm-aes-gcm** command in CloudHSM CLI to unwrap a payload key into the AWS CloudHSM cluster using the AES wrapping key and the `CLOUDHSM-AES-GCM` unwrapping mechanism.

Unwrapped keys can be used in the same ways as the keys generated by AWS CloudHSM. To indicate that they were not generated locally, their `local` attribute is set to `false`.

To use the **key unwrap cloudhsm-aes-gcm** command, you must have the AES wrapping key in your AWS CloudHSM cluster and its `unwrap` attribute must be set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key unwrap cloudhsm-aes-gcm
Usage: key unwrap cloudhsm-aes-gcm [OPTIONS] --filter [<FILTER>...] --tag-length-
bits <TAG_LENGTH_BITS> --key-type-class <KEY_TYPE_CLASS> --label <LABEL> <--data-
path <DATA_PATH>|--data <DATA>>
```

```
Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --share-crypto-users [<SHARE_CRYPTO_USERS;...]
          Space separated list of Crypto User usernames to share the unwrapped key with
      --manage-key-quorum-value <MANAGE_KEY_QUORUM_VALUE;
          The quorum value for key management operations for the unwrapped key
      --use-key-quorum-value <USE_KEY_QUORUM_VALUE;
          The quorum value for key usage operations for the unwrapped key
      --aad <AAD>
          Aes GCM Additional Authenticated Data (AAD) value, in hex
      --tag-length-bits <TAG_LENGTH_BITS>
          Aes GCM tag length in bits
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
 generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation
  -h, --help
          Print help
```

Examples

These examples show how to use the **key unwrap cloudhsm-aes-gcm** command using an AES key with the `unwrap` attribute value set to `true`.

Example Example: Unwrap a payload key from Base64 encoded wrapped key data

```
aws-cloudhsm > key unwrap cloudhsm-aes-gcm --key-type-class aes --label aes-unwrapped --filter attr.label=aes-example --tag-length-bits 64  --aad 0x10 --data 6Rn8nkjEriDYlnP3P8nPkYQ8hplOEJ899zsrF+aTB0i/fIlZ
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001408e8",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
```

```
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example Example: Unwrap a payload key provided through a data path

```
aws-cloudhsm > key unwrap cloudhsm-aes-gcm --key-type-class aes --label aes-unwrapped
 --filter attr.label=aes-example --tag-length-bits 64  --aad 0x10 --data-path payload-
key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001408e8",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
```

```
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a key to unwrap with.

Required: Yes

<DATA_PATH>

Path to the binary file containing the wrapped key data.

Required: Yes (unless provided through Base64 encoded data)

<DATA>

Base64 encoded wrapped key data.

Required: Yes (unless provided through data path)

<ATTRIBUTES>

Space separated list of key attributes in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` for the wrapped key.

Required: No

<AAD>

Aes GCM Additional Authenticated Data (AAD) value, in hex.

Required: No

<TAG_LENGTH_BITS>

Aes GCM tag length in bits.

Required: Yes

<KEY_TYPE_CLASS>

Key type and class of wrapped key [possible values: `aes`, `des3`, `ec-private`, `generic-secret`, `rsa-private`].

Required: Yes

<LABEL>

Label for the unwrapped key.

Required: Yes

<SESSION>

Creates a session key that exists only in the current session. The key cannot be recovered after the session ends.

Required: No

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key management service quorum value of the unwrapping key is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Unwrap a key with RSA-AES using CloudHSM CLI

Use the **key unwrap rsa-aes** command in CloudHSM CLI to unwrap a payload key using an RSA private key and the `RSA-AES` unwrapping mechanism.

Unwrapped keys can be used in the same ways as the keys generated by AWS CloudHSM. To indicate that they were not generated locally, their `local` attribute is set to `false`.

To use the **key unwrap rsa-aes**, you must have the RSA private key of the RSA public wrapping key in your AWS CloudHSM cluster, and its `unwrap` attribute must be set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key unwrap rsa-aes
Usage: key unwrap rsa-aes [OPTIONS] --filter [<FILTER>...] --hash-
function <HASH_FUNCTION> --mgf <MGF> --key-type-class <KEY_TYPE_CLASS> --label <LABEL>
 <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
```

```
          Key reference (e.g. key-reference=0xabc) or space separated list of key
attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --share-crypto-users [<SHARE_CRYPTO_USERS;...]
          Space separated list of Crypto User usernames to share the unwrapped key with
      --manage-key-quorum-value <MANAGE_KEY_QUORUM_VALUE;
          The quorum value for key management operations for the unwrapped key
      --use-key-quorum-value <USE_KEY_QUORUM_VALUE;
          The quorum value for key usage operations for the unwrapped key
      --hash-function <HASH_FUNCTION>
          Hash algorithm [possible values: sha1, sha224, sha256, sha384, sha512]
      --mgf <MGF>
          Mask Generation Function algorithm [possible values: mgf1-sha1, mgf1-sha224,
mgf1-sha256, mgf1-sha384, mgf1-sha512]
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
be recovered after the session ends
      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation
  -h, --help
          Print help
```

Example

These examples show how to use the **key unwrap rsa-aes** command using the RSA private key with the `unwrap` attribute value set to `true`.

Example Example: Unwrap a payload key from Base64 encoded wrapped key data

```
aws-cloudhsm > key unwrap rsa-aes --key-type-class aes --label aes-unwrapped
 --filter attr.label=rsa-private-key-example --hash-function sha256 --
```

```
mgf mgf1-sha256 --data HrSE1DEyLjIeyGdPa9R+ebiqB5TIJGyamPker31ZebPwRA
+NcerbAJO8DJ1lXPygZcI21vIFSZJuWMEiWpe1R9D/5WSYgxLVKex30xCFqebtEzxbKuv4DOmU4meSofqREYvtb3EoIKwjy
+RL5WGXKe4nAboAkC5GO7veI5yHL1SaKlssSJtTL/CFpbSLsAFuYbv/NUCWwMY5mwyVTCSlw+HlgKK
+5TH1MzBaSi8fpfyepLT8sHy2Q/VRl6ifb49p6m0KQFbRVvz/OWUd6l4d97BdgtaEz6ueg==
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001808e2",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
```

```
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example Example: Unwrap a payload key provided through a data path

```
aws-cloudhsm > key unwrap rsa-aes --key-type-class aes --label aes-unwrapped --filter
 attr.label=rsa-private-key-example --hash-function sha256 --mgf mgf1-sha256 --data-
path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001808e2",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
```

```
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```


Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been [configured.](#)

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a key to unwrap with.

Required: Yes

<DATA_PATH>

Path to the binary file containing the wrapped key data.

Required: Yes (unless provided through Base64 encoded data)

<DATA>

Base64 encoded wrapped key data.

Required: Yes (unless provided through data path)

<ATTRIBUTES>

Space separated list of key attributes in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` for the wrapped key.

Required: No

<KEY_TYPE_CLASS>

Key type and class of wrapped key [possible values: `aes`, `des3`, `ec-private`, `generic-secret`, `rsa-private`].

Required: Yes

<HASH_FUNCTION>

Specifies the hash function.

Valid values:

- sha1
- sha224
- sha256
- sha384
- sha512

Required: Yes

<MGF>

Specifies the mask generation function.

> **Note**
>
> The mask generation function hash function must match the signing mechanism hash function.

Valid values:

- mgf1-sha1

- mgf1-sha224
- mgf1-sha256
- mgf1-sha384
- mgf1-sha512

Required: Yes

<LABEL>

Label for the unwrapped key.

Required: Yes

<SESSION>

Creates a session key that exists only in the current session. The key cannot be recovered after the session ends.

Required: No

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key management service quorum value of the unwrapping key is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Unwrap a key with RSA-OAEP using CloudHSM CLI

Use the **key unwrap rsa-oaep** command in CloudHSM CLI to unwrap a payload key using the RSA private key and the `RSA-OAEP` unwrapping mechanism.

Unwrapped keys can be used in the same ways as the keys generated by AWS CloudHSM. To indicate that they were not generated locally, their `local` attribute is set to `false`.

To use the **key unwrap rsa-oaep** command, you must have the RSA private key of the RSA public wrapping key in your AWS CloudHSM cluster, and its `unwrap` attribute must be set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key unwrap rsa-oaep
Usage: key unwrap rsa-oaep [OPTIONS] --filter [<FILTER>...] --hash-
function <HASH_FUNCTION> --mgf <MGF> --key-type-class <KEY_TYPE_CLASS> --label <LABEL>
 <--data-path <DATA_PATH>|--data <DATA>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --share-crypto-users [<SHARE_CRYPTO_USERS;...]
          Space separated list of Crypto User usernames to share the unwrapped key with
      --manage-key-quorum-value <MANAGE_KEY_QUORUM_VALUE;
          The quorum value for key management operations for the unwrapped key
      --use-key-quorum-value <USE_KEY_QUORUM_VALUE;
          The quorum value for key usage operations for the unwrapped key
      --hash-function <HASH_FUNCTION>
          Hash algorithm [possible values: sha1, sha224, sha256, sha384, sha512]
      --mgf <MGF>
```

```
          Mask Generation Function algorithm [possible values: mgf1-sha1, mgf1-sha224,
 mgf1-sha256, mgf1-sha384, mgf1-sha512]
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
 generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation
  -h, --help
          Print help
```

Examples

These examples show how to use the **key unwrap rsa-oaep** command using the RSA private key with the `unwrap` attribute value set to `true`.

Example Example: Unwrap a payload key from Base64 encoded wrapped key data

```
aws-cloudhsm > key unwrap rsa-oaep --key-type-class aes --label aes-unwrapped --filter
 attr.label=rsa-private-example-key --hash-function sha256 --mgf mgf1-sha256 --data
 OjJe4msobPLz9TuSAdULEu17T5rMDWtSlLyBSkLbaZnYzzpdrhsbGLbwZJCtB/jGkDNdB4qyTAOQwEpggGf6v
+Yx6JcesNeKKNU8XZal/YBoHC8noTGUSDI2qr+u2tDc84NPv6d+F2KOONXsSxMhmxzzNG/
gzTVIJhOuy/B1yHjGP4mOXoDZf5+7f5M1CjxBmz4Vva/wrWHGCSG0yOaWblEvOiHAIt3UBdyKmU+/
My4xjfJv7WGGu3DFUUIZ06TihRtKQhUYU1M9u6NPf9riJJfHsk6QCuSZ9yWThDT9as6i7e3htnyDhIhGWaoK8JU855cN/
YNKAUqkNpC4FPL3iw==
{
  "data": {
    "key": {
      "key-reference": "0x00000000001808e9",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
```

```
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example Example: Unwrap a payload key provided through a data path

```
aws-cloudhsm > key unwrap rsa-oaep --key-type-class aes --label aes-unwrapped --filter
 attr.label=rsa-private-example-key --hash-function sha256 --mgf mgf1-sha256 --data-
path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
```

```
      "key-reference": "0x00000000001808e9",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
```

```
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a key to unwrap with.

Required: Yes

<DATA_PATH>

Path to the binary file containing the wrapped key data.

Required: Yes (unless provided through Base64 encoded data)

<DATA>

Base64 encoded wrapped key data.

Required: Yes (unless provided through data path)

<ATTRIBUTES>

Space separated list of key attributes in the form of `KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` for the wrapped key.

Required: No

<KEY_TYPE_CLASS>

Key type and class of wrapped key [possible values: `aes`, `des3`, `ec-private`, `generic-secret`, `rsa-private`].

Required: Yes

<HASH_FUNCTION>

Specifies the hash function.

Valid values:

- sha1
- sha224
- sha256
- sha384
- sha512

Required: Yes

<MGF>

Specifies the mask generation function.

> **Note**
> The mask generation function hash function must match the signing mechanism hash function.

Valid values:

- mgf1-sha1
- mgf1-sha224
- mgf1-sha256
- mgf1-sha384
- mgf1-sha512

Required: Yes

<LABEL>

Label for the unwrapped key.

Required: Yes

<SESSION>

Creates a session key that exists only in the current session. The key cannot be recovered after the session ends.

Required: No

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key management service quorum value of the unwrapping key is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Unwrap a key with RSA-PKCS using CloudHSM CLI

Use the **key unwrap rsa-pkcs** command in CloudHSM CLI to unwrap a payload key using the RSA private key and the `RSA-PKCS` unwrapping mechanism.

Unwrapped keys can be used in the same ways as the keys generated by AWS CloudHSM. To indicate that they were not generated locally, their `local` attribute is set to `false`.

To use the key **unwrap rsa-pkcs** command, you must have the RSA private key of the RSA public wrapping key in your AWS CloudHSM cluster, and its `unwrap` attribute must be set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key unwrap rsa-pkcs
Usage: key unwrap rsa-pkcs [OPTIONS] --filter [<FILTER>...] --key-type-
class <KEY_TYPE_CLASS> --label <LABEL> <--data-path <DATA_PATH>|--data <DATA>>

Options:
      --cluster-id <CLUSTER_ID>
```

```
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --filter [<FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a key
 to unwrap with
      --data-path <DATA_PATH>
          Path to the binary file containing the wrapped key data
      --data <DATA>
          Base64 encoded wrapped key data
      --attributes [<UNWRAPPED_KEY_ATTRIBUTES>...]
          Space separated list of key attributes in the form of
 KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the unwrapped key
      --share-crypto-users [<SHARE_CRYPTO_USERS;...]
          Space separated list of Crypto User usernames to share the unwrapped key with
      --manage-key-quorum-value <MANAGE_KEY_QUORUM_VALUE;
          The quorum value for key management operations for the unwrapped key
      --use-key-quorum-value <USE_KEY_QUORUM_VALUE;
          The quorum value for key usage operations for the unwrapped key
      --key-type-class <KEY_TYPE_CLASS>
          Key type and class of wrapped key [possible values: aes, des3, ec-private,
 generic-secret, rsa-private]
      --label <LABEL>
          Label for the unwrapped key
      --session
          Creates a session key that exists only in the current session. The key cannot
 be recovered after the session ends
      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation
  -h, --help
          Print help
```

Examples

These examples show how to use the **key unwrap rsa-oaep** command using an AES key with the `unwrap` attribute value set to `true`.

Example Example: Unwrap a payload key from Base64 encoded wrapped key data

```
aws-cloudhsm > key unwrap rsa-pkcs --key-type-class aes --label
 aes-unwrapped --filter attr.label=rsa-private-key-example --data
 am0Nc7+YE8FWs+5HvU7sIBcXVb24QA0l65nbNAD+1bK+e18BpSfnaI3P+r8Dp+pLu1ofoUy/
```

```
vtzRjZoCiDofcz4EqCFnGl4GdcJ1/3W/5WRvMatCa2d7cx02swaeZcjKsermPXYRO1lGlfq6NskwMeeTkV8R7Rx9artFrs1
c3XdFJ2+0Bo94c6og/
yfPcpOOobJlITCoXhtMRepSdO4OggYq/6nUDuHCtJ86pPGnNahyr7+sAaSI3a5ECQLUjwaIARUCyoRh7EFK3qPXcg==
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08ef",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": false,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": true,
```

```
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 16
      }
    }
  }
}
```

Example Example: Unwrap a payload key provided through a data path

```
aws-cloudhsm > key unwrap rsa-pkcs --key-type-class aes --label aes-unwrapped --filter
 attr.label=rsa-private-key-example --data-path payload-key.pem
{
  "error_code": 0,
  "data": {
    "key": {
      "key-reference": "0x00000000001c08ef",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "aes",
        "label": "aes-unwrapped",
        "id": "0x",
        "check-value": "0x8d9099",
        "class": "secret-key",
        "encrypt": false,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
```

```
          "extractable": true,
          "local": false,
          "modifiable": true,
          "never-extractable": false,
          "private": true,
          "sensitive": true,
          "sign": true,
          "trusted": false,
          "unwrap": false,
          "verify": true,
          "wrap": false,
          "wrap-with-trusted": false,
          "key-length-bytes": 16
        }
      }
    }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a key to unwrap with.

Required: Yes

<DATA_PATH>

Path to the binary file containing the wrapped key data.

Required: Yes (unless provided through Base64 encoded data)

<DATA>

Base64 encoded wrapped key data.

Required: Yes (unless provided through data path)

<ATTRIBUTES>

Space separated list of key attributes in the form of KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE for the wrapped key.

Required: No

<KEY_TYPE_CLASS>

Key type and class of wrapped key [possible values: `aes`, `des3`, `ec-private`, `generic-secret`, `rsa-private`].

Required: Yes

<LABEL>

Label for the unwrapped key.

Required: Yes

<SESSION>

Creates a session key that exists only in the current session. The key cannot be recovered after the session ends.

Required: No

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if the key management service quorum value of the unwrapping key is greater than 1.

Related topics

- [The key wrap command in CloudHSM CLI]
- [The key unwrap command in CloudHSM CLI]

The key wrap command in CloudHSM CLI

The **key wrap** command in CloudHSM CLI exports an encrypted copy of a symmetric or asymmetric private key from the hardware security module (HSM) to a file. When you run **key wrap**, you specify two things: The key to export and the output file. The key to export is a key on the HSM that will encrypt (wrap) the key that you want to export.

The **key wrap** command does not remove the key from the HSM or prevent you from using it in cryptographic operations. You can export the same key multiple times. To import the encrypted key back into the HSM, use The key unwrap command in CloudHSM CLI. Only the owner of a key, that is the crypto user (CU) who created the key, can wrap the key. Users with whom the key is shared can only use the key in cryptographic operations.

The **key wrap** command consists of the following subcommands:

- aes-gcm
- aes-no-pad
- aes-pkcs5-pad
- aes-zero-pad
- cloudhsm-aes-gcm
- rsa-aes
- rsa-oaep
- rsa-pkcs

Wrap a key with AES-GCM using CloudHSM CLI

Use the **key wrap aes-gcm** command in CloudHSM CLI to wrap a payload key using an AES key on the hardware security module (HSM) and the `AES-GCM` wrapping mechanism. The payload key's `extractable` attribute must be set to `true`.

Only the owner of a key, that is the crypto user (CU) who created the key, can wrap the key. Users who share the key can use the key in cryptographic operations.

To use the **key wrap aes-gcm** command, you must first have an AES key in your AWS CloudHSM cluster. You can generate an AES key for wrapping with the Generate a symmetric AES key with CloudHSM CLI command and the `wrap` attribute set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key wrap aes-gcm
Usage: key wrap aes-gcm [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --wrapping-filter [<WRAPPING_FILTER>...] --tag-length-bits <TAG_LENGTH_BITS>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
      --wrapping-approval <WRAPPING_APPROVALR>
          File path of signed quorum token file to approve operation for wrapping key
      --payload-approval <PAYLOAD_APPROVALR>
          File path of signed quorum token file to approve operation for payload key
      --aad <AAD>
          Aes GCM Additional Authenticated Data (AAD) value, in hex
      --tag-length-bits <TAG_LENGTH_BITS>
          Aes GCM tag length in bits
  -h, --help
          Print help
```

Example

This example shows how to use the **key wrap aes-gcm** command using an AES key.

Example

```
aws-cloudhsm > key wrap aes-gcm --payload-filter attr.label=payload-key --wrapping-filter attr.label=aes-example --tag-length-bits 64  --aad 0x10
{
  "error_code": 0,
  "data": {
    "payload_key_reference": "0x00000000001c08f1",
    "wrapping_key_reference": "0x00000000001c08ea",
    "iv": "0xf90613bb8e337ec0339aad21",
    "wrapped_key_data": "xvslgrtg8kHzrvekny97tLSIeokpPwV8"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<PAYLOAD_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a payload key.

Required: Yes

<PATH>

Path to the binary file where the wrapped key data will be saved.

Required: No

<WRAPPING_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a wrapping key.

Required: Yes

<AAD>

AES GCM Additional Authenticated Data (AAD) value, in hex.

Required: No

<TAG_LENGTH_BITS>

AES GCM tag length in bits.

Required: Yes

<WRAPPING_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for wrapping key. Only required if wrapping key's key management service quorum value is greater than 1.

<PAYLOAD_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for payload key. Only required if payload key's key management service quorum value is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Wrap a key with AES-NO-PAD using CloudHSM CLI

Use the **key wrap aes-no-pad** command in CloudHSM CLI to wrap a payload key using an AES key on the hardware security module (HSM) and the `AES-NO-PAD` wrapping mechanism. The payload key's `extractable` attribute must be set to `true`.

Only the owner of a key, that is the crypto user (CU) who created the key, can wrap the key. Users who share the key can use the key in cryptographic operations.

To use the **key wrap aes-no-pad** command, you must first have an AES key in your AWS CloudHSM cluster. You can generate an AES key for wrapping using the Generate a symmetric AES key with CloudHSM CLI command and the `wrap` attribute set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key wrap aes-no-pad
Usage: key wrap aes-no-pad [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --wrapping-filter [<WRAPPING_FILTER>...]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
      --wrapping-approval <WRAPPING_APPROVALR>
          File path of signed quorum token file to approve operation for wrapping key
      --payload-approval <PAYLOAD_APPROVALR>
          File path of signed quorum token file to approve operation for payload key
  -h, --help
          Print help
```

Example

This example shows how to use the **key wrap aes-no-pad** command using an AES key with the `wrap` attribute value set to `true`.

Example

```
aws-cloudhsm > key wrap aes-no-pad --payload-filter attr.label=payload-key --wrapping-filter attr.label=aes-example
{
```

```
  "error_code": 0,
  "data": {
    "payload_key_reference": "0x00000000001c08f1",
    "wrapping_key_reference": "0x00000000001c08ea",
    "wrapped_key_data": "eXK3PMAOnKM9y3YX6brbhtMoC060EOH9"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<PAYLOAD_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a payload key.

Required: Yes

<PATH>

Path to the binary file where the wrapped key data will be saved.

Required: No

<WRAPPING_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a wrapping key.

Required: Yes

<WRAPPING_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for wrapping key. Only required if wrapping key's key management service quorum value is greater than 1.

<PAYLOAD_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for payload key. Only required if payload key's key management service quorum value is greater than 1.

Related topics

- [The key wrap command in CloudHSM CLI]
- [The key unwrap command in CloudHSM CLI]

Wrap a key with AES-PKCS5-PAD using CloudHSM CLI

Use the **key wrap aes-pkcs5-pad** command in CloudHSM CLI to wrap a payload key using an AES key on the hardware security module (HSM) and the `AES-PKCS5-PAD` wrapping mechanism. The payload key's `extractable` attribute must be set to `true`.

Only the owner of a key, that is the crypto user (CU) who created the key, can wrap the key. Users who share the key can use the key in cryptographic operations.

To use the **key wrap aes-pkcs5-pad** command, you must first have an AES key in your AWS CloudHSM cluster. You can generate an AES key for wrapping using the Generate a symmetric AES key with CloudHSM CLI command and the `wrap` attribute set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key wrap aes-pkcs5-pad
Usage: key wrap aes-pkcs5-pad [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --
wrapping-filter [<WRAPPING_FILTER>...]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
```

```
    --payload-filter [<PAYLOAD_FILTER>...]
        Key reference (e.g. key-reference=0xabc) or space separated list of key
attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
payload key
    --wrapping-filter [<WRAPPING_FILTER>...]
        Key reference (e.g. key-reference=0xabc) or space separated list of key
attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
wrapping key
    --path <PATH>
        Path to the binary file where the wrapped key data will be saved
    --wrapping-approval <WRAPPING_APPROVALR>
        File path of signed quorum token file to approve operation for wrapping key
    --payload-approval <PAYLOAD_APPROVALR>
        File path of signed quorum token file to approve operation for payload key
 -h, --help
        Print help
```

Example

This example shows how to use the **key wrap aes-pkcs5-pad** command using an AES key with the `wrap` attribute value set to `true`.

Example

```
aws-cloudhsm > key wrap aes-pkcs5-pad --payload-filter attr.label=payload-key --
wrapping-filter attr.label=aes-example
{
  "error_code": 0,
  "data": {
    "payload_key_reference": "0x00000000001c08f1",
    "wrapping_key_reference": "0x00000000001c08ea",
    "wrapped_key_data": "MbuYNresfOKyGNnxKWen88nSfX+uUE/0qmGofSisicY="
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<PAYLOAD_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a payload key.

Required: Yes

<PATH>

Path to the binary file where the wrapped key data will be saved.

Required: No

<WRAPPING_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a wrapping key.

Required: Yes

<WRAPPING_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for wrapping key. Only required if wrapping key's key management service quorum value is greater than 1.

<PAYLOAD_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for payload key. Only required if payload key's key management service quorum value is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Wrap a key with AES-ZERO-PAD using CloudHSM CLI

Use the **key wrap aes-zero-pad** command in CloudHSM CLI to wrap a payload key using an AES key on the hardware security module HSM) and the `AES-ZERO-PAD` wrapping mechanism. The payload key’s `extractable` attribute must be set to `true`.

Only the owner of a key, that is the crypto user (CU) who created the key, can wrap the key. Users who share the key can use the key in cryptographic operations.

To use the **key wrap aes-zero-pad** command, you must first have an AES key in your AWS CloudHSM cluster. You can generate an AES key for wrapping using the [Generate a symmetric AES key with CloudHSM CLI]() command with the `wrap` attribute set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key wrap aes-zero-pad
Usage: key wrap aes-zero-pad [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --
wrapping-filter [<WRAPPING_FILTER>...]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
      --wrapping-approval <WRAPPING_APPROVALR>
          File path of signed quorum token file to approve operation for wrapping key
      --payload-approval <PAYLOAD_APPROVALR>
          File path of signed quorum token file to approve operation for payload key
  -h, --help
          Print help
```

Example

This example shows how to use the **key wrap aes-zero-pad** command using an AES key with the `wrap` attribute value set to `true`.

Example

```
aws-cloudhsm > key wrap aes-zero-pad --payload-filter attr.label=payload-key --
wrapping-filter attr.label=aes-example
{
  "error_code": 0,
  "data": {
    "payload_key_reference": "0x00000000001c08f1",
    "wrapping_key_reference": "0x00000000001c08ea",
    "wrapped_key_data": "L1wVlL/YeBNVAw6Mpk3owFJZXBzDLONt"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<PAYLOAD_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a payload key.

Required: Yes

<PATH>

Path to the binary file where the wrapped key data will be saved.

Required: No

<WRAPPING_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a wrapping key.

Required: Yes

<WRAPPING_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for wrapping key. Only required if wrapping key's key management service quorum value is greater than 1.

<PAYLOAD_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for payload key. Only required if payload key's key management service quorum value is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Wrap a key with CLOUDHSM-AES-GCM using CloudHSM CLI

Use the **key wrap cloudhsm-aes-gcm** command in CloudHSM CLI to wrap a payload key using an AES key on the hardware security module (HSM) and the `CLOUDHSM-AES-GCM` wrapping mechanism. The payload key’s `extractable` attribute must be set to `true`.

Only the owner of a key, that is the crypto user (CU) who created the key, can wrap the key. Users who share the key can use the key in cryptographic operations.

To use the **key wrap cloudhsm-aes-gcm** command, you must first have an AES key in your AWS CloudHSM cluster. You can generate an AES key for wrapping with the Generate a symmetric AES key with CloudHSM CLI command and the `wrap` attribute set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key wrap cloudhsm-aes-gcm
Usage: key wrap cloudhsm-aes-gcm [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --
wrapping-filter [<WRAPPING_FILTER>...] --tag-length-bits <TAG_LENGTH_BITS>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
      --wrapping-approval <WRAPPING_APPROVALR>
          File path of signed quorum token file to approve operation for wrapping key
      --payload-approval <PAYLOAD_APPROVALR>
          File path of signed quorum token file to approve operation for payload key
      --aad <AAD>
          Aes GCM Additional Authenticated Data (AAD) value, in hex
      --tag-length-bits <TAG_LENGTH_BITS>
          Aes GCM tag length in bits
  -h, --help
          Print help
```

Example

This example shows how to use the **key wrap cloudhsm-aes-gcm** command using an AES key.

Example

```
aws-cloudhsm > key wrap cloudhsm-aes-gcm --payload-filter attr.label=payload-key --
wrapping-filter attr.label=aes-example --tag-length-bits 64 --aad 0x10
{
  "error_code": 0,
  "data": {
```

```
    "payload_key_reference": "0x00000000001c08f1",
    "wrapping_key_reference": "0x00000000001c08ea",
    "wrapped_key_data": "6Rn8nkjEriDYlnP3P8nPkYQ8hplOEJ899zsrF+aTB0i/fIlZ"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been [configured.](#)

<PAYLOAD_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a payload key.

Required: Yes

<PATH>

Path to the binary file where the wrapped key data will be saved.

Required: No

<WRAPPING_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a wrapping key.

Required: Yes

<AAD>

AES GCM Additional Authenticated Data (AAD) value, in hex.

Required: No

<TAG_LENGTH_BITS>

AES GCM tag length in bits.

Required: Yes

<WRAPPING_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for wrapping key. Only required if wrapping key's key management service quorum value is greater than 1.

<PAYLOAD_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for payload key. Only required if payload key's key management service quorum value is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Wrap a key with RSA-AES using CloudHSM CLI

Use the **key wrap rsa-aes** command in CloudHSM CLI to wrap a payload key using an RSA public key on the hardware security module (HSM) and the RSA-AES wrapping mechanism. The payload key's `extractable` attribute must be set to `true`.

Only the owner of a key, that is the crypto user (CU) who created the key, can wrap the key. Users who share the key can use the key in cryptographic operations.

To use the **key wrap rsa-aes** command, you must first have an RSA key in your AWS CloudHSM cluster. You can generate an RSA key pair using the The generate-asymmetric-pair category in CloudHSM CLI command and the `wrap` attribute set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key wrap rsa-aes
Usage: key wrap rsa-aes [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --wrapping-filter [<WRAPPING_FILTER>...] --hash-function <HASH_FUNCTION> --mgf <MGF>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
      --wrapping-approval <WRAPPING_APPROVALR>
          File path of signed quorum token file to approve operation for wrapping key
      --payload-approval <PAYLOAD_APPROVALR>
          File path of signed quorum token file to approve operation for payload key
      --hash-function <HASH_FUNCTION>
          Hash algorithm [possible values: sha1, sha224, sha256, sha384, sha512]
      --mgf <MGF>
          Mask Generation Function algorithm [possible values: mgf1-sha1, mgf1-sha224,
 mgf1-sha256, mgf1-sha384, mgf1-sha512]
  -h, --help
          Print help
```

Example

This example shows how to use the **key wrap rsa-ae** command using an RSA public key with the `wrap` attribute value set to `true`.

Example

```
aws-cloudhsm > key wrap rsa-aes --payload-filter attr.label=payload-key --wrapping-filter attr.label=rsa-public-key-example --hash-function sha256 --mgf mgf1-sha256
{
```

```
  "error_code": 0,
  "data": {
    "payload-key-reference": "0x00000000001c08f1",
    "wrapping-key-reference": "0x00000000007008da",
    "wrapped-key-data": "HrSE1DEyLjIeyGdPa9R+ebiqB5TIJGyamPker31ZebPwRA
+NcerbAJO8DJ1lXPygZcI21vIFSZJuWMEiWpe1R9D/5WSYgxLVKex30xCFqebtEzxbKuv4DOmU4meSofqREYvtb3EoIKwjy
+RL5WGXKe4nAboAkC5GO7veI5yHL1SaKlssSJtTL/CFpbSLsAFuYbv/NUCWwMY5mwyVTCSlw+HlgKK
+5TH1MzBaSi8fpfyepLT8sHy2Q/VRl6ifb49p6m0KQFbRVvz/OWUd6l4d97BdgtaEz6ueg=="
  }
}
```


Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<PAYLOAD_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a payload key.

Required: Yes

<PATH>

Path to the binary file where the wrapped key data will be saved.

Required: No

<WRAPPING_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a wrapping key.

Required: Yes

<MGF>

Specifies the mask generation function.

> **Note**
>
> The mask generation function hash function must match the signing mechanism hash function.

Valid values

- mgf1-sha1
- mgf1-sha224
- mgf1-sha256
- mgf1-sha384
- mgf1-sha512

Required: Yes

<WRAPPING_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for wrapping key. Only required if wrapping key's key management service quorum value is greater than 1.

<PAYLOAD_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for payload key. Only required if payload key's key management service quorum value is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Wrap a key with RSA-OAEP using CloudHSM CLI

Use the **key wrap rsa-oaep** command in CloudHSM CLI to wrap a payload key using an RSA public key on the hardware security module (HSM) and the `RSA-OAEP` wrapping mechanism. The payload key's `extractable` attribute must be set to `true`.

Only the owner of a key, that is the crypto user (CU) who created the key, can wrap the key. Users who share the key can use the key in cryptographic operations.

To use the **key wrap rsa-oaep** command, you must first have an RSA key in your AWS CloudHSM cluster. You can generate an RSA key pair using the [The generate-asymmetric-pair category in CloudHSM CLI]() command and the `wrap` attribute set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key wrap rsa-oaep
Usage: key wrap rsa-oaep [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --wrapping-filter [<WRAPPING_FILTER>...] --hash-function <HASH_FUNCTION> --mgf <MGF>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
      --wrapping-approval <WRAPPING_APPROVALR>
          File path of signed quorum token file to approve operation for wrapping key
      --payload-approval <PAYLOAD_APPROVALR>
          File path of signed quorum token file to approve operation for payload key
      --hash-function <HASH_FUNCTION>
          Hash algorithm [possible values: sha1, sha224, sha256, sha384, sha512]
      --mgf <MGF>
```

```
        Mask Generation Function algorithm [possible values: mgf1-sha1, mgf1-sha224,
mgf1-sha256, mgf1-sha384, mgf1-sha512]
 -h, --help
        Print help
```

Example

This example shows how to use the **key wrap rsa-oaep** command using an RSA public key with the `wrap` attribute value set to `true`.

Example

```
aws-cloudhsm > key wrap rsa-oaep --payload-filter attr.label=payload-key --wrapping-filter attr.label=rsa-public-key-example --hash-function sha256 --mgf mgf1-sha256
{
  "error_code": 0,
  "data": {
    "payload-key-reference": "0x00000000001c08f1",
    "wrapping-key-reference": "0x00000000007008da",
    "wrapped-key-data": "OjJe4msobPLz9TuSAdULEu17T5rMDWtSlLyBSkLbaZnYzzpdrhsbGLbwZJCtB/
jGkDNdB4qyTAOQwEpggGf6v+Yx6JcesNeKKNU8XZal/YBoHC8noTGUSDI2qr+u2tDc84NPv6d
+F2KOONXsSxMhmxzzNG/gzTVIJhOuy/B1yHjGP4mOXoDZf5+7f5M1CjxBmz4Vva/
wrWHGCSG0yOaWblEvOiHAIt3UBdyKmU+/
My4xjfJv7WGGu3DFUUIZ06TihRtKQhUYU1M9u6NPf9riJJfHsk6QCuSZ9yWThDT9as6i7e3htnyDhIhGWaoK8JU855cN/
YNKAUqkNpC4FPL3iw=="
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<PAYLOAD_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a payload key.

Required: Yes

<PATH>

Path to the binary file where the wrapped key data will be saved.

Required: No

<WRAPPING_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a wrapping key.

Required: Yes

<MGF>

Specifies the mask generation function.

> **Note**
>
> The mask generation function hash function must match the signing mechanism hash function.

Valid values

- mgf1-sha1
- mgf1-sha224
- mgf1-sha256
- mgf1-sha384
- mgf1-sha512

Required: Yes

<WRAPPING_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for wrapping key. Only required if wrapping key's key management service quorum value is greater than 1.

<PAYLOAD_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for payload key. Only required if payload key's key management service quorum value is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Wrap a key with RSA-PKCS using CloudHSM CLI

Use the **key wrap rsa-pkcs** command in CloudHSM CLI to wrap a payload key using an RSA public key on the hardware security module (HSM) and the `RSA-PKCS` wrapping mechanism. The payload key's `extractable` attribute must be set to `true`.

Only the owner of a key, that is the crypto user (CU) who created the key, can wrap the key. Users who share the key can use the key in cryptographic operations.

To use the **key wrap rsa-pkcs** command, you must first have an RSA key in your AWS CloudHSM cluster. You can generate an RSA key pair using the The generate-asymmetric-pair category in CloudHSM CLI command and the `wrap` attribute set to `true`.

User type

The following types of users can run this command.

- Crypto users (CUs)

Requirements

- To run this command, you must be logged in as a CU.

Syntax

```
aws-cloudhsm > help key wrap rsa-pkcs
Usage: key wrap rsa-pkcs [OPTIONS] --payload-filter [<PAYLOAD_FILTER>...] --wrapping-
filter [<WRAPPING_FILTER>...]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error
      --payload-filter [<PAYLOAD_FILTER>...]
```

```
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 payload key
      --wrapping-filter [<WRAPPING_FILTER>...]
          Key reference (e.g. key-reference=0xabc) or space separated list of key
 attributes in the form of attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE to select a
 wrapping key
      --path <PATH>
          Path to the binary file where the wrapped key data will be saved
      --wrapping-approval <WRAPPING_APPROVALR>
          File path of signed quorum token file to approve operation for wrapping key
      --payload-approval <PAYLOAD_APPROVALR>
          File path of signed quorum token file to approve operation for payload key
  -h, --help
          Print help
```

Example

This example shows how to use the **key wrap rsa-pkcs** command using an RSA public key.

Example

```
aws-cloudhsm > key wrap rsa-pkcs --payload-filter attr.label=payload-key --wrapping-
filter attr.label=rsa-public-key-example
{
  "error_code": 0,
  "data": {
    "payload_key_reference": "0x00000000001c08f1",
    "wrapping_key_reference": "0x00000000007008da",
    "wrapped_key_data": "am0Nc7+YE8FWs+5HvU7sIBcXVb24QA0l65nbNAD+1bK+e18BpSfnaI3P+r8Dp
+pLu1ofoUy/
vtzRjZoCiDofcz4EqCFnGl4GdcJ1/3W/5WRvMatCa2d7cx02swaeZcjKsermPXYR011Glfq6NskwMeeTkV8R7Rx9artFrs1
c3XdFJ2+0Bo94c6og/
yfPcpOOobJlITCoXhtMRepSd040ggYq/6nUDuHCtJ86pPGnNahyr7+sAaSI3a5ECQLUjwaIARUCyoRh7EFK3qPXcg=="
  }
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been [configured.](#)

<PAYLOAD_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a payload key.

Required: Yes

<PATH>

Path to the binary file where the wrapped key data will be saved.

Required: No

<WRAPPING_FILTER>

Key reference (for example, `key-reference=0xabc`) or space separated list of key attributes in the form of `attr.KEY_ATTRIBUTE_NAME=KEY_ATTRIBUTE_VALUE` to select a wrapping key.

Required: Yes

<WRAPPING_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for wrapping key. Only required if wrapping key's key management service quorum value is greater than 1.

<PAYLOAD_APPROVALR>

Specifies the file path to a signed quorum token file to approve operation for payload key. Only required if payload key's key management service quorum value is greater than 1.

Related topics

- The key wrap command in CloudHSM CLI
- The key unwrap command in CloudHSM CLI

Log in to an HSM using CloudHSM CLI

You can use the **login** command in CloudHSM CLI to log in and out of each hardware security (HSM) in a AWS CloudHSM cluster. This command has the following sub-command:

- mfa-token-sign

> **Note**
>
> If you exceed five incorrect login attempts, your account is locked out. To unlock the account, an admin must reset your password using the user change-password command in cloudhsm_cli.

To troubleshoot login and logout

If you have more than one HSM in your cluster, you may be allowed additional incorrect login attempts before your account is locked out. This is because the CloudHSM client balances load across various HSMs. Therefore, the login attempt may not begin on the same HSM each time. If you are testing this functionality, we recommend you do so on a cluster with only one active HSM.

If you created your cluster before February 2018, your account is locked out after 20 incorrect login attempts.

User type

The following users can run these commands.

- Unactivated admin
- Admin
- Crypto user (CU)

Syntax

```
aws-cloudhsm > help login
Login to your cluster

USAGE:
    cloudhsm-cli login [OPTIONS] --username <USERNAME> --role <ROLE> [COMMAND]

Commands:
  mfa-token-sign  Login with token-sign mfa
  help            Print this message or the help of the given subcommand(s)

OPTIONS:
        --cluster-id <CLUSTER_ID>
```

```
         Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

       --username <USERNAME>
           Username to access the Cluster

       --role <ROLE>
           Role the user has in the Cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

       --password <PASSWORD>
          Optional: Plaintext user's password. If you do not include this argument you
 will be prompted for it

  -h, --help
         Print help (see a summary with '-h')
```

Example

Example

This command logs you in to all HSMs in a cluster with the credentials of an admin user named `admin1`.

```
aws-cloudhsm > login --username admin1 --role admin
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "admin1",
    "role": "admin"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<USERNAME>

Specifies a friendly name for the user. The maximum length is 31 characters. The only special character permitted is an underscore (_). The username is not case sensitive in this command, username is always displayed in lowercase.

Required: Yes

<ROLE>

Specifies the role assigned to this user. This parameter is required. Valid values are **admin**, **crypto-user**.

To get the user's role, use the **user list** command. For detailed information about the user types on an HSM, see Understanding HSM users.

<PASSWORD>

Specifies the password of the user who is logging in to the HSMs.

Related topics

- Getting Started with CloudHSM CLI
- Activate the Cluster

Log in with MFA to an HSM using CloudHSM CLI

Use the **login mfa-token-sign** command in AWS CloudHSM CloudHSM CLI to log in to a hardware security module (HSM) using multi-factor authentication (MFA). To use this command, you must first set up MFA for CloudHSM CLI.

User type

The following users can run these commands.

- Admin
- Crypto user (CU)

Syntax

```
aws-cloudhsm > help login mfa-token-sign
Login with token-sign mfa

USAGE:
    login --username <username> --role <role> mfa-token-sign --token <token>

OPTIONS:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
      --token <TOKEN>            Filepath where the unsigned token file will be written
  -h, --help                     Print help
```

Example

Example

```
aws-cloudhsm > login --username test_user --role admin mfa-token-sign --token /home/
valid.token
Enter password:
Enter signed token file path (press enter if same as the unsigned token file):
{
  "error_code": 0,
  "data": {
    "username": "test_user",
    "role": "admin"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<TOKEN>

Filepath where the unsigned token file will be written.

Required: Yes

Related topics

- Getting Started with CloudHSM CLI
- Activate the Cluster
- Using CloudHSM CLI to manage MFA

Log out of an HSM using CloudHSM CLI

Use the **logout** command in CloudHSM CLI to log out of each hardware security module (HSM) in an AWS CloudHSM cluster.

User type

The following users can run this command.

- Admin
- Crypto user (CU)

Syntax

```
aws-cloudhsm > help logout
Logout of your cluster

USAGE:
    logout

OPTIONS:
        --cluster-id <CLUSTER_ID> Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
    -h, --help                    Print help information
    -V, --version                 Print version information
```

Example

Example

This command logs you out of all HSMs in a cluster.

```
aws-cloudhsm > logout
{
  "error_code": 0,
  "data": "Logout successful"
}
```

Related topics

- Getting Started with CloudHSM CLI
- Activate the Cluster

The user category in CloudHSM CLI

In the CloudHSM CLI, **user** is a parent category for a group of commands that, when combined with the parent category, create a command specific to users. Currently, the user category consists of the following commands:

- user change-mfa
- user change-password
- user create
- user delete
- user list
- user replicate

The user change-mfa category in CloudHSM CLI

In the CloudHSM CLI, **user change-mfa** is a parent category for a group of commands that, when combined with the parent category, create a command specific to changing multi-factor authentication (MFA) for users.

Currently, this category consists of the following sub-command:

- token-sign

Change a user's MFA setup with CloudHSM CLI

Use the **user change-mfa token-sign** command in CloudHSM CLI to update a user account's multi-factor authentication (MFA) setup. Any user account can run this command. Accounts with the Admin role can run this command for other users.

User type

The following users can run this command.

- Admin
- Crypto user

Syntax

Currently, there is only a single multi-factor strategy available for users: Token Sign.

```
aws-cloudhsm > help user change-mfa
Change a user's Mfa Strategy

Usage:
    user change-mfa <COMMAND>

Commands:
  token-sign  Register or Deregister a public key using token-sign mfa strategy
  help        Print this message or the help of the given subcommand(s)
```

The Token Sign strategy asks for a Token file to write unsigned tokens to.

```
aws-cloudhsm > help user change-mfa token-sign
Register or Deregister a public key using token-sign mfa strategy

Usage: user change-mfa token-sign [OPTIONS] --username <USERNAME> --role <ROLE> <--
token <TOKEN>|--deregister>

Options:
      --cluster-id <CLUSTER_ID>
```

```
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --username <USERNAME>
          Username of the user that will be modified

      --role <ROLE>
          Role the user has in the cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

      --change-password <CHANGE_PASSWORD>
          Optional: Plaintext user's password. If you do not include this argument you
 will be prompted for it

      --token <TOKEN>
          Filepath where the unsigned token file will be written. Required for enabling
 MFA for a user

      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation

      --deregister
          Deregister the MFA public key, if present

      --change-quorum
          Change the Quorum public key along with the MFA key

  -h, --help
          Print help (see a summary with '-h')
```

Example

This command will write one unsigned token per HSM in your cluster to the file specified by `token`. When you are prompted, sign the tokens in the file.

Example : Write one unsigned token per HSM in your cluster

```
aws-cloudhsm > user change-mfa token-sign --username cu1 --change-password password --
role crypto-user --token /path/myfile
```

```
Enter signed token file path (press enter if same as the unsigned token file):
Enter public key PEM file path:/path/mypemfile
{
  "error_code": 0,
  "data": {
    "username": "test_user",
    "role": "admin"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<ROLE>

Specifies the role given to the user account. This parameter is required. For detailed information about the user types on an HSM, see Understanding HSM users.

Valid values

- **Admin**: Admins can manage users, but they cannot manage keys.
- **Crypto user**: Crypto users can create an manage keys and use keys in cryptographic operations.

<USERNAME>

Specifies a friendly name for the user. The maximum length is 31 characters. The only special character permitted is an underscore (_).

You cannot change the name of a user after it is created. In CloudHSM CLI commands, the role and password are case-sensitive, but the username is not.

Required: Yes

<CHANGE_PASSWORD>

Specifies the plaintext new password of the user whose MFA is being registered/deregistered.

Required: Yes

<TOKEN>

File path where the unsigned token file will be written.

Required: Yes

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if quorum user service quorum value is greater than 1.

<DEREGISTER>

Deregisters the MFA public key, if present.

<CHANGE-QUORUM>

Changes the quorum public key along with the MFA key.

Related topics

- Understanding 2FA for HSM users

Change a user's password with CloudHSM CLI

Use the **user change-password** command in CloudHSM CLI to change the password of an existing user in your AWS CloudHSM cluster. To enable MFA for a user, use the `user change-mfa` command.

Any user can change their own password. In addition, users with the admin role can change the password of another user in the cluster. You do not need to enter the current password to make the change.

> **Note**
>
> You cannot change the password of a user who is currently logged in to the cluster.

User type

The following users can run this command.

- Admin

- Crypto user (CU)

Syntax

> **Note**
>
> To enable multi-factor authentication (MFA) for a user, use the **user change-mfa** command.

```
aws-cloudhsm > help user change-password
Change a user's password

    Usage:
        cloudhsm-cli user change-password [OPTIONS] --username <USERNAME> --role <ROLE>
 [--password <PASSWORD>]

    Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --username <USERNAME>
          Username of the user that will be modified

      --role <ROLE>
          Role the user has in the cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

      --password <PASSWORD>
          Optional: Plaintext user's password. If you do not include this argument you
 will be prompted for it

      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation

      --deregister-mfa <DEREGISTER-MFA>
          Deregister the user's mfa public key, if present
```

```
      --deregister-quorum <DEREGISTER-QUORUM>
          Deregister the user's quorum public key, if present
 -h, --help
          Print help (see a summary with '-h')
```

Example

The following examples show how to use **user change-password** to reset the password for the current user or any other user in your cluster.

Example : Change your password

Any user in the cluster can use **user change-password** to change their own password.

The following output shows that Bob is currently logged in as a crypto user(CU).

```
aws-cloudhsm > user change-password --username bob --role crypto-user
Enter password:
Confirm password:
{
  "error_code": 0,
  "data": {
    "username": "bob",
    "role": "crypto-user"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if quorum user service quorum value is greater than 1.

<DEREGISTER-MFA>

Deregisters the MFA public key, if present.

<DEREGISTER-QUORUM>

Deregister the Quorum public key, if present.

<PASSWORD>

Specifies the plaintext new password of the user. The following characters are not permitted ':'

Required: Yes

<ROLE>

Specifies the role given to the user account. This parameter is required. For detailed information about the user types on an HSM, see Understanding HSM users.

Valid values

- **Admin**: Admins can manage users, but they cannot manage keys.
- **Crypto user**: Crypto users can create an manage keys and use keys in cryptographic operations.

<USERNAME>

Specifies a friendly name for the user. The maximum length is 31 characters. The only special character permitted is an underscore (_).

You cannot change the name of a user after it is created. In CloudHSM CLI commands, the role and password are case-sensitive, but the username is not.

Required: Yes

Related topics

- user list
- user create
- user delete

The user change-quorum category in CloudHSM CLI

In the CloudHSM CLI, **user change-quorum** is a parent category for a group of commands that, when combined with the parent category, create a command specific to changing quorum for users.

user change-quorum is used to register user quorum authentication using a specified quorum strategy. As of SDK 5.8.0, there is only a single quorum strategy available for users as shown below.

Currently, this category consists of the following category and sub-command:

- token-sign
  - register

The user change-quorum token-sign category in CloudHSM CLI

In the CloudHSM CLI, **user change-quorum token-sign** is a parent category for commands that, when combined with this parent category, create a command specific to token-sign quorum operations.

Currently, the this category consists of the following commands:

- register

Register a user's token-sign quorum strategy using CloudHSM CLI

Use the **user change-quorum token-sign register** command in CloudHSM CLI to register the token-sign quorum strategy for an admin user.

User type

The following users can run this command.

- Admin

Syntax

```
aws-cloudhsm > help user change-quorum token-sign register
Register a user for quorum authentication with a public key

Usage: user change-quorum token-sign register --public-key <PUBLIC_KEY> --signed-token <SIGNED_TOKEN>

Options:
      --cluster-id <CLUSTER_ID>      Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
```

```
      --public-key <PUBLIC_KEY>      Filepath to public key PEM file
      --signed-token <SIGNED_TOKEN>  Filepath with token signed by user private key
  -h, --help Print help (see a summary with '-h')
```

Example

Example

To run this command you will need to be logged in as the user you wish to **register quorum token-sign** for.

```
aws-cloudhsm > login --username admin1 --role admin
Enter password:
{
  "error_code": 0,
  "data": {
    "username": "admin1",
    "role": "admin"
  }
}
```

The **user change-quorum token-sign register** command will register your public key with the HSM. As a result, it will qualify you as a quorum approver for quorum-required operations that need a user to obtain quorum signatures to meet the necessary quorum value threshold.

```
aws-cloudhsm > user change-quorum token-sign register \
    --public-key /home/mypemfile \
    --signed-token /home/mysignedtoken
{
  "error_code": 0,
  "data": {
    "username": "admin1",
    "role": "admin"
  }
}
```

You can now run the **user list** command and confirm that quorum token-sign has been registered for this user.

```
aws-cloudhsm > user list
{
```

```
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      },
      {
        "username": "admin1",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<PUBLIC-KEY>

Filepath to the public key PEM file.

Required: Yes

<SIGNED-TOKEN>

Filepath with token signed by user private key.

Required: Yes

Related topics

- Using CloudHSM CLI to manage quorum authentication
- Using quorum authentication for admins: first time setup
- Change the quorum minimum value for admins
- Service names and types that support quorum authentication

Create an AWS CloudHSM user with CloudHSM CLI

The **user create** command in CloudHSM CLI creates a user in your AWS CloudHSM cluster. Only user accounts with the admin role can run this command.

User type

The following types of users can run this command.

- Admin

Requirements

To run this command, you must be logged in as an admin user

Syntax

```
aws-cloudhsm > help user create
Create a new user

Usage: cloudhsm-cli user create [OPTIONS] --username <USERNAME> --role <ROLE> [--
password <PASSWORD>]

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --username <USERNAME>
          Username to access the HSM cluster
```

```
      --role <ROLE>
          Role the user has in the cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

      --password <PASSWORD>
          Optional: Plaintext user's password. If you do not include this argument you
 will be prompted for it

      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation

  -h, --help
          Print help (see a summary with '-h')
```

Example

These examples show how to use **user create** to create new users in your HSMs.

Example : Create a crypto user

This example creates an account in your AWS CloudHSM cluster with the crypto user role.

```
aws-cloudhsm > user create --username alice --role crypto-user
Enter password:
Confirm password:
{
  "error_code": 0,
  "data": {
    "username": "alice",
    "role": "crypto-user"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<USERNAME>

Specifies a friendly name for the user. The maximum length is 31 characters. The only special character permitted is an underscore (_). The username is not case sensitive in this command, username is always displayed in lowercase.

Required: Yes

<ROLE>

Specifies the role assigned to this user. This parameter is required. Valid values are **admin**, **crypto-user**.

To get the user's role, use the **user list** command. For detailed information about the user types on an HSM, see Understanding HSM users.

<PASSWORD>

Specifies the password of the user who is logging in to the HSMs. The following characters are not permitted ':'

Required: Yes

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if quorum user service quorum value is greater than 1.

Related topics

- user list
- user delete
- user change-password

Delete an AWS CloudHSM user with CloudHSM CLI

The **user delete** command in CloudHSM CLI deletes a user from your AWS CloudHSM cluster. Only user accounts with the admin role may run this command. You cannot delete a user who is currently logged into an HSM.

User type

The following types of users can run this command.

- Admin

Requirements

- You can't delete user accounts that own keys.
- Your user account must have the admin role to run this command.

Syntax

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
aws-cloudhsm > help user delete
Delete a user

Usage: user delete [OPTIONS] --username <USERNAME> --role <ROLE>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --username <USERNAME>
          Username to access the HSM cluster

      --role <ROLE>
          Role the user has in the cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

      --approval <APPROVAL>
          Filepath of signed quorum token file to approve operation
```

Example

```
aws-cloudhsm > user delete --username alice --role crypto-user
{
  "error_code": 0,
```

```
  "data": {
    "username": "alice",
    "role": "crypto-user"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<USERNAME>

Specifies a friendly name for the user. The maximum length is 31 characters. The only special character permitted is an underscore (_). The username is not case sensitive in this command, username is always displayed in lowercase.

Required: Yes

<ROLE>

Specifies the role assigned to this user. This parameter is required. Valid values are **admin**, **crypto-user**.

To get the user's role, use the **user list** command. For detailed information about the user types on an HSM, see Understanding HSM users.

Required: Yes

<APPROVAL>

Specifies the file path to a signed quorum token file to approve operation. Only required if quorum user service quorum value is greater than 1.

Required: Yes

Related topics

- user list

- user create
- user change-password

List all AWS CloudHSM users with CloudHSM CLI

The **user list** command in the CloudHSM CLI lists the user accounts present in your AWS CloudHSM cluster. You do not need to be logged in to CloudHSM CLI to run this command.

> **Note**
>
> If you add or delete HSMs, update the configuration files that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective on all HSMs in the cluster.

User type

The following types of users can run this command.

- All users. You do not need to be logged in to run this command.

Syntax

```
aws-cloudhsm > help user list
List the users in your cluster

USAGE:
    user list

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
  -h, --help                     Print help
```

Example

This command lists the users present in your CloudHSM cluster.

```
aws-cloudhsm > user list
```

```
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      },
      {
        "username": "test_user",
        "role": "admin",
        "locked": "false",
        "mfa": [
          {
            "strategy": "token-sign",
            "status": "enabled"
          }
        ],
        "cluster-coverage": "full"
      },
      {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

The output includes the following user attributes:

- **Username**: Displays the user-defined friendly name for the user. Username is always displayed in lowercase.
- **Role**: Determines the operations that the user can perform on the HSM.
- **Locked**: Indicates whether this user account has been locked out.
- **MFA**: Indicates the supported multi-factor authentication mechanisms for this user account.
- **Cluster coverage**: Indicates the cluster-wide availability of this user account.

Related topics

- listUsers in key_mgmt_util
- user create
- user delete
- user change-password

Replicate a user with CloudHSM CLI

Use the **user replicate** command in CloudHSM CLI to replicate a user from a source AWS CloudHSM cluster to a destination AWS CloudHSM cluster.

User type

The following types of users can run this command.

- Admins (COs)

Requirements

- The source and destination clusters must be clones. This means one was created from a backup of the other, or they were both created from a common backup. See Creating clusters from backups for more information.
- To run this command, you must be logged in as an admin on both the source and destination clusters.
  - In single command mode, the command will use the CLOUDHSM_PIN and CLOUDHSM_ROLE environmental variables to authenticate on the source cluster. See Single Command mode for more information. To provide credentials for the destination cluster, you need to set two additional environmental variables: DESTINATION_CLOUDHSM_PIN and DESTINATION_CLOUDHSM_ROLE:

    ```
    $ export DESTINATION_CLOUDHSM_ROLE=<role>
    ```

    ```
    $ export DESTINATION_CLOUDHSM_PIN=<username:password>
    ```

  - In interactive mode, users will need to explicitly log into both the source and destination clusters.

Syntax

```
aws-cloudhsm > help user replicate
Replicate a user from a source to a destination cluster

Usage: user replicate --username <USERNAME> --role <ROLE> --source-cluster-
id <SOURCE_CLUSTER_ID> --destination-cluster-id <DESTINATION_CLUSTER_ID>

Options:
      --username <USERNAME>
          Username of the user to replicate

      --role <ROLE>
          Role the user has in the cluster

          Possible values:
          - crypto-user: A CryptoUser has the ability to manage and use keys
          - admin:       An Admin has the ability to manage user accounts

      --source-cluster-id <SOURCE_CLUSTER_ID>
          Source cluster ID

      --destination-cluster-id <DESTINATION_CLUSTER_ID>
          Destination cluster ID

  -h, --help
          Print help (see a summary with '-h')
```

Examples

Example Example: Replicate user

This command replicates a user from a source cluster with to a cloned destination cluster. The example below demonstrates the output when logged in as an admin on both clusters.

```
admin-user@cluster-1234abcdefg > user replicate \
      --username example-admin \
      --role admin \
      --source-cluster-id cluster-1234abcdefg \
      --destination-cluster-id cluster-2345bcdefgh
{
  "error_code": 0,
  "data": {
```

```
    "user": {
      "username": "example-admin",
      "role": "admin",
      "locked": "false",
      "mfa": [],
      "quorum": [],
      "cluster-coverage": "full"
    },
    "message": "Successfully replicated user"
  }
}
```

Arguments

<USERNAME>

Specifies the username of the user to replicate in the source cluster.

Required: Yes

<ROLE>

Specifies the role assigned to this user. This parameter is required. Valid values are **admin**, **crypto-user**.

To get the user's role, use the **user list** command. For detailed information about the user types on an HSM, see Understanding HSM users.

Required: Yes

<SOURCE_CLUSTER_ID>

The source cluster ID.

Required: Yes

<DESTINATION_CLUSTER_ID>

The destination cluster ID.

Required: Yes

Related topics

- Connecting to multiple clusters with CloudHSM CLI

The quorum category in CloudHSM CLI

In the CloudHSM CLI, **quorum** is a parent category for a group of commands that, when combined with **quorum**, creates a command specific to quorum authentication, or M of N operations. Currently, this category consists of the **token-sign** sub-category which consists of its own commands. Click the link below for details.

- token-sign

Admin Services: Quorum authentication is used for admin privileged services like creating users, deleting users, changing user passwords, setting quorum values, and deactivating quorum and MFA capabilities.

Crypto User Services: Quorum authentication is used for crypto-user privileged services associated with a specific key like signing with a key, sharing/unsharing a key, wrapping/unwrapping a key, and setting a key's attribute. The quorum value of an associated key is configured when the key is generated, imported, or unwrapped. The quorum value must be equal to or less than the number of users that the key is associated with, which includes users that the key is shared with and the key owner.

Each service type is further broken down into a qualifying service name, which contains a specific set of quorum supported service operations that can be performed.

Service name	Service type	Service operations
user	Admin	• user create • user delete • user change-password • user change-mfa
quorum	Admin	• quorum token-sign set-quorum-value
cluster[1]	Admin	• cluster mtls register-trust-anchor • cluster mtls deregister-trust-anchor

Service name	Service type	Service operations
		• cluster mtls set-enfor cement
key-management	Crypto User	• key wrap • key unwrap • key share • key unshare • key set-attribute
key-usage	Crypto User	• key sign

[1] Cluster service is exclusively available on hsm2m.medium

Related topics

- Set up quorum authentication for AWS CloudHSM admins using CloudHSM CLI
- Manage quorum authentication (M of N access control) using CloudHSM CLI

The quorum token-sign category in CloudHSM CLI

In the CloudHSM CLI, **quorum token-sign** is a category for a group of commands that, when combined with **quorum token-sign**, create a command specific to quorum authentication, or M of N operations.

Currently, this category consists of the following commands:

- delete
- generate
- list
- list-quorum-values
- set-quorum-value

Delete quorum tokens using CloudHSM CLI

Use the **quorum token-sign delete** command in CloudHSM CLI to delete one or more tokens for a quorum authorized service.

User type

The following users can run this command.

- Admin

Syntax

```
aws-cloudhsm > help quorum token-sign delete
Delete one or more Quorum Tokens

Usage: quorum token-sign delete --scope <SCOPE>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --scope <SCOPE>
          Scope of which token(s) will be deleted

          Possible values:
          - user: Deletes all token(s) of currently logged in user
          - all:  Deletes all token(s) on the HSM
  -h, --help
          Print help (see a summary with '-h')
```

Example

The following example shows how the **quorum token-sign delete** command in CloudHSM CLI can be used to delete one or more tokens for a quorum authorized service.

Example : Delete one or more tokens for a quorum authorized service

```
aws-cloudhsm > quorum token-sign delete --scope all
{
```

```
  "error_code": 0,
  "data": "Deletion of quorum token(s) successful"
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been configured.

<SCOPE>

The scope in which token(s) will be deleted in the AWS CloudHSM cluster.

Valid values

- **User**: Used to delete only tokens owned by the logged in user.
- **All**: Used to delete all tokens in the AWS CloudHSM cluster.

Related topics

- user list
- user create
- user delete

Generate a quorum token using CloudHSM CLI

Use the **quorum token-sign generate** command in CloudHSM CLI to generate a token for a quorum authorized service.

There is a limit to obtaining one active token per user per service on an HSM cluster for services user and quorum. This limit does not apply to tokens related to key services.

> **Note**
>
> Only Admins and Crypto Users may generate specific service tokens. For more information about service types and names, see Service names and types that support quorum authentication

Admin Services: Quorum authentication is used for admin privileged services like creating users, deleting users, changing user passwords, setting quorum values, and deactivating quorum and MFA capabilities.

Crypto User Services: Quorum authentication is used for crypto-user privileged services associated with a specific key like signing with a key, sharing/unsharing a key, wrapping/unwrapping a key, and setting a key's attribute. The quorum value of an associated key is configured when the key is generated, imported, or unwrapped. The quorum value must be equal to or less than the number of users that the key is associated with, which includes users that the key is shared with and the key owner.

Each service type is further broken down into a qualifying service name, which contains a specific set of quorum supported service operations that can be performed.

Service name	Service type	Service operations
user	Admin	• user create • user delete • user change-password • user change-mfa
quorum	Admin	• quorum token-sign set-quorum-value
cluster[1]	Admin	• cluster mtls register-trust-anchor • cluster mtls deregister-trust-anchor • cluster mtls set-enforcement
key-management	Crypto User	• key wrap • key unwrap • key share • key unshare • key set-attribute

Service name	Service type	Service operations
key-usage	Crypto User	• key sign

[1] Cluster service is exclusively available on hsm2m.medium

User type

The following users can run this command.

- Admin
- Crypto user (CU)

Syntax

```
aws-cloudhsm > help quorum token-sign generate
Generate a token

Usage: quorum token-sign generate --service <SERVICE> --token <TOKEN>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --service <SERVICE>
          Service the token will be used for

          Possible values:
          - user:
            User management service is used for executing quorum authenticated user
 management operations
          - quorum:
            Quorum management service is used for setting quorum values for any quorum
 service
          - cluster:
            Cluster management service is used for executing quorum for cluster
 wide configuration managements like mtls enforcement, mtls registration and mtls
 deregistration
          - registration:
```

```
            Registration service is used for registering a public key for quorum
 authentication
          - key-usage:
            Key usage service is used for executing quorum authenticated key usage
 operations
          - key-management:
            Key management service is used for executing quorum authenticated key
 management operations

      --token <TOKEN>
          Filepath where the unsigned token file will be written
  -h, --help                     Print help
```

Example

This command will write one unsigned token per HSM in your cluster to the file specified by `token`.

Example : Write one unsigned token per HSM in your cluster

```
aws-cloudhsm > quorum token-sign generate --service user --token /home/tfile
{
  "error_code": 0,
  "data": {
    "filepath": "/home/tfile"
  }
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been [configured.]

<SERVICE>

Specifies the quorum authorized service for which to generate a token. This parameter is required.

Valid values

- **user**: The user management service that is used for executing quorum authorized user management operations.

- **quorum**: The quorum management service that is used for setting quorum authorized quorum values for any quorum authorized service.
- **cluster**: The cluster management service that is used for executing quorum for cluster wide configuration managements like mtls enforcement, mtls registration and mtls deregistration.
- **registration**: Generates an unsigned token for use in registering a public key for quorum authorization.
- **key-usage**: Generates an unsigned token that is used for executing quorum authorized key usage operations.
- **key-management**: Generates an unsigned token that is used for executing quorum authorized key management operations.

Required: Yes

<TOKEN>

Filepath where the unsigned token file will be written.

Required: Yes

Related topics

- Service names and types that support quorum authentication

List quorum tokens using CloudHSM CLI

Use the **quorum token-sign list** command in CloudHSM CLI to list all token-sign quorum tokens present in your AWS CloudHSM cluster. This includes tokens generated by other users. A token is bound to a user, so while you may see tokens from other users, you will only be able to use tokens associated with the currently logged in user.

For more information about service types and names, see Service names and types that support quorum authentication. For more information about the content displayed from listed tokens, see the section called “Key management and usage with quorum (M of N)” for tokens associated with `key-management` and `key-usage` services, and see the section called “User management with quorum (M of N)” for tokens associated with `user`, `quorum`, or `cluster` service, respectively.

User type

The following users can run this command.

- Admin
- Crypto user (CU)

Syntax

```
aws-cloudhsm > help quorum token-sign list
List the token-sign tokens in your cluster

Usage: quorum token-sign list

Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
  -h, --help                     Print help
```

Example

This command will list all token-sign tokens present in your AWS CloudHSM cluster. This includes tokens generated by other users. A token is bound to a user, so while you may see tokens from other users, you will only be able to use tokens associated with the currently logged in user.

Example

```
aws-cloudhsm > quorum token-sign list
{
  "error_code": 0,
  "data": {
    "tokens": [
      {
        "username": "admin",
        "service": "quorum",
        "approvals-required": 2,
        "number-of-approvals": 0,
        "token-timeout-seconds": 397,
        "cluster-coverage": "full"
      },
      {
        "username": "admin",
        "service": "user",
        "approvals-required": 2,
```

```
          "number-of-approvals": 0,
          "token-timeout-seconds": 588,
          "cluster-coverage": "full"
        },
        {
          "username": "crypto_user1",
          "service": "key-management",
          "key-reference": "0x00000000002c33f7",
          "minimum-token-count": 1
        },
        {
          "username": "crypto_user1",
          "service": "key-usage",
          "key-reference": "0x00000000002c33f7",
          "minimum-token-count": 1
        }
      ]
    }
}
```

Related topics

- quorum token-sign generate

Show quorum values using CloudHSM CLI

Use the **quorum token-sign list-quorum-values** command in CloudHSM CLI to lists the quorum values set in your AWS CloudHSM cluster.

User type

The following users can run this command.

- All users. You do not need to be logged in to run this command.

Syntax

```
aws-cloudhsm > help quorum token-sign list-quorum-values
List current quorum values

Usage: quorum token-sign list-quorum-values
```

```
Options:
      --cluster-id <CLUSTER_ID>  Unique Id to choose which of the clusters in the
 config file to run the operation against. If not provided, will fall back to the value
 provided when interactive mode was started, or error
  -h, --help                     Print help
```

Example

This command lists quorum values set in your AWS CloudHSM cluster for each service.

Example

hsm1.medium:

```
aws-cloudhsm > quorum token-sign list-quorum-values
{
  "error_code": 0,
  "data": {
    "user": 1,
    "quorum": 1
  }
}
```

hsm2m.medium:

```
aws-cloudhsm > quorum token-sign list-quorum-values
{
  "error_code": 0,
  "data": {
    "user": 1,
    "quorum": 1,
    "cluster": 1
  }
}
```

Related topics

- Service names and types that support quorum authentication
- Setup mTLS (recommended)

Update a quorum value using CloudHSM CLI

Use the **quorum token-sign set-quorum-value** command in CloudHSM CLI to set a new quorum value for a quorum authorized service.

User type

The following users can run this command.

- Admin

Syntax

```
aws-cloudhsm > help quorum token-sign set-quorum-value
Set a quorum value

Usage: quorum token-sign set-quorum-value [OPTIONS] --service <SERVICE> --value <VALUE>

Options:
      --cluster-id <CLUSTER_ID>
          Unique Id to choose which of the clusters in the config file to run the
 operation against. If not provided, will fall back to the value provided when
 interactive mode was started, or error

      --service <SERVICE>
          Service the token will be used for

          Possible values:
          - user:
            User management service is used for executing quorum authenticated user
 management operations
          - quorum:
            Quorum management service is used for setting quorum values for any quorum
 service
          - cluster:
            Cluster management service is used for executing quorum for cluster
 wide configuration managements like mtls enforcement, mtls registration and mtls
 deregistration

      --value <VALUE>
          Value to set for service

      --approval <APPROVAL>
```

```
          Filepath of signed quorum token file to approve operation

  -h, --help
          Print help (see a summary with '-h')
```

Example

Example

In the following example, this command writes one unsigned token per HSM in your cluster to the file specified by token. When you are prompted, sign the tokens in the file.

```
aws-cloudhsm > quorum token-sign set-quorum-value --service quorum --value 2
{
  "error_code": 0,
  "data": "Set Quorum Value successful"
}
```

You can then run the **list-quorum-values** command to confirm that the quorum value for the quorum management service has been set:

hsm1.medium:

```
aws-cloudhsm > quorum token-sign list-quorum-values
{
  "error_code": 0,
  "data": {
    "user": 1,
    "quorum": 2
  }
}
```

hsm2m.medium:

```
aws-cloudhsm > quorum token-sign list-quorum-values
{
  "error_code": 0,
  "data": {
    "user": 1,
    "quorum": 2,
    "cluster": 1
  }
```

```
}
```

Arguments

<CLUSTER_ID>

The ID of the cluster to run this operation on.

Required: If multiple clusters have been [configured.](#)

<APPROVAL>

The filepath of the signed token file to be approved on the HSM.

<SERVICE>

Specifies the quorum authorized service for which to generate a token. This parameter is required. For more information about service types and names, see [Service names and types that support quorum authentication](#).

Valid values

- **user**: The user management service. Service used for executing quorum authorized user management operations.
- **quorum**: The quorum management service. Service used for setting a quorum authorized quorum values for any quorum authorized service.
- **cluster**: The cluster management service that is used for executing quorum for cluster wide configuration managements like mtls enforcement, mtls registration and mtls deregistration.
- **registration**: Generates a unsigned token for use in registering a public key for quorum authorization.

Required: Yes

<VALUE>

Specifies The quorum value to be set. The maximum quorum value is eight (8).

Require: Yes

Related topics

- [quorum token-sign list-quorum-values](#)

- Service names and types that support quorum authentication
- Setup mTLS (recommended)

AWS CloudHSM Management Utility (CMU)

The **cloudhsm_mgmt_util** command line tool helps crypto officers manage users in the hardware security modules (HSMs) in AWS CloudHSM clusters. The AWS CloudHSM Management Utility (CMU) includes tools that create, delete, and list users, and change user passwords.

The CMU and Key Management Utility (KMU) are part of the Client SDK 3 suite. Client SDK 3 and its related command line tools (Key Management Utility and CloudHSM Management Utility) are only available in the HSM type *hsm1.medium*.

cloudhsm_mgmt_util also includes commands that allow crypto users (CUs) to share keys and get and set key attributes. These commands complement the key management commands in the primary key management tool, key_mgmt_util.

For a quick start, see Cloned clusters in AWS CloudHSM. For detailed information about the cloudhsm_mgmt_util commands and examples of using the commands, see Reference for AWS CloudHSM Management Utility commands .

Topics

- Supported platforms for AWS CloudHSM Management Utility
- Getting started with AWS CloudHSM Management Utility (CMU)
- Install and configure the AWS CloudHSM client for CMU (Linux)
- Install and configure the AWS CloudHSM client for CMU (Windows)
- Reference for AWS CloudHSM Management Utility commands

Supported platforms for AWS CloudHSM Management Utility

This topic describes the Linux and Windows platforms that the AWS CloudHSM Management Utility (CMU) supports.

Linux support

- Amazon Linux
- Amazon Linux 2

- CentOS 6.10+
- CentOS 7.3+
- CentOS 8
- Red Hat Enterprise Linux (RHEL) 6.10+
- Red Hat Enterprise Linux (RHEL) 7.9+
- Red Hat Enterprise Linux (RHEL) 8
- Ubuntu 16.04 LTS
- Ubuntu 18.04 LTS

Windows support

- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2
- Microsoft Windows Server 2016
- Microsoft Windows Server 2019

Getting started with AWS CloudHSM Management Utility (CMU)

AWS CloudHSM Management Utility (CMU) enables you to manage hardware security module (HSM) users. Use this topic to get started with basic HSM user management tasks, such as creating users, listing users, and connecting CMU to the cluster.

1. To use CMU, you must first use the configure tool to update the local CMU configuration with the `--cmu` parameter and an IP address from one of the HSMs in your cluster. Do this each time you use CMU to ensure you're managing HSM users on every HSM in the cluster.

   Linux

   ```
   $ sudo /opt/cloudhsm/bin/configure --cmu <IP address>
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" --cmu <IP address>
   ```

2. Use the following command to start the CLI in interactive mode.

Linux

```
$ /opt/cloudhsm/bin/cloudhsm_mgmt_util /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\cloudhsm_mgmt_util.exe" C:
\ProgramData\Amazon\CloudHSM\data\cloudhsm_mgmt_util.cfg
```

Output should be similar to the following depending on how many HSMs you have.

```
Connecting to the server(s), it may take time
depending on the server(s) load, please wait...

Connecting to server '10.0.2.9': hostname '10.0.2.9', port 2225...
Connected to server '10.0.2.9': hostname '10.0.2.9', port 2225.

Connecting to server '10.0.3.11': hostname '10.0.3.11', port 2225...
Connected to server '10.0.3.11': hostname '10.0.3.11', port 2225.

Connecting to server '10.0.1.12': hostname '10.0.1.12', port 2225...
Connected to server '10.0.1.12': hostname '10.0.1.12', port 2225.
```

The prompt changes to `aws-cloudhsm>` when cloudhsm_mgmt_util is running.

3. Use the **loginHSM** command to log in to the cluster. Any type user can use this command to log in to the cluster.

   The command in the following example logs in *admin*, which is the default crypto officer (CO). You set this user's password when you activated the cluster. You can use the `-hpswd` parameter to hide your password.

   ```
   aws-cloudhsm>loginHSM CO admin -hpswd
   ```

   The system prompts you for your password. You enter the password, the system hides the password, and the output shows that the command was successful and that the you have connected to all the HSMs on the cluster.

```
Enter password:


loginHSM success on server 0(10.0.2.9)
loginHSM success on server 1(10.0.3.11)
loginHSM success on server 2(10.0.1.12)
```

4. Use **listUsers** to list all the users on the cluster.

```
aws-cloudhsm>listUsers
```

CMU lists all the users on the cluster.

```

Users on server 0(10.0.2.9):
Number of users found:2

    User Id             User Type       User Name                          
 MofnPubKey    LoginFailureCnt         2FA
         1              CO              admin                                    NO
               0               NO
         2              AU              app_user                                 NO
               0               NO
Users on server 1(10.0.3.11):
Number of users found:2

    User Id             User Type       User Name                          
 MofnPubKey    LoginFailureCnt         2FA
         1              CO              admin                                    NO
               0               NO
         2              AU              app_user                                 NO
               0               NO
Users on server 2(10.0.1.12):
Number of users found:2

    User Id             User Type       User Name                          
 MofnPubKey    LoginFailureCnt         2FA
         1              CO              admin                                    NO
               0               NO
         2              AU              app_user                                 NO
               0               NO
```

5. Use **createUser** to create a CU user named **`example_user`** with a password of **`password1`**.

   You use CU users in your applications to perform cryptographic and key management operations. You can create CU users because in step 3 you logged in as a CO user. Only CO users can perform user management tasks with CMU, such as creating and deleting users and changing the passwords of other users.

   ```
   aws-cloudhsm>createUser CU example_user password1
   ```

   CMU prompts you about the create user operation.

   ```

   *************************CAUTION********************************
   This is a CRITICAL operation, should be done on all nodes in the
   cluster. AWS does NOT synchronize these changes automatically with the
   nodes on which this operation is not executed or failed, please
   ensure this operation is executed on all nodes in the cluster.
   ****************************************************************

   Do you want to continue(y/n)?
   ```

6. To create the CU user **`example_user`**, type **y**.
7. Use **listUsers** to list all the users on the cluster.

   ```
   aws-cloudhsm>listUsers
   ```

   CMU lists all the users on the cluster, including the new CU user you just created.

   ```

   Users on server 0(10.0.2.9):
   Number of users found:3

       User Id             User Type       User Name
    MofnPubKey    LoginFailureCnt         2FA
            1              CO              admin                            NO
                  0               NO
            2              AU              app_user                         NO
                  0               NO
            3              CU              example_user                     NO
                  0               NO
   Users on server 1(10.0.3.11):
   ```

```
Number of users found:3

    User Id             User Type       User Name                                MofnPubKey    LoginFailureCnt         2FA
         1              CO              admin                                    NO               0               NO
         2              AU              app_user                                 NO               0               NO
         3              CU              example_user                             NO               0               NO
Users on server 2(10.0.1.12):
Number of users found:3

    User Id             User Type       User Name                                MofnPubKey    LoginFailureCnt         2FA
         1              CO              admin                                    NO               0               NO
         2              AU              app_user                                 NO               0               NO
         3              CU              example_user                             NO               0               NO
```

8. Use the **logoutHSM** command to log out of the HSMs.

```
aws-cloudhsm>logoutHSM


logoutHSM success on server 0(10.0.2.9)
logoutHSM success on server 1(10.0.3.11)
logoutHSM success on server 2(10.0.1.12)
```

9. Use the **quit** command to stop cloudhsm_mgmt_util.

```
aws-cloudhsm>quit


disconnecting from servers, please wait...
```

Install and configure the AWS CloudHSM client for CMU (Linux)

To interact with the hardware security module (HSM) in your AWS CloudHSM cluster using the cloudhsm_mgmt_util (CMU), you need the AWS CloudHSM client software for Linux. You should install it on the Linux Amazon EC2 client instance that you created previously. You can also install a client if you are using Windows. For more information, see Install and configure the AWS CloudHSM client for CMU (Windows).

Tasks

Step 1. Install the AWS CloudHSM client and command line tools

Connect to your client instance and run the following commands to download and install the AWS CloudHSM client and command line tools.

Amazon Linux

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-latest.el6.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 7

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 8

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-latest.el8.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el8.x86_64.rpm
```

RHEL 7

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm
```

RHEL 8

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-latest.el8.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client_latest_amd64.deb
```

```
sudo apt install ./cloudhsm-client_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client_latest_u18.04_amd64.deb
```

```
sudo apt install ./cloudhsm-client_latest_u18.04_amd64.deb
```

Step 2. Edit the client configuration

Before you can use the AWS CloudHSM client to connect to your cluster, you must edit the client configuration.

To edit the client configuration

1. If installing Client SDK 3 on cloudhsm_mgmt_util, complete the following steps to ensure all the nodes in the cluster are synced.

   a. Run **configure -a** ***<IP of one of the HSMs>***.

   b. Restart the client service.

   c. Run **configure -m**.

2. Copy your issuing certificate—the one that you used to sign the cluster's certificate—to the following location on the client instance: `/opt/cloudhsm/etc/customerCA.crt`. You need instance root user permissions on the client instance to copy your certificate to this location.

3. Use the following configure command to update the configuration files for the AWS CloudHSM client and command line tools, specifying the IP address of the HSM in your cluster. To get the HSM's IP address, view your cluster in the AWS CloudHSM console, or run the **describe-clusters** AWS CLI command. In the command's output, the HSM's IP address is the value of the `EniIp` field. If you have more than one HSM, choose the IP address for any of the HSMs; it doesn't matter which one.

   ```
   sudo /opt/cloudhsm/bin/configure -a <IP address>

   Updating server config in /opt/cloudhsm/etc/cloudhsm_client.cfg
   Updating server config in /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
   ```

4. Go to Activate the cluster in AWS CloudHSM.

Install and configure the AWS CloudHSM client for CMU (Windows)

To work with a hardware security module (HSM) in your AWS CloudHSM cluster on Windows using the cloudhsm_mgmt_util (CMU), you need the AWS CloudHSM client software for Windows. You should install it on the Windows Server instance that you created previously.

> **Note**
>
> - If you are updating the client, existing configuration files from previous installations are *not* overwritten.
> - The AWS CloudHSM client installer for Windows automatically registers the Cryptography API: Next Generation (CNG) and Key Storage Provider (KSP). To uninstall the client, run the installer again and follow the uninstall instructions.
> - If you are using Linux, you can install the Linux client. For more information, see Install and configure the AWS CloudHSM client for CMU (Linux).

To install (or update) the latest Windows client and command line tools

1. Connect to your Windows Server instance.
2. Download the AWSCloudHSMClient-latest.msi installer.
3. If installing Client SDK 3 on cloudhsm_mgmt_util, complete the following steps to ensure all the nodes in the cluster are synced.
   a. Run **configure.exe -a** ***<IP of one of the HSMs>***.
   b. Restart the client service.
   c. Run **configure.exe -m**.
4. Go to your download location and run the installer (**AWSCloudHSMClient-latest.msi**) with administrative privilege.
5. Follow the installer instructions, then choose **Close** after the installer has finished.
6. Copy your self-signed issuing certificate—the one that you used to sign the cluster certificate—to the `C:\ProgramData\Amazon\CloudHSM` folder.
7. Run the following command to update your configuration files. Be sure to stop and start the client during reconfiguration if you are updating it:

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" -a <HSM IP address>
```

8. Go to Activate the cluster in AWS CloudHSM.

Reference for AWS CloudHSM Management Utility commands

The AWS CloudHSM cloudhsm_mgmt_util command line tool helps crypto officers manage users in the hardware security modules (HSMs) in the AWS CloudHSM cluster. It also includes commands that allow crypto users (CUs) to share keys, and get and set key attributes. These commands complement the primary key management commands in the key_mgmt_util command line tool.

For a quick start, see Cloned clusters in AWS CloudHSM.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util and log in to the HSM. Be sure that you log in with the user account type that can run the commands you plan to use.

To list all cloudhsm_mgmt_util commands, run the following command:

```
aws-cloudhsm> help
```

To get the syntax for a cloudhsm_mgmt_util command, run the following command:

```
aws-cloudhsm> help <command-name>
```

> **Note**
>
> Use the syntax as per the documentation. While the built-in software help may provide additional options, these should not be considered supported and should not be utilized in production code.

To run a command, enter the command name, or enough of the name to distinguish it from the names of other cloudhsm_mgmt_util commands.

For example, to get a list of users on the HSMs, enter **listUsers** or **listU**.

```
aws-cloudhsm> listUsers
```

To end your cloudhsm_mgmt_util session, run the following command:

```
aws-cloudhsm> quit
```

For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

The following topics describe commands in cloudhsm_mgmt_util.

> **Note**
>
> Some commands in key_mgmt_util and cloudhsm_mgmt_util have the same names. However, the commands typically have different syntax, different output, and slightly different functionality.

Command	Description	User Type
changePswd	Changes the passwords of users on the HSMs. Any user can change their own password. COs can change anyone's password.	CO
createUser	Creates users of all types on the HSMs.	CO
deleteUser	Deletes users of all types from the HSMs.	CO
findAllKeys	Gets the keys that a user owns or shares. Also gets a hash of the key ownership and sharing data for all keys on each HSM.	CO, AU
getAttribute	Gets an attribute value for an AWS CloudHSM key and	CU

Command	Description	User Type
	writes it to a file or stdout (standard output).	
getHSMInfo	Gets information about the hardware on which an HSM is running.	All. Login is not required.
getKeyInfo	Gets owners, shared users, and the quorum authentic ation status of a key.	All. Login is not required.
info	Gets information about an HSM, including the IP address, hostname, port, and current user.	All. Login is not required.
listUsers	Gets the users in each of the HSMs, their user type and ID, and other attributes.	All. Login is not required.
loginHSM and logoutHSM	Log in and log out of an HSM.	All.
quit	Quits cloudhsm_mgmt_util.	All. Login is not required.
server	Enters and exits server mode on an HSM.	All.
registerQuorumPubKey	Associates an HSM user with an asymmetric RSA-2048 key pair.	CO
setAttribute	Changes the values of the label, encrypt, decrypt, wrap, and unwrap attributes of an existing key.	CU

Command	Description	User Type
shareKey	Shares an existing key with other users.	CU
syncKey	Syncs a key across cloned AWS CloudHSM clusters.	CU, CO
syncUser	Syncs a user across cloned AWS CloudHSM clusters.	CO

Change a user's password using CMU

Use the **changePswd** command in the AWS CloudHSM cloudhsm_mgmt_util (CMU) to change the password of an existing user on the hardware security modules (HSM) in the AWS CloudHSM cluster.

Any user can change their own password. In addition, Crypto officers (COs and PCOs) can change the password of another CO or crypto user (CU). You do not need to enter the current password to make the change.

> **Note**
>
> You cannot change the password of a user who is currently logged into the AWS CloudHSM client or key_mgmt_util.

To troubleshoot changePswd

Before you run any CMU command, you must start CMU and log in to the HSM. Be sure that you log in with a user type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files for CMU. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User type

The following users can run this command.

- Crypto officers (CO)
- Crypto users (CU)

Syntax

Enter the arguments in the order specified in the syntax diagram. Use the `-hpswd` parameter to mask your password. To enable two-factor authentication (2FA) for a CO user, use the `-2fa` parameter and include a file path. For more information, see the section called "Arguments".

```
changePswd <user-type> <user-name> <password |-hpswd> [-2fa </path/to/authdata>]
```

Examples

The following examples show how to use **changePassword** to reset the password for the current user or any other user in your HSMs.

Example : Change your password

Any user on the HSMs can use **changePswd** to change their own password. Before you change the password, use info to get information about each of the HSMs in the cluster, including the username and the user type of the logged in user.

The following output shows that Bob is currently logged in as a crypto user(CU).

```
        aws-cloudhsm> info server 0

Id       Name                     Hostname          Port    State           Partition
  LoginState
0        10.1.9.193               10.1.9.193       2225    Connected       hsm-jqici4covtv
  Logged in as 'bob(CU)'

aws-cloudhsm> info server 1

Id       Name                     Hostname          Port    State           Partition
  LoginState
1        10.1.10.7                10.1.10.7        2225    Connected       hsm-ogi3sywxbqx
  Logged in as 'bob(CU)'
```

To change password, Bob runs **changePswd** followed with the user type, username, and a new password.

```
aws-cloudhsm> changePswd CU bob newPassword

*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)?y
Changing password for bob(CU) on 2 nodes
```

Example : Change the password of another user

You must be a CO or PCO to change the password of another CO, or CU on the HSMs. Before you change the password for another user, use the info command to confirm that your user type is either CO or PCO.

The following output confirms that Alice, who is a CO, is currently logged in.

```

aws-cloudhsm>info server 0

Id      Name                Hostname          Port    State           Partition
 LoginState
0       10.1.9.193          10.1.9.193         2225    Connected       hsm-jqici4covtv
 Logged in as 'alice(CO)'


aws-cloudhsm>info server 1

Id      Name                Hostname          Port    State           Partition
 LoginState
0       10.1.10.7           10.1.10.7         2225    Connected       hsm-ogi3sywxbqx
 Logged in as 'alice(CO)'
```

Alice wants to reset the password of another user, John. Before she changes the password, she uses the listUsers command to verify John's user type.

The following output lists John as a CO user.

```
aws-cloudhsm> listUsers
Users on server 0(10.1.9.193):
Number of users found:5

    User Id             User Type       User Name            MofnPubKey
 LoginFailureCnt         2FA
         1              PCO             admin                     YES              0
             NO
         2              AU              jane                       NO              0
             NO
         3              CU              bob                        NO              0
             NO
         4              CU              alice                      NO              0
             NO
         5              CO              john                       NO              0
             NO
Users on server 1(10.1.10.7):
Number of users found:5

    User Id             User Type       User Name            MofnPubKey
 LoginFailureCnt         2FA
         1              PCO             admin                     YES              0
             NO
         2              AU              jane                       NO              0
             NO
         3              CU              bob                        NO              0
             NO
         4              CO              alice                      NO              0
             NO
         5              CO              john                       NO              0
             NO
```

To change the password, Alice runs **changePswd** followed with John's user type, username, and a new password.

```
aws-cloudhsm>changePswd CO john newPassword

*************************CAUTION********************************
```

```
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)?y
Changing password for john(CO) on 2 nodes
```

Arguments

Enter the arguments in the order specified in the syntax diagram. Use the `-hpswd` parameter to mask your password. To enable 2FA for a CO user, use the `-2fa` parameter and include a file path. For more information about working with 2FA, see Manage user 2FA

```
changePswd <user-type> <user-name> <password |-hpswd> [-2fa </path/to/authdata>]
```

<user-type>

Specifies the current type of the user whose password you are changing. You cannot use **changePswd** to change the user type.

Valid values are CO, CU, PCO, and PRECO.

To get the user type, use listUsers. For detailed information about the user types on an HSM, see HSM user types for AWS CloudHSM Management Utility.

Required: Yes

<user-name>

Specifies the user's friendly name. This parameter is not case-sensitive. You cannot use **changePswd** to change the user name.

Required: Yes

<password | -hpswd >

Specifies a new password for the user. Enter a string of 7 to 32 characters. This value is case sensitive. The password appears in plaintext when you type it. To hide your password, use the `-hpswd` parameter in place of the password and follow the prompts.

Required: Yes

[-2fa </path/to/authdata>]

Specifies enabling 2FA for this CO user. To get the data necessary for setting up 2FA, include a path to a location in the file system with a file name after the `-2fa` parameter. For more information about working with 2FA, see Manage user 2FA.

Required: No

Related topics

- info
- listUsers
- createUser
- deleteUser

Create an AWS CloudHSM user with CMU

Use the **createUser** command in cloudhsm_mgmt_util (CMU) to create a user on the hardware security modules (HSM) in the AWS CloudHSM cluster. Only crypto officers (COs and PRECOs) can run this command. When the command succeeds, it creates the user in all HSMs in the cluster.

To troubleshoot createUser

If your HSM configuration is inaccurate, the user might not be created on all HSMs. To add the user to any HSMs in which it is missing, use the syncUser or createUser command only on the HSMs that are missing that user. To prevent configuration errors, run the configure tool with the `-m` option.

Before you run any CMU command, you must start CMU and log in to the HSM. Be sure that you log in with a user type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files for CMU. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User type

The following types of users can run this command.

- Crypto officers (CO, PRECO)

Syntax

Enter the arguments in the order specified in the syntax diagram. Use the `-hpswd` parameter to mask your password. To create a CO user with two-factor authentication (2FA), use the `-2fa` parameter and include a file path. For more information, see the section called “Arguments”.

```
createUser <user-type> <user-name> <password> |-hpswd> [-2fa </path/to/authdata>]
```

Examples

These examples show how to use **createUser** to create new users in your HSMs.

Example : Create a crypto officer

This example creates a crypto officer (CO) on the HSMs in a cluster. The first command uses loginHSM to log in to the HSM as a crypto officer.

```
aws-cloudhsm> loginHSM CO admin 735782961

loginHSM success on server 0(10.0.0.1)
loginHSM success on server 1(10.0.0.2)
loginHSM success on server 1(10.0.0.3)
```

The second command uses the **createUser** command to create `alice`, a new crypto officer on the HSM.

The caution message explains that the command creates users on all of the HSMs in the cluster. But, if the command fails on any HSMs, the user will not exist on those HSMs. To continue, type `y`.

The output shows that the new user was created on all three HSMs in the cluster.

```
aws-cloudhsm> createUser CO alice 391019314

*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)?Invalid option, please type 'y' or 'n'
```

```
Do you want to continue(y/n)?y
Creating User alice(CO) on 3 nodes
```

When the command completes, `alice` has the same permissions on the HSM as the `admin` CO user, including changing the password of any user on the HSMs.

The final command uses the listUsers command to verify that `alice` exists on all three HSMs on the cluster. The output also shows that `alice` is assigned user ID `3.`. You use the user ID to identify `alice` in other commands, such as findAllKeys.

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:3

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                                   YES
           0               NO
         2              AU              app_user                                 NO
           0               NO
         3              CO              alice                                    NO
           0               NO
Users on server 1(10.0.0.2):
Number of users found:3

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                                   YES
           0               NO
         2              AU              app_user                                 NO
           0               NO
         3              CO              alice                                    NO
           0               NO

Users on server 1(10.0.0.3):
Number of users found:3

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
         1              PRECO           admin                                   YES
           0               NO
         2              AU              app_user                                 NO
           0               NO
```

```
          3              CO              alice                                  NO
            0              NO
```

Example : Create a crypto user

This example creates a crypto user (CU), bob, on the HSM. Crypto users can create and manage keys, but they cannot manage users.

After you type y to respond to the caution message, the output shows that bob was created on all three HSMs in the cluster. The new CU can log in to the HSM to create and manage keys.

The command used a password value of defaultPassword. Later, bob or any CO can use the changePswd command to change his password.

```
aws-cloudhsm> createUser CU bob defaultPassword

*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)?Invalid option, please type 'y' or 'n'

Do you want to continue(y/n)?y
Creating User bob(CU) on 3 nodes
```

Arguments

Enter the arguments in the order specified in the syntax diagram. Use the -hpswd parameter to mask your password. To create a CO user with 2FA enabled, use the -2fa parameter and include a file path. For more information about 2FA, see Manage user 2FA.

```
createUser <user-type> <user-name> <password> |-hpswd> [-2fa </path/to/authdata>]
```

<user-type>

Specifies the type of user. This parameter is required.

For detailed information about the user types on an HSM, see HSM user types for AWS CloudHSM Management Utility.

Valid values:

- **CO**: Crypto officers can manage users, but they cannot manage keys.
- **CU**: Crypto users can create an manage keys and use keys in cryptographic operations.

The PRECO is converted to a CO when you assign a password during HSM activation.

Required: Yes

<user-name>

Specifies a friendly name for the user. The maximum length is 31 characters. The only special character permitted is an underscore (_).

You cannot change the name of a user after it is created. In cloudhsm_mgmt_util commands, the user type and password are case-sensitive, but the user name is not.

Required: Yes

<password | -hpswd >

Specifies a password for the user. Enter a string of 7 to 32 characters. This value is case-sensitive. The password appears in plaintext when you type it. To hide your password, use the `-hpswd` parameter in place of the password and follow the prompts.

To change a user password, use changePswd. Any HSM user can change their own password, but CO users can change the password of any user (of any type) on the HSMs.

Required: Yes

[-2fa </path/to/authdata>]

Specifies the creation of a CO user with 2FA enabled. To get the data necessary for setting up 2FA authentication, include a path to a location in the file system with a file name after the `-2fa` parameter. For more information about setting up and working with 2FA, see Manage user 2FA.

Required: No

Related topics

- listUsers

- deleteUser
- syncUser
- changePswd

Delete an AWS CloudHSM user using CMU

Use the **deleteUser** command in the AWS CloudHSM cloudhsm_mgmt_util (CMU) to delete a user from the hardware security modules (HSM) in the AWS CloudHSM cluster. Only crypto officers (CO) can run this command. You cannot delete a user who is currently logged into an HSM. For more information about deleting users, see How to Delete HSM Users.

> **Tip**
>
> You can't delete crypto users (CU) that own keys.

User type

The following types of users can run this command.

- CO

Syntax

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
deleteUser <user-type> <user-name>
```

Example

This example deletes a crypto officer (CO) from the HSMs in a cluster. The first command uses listUsers to list all users on the HSMs.

The output shows that user 3, `alice`, is a CO on the HSMs.

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
```

```
Number of users found:3

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                                    YES
           0               NO
         2              AU              app_user                                  NO
           0               NO
         3              CO              alice                                     NO
           0               NO
Users on server 1(10.0.0.2):
Number of users found:3

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                                    YES
           0               NO
         2              AU              app_user                                  NO
           0               NO
         3              CO              alice                                     NO
           0               NO

Users on server 1(10.0.0.3):
Number of users found:3

    User Id             User Type       User Name                          MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                                    YES
           0               NO
         2              AU              app_user                                  NO
           0               NO
         3              CO              alice                                     NO
           0               NO
```

The second command uses the **deleteUser** command to delete `alice` from the HSMs.

The output shows that the command succeeded on all three HSMs in the cluster.

```
aws-cloudhsm> deleteUser CO alice
Deleting user alice(CO) on 3 nodes
deleteUser success on server 0(10.0.0.1)
deleteUser success on server 0(10.0.0.2)
deleteUser success on server 0(10.0.0.3)
```

The final command uses the **listUsers** command to verify that `alice` is deleted from all three of the HSMs on the cluster.

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:2

    User Id             User Type       User Name                           MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                                    YES
           0               NO
         2              AU              app_user                                  NO
           0               NO
Users on server 1(10.0.0.2):
Number of users found:2

    User Id             User Type       User Name                           MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                                    YES
           0               NO
         2              AU              app_user                                  NO
           0               NO
Users on server 1(10.0.0.3):
Number of users found:2

    User Id             User Type       User Name                           MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                                    YES
           0               NO
         2              AU              app_user                                  NO
           0               NO
```

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
deleteUser <user-type> <user-name>
```

<user-type>

Specifies the type of user. This parameter is required.

> **Tip**
>
> You can't delete crypto users (CU) that own keys.

Valid values are **CO**, **CU**.

To get the user type, use listUsers. For detailed information about the user types on an HSM, see HSM user types for AWS CloudHSM Management Utility.

Required: Yes

<user-name>

Specifies a friendly name for the user. The maximum length is 31 characters. The only special character permitted is an underscore (_).

You cannot change the name of a user after it is created. In cloudhsm_mgmt_util commands, the user type and password are case-sensitive, but the user name is not.

Required: Yes

Related topics

- listUsers
- createUser
- syncUser
- changePswd

List the keys that an AWS CloudHSM crypto user owns using CMU

Use the **findAllKeys** command in the AWS CloudHSM cloudhsm_mgmt_util (CMU) to get the keys that a specified crypto user (CU) of AWS CloudHSM owns or shares. The command also returns a hash of the user data on each of the HSMs. You can use the hash to determine at a glance whether the users, key ownership, and key sharing data are the same on all HSMs in the cluster. In the output, the keys owned by the user are annotated by `(o)` and shared keys are annotated by `(s)`.

findAllKeys returns public keys only when the specified CU owns the key, even though all CUs on the HSM can use any public key. This behavior is different from findKey in key_mgmt_util, which returns public keys for all CU users.

Only crypto officers (COs and PCOs) and appliance users (AUs) can run this command. Crypto users (CUs) can run the following commands:

- listUsers to find all users
- findKey in key_mgmt_util to find the keys that they can use
- getKeyInfo in key_mgmt_util to find the owner and shared users of a particular key they own or share

Before you run any CMU command, you must start CMU and log in to the HSM. Be sure that you log in with a user type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files for CMU. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User type

The following users can run this command.

- Crypto officers (CO, PCO)
- Appliance users (AU)

Syntax

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
findAllKeys <user id> <key hash (0/1)> [<output file>]
```

Examples

These examples show how to use `findAllKeys` to find all keys for a user and get a hash of key user information on each of the HSMs.

Example : Find the keys for a CU

This example uses **findAllKeys** to find the keys in the HSMs that user 4 owns and shares. The command uses a value of `0` for the second argument to suppress the hash value. Because it omits the optional file name, the command writes to stdout (standard output).

The output shows that user 4 can use 6 keys: 8, 9, 17, 262162, 19, and 31. The output uses an `(s)` to indicate keys that are explicitly shared by the user. The keys that the user owns are indicated by an `(o)` and include symmetric and private keys that the user does not share, and public keys that are available to all crypto users.

```
aws-cloudhsm> findAllKeys 4 0
Keys on server 0(10.0.0.1):
Number of keys found 6
number of keys matched from start index 0::6
8(s),9(s),17,262162(s),19(o),31(o)
findAllKeys success on server 0(10.0.0.1)

Keys on server 1(10.0.0.2):
Number of keys found 6
number of keys matched from start index 0::6
8(s),9(s),17,262162(s),19(o),31(o)
findAllKeys success on server 1(10.0.0.2)

Keys on server 1(10.0.0.3):
Number of keys found 6
number of keys matched from start index 0::6
8(s),9(s),17,262162(s),19(o),31(o)
findAllKeys success on server 1(10.0.0.3)
```

Example : Verify that user data is synchronized

This example uses **findAllKeys** to verify that all of the HSMs in the cluster contain the same users, key ownership, and key sharing values. To do this, it gets a hash of the key user data on each HSM and compares the hash values.

To get the key hash, the command uses a value of `1` in the second argument. The optional file name is omitted, so the command writes the key hash to stdout.

The example specifies user `6`, but the hash value will be the same for any user that owns or shares any of the keys on the HSMs. If the specified user does not own or share any keys, such as a CO, the command does not return a hash value.

The output shows that the key hash is identical to both of the HSMs in the cluster. If one of the HSM had different users, different key owners, or different shared users, the key hash values would not be equal.

```
aws-cloudhsm> findAllKeys 6 1
```

```
Keys on server 0(10.0.0.1):
Number of keys found 3
number of keys matched from start index 0::3
8(s),9(s),11,17(s)
Key Hash:
55655676c95547fd4e82189a072ee1100eccfca6f10509077a0d6936a976bd49

findAllKeys success on server 0(10.0.0.1)
Keys on server 1(10.0.0.2):
Number of keys found 3
number of keys matched from start index 0::3
8(s),9(s),11(o),17(s)
Key Hash:
55655676c95547fd4e82189a072ee1100eccfca6f10509077a0d6936a976bd49

findAllKeys success on server 1(10.0.0.2)
```

This command demonstrates that the hash value represents the user data for all keys on the HSM. The command uses the **findAllKeys** for user 3. Unlike user 6, who owns or shares just 3 keys, user 3 own or shares 17 keys, but the key hash value is the same.

```
aws-cloudhsm> findAllKeys 3 1
Keys on server 0(10.0.0.1):
Number of keys found 17
number of keys matched from start index 0::17
6(o),7(o),8(s),11(o),12(o),14(o),262159(o),262160(o),17(s),262162(s),19(s),20(o),21(o),262177(o
Key Hash:
55655676c95547fd4e82189a072ee1100eccfca6f10509077a0d6936a976bd49

findAllKeys success on server 0(10.0.0.1)
Keys on server 1(10.0.0.2):
Number of keys found 17
number of keys matched from start index 0::17
6(o),7(o),8(s),11(o),12(o),14(o),262159(o),262160(o),17(s),262162(s),19(s),20(o),21(o),262177(o
Key Hash:
55655676c95547fd4e82189a072ee1100eccfca6f10509077a0d6936a976bd49

findAllKeys success on server 1(10.0.0.2)
```

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
findAllKeys <user id> <key hash (0/1)> [<output file>]
```

<user id>

Gets all keys that the specified user owns or shares. Enter the user ID of a user on the HSMs. To find the user IDs of all users, use listUsers.

All user IDs are valid, but `findAllKeys` returns keys only for crypto users (CUs).

Required: Yes

<key hash>

Includes (1) or excludes (`0`) a hash of the user ownership and sharing data for all keys in each HSM.

When the `user id` argument represents a user who owns or shares keys, the key hash is populated. The key hash value is identical for all users who own or share keys on the HSM, even though they own and share different keys. However, when the `user id` represents a user who does not own or share any keys, such as a CO, the hash value is not populated.

Required: Yes

<output file>

Writes the output to the specified file.

Required: No

Default: Stdout

Related topics

- changePswd
- deleteUser
- listUsers
- syncUser
- findKey in key_mgmt_util
- getKeyInfo in key_mgmt_util

Get an AWS CloudHSM key attribute value using CMU

Use the **getAttribute** command in the AWS CloudHSM cloudhsm_mgmt_util (CMU) to get one attribute value for a key from all hardware security modules (HSM) in the AWS CloudHSM cluster and writes it to stdout (standard output) or to a file. Only crypto users (CUs) can run this command.

Key attributes are properties of a key. They include characteristics, like the key type, class, label, and ID, and values that represent actions that you can perform on the key, like encrypt, decrypt, wrap, sign, and verify.

You can use **getAttribute** only on keys that you own and key that are shared with you. You can run this command or the getAttribute command in key_mgmt_util, which writes one or all of the attribute values of a key to a file.

To get a list of attributes and the constants that represent them, use the listAttributes command. To change the attribute values of existing keys, use setAttribute in key_mgmt_util and setAttribute in cloudhsm_mgmt_util. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

Before you run any CMU command, you must start CMU and log in to the HSM. Be sure that you log in with a user type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files for CMU. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User type

The following users can run this command.

- Crypto users (CU)

Syntax

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
getAttribute <key handle> <attribute id> [<filename>]
```

Example

This example gets the value of the extractable attribute for a key in the HSMs. You can use a command like this to determine whether you can export a key from the HSMs.

The first command uses listAttributes to find the constant that represents the extractable attribute. The output shows that the constant for `OBJ_ATTR_EXTRACTABLE` is `354`. You can also find this information with descriptions of the attributes and their values in the AWS CloudHSM key attribute reference for KMU.

```
aws-cloudhsm> listAttributes

Following are the possible attribute values for getAttribute:

      OBJ_ATTR_CLASS                  = 0
      OBJ_ATTR_TOKEN                  = 1
      OBJ_ATTR_PRIVATE                = 2
      OBJ_ATTR_LABEL                  = 3
      OBJ_ATTR_TRUSTED                = 134
      OBJ_ATTR_KEY_TYPE               = 256
      OBJ_ATTR_ID                     = 258
      OBJ_ATTR_SENSITIVE              = 259
      OBJ_ATTR_ENCRYPT                = 260
      OBJ_ATTR_DECRYPT                = 261
      OBJ_ATTR_WRAP                   = 262
      OBJ_ATTR_UNWRAP                 = 263
      OBJ_ATTR_SIGN                   = 264
      OBJ_ATTR_VERIFY                 = 266
      OBJ_ATTR_DERIVE                 = 268
      OBJ_ATTR_LOCAL                  = 355
      OBJ_ATTR_MODULUS                = 288
      OBJ_ATTR_MODULUS_BITS           = 289
      OBJ_ATTR_PUBLIC_EXPONENT        = 290
      OBJ_ATTR_VALUE_LEN              = 353
      OBJ_ATTR_EXTRACTABLE            = 354
      OBJ_ATTR_NEVER_EXTRACTABLE      = 356
      OBJ_ATTR_ALWAYS_SENSITIVE       = 357
      OBJ_ATTR_DESTROYABLE            = 370
      OBJ_ATTR_KCV                    = 371
      OBJ_ATTR_WRAP_WITH_TRUSTED      = 528
      OBJ_ATTR_WRAP_TEMPLATE          = 1073742353
      OBJ_ATTR_UNWRAP_TEMPLATE        = 1073742354
      OBJ_ATTR_ALL                    = 512
```

The second command uses **getAttribute** to get the value of the extractable attribute for the key with key handle `262170` in the HSMs. To specify the extractable attribute, the command uses `354`, the constant that represents the attribute. Because the command does not specify a file name, **getAttribute** writes the output to stdout.

The output shows that the value of the extractable attribute is 1 on all of the HSM. This value indicates that the owner of the key can export it. When the value is 0 (0x0), it cannot be exported from the HSMs. You set the value of the extractable attribute when you create a key, but you cannot change it.

```
aws-cloudhsm> getAttribute 262170 354

Attribute Value on server 0(10.0.1.10):
OBJ_ATTR_EXTRACTABLE
0x00000001

Attribute Value on server 1(10.0.1.12):
OBJ_ATTR_EXTRACTABLE
0x00000001

Attribute Value on server 2(10.0.1.7):
OBJ_ATTR_EXTRACTABLE
0x00000001
```

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
getAttribute <key handle> <attribute id> [<filename>]
```

<key-handle>

Specifies the key handle of the target key. You can specify only one key in each command. To get the key handle of a key, use findKey in key_mgmt_util.

You must own the specified key or it must be shared with you. To find the users of a key, use getKeyInfo in key_mgmt_util.

Required: Yes

<attribute id>

Identifies the attribute. Enter a constant that represents an attribute, or `512`, which represents all attributes. For example, to get the key type, enter `256`, which is the constant for the `OBJ_ATTR_KEY_TYPE` attribute.

To list the attributes and their constants, use listAttributes. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

Required: Yes

<filename>

Writes the output to the specified file. Enter a file path.

If the specified file exists, **getAttribute** overwrites the file without warning.

Required: No

Default: Stdout

Related topics

- getAttribute in key_mgmt_util
- listAttributes
- setAttribute in cloudhsm_mgmt_util
- setAttribute in key_mgmt_util
- Key Attribute Reference

Get hardware information for each HSM in an AWS CloudHSM cluster with CMU

Use the **getHSMInfo** command in the AWS CloudHSM cloudhsm_mgmt_util (CMU) to get information about the hardware on which each hardware security module (HSM) runs, including the model, serial number, FIPS state, memory, temperature, and the version numbers of the hardware and firmware. The information also includes the server ID that cloudhsm_mgmt_util uses to refer to the HSM.

Before you run any CMU command, you must start CMU and log in to the HSM. Be sure that you log in with a user type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files for CMU. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User type

The following types of users can run this command.

- All users. You do not have to be logged in to run this command.

Syntax

This command has no parameters.

```
getHSMInfo
```

Example

This example uses **getHSMInfo** to get information about the HSMs in the cluster.

```
aws-cloudhsm> getHSMInfo
Getting HSM Info on 3 nodes
                *** Server 0 HSM Info ***

        Label                :cavium
        Model                :NITROX-III CNN35XX-NFBE

        Serial Number        :3.0A0101-ICM000001
        HSM Flags            :0
        FIPS state           :2 [FIPS mode with single factor authentication]

        Manufacturer ID      :
        Device ID            :10
        Class Code           :100000
        System vendor ID     :177D
        SubSystem ID         :10


        TotalPublicMemory    :560596
        FreePublicMemory     :294568
        TotalPrivateMemory   :0
        FreePrivateMemory    :0

        Hardware Major       :3
```

```
        Hardware Minor       :0

        Firmware Major       :2
        Firmware Minor       :03

        Temperature          :56 C

        Build Number         :13

        Firmware ID          :xxxxxxxxxxxxxxx

...
```

Related topics

- info

Get AWS CloudHSM user info about a key using CMU

Use the **getKeyInfo** command in the AWS CloudHSM key_mgmt_util (KMU) to return the hardware security module (HSM) user IDs of users who can use the key, including the owner and crypto users (CU) with whom the key is shared. When quorum authentication is enabled on a key, **getKeyInfo** also returns the number of users who must approve cryptographic operations that use the key. You can run **getKeyInfo** only on keys that you own and keys that are shared with you.

When you run **getKeyInfo** on public keys, **getKeyInfo** returns only the key owner, even though all users of the HSM can use the public key. To find the HSM user IDs of users in your HSMs, use listUsers. To find the keys for a particular user, use findKey `-u` in key_mgmt_util. Crypto officers can use findAllKeys in cloudhsm_mgmt_util.

You own the keys that you create. You can share a key with other users when you create it. Then, to share or unshare an existing key, use shareKey in cloudhsm_mgmt_util.

Before you run any CMU command, you must start CMU and log in to the HSM. Be sure that you log in with a user type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files for CMU. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User type

The following types of users can run this command.

- Crypto users (CU)

Syntax

```
getKeyInfo -k <key-handle> [<output file>]
```

Examples

These examples show how to use **getKeyInfo** to get information about the users of a key.

Example : Get the users for an asymmetric key

This command gets the users who can use the AES (asymmetric) key with key handle 262162. The output shows that user 3 owns the key and has shares it with users 4 and 6.

Only users 3, 4, and 6 can run **getKeyInfo** on key 262162.

```
aws-cloudhsm>getKeyInfo 262162
Key Info on server 0(10.0.0.1):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 2 user(s):

                 4
                 6
Key Info on server 1(10.0.0.2):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 2 user(s):

                 4
                 6
```

Example : Get the users for a symmetric key pair

These commands use **getKeyInfo** to get the users who can use the keys in an ECC (symmetric) key pair. The public key has key handle 262179. The private key has key handle 262177.

When you run **getKeyInfo** on the private key (262177), it returns the key owner (3) and crypto users (CUs) 4, with whom the key is shared.

```
aws-cloudhsm>getKeyInfo -k 262177
Key Info on server 0(10.0.0.1):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 1 user(s):

                 4
Key Info on server 1(10.0.0.2):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 1 user(s):

                 4
```

When you run **getKeyInfo** on the public key (262179), it returns only the key owner, user 3.

```
aws-cloudhsm>getKeyInfo -k 262179
Key Info on server 0(10.0.3.10):

        Token/Flash Key,

        Owned by user 3

Key Info on server 1(10.0.3.6):

        Token/Flash Key,

        Owned by user 3
```

To confirm that user 4 can use the public key (and all public keys on the HSM), use the `-u` parameter of [findKey] in key_mgmt_util.

The output shows that user 4 can use both the public (262179) and private (262177) key in the key pair. User 4 can also use all other public keys and any private keys that they have created or that have been shared with them.

```
Command:  findKey -u 4

Total number of keys present 8

 number of keys matched from start index 0::7
11, 12, 262159, 262161, 262162, 19, 20, 21, 262177, 262179

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Example : Get the quorum authentication value (m_value) for a key

This example shows how to get the `m_value` for a key. The m_value is the number of users in the quorum who must approve any cryptographic operations that use the key and operations to share the unshare the key.

When quorum authentication is enabled on a key, a quorum of users must approve any cryptographic operations that use the key. To enable quorum authentication and set the quorum size, use the `-m_value` parameter when you create the key.

This command uses [genSymKey] to create a 256-bit AES key that is shared with user 4. It uses the `m_value` parameter to enable quorum authentication and set the quorum size to two users. The number of users must be large enough to provide the required approvals.

The output shows that the command created key 10.

```
 Command:  genSymKey -t 31 -s 32 -l aes256m2 -u 4 -m_value 2

        Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 10
```

```
        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

This command uses **getKeyInfo** in cloudhsm_mgmt_util to get information about the users of key 10. The output shows that the key is owned by user 3 and shared with user 4. It also shows that a quorum of two users must approve every cryptographic operation that uses the key.

```
aws-cloudhsm>getKeyInfo 10

Key Info on server 0(10.0.0.1):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 1 user(s):

                 4
         2 Users need to approve to use/manage this key
Key Info on server 1(10.0.0.2):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 1 user(s):

                 4
         2 Users need to approve to use/manage this key
```

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
getKeyInfo -k <key-handle> <output file>
```

<key-handle>

Specifies the key handle of one key in the HSM. Enter the key handle of a key that you own or share. This parameter is required.

Required: Yes

<output file>

Writes the output to the specified file, instead of stdout. If the file exists, the command overwrites it without warning.

Required: No

Default: stdout

Related topics

- getKeyInfo in key_mgmt_util
- findKey in key_mgmt_util
- findAllKeys in cloudhsm_mgmt_util
- listUsers
- shareKey

Get information for each HSM in an AWS CloudHSM cluster using CMU

Use the **info** command in the AWS CloudHSM cloudhsm_mgmt_util (CMU) to get information about each of the hardware security modules (HSM) in the AWS CloudHSM cluster, including the host name, port, IP address and the name and type of the user who is logged in to cloudhsm_mgmt_util on the HSM.

Before you run any CMU command, you must start CMU and log in to the HSM. Be sure that you log in with a user type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files for CMU. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User type

The following types of users can run this command.

- All users. You do not have to be logged in to run this command.

Syntax

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
info server <server ID>
```

Example

This example uses **info** to get information about an HSM in the cluster. The command uses 0 to refer to the first HSM in the cluster. The output shows the IP address, port, and the type and name of the current user.

```
aws-cloudhsm> info server 0
Id      Name                    Hostname         Port    State           Partition
         LoginState
0       10.0.0.1                10.0.0.1        2225    Connected       hsm-udw0tkfg1ab
         Logged in as 'testuser(CU)'
```

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
info server <server ID>
```

<server id>

Specifies the server ID of the HSM. The HSMs are assigned ordinal numbers that represent the order in which they are added to the cluster, beginning with 0. To find the server ID of an HSM, use getHSMInfo.

Required: Yes

Related topics

- getHSMInfo
- loginHSM and logoutHSM

List the attributes of an AWS CloudHSM key using CMU

Use the **listAttributes** command in the AWS CloudHSM cloudhsm_mgmt_util (CMU) to list the attributes of an AWS CloudHSM key and the constants that represent them. You use these constants to identify the attributes in getAttribute and setAttribute commands.

For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

User type

The following users can run this command.

- All users. You do not have to be logged in to run this command.

Syntax

```
listAttributes [-h]
```

Example

This command lists the key attributes that you can get and change in key_mgmt_util and the constants that represent them. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU. To represent all attributes, use 512.

```
Command: listAttributes

    Description
===========
The following are all of the possible attribute values for getAttribute.

      OBJ_ATTR_CLASS                  = 0
      OBJ_ATTR_TOKEN                  = 1
      OBJ_ATTR_PRIVATE                = 2
      OBJ_ATTR_LABEL                  = 3
      OBJ_ATTR_TRUSTED                = 134
      OBJ_ATTR_KEY_TYPE               = 256
      OBJ_ATTR_ID                     = 258
      OBJ_ATTR_SENSITIVE              = 259
```

```
        OBJ_ATTR_ENCRYPT                = 260
        OBJ_ATTR_DECRYPT                = 261
        OBJ_ATTR_WRAP                   = 262
        OBJ_ATTR_UNWRAP                 = 263
        OBJ_ATTR_SIGN                   = 264
        OBJ_ATTR_VERIFY                 = 266
        OBJ_ATTR_DERIVE                 = 268
        OBJ_ATTR_LOCAL                  = 355
        OBJ_ATTR_MODULUS                = 288
        OBJ_ATTR_MODULUS_BITS           = 289
        OBJ_ATTR_PUBLIC_EXPONENT        = 290
        OBJ_ATTR_VALUE_LEN              = 353
        OBJ_ATTR_EXTRACTABLE            = 354
        OBJ_ATTR_NEVER_EXTRACTABLE      = 356
        OBJ_ATTR_ALWAYS_SENSITIVE       = 357
        OBJ_ATTR_DESTROYABLE            = 370
        OBJ_ATTR_KCV                     = 371
        OBJ_ATTR_WRAP_WITH_TRUSTED       = 528
        OBJ_ATTR_WRAP_TEMPLATE           = 1073742353
        OBJ_ATTR_UNWRAP_TEMPLATE        = 1073742354
        OBJ_ATTR_ALL                    = 512
```

Parameters

-h

Displays help for the command.

Required: Yes

Related topics

- getAttribute
- setAttribute
- Key Attribute Reference

List all AWS CloudHSM users using CMU

Use the **listUsers** command in the AWS CloudHSM cloudhsm_mgmt_util to get the users in each of the hardware security modules (HSM), along with their user type and other attributes. All types of users can run this command. You do not even need to be logged in to cloudhsm_mgmt_util to run this command.

Before you run any CMU command, you must start CMU and log in to the HSM. Be sure that you log in with a user type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files for CMU. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User type

The following types of users can run this command.

- All users. You do not need to be logged in to run this command.

Syntax

This command has no parameters.

```
listUsers
```

Example

This command lists the users on each of the HSMs in the cluster and displays their attributes. You can use the `User ID` attribute to identify users in other commands, such as **deleteUser**, **changePswd**, and **findAllKeys**.

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:6

    User Id             User Type       User Name           MofnPubKey
 LoginFailureCnt         2FA
         1              PCO             admin                    YES               0
             NO
         2              AU              app_user                  NO               0
             NO
         3              CU              crypto_user1              NO               0
             NO
         4              CU              crypto_user2              NO               0
             NO
```

```
         5              CO              officer1                YES              0
             NO
         6              CO              officer2                 NO              0
             NO
Users on server 1(10.0.0.2):
Number of users found:5

    User Id             User Type       User Name             MofnPubKey
 LoginFailureCnt         2FA
         1              PCO             admin                   YES              0
             NO
         2              AU              app_user                 NO              0
             NO
         3              CU              crypto_user1             NO              0
             NO
         4              CU              crypto_user2             NO              0
             NO
         5              CO              officer1                YES              0
             NO
```

The output includes the following user attributes:

- **User ID**: Identifies the user in key_mgmt_util and cloudhsm_mgmt_util commands.
- User type: Determines the operations that the user can perform on the HSM.
- **User Name**: Displays the user-defined friendly name for the user.
- **MofnPubKey**: Indicates whether the user has registered a key pair for signing quorum authentication tokens.
- **LoginFailureCnt**: Indicates the number of times the user has unsuccessfully logged in.
- **2FA**: Indicates that the user has enabled multi-factor authentication.

Related topics

- listUsers in key_mgmt_util
- createUser
- deleteUser
- changePswd

Log in and out of an HSM using AWS CloudHSM Management Utility

Use the **loginHSM** and **logoutHSM** commands in the AWS CloudHSM cloudhsm_mgmt_util to log in and out of each HSM in a cluster. Any user of any type can use these commands.

> **Note**
>
> If you exceed five incorrect login attempts, your account is locked out. To unlock the account, a cryptographic officer (CO) must reset your password using the changePswd command in cloudhsm_mgmt_util.

To troubleshoot loginHSM and logoutHSM

Before you run these cloudhsm_mgmt_util commands, you must start cloudhsm_mgmt_util.

If you add or delete HSMs, update the configuration files that the AWS CloudHSM client and the command line tools use. Otherwise, the changes that you make might not be effective on all HSMs in the cluster.

If you have more than one HSM in your cluster, you may be allowed additional incorrect login attempts before your account is locked out. This is because the CloudHSM client balances load across various HSMs. Therefore, the login attempt may not begin on the same HSM each time. If you are testing this functionality, we recommend you do so on a cluster with only one active HSM.

If you created your cluster before February 2018, your account is locked out after 20 incorrect login attempts.

User type

The following users can run these commands.

- Pre-crypto officer (PRECO)
- Crypto officer (CO)
- Crypto user (CU)

Syntax

Enter the arguments in the order specified in the syntax diagram. Use the `-hpswd` parameter to mask your password. To login with two-factor authentication (2FA), use the `-2fa` parameter and include a file path. For more information, see the section called "Arguments".

```
loginHSM <user-type> <user-name> <password> |-hpswd> [-2fa </path/to/authdata>]
```

```
logoutHSM
```

Examples

These examples show how to use **loginHSM** and **logoutHSM** to log in and out of all HSMs in a cluster.

Example : Log in to the HSMs in a cluster

This command logs you in to all HSMs in a cluster with the credentials of a CO user named `admin` and a password of `co12345`. The output shows that the command was successful and that you have connected to the HSMs (which, in this case, are `server 0` and `server 1`).

```
aws-cloudhsm>loginHSM CO admin co12345

loginHSM success on server 0(10.0.2.9)
loginHSM success on server 1(10.0.3.11)
```

Example : Log in with a hidden password

This command is the same as the example above, except this time you specify that the system should hide the password.

```
aws-cloudhsm>loginHSM CO admin -hpswd
```

The system prompts you for your password. You enter the password, the system hides the password, and the output shows that the command was successful and that the you have connected to the HSMs.

```
Enter password:

loginHSM success on server 0(10.0.2.9)
```

```
loginHSM success on server 1(10.0.3.11)

aws-cloudhsm>
```

Example : Log out of an HSM

This command logs you out of the HSMs that you are currently logged in to (which, in this case, are `server 0` and `server 1`). The output shows that the command was successful and that you have disconnected from the HSMs.

```
aws-cloudhsm>logoutHSM

logoutHSM success on server 0(10.0.2.9)
logoutHSM success on server 1(10.0.3.11)
```

Arguments

Enter the arguments in the order specified in the syntax diagram. Use the `-hpswd` parameter to mask your password. To login with two-factor authentication (2FA), use the `-2fa` parameter and include a file path. For more information about working with 2FA, see Manage user 2FA

```
loginHSM <user-type> <user-name> <password> |-hpswd> [-2fa </path/to/authdata>]
```

<user type>

Specifies the type of user who is logging in to the HSMs. For more information, see User Type above.

Required: Yes

<user name>

Specifies the user name of the user who is logging in to the HSMs.

Required: Yes

<password | -hpswd >

Specifies the password of the user who is logging in to the HSMs. To hide your password, use the `-hpswd` parameter in place of the password and follow the prompt.

Required: Yes

[-2fa </path/to/authdata>]

Specifies that the system should use a second factor to authenticate this 2FA-enabled CO user. To get the necessary data for logging in with 2FA, include a path to a location in the file system with a file name after the `-2fa` parameter. For more information about working with 2FA, see Manage user 2FA.

Required: No

Related topics

- Getting Started with cloudhsm_mgmt_util
- Activate the Cluster

Associate AWS CloudHSM users with keys using CMU

Use the **registerQuorumPubKey** command in the AWS CloudHSM cloudhsm_mgmt_util to associate hardware security module (HSM) users with asymmetric RSA-2048 key pairs. Once you associate HSM users with keys, those users can use the private key to approve quorum requests and the cluster can use the registered public key to verify the signature is from the user. For more information about quorum authentication, see Managing Quorum Authentication (M of N Access Control).

> **Tip**
>
> In the AWS CloudHSM documentation, quorum authentication is sometimes referred to as M of N (MofN), which means a minimum of *M* approvers out of a total number *N* approvers.

User type

The following types of users can run this command.

- Crypto officers (CO)

Syntax

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
registerQuorumPubKey <user-type> <user-name> <registration-token> <signed-registration-token> <public-key>
```

Examples

This example shows how to use **registerQuorumPubKey** to register crypto officers (CO) as approvers on quorum authentication requests. To run this command, you must have an asymmetric RSA-2048 key pair, a signed token, and an unsigned token. For more information about these requirements, see the section called “Arguments”.

Example : Register an HSM user for quorum authentication

This example registers a CO named `quorum_officer` as an approver for quorum authentication.

```
aws-cloudhsm> registerQuorumPubKey CO <quorum_officer> </path/to/quorum_officer.token> </path/to/quorum_officer.token.sig> </path/to/quorum_officer.pub>

*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)?y
registerQuorumPubKey success on server 0(10.0.0.1)
```

The final command uses the listUsers command to verify that `quorum_officer` is registered as an MofN user.

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:3

    User Id             User Type       User Name                           MofnPubKey
  LoginFailureCnt         2FA
         1              PCO             admin                                     NO
           0               NO
         2              AU              app_user                                  NO
           0               NO
```

```
        3              CO              quorum_officer                              YES
          0                NO
```

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
registerQuorumPubKey <user-type> <user-name> <registration-token> <signed-registration-
token> <public-key>
```

<user-type>

Specifies the type of user. This parameter is required.

For detailed information about the user types on an HSM, see HSM user types for AWS CloudHSM Management Utility.

Valid values:

- **CO**: Crypto officers can manage users, but they cannot manage keys.

Required: Yes

<user-name>

Specifies a friendly name for the user. The maximum length is 31 characters. The only special character permitted is an underscore (_).

You cannot change the name of a user after it is created. In cloudhsm_mgmt_util commands, the user type and password are case-sensitive, but the user name is not.

Required: Yes

<registration-token>

Specifies the path to a file that contains an unsigned registration token. Can have any random data of max file size of 245 bytes. For more information about creating an unsigned registration token, see Create and Sign a Registration Token.

Required: Yes

<signed-registration-token>

Specifies the path to a file that contains the SHA256_PKCS mechanism signed hash of the registration-token. For more information, see Create and Sign a Registration Token.

Required: Yes

<public-key>

Specifies the path to a file that contains the public key of an asymmetric RSA-2048 key pair. Use the private key to sign the registration token. For more information, see Create an RSA Key Pair.

Required: Yes

> **Note**
>
> The cluster uses the same key for quorum authentication and for two-factor authentication (2FA). This means you can't rotate a quorum key for a user that has 2FA enabled using **registerQuorumPubKey**. To rotate the key, you must use **changePswd**. For more information about using quorum authentication and 2FA, see Quorum Authentication and 2FA.

Related topics

- Create an RSA Key Pair
- Create and Sign a Registration Token
- Register the Public Key with the HSM
- Managing Quorum Authentication (M of N Access Control)
- Quorum Authentication and 2FA
- listUsers

Interact with one HSM in an AWS CloudHSM cluster using CMU

Use the **server** command in the AWS CloudHSM cloudhsm_mgmt_util to enter *server mode* and interact directly with a particular hardware security module (HSM) instance.

Normally, when you issue a command in cloudhsm_mgmt_util, the command effects all HSMs in the designated cluster (*global mode)*. However, there may be circumstances for which you need to issue commands to a single HSM. For instance, in the event that automatic synchronization fails, you may need to sync keys and users on an HSM in order to maintain consistency across the cluster.

Upon successful initiation, the `aws-cloudhsm >` command prompt is replaced with the `server >` command prompt.

In order to exit server mode, use the `exit` command. Upon successful exit, you will be returned to the cloudhsm_mgmt_util command prompt.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util.

User type

The following users can run this command.

- All users.

Prerequisites

In order to enter server mode, you must first know the server number of the target HSM. Server numbers are listed in the trace output generated by cloudhsm_mgmt_util upon initiation. Server numbers are assigned in the same order that the HSMs appear in the configuration file. For this example, we assume that `server 0` is the server that corresponds to the desired HSM.

Syntax

To start server mode:

```
server <server-number>
```

To exit server mode:

```
server> exit
```

Example

This command enters server mode on an HSM with server number `0`.

```
aws-cloudhsm> server 0
```

```
Server is in 'E2' mode...
```

In order to exit server mode, use the **exit** command.

```
server0> exit
```

Arguments

```
server <server-number>
```

<server-number>

Specifies the server number of the target HSM.

Required: Yes

There are no arguments for the `exit` command.

Related topics

- syncKey
- createUser
- deleteUser

Set the attributes of AWS CloudHSM keys using CMU

Use the **setAttribute** command in the AWS CloudHSM cloudhsm_mgmt_util to change the value of the label, encrypt, decrypt, wrap, and unwrap attributes of a key in the HSMs. You can also use the setAttribute command in key_mgmt_util to convert a session key to a persistent key. You can only change the attributes of keys that you own.

Before you run any CMU command, you must start CMU and log in to the HSM. Be sure that you log in with a user type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files for CMU. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User type

The following users can run this command.

- Crypto users (CU)

Syntax

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
setAttribute <key handle> <attribute id>
```

Example

This example shows how to disable the decrypt functionality of a symmetric key. You can use a command like this one to configure a wrapping key, which should be able to wrap and unwrap other keys but not encrypt or decrypt data.

The first step is to create the wrapping key. This command uses genSymKey in key_mgmt_util to generate a 256-bit AES symmetric key. The output shows that the new key has key handle 14.

```
$  genSymKey -t 31 -s 32 -l aes256

Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 14

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Next, we want to confirm the current value of the decrypt attribute. To get the attribute ID of the decrypt attribute, use listAttributes. The output shows that the constant that represents the `OBJ_ATTR_DECRYPT` attribute is 261. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

```
aws-cloudhsm> listAttributes
```

```
Following are the possible attribute values for getAttribute:

      OBJ_ATTR_CLASS                  = 0
      OBJ_ATTR_TOKEN                  = 1
      OBJ_ATTR_PRIVATE                = 2
      OBJ_ATTR_LABEL                  = 3
      OBJ_ATTR_TRUSTED                = 134
      OBJ_ATTR_KEY_TYPE               = 256
      OBJ_ATTR_ID                     = 258
      OBJ_ATTR_SENSITIVE              = 259
      OBJ_ATTR_ENCRYPT                = 260
      OBJ_ATTR_DECRYPT                = 261
      OBJ_ATTR_WRAP                   = 262
      OBJ_ATTR_UNWRAP                 = 263
      OBJ_ATTR_SIGN                   = 264
      OBJ_ATTR_VERIFY                 = 266
      OBJ_ATTR_DERIVE                 = 268
      OBJ_ATTR_LOCAL                  = 355
      OBJ_ATTR_MODULUS                = 288
      OBJ_ATTR_MODULUS_BITS           = 289
      OBJ_ATTR_PUBLIC_EXPONENT        = 290
      OBJ_ATTR_VALUE_LEN              = 353
      OBJ_ATTR_EXTRACTABLE            = 354
      OBJ_ATTR_NEVER_EXTRACTABLE      = 356
      OBJ_ATTR_ALWAYS_SENSITIVE       = 357
      OBJ_ATTR_DESTROYABLE            = 370
      OBJ_ATTR_KCV                    = 371
      OBJ_ATTR_WRAP_WITH_TRUSTED      = 528
      OBJ_ATTR_WRAP_TEMPLATE          = 1073742353
      OBJ_ATTR_UNWRAP_TEMPLATE        = 1073742354
      OBJ_ATTR_ALL                    = 512
```

To get the current value of the decrypt attribute for key 14, the next command uses getAttribute in cloudhsm_mgmt_util.

The output shows that the value of the decrypt attribute is true (1) on both HSMs in the cluster.

```
aws-cloudhsm> getAttribute 14 261

Attribute Value on server 0(10.0.0.1):
OBJ_ATTR_DECRYPT
0x00000001
```

```
Attribute Value on server 1(10.0.0.2):
OBJ_ATTR_DECRYPT
0x00000001
```

This command uses **setAttribute** to change the value of the decrypt attribute (attribute 261) of key 14 to `0`. This disables the decrypt functionality on the key.

The output shows that the command succeeded on both HSMs in the cluster.

```
aws-cloudhsm> setAttribute 14 261 0
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)? y
setAttribute success on server 0(10.0.0.1)
setAttribute success on server 1(10.0.0.2)
```

The final command repeats the **getAttribute** command. Again, it gets the decrypt attribute (attribute 261) of key 14.

This time, the output shows that the value of the decrypt attribute is false (0) on both HSMs in the cluster.

```
aws-cloudhsm > getAttribute 14 261
Attribute Value on server 0(10.0.3.6):
OBJ_ATTR_DECRYPT
0x00000000

Attribute Value on server 1(10.0.1.7):
OBJ_ATTR_DECRYPT
0x00000000
```

Arguments

```
setAttribute <key handle> <attribute idb
```

<key-handle>

Specifies the key handle of a key that you own. You can specify only one key in each command. To get the key handle of a key, use findKey in key_mgmt_util. To find the users of a key, use getKeyInfo.

Required: Yes

<attribute id>

Specifies the constant that represents the attribute that you want to change. You can specify only one attribute in each command. To get the attributes and their integer values, use listAttributes. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

Valid values:

- **3** – `OBJ_ATTR_LABEL`.
- **134** – `OBJ_ATTR_TRUSTED`.
- **260** – `OBJ_ATTR_ENCRYPT`.
- **261** – `OBJ_ATTR_DECRYPT`.
- **262** – `OBJ_ATTR_WRAP`.
- **263** – `OBJ_ATTR_UNWRAP`.
- **264** – `OBJ_ATTR_SIGN`.
- **266** – `OBJ_ATTR_VERIFY`.
- **268** – `OBJ_ATTR_DERIVE`.
- **370** – `OBJ_ATTR_DESTROYABLE`.
- **528** – `OBJ_ATTR_WRAP_WITH_TRUSTED`.
- **1073742353** – `OBJ_ATTR_WRAP_TEMPLATE`.
- **1073742354** – `OBJ_ATTR_UNWRAP_TEMPLATE`.

Required: Yes

Related topics

- setAttribute in key_mgmt_util
- getAttribute

- [listAttributes]
- [Key Attribute Reference]

Exit the CMU

Use the **quit** command in the AWS CloudHSM cloudhsm_mgmt_util to exit the cloudhsm_mgmt_util. Any user of any type can use this command.

Before you run any cloudhsm_mgmt_util command, you must start cloudhsm_mgmt_util.

User type

The following users can run this command.

- All users. You do not need to be logged in to run this command.

Syntax

```
quit
```

Example

This command exits cloudhsm_mgmt_util. Upon successful completion, you are returned to your regular command line. This command has no output parameters.

```
aws-cloudhsm> quit

disconnecting from servers, please wait...
```

Related topics

- [Getting Started with cloudhsm_mgmt_util]

Share AWS CloudHSM keys using CMU

Use the **shareKey** command in the AWS CloudHSM cloudhsm_mgmt_util to share and unshare keys that you own with other crypto users. Only the key owner can share and unshare a key. You can also share a key when you create it.

Users who share the key can use the key in cryptographic operations, but they cannot delete, export, share, or unshare the key, or change its attributes. When quorum authentication is enabled on a key, the quorum must approve any operations that share or unshare the key.

Before you run any CMU command, you must start CMU and log in to the HSM. Be sure that you log in with a user type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files for CMU. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User type

The following types of users can run this command.

- Crypto users (CU)

Syntax

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

User Type: Crypto user (CU)

```
shareKey <key handle> <user id> <(share/unshare key?) 1/0>
```

Example

The following examples show how to use **shareKey** to share and unshare keys that you own with other crypto users.

Example : Share a key

This example uses **shareKey** to share an ECC private key that the current user owns with another crypto user on the HSMs. Public keys are available to all users of the HSM, so you cannot share or unshare them.

The first command uses getKeyInfo to get the user information for key 262177, an ECC private key on the HSMs.

The output shows that key 262177 is owned by user 3, but is not shared.

```
aws-cloudhsm>getKeyInfo 262177
```

```
Key Info on server 0(10.0.3.10):

        Token/Flash Key,

        Owned by user 3

Key Info on server 1(10.0.3.6):

        Token/Flash Key,

        Owned by user 3
```

This command uses **shareKey** to share key 262177 with user 4, another crypto user on the HSMs. The final argument uses a value of 1 to indicate a share operation.

The output shows that the operation succeeded on both HSMs in the cluster.

```
aws-cloudhsm>shareKey 262177 4 1
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)?y
shareKey success on server 0(10.0.3.10)
shareKey success on server 1(10.0.3.6)
```

To verify that the operation succeeded, the example repeats the first **getKeyInfo** command.

The output shows that key 262177 is now shared with user 4.

```
aws-cloudhsm>getKeyInfo 262177

Key Info on server 0(10.0.3.10):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 1 user(s):
```

```
                 4
Key Info on server 1(10.0.3.6):

        Token/Flash Key,

        Owned by user 3

        also, shared to following 1 user(s):

                 4
```

Example : Unshare a key

This example unshares a symmetric key, that is, it removes a crypto user from the list of shared users for the key.

This command uses **shareKey** to remove user 4 from the list of shared users for key 6. The final argument uses a value of `0` to indicate an unshare operation.

The output shows that the command succeeded on both HSMs. As a result, user 4 can no longer use key 6 in cryptographic operations.

```
aws-cloudhsm>shareKey 6 4 0
*************************CAUTION********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. AWS does NOT synchronize these changes automatically with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
****************************************************************

Do you want to continue(y/n)?y
shareKey success on server 0(10.0.3.10)
shareKey success on server 1(10.0.3.6)
```

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
shareKey <key handle> <user id> <(share/unshare key?) 1/0>
```

<key-handle>

Specifies the key handle of a key that you own. You can specify only one key in each command. To get the key handle of a key, use findKey in key_mgmt_util. To verify that you own a key, use getKeyInfo.

Required: Yes

<user id>

Specifies the user ID the crypto user (CU) with whom you are sharing or unsharing the key. To find the user ID of a user, use listUsers.

Required: Yes

<share 1 or unshare 0>

To share the key with the specified user, type 1. To unshare the key, that is, to remove the specified user from the list of shared users for the key, type `0`.

Required: Yes

Related topics

- getKeyInfo

Synchronize keys across the AWS CloudHSM cluster using CMU

Use the **syncKey** command in the AWS CloudHSM cloudhsm_mgmt_util to manually synchronize keys across HSM instances within a cluster or across cloned clusters. In general, you will not need to use this command, as HSM instances within a cluster sync keys automatically. However, key synchronization across cloned clusters must be done manually. Cloned clusters are usually created in different AWS Regions in order to simplify the global scaling and disaster recovery processes.

You cannot use **syncKey** to synchronize keys across arbitrary clusters: one of the clusters must have been created from a backup of the other. Additionally, both clusters must have consistent CO and CU credentials in order for the operation to be successful. For more information, see HSM Users.

To use **syncKey**, you must first create an AWS CloudHSM configuration file that specifies one HSM from the source cluster and one from the destination cluster. This will allow cloudhsm_mgmt_util to connect to both HSM instances. Use this configuration file to start cloudhsm_mgmt_util. Then log in with the credentials of a CO or a CU who owns the keys you want to synchronize.

User type

The following types of users can run this command.

- Crypto officers (CO)
- Crypto users (CU)

> **Note**
>
> COs can use **syncKey** on any keys, while CUs can only use this command on keys that they own. For more information, see the section called "User types".

Prerequisites

Before you begin, you must know the `key handle` of the key on the source HSM to be synchronized with the destination HSM. To find the `key handle`, use the listUsers command to list all identifiers for named users. Then, use the findAllKeys command to find all keys that belong to a particular user.

You also need to know the `server IDs` assigned to the source and destination HSMs, which are shown in the trace output returned by cloudhsm_mgmt_util upon initiation. These are assigned in the same order that the HSMs appear in the configuration file.

Follow the instructions in Using CMU Across Cloned Clusters and initialize cloudhsm_mgmt_util with the new config file. Then, enter server mode on the source HSM by issuing the server command.

Syntax

> **Note**
>
> To run **syncKey**, first enter server mode on the HSM which contains the key to be synchronized.

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

User Type: Crypto user (CU)

```
syncKey <key handle> <destination hsm>
```

Example

Run the **server** command to log into the source HSM and enter server mode. For this example, we assume that `server 0` is the source HSM.

```
aws-cloudhsm> server 0
```

Now run the **syncKey** command. In this example, we assume key `261251` is to be synced to `server 1`.

```
aws-cloudhsm> syncKey 261251 1
syncKey success
```

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
syncKey <key handle> <destination hsm>
```

<key handle>

Specifies the key handle of the key to sync. You can specify only one key in each command. To get the key handle of a key, use findAllKeys while logged in to an HSM server.

Required: Yes

<destination hsm>

Specifies the number of the server to which you are syncing a key.

Required: Yes

Related topics

- listUsers
- findAllKeys
- describe-clusters in AWS CLI

- server

Synchronize users across the AWS CloudHSM cluster using CMU

Use the **syncUser** command in the AWS CloudHSM cloudhsm_mgmt_util to manually synchronize crypto users (CUs) or crypto officers (COs) across HSM instances within a cluster or across cloned clusters. AWS CloudHSM does not automatically synchronize users. Generally, you manage users in global mode so that all HSMs in a cluster are updated together. You might need to use **syncUser** if an HSM is accidentally desynchronized (for example, due to password changes) or if you want to rotate user credentials across cloned clusters. Cloned clusters are usually created in different AWS Regions to simplify the global scaling and disaster recovery processes.

Before you run any CMU command, you must start CMU and log in to the HSM. Be sure that you log in with a user type that can run the commands you plan to use.

If you add or delete HSMs, update the configuration files for CMU. Otherwise, the changes that you make might not be effective for all HSMs in the cluster.

User type

The following types of users can run this command.

- Crypto officers (CO)

Prerequisites

Before you begin, you must know the `user ID` of the user on the source HSM to be synchronized with the destination HSM. To find the `user ID`, use the listUsers command to list all users on the HSMs in a cluster.

You also need to know the `server ID` assigned to the source and destination HSMs, which are shown in the trace output returned by cloudhsm_mgmt_util upon initiation. These are assigned in the same order that the HSMs appear in the configuration file.

If you are synchronizing HSMs across cloned clusters, follow the instructions in Using CMU Across Cloned Clusters and initialize cloudhsm_mgmt_util with the new config file.

When you are ready to run **syncUser**, enter server mode on the source HSM by issuing the **server** command.

Syntax

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
syncUser <user ID> <server ID>
```

Example

Run the **server** command to log into the source HSM and enter server mode. For this example, we assume that `server 0` is the source HSM.

```
aws-cloudhsm> server 0
```

Now run the **syncUser** command. For this example, we assume that user 6 is the user to be synced, and `server 1` is the destination HSM.

```
server 0> syncUser 6 1
ExtractMaskedObject: 0x0 !
InsertMaskedObject: 0x0 !
syncUser success
```

Arguments

Because this command does not have named parameters, you must enter the arguments in the order specified in the syntax diagram.

```
syncUser <user ID> <server ID>
```

<user ID>

Specifies the ID of the user to sync. You can specify only one user in each command. To get the ID of a user, use listUsers.

Required: Yes

<server ID>

Specifies the server number of the HSM to which you are syncing a user.

Required: Yes

Related topics

- listUsers
- describe-clusters in AWS CLI
- server

AWS CloudHSM Key Management Utility (KMU)

The key management utility (KMU) is a command line tool for AWS CloudHSM that helps crypto users (CU) manage keys on the hardware security modules (HSM). KMU includes multiple commands that generate, delete, import, and export keys, get and set attributes, find keys, and perform cryptographic operations.

KMU and CMU are part of the Client SDK 3 suite.

For a quick start, see Getting started with AWS CloudHSM key_mgmt_util. For detailed information about the commands, see Reference for AWS CloudHSM Key Management Utility commands. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

To use key_mgmt_util if you are using Linux, connect to your client instance and then see Install and configure the AWS CloudHSM client for KMU (Linux). If you are using Windows, see Install and configure the AWS CloudHSM client for KMU (Windows).

Topics

- Getting started with AWS CloudHSM key_mgmt_util
- Install and configure the AWS CloudHSM client for KMU (Linux)
- Install and configure the AWS CloudHSM client for KMU (Windows)
- Reference for AWS CloudHSM Key Management Utility commands

Getting started with AWS CloudHSM key_mgmt_util

AWS CloudHSM includes two command line tools with the AWS CloudHSM client software. The cloudhsm_mgmt_util tool includes commands to manage HSM users. The key_mgmt_util tool includes commands to manage keys. To get started with the key_mgmt_util command line tool, see the following topics.

Topics

- Set up AWS CloudHSM key_mgmt_util
- Log in to the HSMs in an AWS CloudHSM cluster using KMU
- Log out from the HSMs in an AWS CloudHSM cluster using KMU
- Stop the AWS CloudHSM key_mgmt_util

If you encounter an error message or unexpected outcome for a command, see the Troubleshooting AWS CloudHSM topics for help. For details about the key_mgmt_util commands, see Reference for AWS CloudHSM Key Management Utility commands.

Set up AWS CloudHSM key_mgmt_util

Complete the following setup before you use AWS CloudHSM key_mgmt_util (KMU).

Topics

- Step 1. Start the AWS CloudHSM client
- Step 2. Start key_mgmt_util

Step 1. Start the AWS CloudHSM client

Before you use key_mgmt_util, you must start the AWS CloudHSM client. The client is a daemon that establishes end-to-end encrypted communication with the HSMs in your cluster. The key_mgmt_util tool uses the client connection to communicate with the HSMs in your cluster. Without it, key_mgmt_util doesn't work.

To start the AWS CloudHSM client

Use the following command to start the AWS CloudHSM client.

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- For Windows client 1.1.2+:

  ```
  C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
  ```

- For Windows clients 1.1.1 and older:

  ```
  C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe C:
  \ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
  ```

Step 2. Start key_mgmt_util

After you start the AWS CloudHSM client, use the following command to start key_mgmt_util.

Amazon Linux

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

Amazon Linux 2

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

CentOS 7

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

CentOS 8

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

RHEL 7

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

RHEL 8

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

Ubuntu 16.04 LTS

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

Ubuntu 18.04 LTS

```
$ /opt/cloudhsm/bin/key_mgmt_util
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\key_mgmt_util.exe"
```

The prompt changes to `Command:` when key_mgmt_util is running.

If the command fails, such as returning a `Daemon socket connection error` message, try updating your configuration file.

Log in to the HSMs in an AWS CloudHSM cluster using KMU

Use the **loginHSM** command in key_mgmt_util (KMU) to log in to the hardware security modules (HSM) in an AWS CloudHSM cluster. The following command logs in as a crypto user (CU) named `example_user`. The output indicates a successful login for all three HSMs in the cluster.

```
Command:  loginHSM -u CU -s example_user -p <PASSWORD>
Cfm3LoginHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

The following shows the syntax for the **loginHSM** command.

```
Command:  loginHSM -u <USER TYPE> -s <USERNAME> -p <PASSWORD>
```

Log out from the HSMs in an AWS CloudHSM cluster using KMU

Use the **logoutHSM** command in key_mgmt_util (KMU) to log out from the hardware security modules (HSM) in an AWS CloudHSM cluster.

```
Command:  logoutHSM
Cfm3LogoutHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Stop the AWS CloudHSM key_mgmt_util

Use the **exit** command to stop the AWS CloudHSM key_mgmt_util.

```
Command:  exit
```

Install and configure the AWS CloudHSM client for KMU (Linux)

To interact with the hardware security module (HSM) in your AWS CloudHSM cluster using the key_mgmt_util (KMU), you need the AWS CloudHSM client software for Linux. You should install it on the Linux EC2 client instance that you created previously. You can also install a client if you are using Windows. For more information, see Install and configure the AWS CloudHSM client for KMU (Windows).

Tasks

- Step 1. Install the AWS CloudHSM client and command line tools
- Step 2. Edit the client configuration

Step 1. Install the AWS CloudHSM client and command line tools

Connect to your client instance and run the following commands to download and install the AWS CloudHSM client and command line tools.

Amazon Linux

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-latest.el6.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 7

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 8

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-latest.el8.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el8.x86_64.rpm
```

RHEL 7

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm
```

RHEL 8

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-latest.el8.x86_64.rpm
```

```
sudo yum install ./cloudhsm-client-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client_latest_amd64.deb
```

```
sudo apt install ./cloudhsm-client_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client_latest_u18.04_amd64.deb
```

```
sudo apt install ./cloudhsm-client_latest_u18.04_amd64.deb
```

Step 2. Edit the client configuration

Before you can use the AWS CloudHSM client to connect to your cluster, you must edit the client configuration.

To edit the client configuration

1. Copy your issuing certificate—the one that you used to sign the cluster's certificate—to the following location on the client instance: `/opt/cloudhsm/etc/customerCA.crt`. You need instance root user permissions on the client instance to copy your certificate to this location.
2. Use the following configure command to update the configuration files for the AWS CloudHSM client and command line tools, specifying the IP address of the HSM in your cluster. To get the HSM's IP address, view your cluster in the AWS CloudHSM console, or run the **describe-clusters** AWS CLI command. In the command's output, the HSM's IP address is the value of the `EniIp` field. If you have more than one HSM, choose the IP address for any of the HSMs; it doesn't matter which one.

```
sudo /opt/cloudhsm/bin/configure -a <IP address>

Updating server config in /opt/cloudhsm/etc/cloudhsm_client.cfg
Updating server config in /opt/cloudhsm/etc/cloudhsm_mgmt_util.cfg
```

3. Go to Activate the cluster in AWS CloudHSM.

Install and configure the AWS CloudHSM client for KMU (Windows)

To work with a hardware security module (HSM) in your AWS CloudHSM cluster on Windows using the key_mgmt_util (KMU), you need the AWS CloudHSM client software for Windows. You should install it on the Windows Server instance that you created previously.

To install (or update) the latest Windows client and command line tools

1. Connect to your Windows Server instance.
2. Download the latest (**AWSCloudHSMClient-latest.msi**) from the downloads page.
3. Go to your download location and run the installer (**AWSCloudHSMClient-latest.msi**) with administrative privilege.
4. Follow the installer instructions, then choose **Close** after the installer has finished.
5. Copy your self-signed issuing certificate—the one that you used to sign the cluster certificate—to the `C:\ProgramData\Amazon\CloudHSM` folder.
6. Run the following command to update your configuration files. Be sure to stop and start the client during reconfiguration if you are updating it:

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" -a <HSM IP address>
```

7. Go to Activate the cluster in AWS CloudHSM.

Notes:

- If you are updating the client, existing configuration files from previous installations are *not* overwritten.
- The AWS CloudHSM client installer for Windows automatically registers the Cryptography API: Next Generation (CNG) and Key Storage Provider (KSP). To uninstall the client, run the installer again and follow the uninstall instructions.
- If you are using Linux, you can install the Linux client. For more information, see Install and configure the AWS CloudHSM client for KMU (Linux).

Reference for AWS CloudHSM Key Management Utility commands

The **key_mgmt_util** command line tool helps you to manage keys in the hardware security modules (HSM) in your AWS CloudHSM cluster, including creating, deleting, and finding keys and their attributes. It includes multiple commands, each of which is described in detail in this topic.

For a quick start, see Getting started with AWS CloudHSM key_mgmt_util. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU. For information about the cloudhsm_mgmt_util command line tool, which includes commands to manage the HSM and users in your cluster, see AWS CloudHSM Management Utility (CMU).

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

To list all key_mgmt_util commands, type:

```
Command: help
```

To get help for a particular key_mgmt_util command, type:

```
Command: <command-name> -h
```

To end your key_mgmt_util session, type:

```
Command: exit
```

The following topics describe commands in key_mgmt_util.

Note

Some commands in key_mgmt_util and cloudhsm_mgmt_util have the same names. However, the commands typically have different syntax, different output, and slightly different functionality.

Command	Description
aesWrapUnwrap	Encrypts and decrypts the contents of a key in a file.
deleteKey	Deletes a key from the HSMs.
Error2String	Gets the error that corresponds to a key_mgmt_util hexadecimal error code.
exit	Exits the key_mgmt_util.
exportPrivateKey	Exports a copy of a private key from an HSM to a file on disk.

Command	Description
exportPubKey	Exports a copy of a public key from an HSM to a file.
exSymKey	Exports a plaintext copy of a symmetric key from the HSMs to a file.
extractMaskedObject	Extracts a key from an HSM as a masked object file.
findKey	Search for keys by key attribute value.
findSingleKey	Verifies that a key exists on all HSMs in the cluster.
genDSAKeyPair	Generates a Digital Signing Algorithm (DSA) key pair in your HSMs.
genECCKeyPair	Generates an Elliptic Curve Cryptography (ECC) key pair in your HSMs.
genRSAKeyPair	Generates an RSA asymmetric key pair in your HSMs.
genSymKey	Generates a symmetric key in your HSMs
getAttribute	Gets the attribute values for an AWS CloudHSM key and writes them to a file.
getCaviumPrivKey	Creates a fake PEM-format version of a private key and exports it to a file.
getCert	Retrieves an HSM's partitions certificates and saves them to a file.

Command	Description
getKeyInfo	Gets the HSM user IDs of users who can use the key. If the key is quorum controlled, it gets the number of users in the quorum.
help	Displays help information about the commands available in key_mgmt_util.
importPrivateKey	Imports a private key into an HSM.
importPubKey	Imports a public key into an HSM.
imSymKey	Imports a plaintext copy of a symmetric key from a file into the HSM.
insertMaskedObject	Inserts a masked object from a file on disk into an HSM contained by related cluster to the object's origin cluster. Related clusters are any clusters generated from a backup of the origin cluster.
???	Determines whether or not a given file contains a real private key or a example PEM key.
listAttributes	Lists the attributes of an AWS CloudHSM key and the constants that represent them.
listUsers	Gets the users in the HSMs, their user type and ID, and other attributes.
loginHSM and logoutHSM	Log in and out of the HSMs in a cluster.
setAttribute	Converts a session key to a persistent key.
sign	Generate a signature for a file using a chosen private key.

Command	Description
unWrapKey	Imports a wrapped (encrypted) key from a file into the HSMs.
verify	Verifies whether a given key was used to sign a given file.
wrapKey	Exports an encrypted copy of a key from the HSM to a file.

Encrypt and decrypt an AWS CloudHSM file using KMU

Use the **aesWrapUnwrap** command in AWS CloudHSM key_mgmt_util to encrypt or decrypt the contents of a file on disk. This command is designed to wrap and unwrap encryption keys, but you can use it on any file that contains less than 4 KB (4096 bytes) of data.

aesWrapUnwrap uses AES Key Wrap. It uses an AES key on the HSM as the wrapping or unwrapping key. Then it writes the result to another file on disk.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
aesWrapUnwrap -h

aesWrapUnwrap -m <wrap-unwrap mode>
              -f <file-to-wrap-unwrap>
              -w <wrapping-key-handle>
              [-i <wrapping-IV>]
              [-out <output-file>]
```

Examples

These examples show how to use **aesWrapUnwrap** to encrypt and decrypt an encryption key in a file.

Example : Wrap an encryption key

This command uses **aesWrapUnwrap** to wrap a Triple DES symmetric key that was exported from the HSM in plaintext into the `3DES.key` file. You can use a similar command to wrap any key saved in a file.

The command uses the `-m` parameter with a value of `1` to indicate wrap mode. It uses the `-w` parameter to specify an AES key in the HSM (key handle 6) as the wrapping key. It writes the resulting wrapped key to the `3DES.key.wrapped` file.

The output shows that the command was successful and that the operation used the default IV, which is preferred.

```
 Command:  aesWrapUnwrap -f 3DES.key -w 6 -m 1 -out 3DES.key.wrapped

        Warning: IV (-i) is missing.
                 0xA6A6A6A6A6A6A6A6 is considered as default IV
result data:
49 49 E2 D0 11 C1 97 22
17 43 BD E3 4E F4 12 75
8D C1 34 CF 26 10 3A 8D
6D 0A 7B D5 D3 E8 4D C2
79 09 08 61 94 68 51 B7

result written to file 3DES.key.wrapped

        Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS
```

Example : Unwrap an encryption key

This example shows how to use **aesWrapUnwrap** to unwrap (decrypt) a wrapped (encrypted) key in a file. You might want to do an operation like this one before importing a key to the HSM. For example, if you try to use the imSymKey command to import an encrypted key, it returns an error because the encrypted key doesn't have the format that is required for a plaintext key of that type.

The command unwraps the key in the `3DES.key.wrapped` file and writes the plaintext to the `3DES.key.unwrapped` file. The command uses the `-m` parameter with a value of `0` to indicate unwrap mode. It uses the `-w` parameter to specify an AES key in the HSM (key handle 6) as the wrapping key. It writes the resulting wrapped key to the `3DES.key.unwrapped` file.

```
 Command:  aesWrapUnwrap -m 0 -f 3DES.key.wrapped -w 6 -out 3DES.key.unwrapped
```

```
        Warning: IV (-i) is missing.
                 0xA6A6A6A6A6A6A6A6 is considered as default IV
result data:
14 90 D7 AD D6 E4 F5 FA
A1 95 6F 24 89 79 F3 EE
37 21 E6 54 1F 3B 8D 62

result written to file 3DES.key.unwrapped

        Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS
```

Parameters

-h

Displays help for the command.

Required: Yes

-m

Specifies the mode. To wrap (encrypt) the file content, type 1; to unwrap (decrypt) the file content, type `0`.

Required: Yes

-f

Specifies the file to wrap. Enter a file that contains less than 4 KB (4096 bytes) of data. This operation is designed to wrap and unwrap encryption keys.

Required: Yes

-w

Specifies the wrapping key. Enter the key handle of an AES key on the HSM. This parameter is required. To find key handles, use the findKey command.

To create a wrapping key, use genSymKey to generate an AES key (type 31).

Required: Yes

-i

Specifies an alternate initial value (IV) for the algorithm. Use the default value unless you have a special condition that requires an alternative.

Default: `0xA6A6A6A6A6A6A6A6`. The default value is defined in the AES Key Wrap algorithm specification.

Required: No

-out

Specifies an alternate name for the output file that contains the wrapped or unwrapped key. The default is `wrapped_key` (for wrap operations) and `unwrapped_key` (for unwrap operations) in the local directory.

If the file exists, the **aesWrapUnwrap** overwrites it without warning. If the command fails, **aesWrapUnwrap** creates an output file with no contents.

Default: For wrap: `wrapped_key`. For unwrap: `unwrapped_key`.

Required: No

Related topics

- exSymKey
- imSymKey
- unWrapKey
- wrapKey

Delete an AWS CloudHSM key using KMU

Use the **deleteKey** command in the AWS CloudHSM key_mgmt_util to delete a key from the hardware security module (HSM) in an AWS CloudHSM cluster. You can only delete one key at a time. Deleting one key in a key pair has no effect on the other key in the pair.

Only the key owner can delete a key. Users who share the key can use it in cryptographic operations, but not delete it.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
deleteKey -h
```

```
deleteKey -k
```

Examples

These examples show how to use **deleteKey** to delete keys from your HSMs.

Example : Delete a key

This command deletes the key with key handle 6. When the command succeeds, **deleteKey** returns success messages from each HSM in the cluster.

```
Command: deleteKey -k 6

        Cfm3DeleteKey returned: 0x00 : HSM Return: SUCCESS

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : Delete a key (failure)

When the command fails because no key has the specified key handle, **deleteKey** returns an invalid object handle error message.

```
Command: deleteKey -k 252126

        Cfm3FindKey returned: 0xa8 : HSM Error: Invalid object handle is passed to this
 operation

        Cluster Error Status
        Node id 1 and err state 0x000000a8 : HSM Error: Invalid object handle is passed
 to this operation
        Node id 2 and err state 0x000000a8 : HSM Error: Invalid object handle is passed
 to this operation
```

When the command fails because the current user is not the owner of the key, the command returns an access denied error.

```
Command:  deleteKey -k 262152
```

```
Cfm3DeleteKey returned: 0xc6 : HSM Error: Key Access is denied.
```

Parameters

-h

Displays command line help for the command.

Required: Yes

-k

Specifies the key handle of the key to delete. To find the key handles of keys in the HSM, use findKey.

Required: Yes

Related topics

- findKey

Describe an AWS CloudHSM error using KMU

Use the **Error2String** helper command in the AWS CloudHSM key_mgmt_util to return the error that corresponds to a key_mgmt_util hexadecimal error code. You can use this command when troubleshooting your commands and scripts.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
Error2String -h

Error2String -r <response-code>
```

Examples

These examples show how to use **Error2String** to get the error string for a key_mgmt_util error code.

Example : Get an error description

This command gets the error description for the `0xdb` error code. The description explains that an attempt to log in to key_mgmt_util failed because the user has the wrong user type. Only crypto users (CU) can log in to key_mgmt_util.

```
Command:  Error2String -r 0xdb

Error Code db maps to HSM Error: Invalid User Type.
```

Example : Find the error code

This example shows where to find the error code in a key_mgmt_util error. The error code, `0xc6`, appears after the string: `Cfm3<command-name> returned: .`

In this example, getKeyInfo indicates that the current user (user 4) can use the key in cryptographic operations. Nevertheless, when the user tries to use deleteKey to delete the key, the command returns error code `0xc6`.

```
 Command:  deleteKey -k 262162

 Cfm3DeleteKey returned: <0xc6> : HSM Error: Key Access is denied

 Cluster Error Status

 Command:  getKeyInfo -k 262162

 Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

Owned by user 3

also, shared to following 1 user(s):

         4
```

If the `0xc6` error is reported to you, you can use an **Error2String** command like this one to look up the error. In this case, the `deleteKey` command failed with an access denied error because the key is shared with the current user but owned by a different user. Only key owners have permission to delete a key.

```
        Command:  Error2String -r 0xa8

        Error Code c6 maps to HSM Error: Key Access is denied
```

Parameters

-h

Displays help for the command.

Required: Yes

-r

Specifies a hexadecimal error code. The `0x` hexadecimal indicator is required.

Required: Yes

Exit the AWS CloudHSM KMU

Use the **exit** command in the AWS CloudHSM key_mgmt_util to exit the key_mgmt_util. Upon successful exit, you will be returned to your standard command line.

Before you run any key_mgmt_util command, you must start key_mgmt_util.

Syntax

```
exit
```

Parameters

There are no parameters for this command.

Related topics

- Start key_mgmt_util

Export a private AWS CloudHSM key using KMU

Use the **exportPrivateKey** command in the AWS CloudHSM key_mgmt_util to export an asymmetric private key from a hardware security module (HSM) to a file. The HSM does not allow direct export of keys in cleartext. The command wraps the private key using an AES wrapping key you specify, decrypts the wrapped bytes, and copies the cleartext private key to a file.

The **exportPrivateKey** command does not remove the key from the HSM, change its key attributes, or prevent you from using the key in further cryptographic operations. You can export the same key multiple times.

You can only export private keys that have `OBJ_ATTR_EXTRACTABLE` attribute value 1. You must specify an AES wrapping key that has `OBJ_ATTR_WRAP` and `OBJ_ATTR_DECRYPT` attributes value 1. To find a key's attributes, use the **getAttribute** command.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
exportPrivateKey -h

exportPrivateKey -k <private-key-handle>
                 -w <wrapping-key-handle>
                 -out <key-file>
                 [-m <wrapping-mechanism>]
                 [-wk <wrapping-key-file>]
```

Examples

This example shows how to use **exportPrivateKey** to export a private key out of an HSM.

Example : Export a private key

This command exports a private key with handle 15 using a wrapping key with handle 16 to a PEM file called `exportKey.pem`. When the command succeeds, **exportPrivateKey** returns a success message.

```
Command: exportPrivateKey -k 15 -w 16 -out exportKey.pem

Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS
```

```
PEM formatted private key is written to exportKey.pem
```

Parameters

This command takes the following parameters.

-h

Displays command line help for the command.

Required: Yes

-k

Specifies the key handle of the private key to be exported.

Required: Yes

-w

Specifies the key handle of the wrapping key. This parameter is required. To find key handles, use the **findKey** command.

To determine whether a key can be used as a wrapping key, use **getAttribute** to get the value of the `OBJ_ATTR_WRAP` attribute (262). To create a wrapping key, use **genSymKey** to create an AES key (type 31).

If you use the `-wk` parameter to specify an external unwrapping key, the `-w` wrapping key is used to wrap, but not unwrap, the key during export.

Required: Yes

-out

Specifies the name of the file to which the exported private key will be written.

Required: Yes

-m

Specifies the wrapping mechanism with which to wrap the private key being exported. The only valid value is 4, which represents the `NIST_AES_WRAP mechanism.`

Default: 4 (`NIST_AES_WRAP`)

Required: No

-wk

Specifies the key to be used to unwrap the key being exported. Enter the path and name of a file that contains a plaintext AES key.

When you include this parameter, **exportPrivateKey** uses the key in the `-w` file to wrap the key being exported and uses the key specified by the `-wk` parameter to unwrap it.

Default: Use the wrapping key specified in the `-w` parameter to both wrap and unwrap.

Required: No

Related topics

- importPrivateKey
- wrapKey
- unWrapKey
- genSymKey

Export a public AWS CloudHSM key using KMU

Use the **exportPubKey** command in the AWS CloudHSM key_mgmt_util to export a public key in an HSM to a file. You can use it to export public keys that you generate in an HSM. You can also use this command to export public keys that were imported into an HSM, such as those imported with the **importPubKey** command.

The **exportPubKey** operation copies the key material to a file that you specify. But it does not remove the key from the HSM, change its key attributes, or prevent you from using the key in further cryptographic operations. You can export the same key multiple times.

You can only export public keys that have a `OBJ_ATTR_EXTRACTABLE` value of `1`. To find a key's attributes, use the **getAttribute** command.

Before you run any `key_mgmt_util` command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
exportPubKey -h

exportPubKey -k <public-key-handle>
             -out <key-file>
```

Examples

This example shows how to use **exportPubKey** to export a public key from an HSM.

Example : Export a public key

This command exports a public key with handle `10` to a file called `public.pem`. When the command succeeds, **exportPubKey** returns a success message.

```
Command: exportPubKey -k 10 -out public.pem

PEM formatted public key is written to public.pem

Cfm3ExportPubKey returned: 0x00 : HSM Return: SUCCESS
```

Parameters

This command takes the following parameters.

-h

Displays command line help for the command.

Required: Yes

-k

Specifies the key handle of the public key to be exported.

Required: Yes

-out

Specifies the name of the file to which the exported public key will be written.

Required: Yes

Related topics

- importPubKey
- Generate Keys

Export a plaintext copy of an AWS CloudHSM key using KMU

Use the **exSymKey** command in the AWS CloudHSM key_mgmt_util tool to export a plaintext copy of a symmetric key from the hardware security module (HSM) and saves it in a file on disk. To export an encrypted (wrapped) copy of a key, use wrapKey. To import a plaintext key, like the ones that `exSymKey` exports, use imSymKey.

During the export process, **exSymKey** uses an AES key that you specify (the *wrapping key*) to *wrap* (encrypt) and then *unwrap* (decrypt) the key to be exported. However, the result of the export operation is a plaintext (*unwrapped*) key on disk.

Only the owner of a key, that is, the CU user who created the key, can export it. Users who share the key can use it in cryptographic operations, but they cannot export it.

The **exSymKey** operation copies the key material to a file that you specify, but it does not remove the key from the HSM, change its key attributes, or prevent you from using the key in cryptographic operations. You can export the same key multiple times.

exSymKey exports only symmetric keys. To export public keys, use exportPubKey. To export private keys, use exportPrivateKey.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
exSymKey -h

exSymKey -k <key-to-export>
         -w <wrapping-key>
         -out <key-file>
         [-m 4]
```

```
        [-wk <unwrapping-key-file> ]
```

Examples

These examples show how to use **exSymKey** to export symmetric keys that you own from your HSMs.

Example : Export a 3DES symmetric key

This command exports a Triple DES (3DES) symmetric key (key handle 7). It uses an existing AES key (key handle 6) in the HSM as the wrapping key. Then it writes the plaintext of the 3DES key to the `3DES.key` file.

The output shows that key 7 (the 3DES key) was successfully wrapped and unwrapped, and then written to the `3DES.key` file.

> **Warning**
>
> Although the output says that a "Wrapped Symmetric Key" was written to the output file, the output file contains a plaintext (unwrapped) key.

```
        Command: exSymKey -k 7 -w 6 -out 3DES.key

       Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS


Wrapped Symmetric Key written to file "3DES.key"
```

Example : Exporting with session-only wrapping key

This example shows how to use a key that exists only in the session as the wrapping key. Because the key to be exported is wrapped, immediately unwrapped, and delivered as plaintext, there is no need to retain the wrapping key.

This series of commands exports an AES key with key handle 8 from the HSM. It uses an AES session key created especially for the purpose.

The first command uses genSymKey to create a 256-bit AES key. It uses the `-sess` parameter to create a key that exists only in the current session.

The output shows that the HSM creates key 262168.

```
        Command:  genSymKey -t 31 -s 32 -l AES-wrapping-key -sess

        Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 262168

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

Next, the example verifies that key 8, the key to be exported, is a symmetric key that is extractable. It also verifies that the wrapping key, key 262168, is an AES key that exists only in the session. You can use the findKey command, but this example exports the attributes of both keys to files and then uses `grep` to find the relevant attribute values in the file.

These commands use `getAttribute` with an `-a` value of 512 (all) to get all attributes for keys 8 and 262168. For information about the key attributes, see the the section called "Key attribute reference".

```
getAttribute -o 8 -a 512 -out attributes/attr_8
getAttribute -o 262168 -a 512 -out attributes/attr_262168
```

These commands use `grep` to verify the attributes of the key to be exported (key 8) and the session-only wrapping key (key 262168).

```
    // Verify that the key to be exported is a symmetric key.
    $  grep -A 1 "OBJ_ATTR_CLASS" attributes/attr_8
    OBJ_ATTR_CLASS
    0x04

    // Verify that the key to be exported is extractable.
    $  grep -A 1 "OBJ_ATTR_KEY_TYPE" attributes/attr_8
    OBJ_ATTR_EXTRACTABLE
    0x00000001

    // Verify that the wrapping key is an AES key
    $  grep -A 1 "OBJ_ATTR_KEY_TYPE" attributes/attr_262168
```

```
OBJ_ATTR_KEY_TYPE
0x1f

// Verify that the wrapping key is a session key
$  grep -A 1 "OBJ_ATTR_TOKEN" attributes/attr_262168
OBJ_ATTR_TOKEN
0x00

// Verify that the wrapping key can be used for wrapping
 $  grep -A 1 "OBJ_ATTR_WRAP" attributes/attr_262168
OBJ_ATTR_WRAP
0x00000001
```

Finally, we use an **exSymKey** command to export key 8 using the session key (key 262168) as the wrapping key.

When the session ends, key 262168 no longer exists.

```
        Command:  exSymKey -k 8 -w 262168 -out aes256_H8.key

        Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS


Wrapped Symmetric Key written to file "aes256_H8.key"
```

Example : Use an external unwrapping key

This example shows how to use an external unwrapping key to export a key from the HSM.

When you export a key from the HSM, you specify an AES key on the HSM to be the wrapping key. By default, that wrapping key is used to wrap and unwrap the key to be exported. However, you can use the `-wk` parameter to tell **exSymKey** to use an external key in a file on disk for unwrapping. When you do, the key specified by the `-w` parameter wraps the target key, and the key in the file specified by the `-wk` parameter unwraps the key.

Because the wrapping key must be an AES key, which is symmetric, the wrapping key in the HSM and unwrapping key on disk must be have the same key material. To do this, you must import the wrapping key to the HSM or export the wrapping key from the HSM before the export operation.

This example creates a key outside of the HSM and imports it into the HSM. It uses the internal copy of the key to wrap a symmetric key that is being exported, and the copy of key in the file to unwrap it.

The first command uses OpenSSL to generate a 256-bit AES key. It saves the key to the `aes256-forImport.key` file. The OpenSSL command does not return any output, but you can use several commands to confirm its success. This example uses the **wc** (word count) tool, which confirms that the file that contains 32 bytes of data.

```
$  openssl rand -out keys/aes256-forImport.key 32

$ wc keys/aes256-forImport.key
 0  2 32 keys/aes256-forImport.key
```

This command uses the imSymKey command to import the AES key from the `aes256-forImport.key` file to the HSM. When the command completes, the key exists in the HSM with key handle 262167 and in the `aes256-forImport.key` file.

```
Command:  imSymKey -f keys/aes256-forImport.key -t 31 -l aes256-imported -w 6

        Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Unwrapped.  Key Handle: 262167

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

This command uses the key in an export operation. The command uses **exSymKey** to export key 21, a 192-bit AES key. To wrap the key, it uses key 262167, which is the copy that was imported into the HSM. To unwrap the key, it uses the same key material in the `aes256-forImport.key` file. When the command completes, key 21 is exported to the `aes192_h21.key` file.

```
        Command:  exSymKey -k 21 -w 262167 -out aes192_H21.key -wk aes256-forImport.key

        Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS
```

```
Wrapped Symmetric Key written to file "aes192_H21.key"
```

Parameters

-h

Displays help for the command.

Required: Yes

-k

Specifies the key handle of the key to export. This parameter is required. Enter the key handle of a symmetric key that you own. This parameter is required. To find key handles, use the findKey command.

To verify that a key can be exported, use the getAttribute command to get the value of the `OBJ_ATTR_EXTRACTABLE` attribute, which is represented by constant `354`. Also, you can export only keys that you own. To find the owner of a key, use the getKeyInfo command.

Required: Yes

-w

Specifies the key handle of the wrapping key. This parameter is required. To find key handles, use the findKey command.

A *wrapping key* is a key in the HSM that is used to encrypt (wrap) and then decrypt (unwrap) the key to be exported. Only AES keys can be used as wrapping keys.

You can use any AES key (of any size) as a wrapping key. Because the wrapping key wraps, and then immediately unwraps, the target key, you can use as session-only AES key as a wrapping key. To determine whether a key can be used as a wrapping key, use getAttribute to get the value of the `OBJ_ATTR_WRAP` attribute, which is represented by the constant 262. To create a wrapping key, use genSymKey to create an AES key (type 31).

If you use the `-wk` parameter to specify an external unwrapping key, the `-w` wrapping key is used to wrap, but not to unwrap, the key during export.

> **Note**
>
> Key 4 represents an unsupported internal key. We recommend that you use an AES key that you create and manage as the wrapping key.

Required: Yes

-out

Specifies the path and name of the output file. When the command succeeds, this file contains the exported key in plaintext. If the file already exists, the command overwrites it without warning.

Required: Yes

-m

Specifies the wrapping mechanism. The only valid value is `4`, which represents the `NIST_AES_WRAP` mechanism.

Required: No

Default: 4

-wk

Use the AES key in the specified file to unwrap the key that is being exported. Enter the path and name of a file that contains a plaintext AES key.

When you include this parameter. `exSymKey` uses the key in the HSM that is specified by the `-w` parameter to wrap the key that is being exported and it uses the key in the `-wk` file to unwrap it. The `-w` and `-wk` parameter values must resolve to the same plaintext key.

Required: No

Default: Use the wrapping key on the HSM to unwrap.

Related topics

- genSymKey
- imSymKey
- wrapKey

Extract an AWS CloudHSM key using KMU

Use the **extractMaskedObject** command in the AWS CloudHSM key_mgmt_util to extract a key from a hardware security module (HSM) and saves it to a file as a masked object. Masked objects are *cloned* objects that can only be used after inserting them back into the original cluster by using the **insertMaskedObject** command. You can only insert a masked object into the same cluster from which it was generated, or a clone of that cluster. This includes any cloned versions of the cluster generated by copying a backup across regions and using that backup to create a new cluster.

Masked objects are an efficient way to offload and synchronize keys, including nonextractable keys (that is, keys that have a `OBJ_ATTR_EXTRACTABLE` value of `0`). This way, keys can be securely synced across related clusters in different regions without the need to update the AWS CloudHSM configure file.

> **Important**
>
> Upon insertion, masked objects are decrypted and given a key handle that is different from the key handle of the original key. A masked object includes all metadata associated with the original key, including attributes, ownership and sharing information, and quorum settings. If you need to sync keys across clusters in an application, use syncKey in the cloudhsm_mgmt_util instead.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM. The **extractMaskedObject** command can be used either by the CU who owns the key or any CO.

Syntax

```
extractMaskedObject -h

extractMaskedObject -o <object-handle>
                    -out <object-file>
```

Examples

This example shows how to use **extractMaskedObject** to extract a key from an HSM as a masked object.

Example : Extract a masked object

This command extracts a masked object out of an HSM from a key with handle `524295` and saves it as a file called `maskedObj`. When the command succeeds, **extractMaskedObject** returns a success message.

```
Command: extractMaskedObject -o 524295 -out maskedObj

Object was masked and written to file "maskedObj"

        Cfm3ExtractMaskedObject returned: 0x00 : HSM Return: SUCCESS
```

Parameters

This command takes the following parameters.

-h

Displays command line help for the command.

Required: Yes

-o

Specifies the handle of the key to extract as a masked object.

Required: Yes

-out

Specifies the name of the file to which the masked object will be saved.

Required: Yes

Related topics

- insertMaskedObject
- syncKey
- Copying a Backup Across Regions
- Creating an AWS CloudHSM Cluster from a Previous Backup

Search for AWS CloudHSM keys by attributes using KMU

Use the **findKey** command in the AWS CloudHSM key_mgmt_util to search for keys by the values of the key attributes. When a key matches all the criteria that you set, **findKey** returns the key handle. With no parameters, **findKey** returns the key handles of all the keys that you can use in the HSM. To find the attribute values of a particular key, use getAttribute.

Like all key_mgmt_util commands, **findKey** is user specific. It returns only the keys that the current user can use in cryptographic operations. This includes keys that current user owns and keys that have been shared with the current user.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
findKey -h

findKey [-c <key class>]
        [-t <key type>]
        [-l <key label>]
        [-id <key ID>]
        [-sess (0 | 1)]
        [-u <user-ids>]
        [-m <modulus>]
        [-kcv <key_check_value>]
```

Examples

These examples show how to use **findKey** to find and identify keys in your HSMs.

Example : Find all keys

This command finds all keys for the current user in the HSM. The output includes keys that the user owns and shares, and all public keys in the HSMs.

To get the attributes of a key with a particular key handle, use getAttribute. To determine whether the current user owns or shares a particular key, use getKeyInfo or findAllKeys in cloudhsm_mgmt_util.

```
Command: findKey

Total number of keys present 13

 number of keys matched from start index 0::12
6, 7, 524296, 9, 262154, 262155, 262156, 262157, 262158, 262159, 262160, 262161, 262162

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

```
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Example : Find keys by type, user, and session

This command finds persistent AES keys that the current user and user 3 can use. (User 3 might be able to use other keys that the current user cannot see.)

```
Command: findKey -t 31 -sess 0 -u 3
```

Example : Find keys by class and label

This command finds all public keys for the current user with the `2018-sept` label.

```
Command: findKey -c 2 -l 2018-sept
```

Example : Find RSA keys by modulus

This command finds RSA keys (type 0) for the current user that were created by using the modulus in the `m4.txt` file.

```
Command: findKey -t 0 -m m4.txt
```

Parameters

-h

Displays help for the command.

Required: Yes

-t

Finds keys of the specified type. Enter the constant that represents the key class. For example, to find 3DES keys, type `-t 21`.

Valid values:

- 0: RSA
- 1: DSA
- 3: EC

- 16: GENERIC_SECRET
- 18: RC4
- 21: Triple DES (3DES)
- 31: AES

Required: No

-c

Finds keys in the specified class. Enter the constant that represents the key class. For example, to find public keys, type `-c 2`.

Valid values for each key type:

- 2: Public. This class contains the public keys of public–private key pairs.
- 3: Private. This class contains the private keys of public–private key pairs.
- 4: Secret. This class contains all symmetric keys.

Required: No

-l

Finds keys with the specified label. Type the exact label. You cannot use wildcard characters or regular expressions in the `--l` value.

Required: No

-id

Finds the key with the specified ID. Type the exact ID string. You cannot use wildcard characters or regular expressions in the `-id` value.

Required: No

-sess

Finds keys by session status. To find keys that are valid only in the current session, type `1`. To find persistent keys, type `0`.

Required: No

-u

Finds keys the specified users and the current user share. Type a comma-separated list of HSM user IDs, such as `-u 3` or `-u 4,7`. To find the IDs of users on an HSM, use listUsers.

When you specify one user ID, **findKey** returns the keys for that user. When you specify multiple user IDs, **findKey** returns the keys that all the specified users can use.

Because **findKey** only returns keys that the current user can use, the `-u` results are always identical to or a subset of the current user's keys. To get all keys that are owned by or shared with any user, crypto officers (COs) can use findAllKeys in cloudhsm_mgmt_util.

Required: No

-m

Finds keys that were created by using the RSA modulus in the specified file. Type the path to file that stores the modulus.

-m specifies the binary file containing RSA modulus to match with (optional).

Required: No

-kcv

Finds keys with the specified key check value.

The *key check value* (KCV) is a 3-byte hash or checksum of a key that is generated when the HSM imports or generates a key. You can also calculate a KCV outside of the HSM, such as after you export a key. You can then compare the KCV values to confirm the identity and integrity of the key. To get the KCV of a key, use getAttribute.

AWS CloudHSM uses the following standard method to generate a key check value:

- **Symmetric keys**: First 3 bytes of the result of encrypting a zero-block with the key.
- **Asymmetric key pairs**: First 3 bytes of the SHA-1 hash of the public key.
- **HMAC keys**: KCV for HMAC keys is not supported at this time.

Required: No

Output

The **findKey** output lists the total number of matching keys and their key handles.

```
        Command:  findKey
Total number of keys present 10
```

```
 number of keys matched from start index 0::9
6, 7, 8, 9, 10, 11, 262156, 262157, 262158, 262159

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Related topics

- findSingleKey
- getKeyInfo
- getAttribute
- findAllKeys in cloudhsm_mgmt_util
- Key Attribute Reference

Verify an AWS CloudHSM key using KMU

Use the **findSingleKey** command in the AWS CloudHSM key_mgmt_util tool to verify that a key exists on all hardware security modules (HSM) in the AWS CloudHSM cluster.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
findSingleKey -h

findSingleKey -k <key-handle>
```

Example

Example

This command verifies that key 252136 exists on all three HSMs in the cluster.

```
Command: findSingleKey -k 252136
```

```
Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS

        Cluster Error Status
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Parameters

-h

Displays help for the command.

Required: Yes

-k

Specifies the key handle of one key in the HSM. This parameter is required.

To find key handles, use the findKey command.

Required: Yes

Related topics

- findKey
- getKeyInfo
- getAttribute

Generate an AWS CloudHSM DSA key pair using KMU

Use the **genDSAKeyPair** command in the AWS CloudHSM key_mgmt_util tool to generate a Digital Signing Algorithm (DSA) key pair in your hardware security modules (HSM). You must specify the modulus length; the command generates the modulus value. You can also assign an ID, share the key with other HSM users, create nonextractable keys, and create keys that expire when the session ends. When the command succeeds, it returns the *key handles* that the HSM assigns to the public and private keys. You can use the key handles to identify the keys to other commands.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

> **Tip**
>
> To find the attributes of a key that you have created, such as the type, length, label, and ID, use getAttribute. To find the keys for a particular user, use getKeyInfo. To find keys based on their attribute values, use findKey.

Syntax

```
genDSAKeyPair -h

genDSAKeyPair -m <modulus length>
              -l <label>
              [-id <key ID>]
              [-min_srv <minimum number of servers>]
              [-m_value <0..8>]
              [-nex]
              [-sess]
              [-timeout <number of seconds> ]
              [-u <user-ids>]
              [-attest]
```

Examples

These examples show how to use **genDSAKeyPair** to create a DSA key pair.

Example : Create a DSA key pair

This command creates a DSA key pair with a DSA label. The output shows that the key handle of the public key is 19 and the handle of the private key is 21.

```
Command: genDSAKeyPair -m 2048 -l DSA

        Cfm3GenerateKeyPair: returned: 0x00 : HSM Return: SUCCESS

        Cfm3GenerateKeyPair:    public key handle: 19    private key handle: 21

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : Create a session-only DSA key pair

This command creates a DSA key pair that is valid only in the current session. The command assigns a unique ID of `DSA_temp_pair` in addition to the required (nonunique) label. You might want to create a key pair like this to sign and verify a session-only token. The output shows that the key handle of the public key is 12 and the handle of the private key is 14.

```
Command: genDSAKeyPair -m 2048 -l DSA-temp -id DSA_temp_pair -sess

        Cfm3GenerateKeyPair: returned: 0x00 : HSM Return: SUCCESS

        Cfm3GenerateKeyPair:    public key handle: 12    private key handle: 14

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

To confirm that the key pair exists only in the session, use the `-sess` parameter of findKey with a value of 1 (true).

```
  Command: findKey -sess 1

  Total number of keys present 2

 number of keys matched from start index 0::1
12, 14

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Example : Create a shared, nonextractable DSA key pair

This command creates a DSA key pair. The private key is shared with three other users, and it cannot be exported from the HSM. Public keys can be used by any user and can always be extracted.

```
        Command:  genDSAKeyPair -m 2048 -l DSA -id DSA_shared_pair -nex -u 3,5,6

        Cfm3GenerateKeyPair: returned: 0x00 : HSM Return: SUCCESS
```

```
        Cfm3GenerateKeyPair:    public key handle: 11    private key handle: 19

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : Create a quorum-controlled key pair

This command creates a DSA key pair with the label `DSA-mV2`. The command uses the `-u` parameter to share the private key with user 4 and 6. It uses the `-m_value` parameter to require a quorum of at least two approvals for any cryptographic operations that use the private key. The command also uses the `-attest` parameter to verify the integrity of the firmware on which the key pair is generated.

The output shows that the command generates a public key with key handle 12 and a private key with key handle 17, and that the attestation check on the cluster firmware passed.

```
        Command:  genDSAKeyPair -m 2048 -l DSA-mV2 -m_value 2 -u 4,6 -attest

        Cfm3GenerateKeyPair: returned: 0x00 : HSM Return: SUCCESS

        Cfm3GenerateKeyPair:    public key handle: 12    private key handle: 17

        Attestation Check : [PASS]

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

This command uses getKeyInfo on the private key (key handle 17). The output confirms that the key is owned by the current user (user 3) and that it is shared with users 4 and 6 (and no others). The output also shows that quorum authentication is enabled and the quorum size is two.

```
        Command:  getKeyInfo -k 17

        Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

        Owned by user 3
```

```
        also, shared to following 2 user(s):

                 4
                 6
         2 Users need to approve to use/manage this key
```

Parameters

-h

Displays help for the command.

Required: Yes

-m

Specifies the length of the modulus in bits. The only valid value is `2048`.

Required: Yes

-l

Specifies a user-defined label for the key pair. Type a string. The same label applies to both keys in the pair. The maximum size allowable for `label` is 127 characters.

You can use any phrase that helps you to identify the key. Because the label does not have to be unique, you can use it to group and categorize keys.

Required: Yes

-id

Specifies a user-defined identifier for the key pair. Type a string that is unique in the cluster. The default is an empty string. The ID that you specify applies to both keys in the pair.

Default: No ID value.

Required: No

-min_srv

Specifies the minimum number of HSMs on which the key is synchronized before the value of the `-timeout` parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of `min_srv` to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-m_value

Specifies the number of users who must approve any cryptographic operation that uses the private key in the pair. Type a value from `0` to `8`.

This parameter establishes a quorum authentication requirement for the private key. The default value, `0`, disables the quorum authentication feature for the key. When quorum authentication is enabled, the specified number of users must sign a token to approve cryptographic operations that use the private key, and operations that share or unshare the private key.

To find the `m_value` of a key, use getKeyInfo.

This parameter is valid only when the `-u` parameter in the command shares the key pair with enough users to satisfy the `m_value` requirement.

Default: 0

Required: No

-nex

Makes the private key nonextractable. The private key that is generated cannot be exported from the HSM. Public keys are always extractable.

Default: Both the public and private keys in the key pair are extractable.

Required: No

-sess

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use setAttribute.

Default: The key is persistent.

Required: No

-timeout

Specifies how long (in seconds) the command waits for a key to be synchronized to the number of HSMs specified by the `min_srv` parameter.

This parameter is valid only when the `min_srv` parameter is also used in the command.

Default: No timeout. The command waits indefinitely and returns only when the key is synchronized to the minimum number of servers.

Required: No

-u

Shares the private key in the pair with the specified users. This parameter gives other HSM crypto users (CUs) permission to use the private key in cryptographic operations. Public keys can be used by any user without sharing.

Type a comma-separated list of HSM user IDs, such as `-u 5,6`. Do not include the HSM user ID of the current user. To find HSM user IDs of CUs on the HSM, use listUsers. To share and unshare existing keys, use shareKey in the cloudhsm_mgmt_util.

Default: Only the current user can use the private key.

Required: No

-attest

Runs an integrity check that verifies that the firmware on which the cluster runs has not been tampered with.

Default: No attestation check.

Required: No

Related topics

- genRSAKeyPair
- genSymKey

- genECCKeyPair

Generate an AWS CloudHSM ECC key pair using KMU

Use the `genECCKeyPair` command in the AWS CloudHSM key_mgmt_util tool to generate an Elliptic Curve Cryptography (ECC) key pair in your hardware security modules (HSM). When running the `genECCKeyPair` command, you must specify the elliptic curve identifier and a label for the key pair. You can also share the private key with other CU users, create non-extractable keys, quorum-controlled keys, and keys that expire when the session ends. When the command succeeds, it returns the key handles that the HSM assigns to the public and private ECC keys. You can use the key handles to identify the keys to other commands.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

> **Tip**
>
> To find the attributes of a key that you have created, such as the type, length, label, and ID, use getAttribute. To find the keys for a particular user, use getKeyInfo. To find keys based on their attribute values, use findKey.

Syntax

```
genECCKeyPair -h

genECCKeyPair -i <EC curve id>
              -l <label>
              [-id <key ID>]
              [-min_srv <minimum number of servers>]
              [-m_value <0..8>]
              [-nex]
              [-sess]
              [-timeout <number of seconds> ]
              [-u <user-ids>]
              [-attest]
```

Examples

The following examples show how to use **genECCKeyPair** to create ECC key pairs in your HSMs.

Example : Create and examine an ECC key pair

This command uses an NID_secp384r1 elliptic curve and an `ecc14` label to create an ECC key pair. The output shows that the key handle of the private key is 262177 and the key handle of the public key is 262179. The label applies to both the public and private keys.

```
Command: genECCKeyPair -i 14 -l ecc14

        Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

        Cfm3GenerateKeyPair:    public key handle: 262179    private key handle: 262177

        Cluster Error Status
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

After generating the key, you can examine its attributes. Use getAttribute to write all of the attributes (represented by the constant 512) of the new ECC private key to the `attr_262177` file.

```
Command: getAttribute -o 262177 -a 512 -out attr_262177
got all attributes of size 529 attr cnt 19
Attributes dumped into attr_262177

        Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS
```

Then use the `cat` command to view the contents of the `attr_262177` attribute file. The output shows the key is an elliptic curve private key that can be used for signing, but not for encrypting, decrypting, wrapping, unwrapping, or verifying. The key is persistent and exportable.

```
$  cat attr_262177

OBJ_ATTR_CLASS
0x03
OBJ_ATTR_KEY_TYPE
0x03
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
0x01
OBJ_ATTR_ENCRYPT
```

```
0x00
OBJ_ATTR_DECRYPT
0x00
OBJ_ATTR_WRAP
0x00
OBJ_ATTR_UNWRAP
0x00
OBJ_ATTR_SIGN
0x01
OBJ_ATTR_VERIFY
0x00
OBJ_ATTR_LOCAL
0x01
OBJ_ATTR_SENSITIVE
0x01
OBJ_ATTR_EXTRACTABLE
0x01
OBJ_ATTR_LABEL
ecc2
OBJ_ATTR_ID

OBJ_ATTR_VALUE_LEN
0x0000008a
OBJ_ATTR_KCV
0xbbb32a
OBJ_ATTR_MODULUS
044a0f9d01d10f7437d9fa20995f0cc742552e5ba16d3d7e9a65a33e20ad3e569e68eb62477a9960a87911e6121d112
OBJ_ATTR_MODULUS_BITS
0x0000019f
```

Example Using an invalid EEC curve

This command attempts to create an ECC key pair by using an NID_X9_62_prime192v1 curve. Because this elliptic curve is not valid for FIPS-mode HSMs, the command fails. The message reports that a server in the cluster is unavailable, but this does not typically indicate a problem with the HSMs in the cluster.

```
Command:  genECCKeyPair -i 1 -l ecc1

        Cfm3GenerateKeyPair returned: 0xb3 : HSM Error: This operation violates the
 current configured/FIPS policies

        Cluster Error Status
```

```
        Node id 0 and err state 0x30000085 : HSM CLUSTER ERROR: Server in cluster is
unavailable
```

Parameters

-h

Displays help for the command.

Required: Yes

-i

Specifies the identifier for the elliptic curve. Enter an identifier.

Valid values:

- **2**: NID_X9_62_prime256v1
- **14**: NID_secp384r1
- **16**: NID_secp256k1

Required: Yes

-l

Specifies a user-defined label for the key pair. Type a string. The same label applies to both keys in the pair. The maximum size allowable for `label` is 127 characters.

You can use any phrase that helps you to identify the key. Because the label does not have to be unique, you can use it to group and categorize keys.

Required: Yes

-id

Specifies a user-defined identifier for the key pair. Type a string that is unique in the cluster. The default is an empty string. The ID that you specify applies to both keys in the pair.

Default: No ID value.

Required: No

-min_srv

Specifies the minimum number of HSMs on which the key is synchronized before the value of the `-timeout` parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of `min_srv` to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-m_value

Specifies the number of users who must approve any cryptographic operation that uses the private key in the pair. Type a value from `0` to `8`.

This parameter establishes a quorum authentication requirement for the private key. The default value, `0`, disables the quorum authentication feature for the key. When quorum authentication is enabled, the specified number of users must sign a token to approve cryptographic operations that use the private key, and operations that share or unshare the private key.

To find the `m_value` of a key, use getKeyInfo.

This parameter is valid only when the `-u` parameter in the command shares the key pair with enough users to satisfy the `m_value` requirement.

Default: 0

Required: No

-nex

Makes the private key nonextractable. The private key that is generated cannot be exported from the HSM. Public keys are always extractable.

Default: Both the public and private keys in the key pair are extractable.

Required: No

-sess

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use setAttribute.

Default: The key is persistent.

Required: No

-timeout

Specifies how long (in seconds) the command waits for a key to be synchronized to the number of HSMs specified by the `min_srv` parameter.

This parameter is valid only when the `min_srv` parameter is also used in the command.

Default: No timeout. The command waits indefinitely and returns only when the key is synchronized to the minimum number of servers.

Required: No

-u

Shares the private key in the pair with the specified users. This parameter gives other HSM crypto users (CUs) permission to use the private key in cryptographic operations. Public keys can be used by any user without sharing.

Type a comma-separated list of HSM user IDs, such as `-u 5,6`. Do not include the HSM user ID of the current user. To find HSM user IDs of CUs on the HSM, use listUsers. To share and unshare existing keys, use shareKey in the cloudhsm_mgmt_util.

Default: Only the current user can use the private key.

Required: No

-attest

Runs an integrity check that verifies that the firmware on which the cluster runs has not been tampered with.

Default: No attestation check.

Required: No

Related topics

- genSymKey
- genRSAKeyPair
- genDSAKeyPair

Generate an AWS CloudHSM RSA key pair using KMU

Use the **genRSAKeyPair** command in the AWS CloudHSM key_mgmt_util tool to generate an RSA asymmetric key pair. You specify the key type, modulus length, and a public exponent. The command generates a modulus of the specified length and creates the key pair. You can assign an ID, share the key with other HSM users, create nonextractable keys and keys that expire when the session ends. When the command succeeds, it returns a key handle that the HSM assigns to the key. You can use the key handle to identify the key to other commands.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

> **Tip**
>
> To find the attributes of a key that you have created, such as the type, length, label, and ID, use getAttribute. To find the keys for a particular user, use getKeyInfo. To find keys based on their attribute values, use findKey.

Syntax

```
genRSAKeyPair -h

genRSAKeyPair -m <modulus length>
              -e <public exponent>
              -l <label>
              [-id <key ID>]
              [-min_srv <minimum number of servers>]
              [-m_value <0..8>]
```

```
               [-nex]
               [-sess]
               [-timeout <number of seconds> ]
               [-u <user-ids>]
               [-attest]
```

Examples

These examples show how to use **genRSAKeyPair** to create asymmetric key pairs in your HSMs.

Example : Create and examine an RSA key pair

This command creates an RSA key pair with a 2048-bit modulus and an exponent of 65537. The output shows that the public key handle is `2100177` and the private key handle is `2100426`.

```
Command: genRSAKeyPair -m 2048 -e 65537 -l rsa_test

Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

        Cfm3GenerateKeyPair:    public key handle: 2100177    private key handle:
 2100426

        Cluster Status:
        Node id 0 status: 0x00000000 : HSM Return: SUCCESS
        Node id 1 status: 0x00000000 : HSM Return: SUCCESS
```

The next command uses getAttribute to get the attributes of the public key that we just created. It writes the output to the `attr_2100177` file. It is followed by a **cat** command that gets the content of the attribute file. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

The resulting hexadecimal values confirm that it is a public key (`OBJ_ATTR_CLASS 0x02`) with a type of RSA (`OBJ_ATTR_KEY_TYPE 0x00`). You can use this public key to encrypt (`OBJ_ATTR_ENCRYPT 0x01`), but not to decrypt (`OBJ_ATTR_DECRYPT 0x00`). The results also include the key length (512, `0x200`), the modulus, the modulus length (2048, `0x800`), and the public exponent (65537, `0x10001`).

```
Command:  getAttribute -o 2100177 -a 512 -out attr_2100177

Attribute size: 801, count: 26
Written to: attr_2100177 file
```

```
        Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS

$  cat attr_2100177
OBJ_ATTR_CLASS
0x02
OBJ_ATTR_KEY_TYPE
0x00
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
0x01
OBJ_ATTR_ENCRYPT
0x01
OBJ_ATTR_DECRYPT
0x00
OBJ_ATTR_WRAP
0x01
OBJ_ATTR_UNWRAP
0x00
OBJ_ATTR_SIGN
0x00
OBJ_ATTR_VERIFY
0x01
OBJ_ATTR_LOCAL
0x01
OBJ_ATTR_SENSITIVE
0x00
OBJ_ATTR_EXTRACTABLE
0x01
OBJ_ATTR_LABEL
rsa_test
OBJ_ATTR_ID

OBJ_ATTR_VALUE_LEN
0x00000200
OBJ_ATTR_KCV
0xc51c18
OBJ_ATTR_MODULUS
0xbb9301cc362c1d9724eb93da8adab0364296bde7124a241087d9436b9be57e4f7780040df03c2c
1c0fe6e3b61aa83c205280119452868f66541bbbffacbbe787b8284fc81deaeef2b8ec0ba25a077d
6983c77a1de7b17cbe8e15b203868704c6452c2810344a7f2736012424cf0703cf15a37183a1d2d0
97240829f8f90b063dd3a41171402b162578d581980976653935431da0c1260bfe756d85dca63857
d9f27a541676cb9c7def0ef6a2a89c9b9304bcac16fdf8183c0a555421f9ad5dfeb534cf26b65873
970cdf1a07484f1c128b53e10209cc6f7ac308669112968c81a5de408e7f644fe58b1a9ae1286fec
```

```
b3e4203294a96fae06f8f0db7982cb5d7f
OBJ_ATTR_MODULUS_BITS
0x00000800
OBJ_ATTR_PUBLIC_EXPONENT
0x010001
OBJ_ATTR_TRUSTED
0x00
OBJ_ATTR_WRAP_WITH_TRUSTED
0x00
OBJ_ATTR_DESTROYABLE
0x01
OBJ_ATTR_DERIVE
0x00
OBJ_ATTR_ALWAYS_SENSITIVE
0x00
OBJ_ATTR_NEVER_EXTRACTABLE
0x00
```

Example : Generate a shared RSA key pair

This command generates an RSA key pair and shares the private key with user 4, another CU on the HSM. The command uses the `m_value` parameter to require at least two approvals before the private key in the pair can be used in a cryptographic operation. When you use the `m_value` parameter, you must also use `-u` in the command and the `m_value` cannot exceed the total number of users (number of values in `-u` + owner).

```
Command:  genRSAKeyPair -m 2048 -e 65537 -l rsa_mofn -id rsa_mv2 -u 4 -m_value 2

        Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

        Cfm3GenerateKeyPair:    public key handle: 27    private key handle: 28

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

Parameters

-h

Displays help for the command.

Required: Yes

-m

Specifies the length of the modulus in bits. The minimum value is 2048.

Required: Yes

-e

Specifies the public exponent. The value must be an odd number greater than or equal to 65537.

Required: Yes

-l

Specifies a user-defined label for the key pair. Type a string. The same label applies to both keys in the pair. The maximum size allowable for `label` is 127 characters.

You can use any phrase that helps you to identify the key. Because the label does not have to be unique, you can use it to group and categorize keys.

Required: Yes

-id

Specifies a user-defined identifier for the key pair. Type a string that is unique in the cluster. The default is an empty string. The ID that you specify applies to both keys in the pair.

Default: No ID value.

Required: No

-min_srv

Specifies the minimum number of HSMs on which the key is synchronized before the value of the `-timeout` parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of `min_srv` to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-m_value

Specifies the number of users who must approve any cryptographic operation that uses the private key in the pair. Type a value from `0` to `8`.

This parameter establishes a quorum authentication requirement for the private key. The default value, `0`, disables the quorum authentication feature for the key. When quorum authentication is enabled, the specified number of users must sign a token to approve cryptographic operations that use the private key, and operations that share or unshare the private key.

To find the `m_value` of a key, use getKeyInfo.

This parameter is valid only when the `-u` parameter in the command shares the key pair with enough users to satisfy the `m_value` requirement.

Default: 0

Required: No

-nex

Makes the private key nonextractable. The private key that is generated cannot be exported from the HSM. Public keys are always extractable.

Default: Both the public and private keys in the key pair are extractable.

Required: No

-sess

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use setAttribute.

Default: The key is persistent.

Required: No

-timeout

Specifies how long (in seconds) the command waits for a key to be synchronized to the number of HSMs specified by the `min_srv` parameter.

This parameter is valid only when the `min_srv` parameter is also used in the command.

Default: No timeout. The command waits indefinitely and returns only when the key is synchronized to the minimum number of servers.

Required: No

-u

Shares the private key in the pair with the specified users. This parameter gives other HSM crypto users (CUs) permission to use the private key in cryptographic operations. Public keys can be used by any user without sharing.

Type a comma-separated list of HSM user IDs, such as `-u 5,6`. Do not include the HSM user ID of the current user. To find HSM user IDs of CUs on the HSM, use listUsers. To share and unshare existing keys, use shareKey in the cloudhsm_mgmt_util.

Default: Only the current user can use the private key.

Required: No

-attest

Runs an integrity check that verifies that the firmware on which the cluster runs has not been tampered with.

Default: No attestation check.

Required: No

Related topics

- genSymKey
- genDSAKeyPair

- genECCKeyPair

Generate an AWS CloudHSM symmetric key using KMU

Use the **genSymKey** command in the AWS CloudHSM key_mgmt_util tool to generate a symmetric key in your hardware security modules (HSM). You can specify the key type and size, assign an ID and label, and share the key with other HSM users. You can also create nonextractable keys and keys that expire when the session ends. When the command succeeds, it returns a key handle that the HSM assigns to the key. You can use the key handle to identify the key to other commands.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
genSymKey -h

genSymKey -t <key-type>
          -s <key-size>
          -l <label>
          [-id <key-ID>]
          [-min_srv <minimum-number-of-servers>]
          [-m_value <0..8>]
          [-nex]
          [-sess]
          [-timeout <number-of-seconds> ]
          [-u <user-ids>]
          [-attest]
```

Examples

These examples show how to use **genSymKey** to create symmetric keys in your HSMs.

> **Tip**
>
> To use the keys you make with these examples for HMAC operations, you must set `OBJ_ATTR_SIGN` and `OBJ_ATTR_VERIFY` to `TRUE` after you generate the key. To set these values, use **setAttribute** in CloudHSM Management Utility (CMU). For more information, see setAttribute.

Example : Generate an AES key

This command creates a 256-bit AES key with an `aes256` label. The output shows that the key handle of the new key is 6.

```
Command: genSymKey -t 31 -s 32 -l aes256

        Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 6

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : Create a session key

This command creates a nonextractable 192-bit AES key that is valid only in the current session. You might want to create a key like this to wrap (and then immediately unwrap) a key that is being exported.

```
Command: genSymKey -t 31 -s 24 -l tmpAES -id wrap01 -nex -sess
```

Example : Return quickly

This command creates a generic 512-byte key with a label of `IT_test_key`. The command does not wait for the key to be synchronized to all HSMs in the cluster. Instead, it returns as soon as the key is created on any one HSM (`-min_srv 1`) or in 1 second (`-timeout 1`), whichever is shorter. If the key is not synchronized to the specified minimum number of HSMs before the timeout expires, it is not generated. You might want to use a command like this in a script that creates numerous keys, like the `for` loop in the following example.

```
Command: genSymKey -t 16 -s 512 -l IT_test_key -min_srv 1 -timeout 1

$  for i in {1..30};
     do /opt/cloudhsm/bin/key_mgmt_util singlecmd loginHSM -u CU -s example_user -p
 example_pwd genSymKey -l aes -t 31 -s 32 -min_srv 1 -timeout 1;
 done;
```

Example : Create a quorum authorized generic key

This command creates a 2048-bit generic secret key with the label `generic-mV2`. The command uses the `-u` parameter to share the key with another CU, user 6. It uses the `-m_value` parameter to require a quorum of at least two approvals for any cryptographic operations that use the key. The command also uses the `-attest` parameter to verify the integrity of the firmware on which the key is generated.

The output shows that the command generated a key with key handle 9 and that the attestation check on the cluster firmware passed.

```
                Command:  genSymKey -t 16 -s 2048 -l generic-mV2 -m_value 2 -u 6 -
attest

        Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 9

        Attestation Check : [PASS]

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : Create and examine a key

This command creates a Triple DES key with a `3DES_shared` label and an ID of `IT-02`. The key can be used by the current user, and users 4 and 5. The command fails if the ID is not unique in the cluster or if the current user is user 4 or 5.

The output shows that the new key has key handle 7.

```
Command: genSymKey -t 21 -s 24 -l 3DES_shared -id IT-02 -u 4,5

       Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 7

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

To verify that the new 3DES key is owned by the current user and shared with users 4 and 5, use **getKeyInfo**. The command uses the handle that was assigned to the new key (`Key Handle: 7`).

The output confirms that the key is owned by user 3 and shared with users 4 and 5.

```
Command:  getKeyInfo -k 7

        Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

        Owned by user 3

        also, shared to following 2 user(s):

                 4, 5
```

To confirm the other properties of the key, use getAttribute. The first command uses `getAttribute` to get all attributes (`-a 512`) of key handle 7 (`-o 7`). It writes them to the `attr_7` file. The second command uses `cat` to get the contents of the `attr_7` file.

This command confirms that key 7 is a 192-bit (`OBJ_ATTR_VALUE_LEN 0x00000018` or 24-byte) 3DES (`OBJ_ATTR_KEY_TYPE 0x15`) symmetric key (`OBJ_ATTR_CLASS 0x04`) with a label of `3DES_shared` (`OBJ_ATTR_LABEL 3DES_shared`) and an ID of `IT_02` (`OBJ_ATTR_ID IT-02`). The key is persistent (`OBJ_ATTR_TOKEN 0x01`) and extractable (`OBJ_ATTR_EXTRACTABLE 0x01`) and can be used for encryption, decryption, and wrapping.

Tip

To find the attributes of a key that you have created, such as the type, length, label, and ID, use getAttribute. To find the keys for a particular user, use getKeyInfo. To find keys based on their attribute values, use findKey.

For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

```
Command:  getAttribute -o 7 -a 512 -out attr_7

got all attributes of size 444 attr cnt 17
Attributes dumped into attr_7 file

        Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS
```

```
$  cat attr_7

OBJ_ATTR_CLASS
0x04
OBJ_ATTR_KEY_TYPE
0x15
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
0x01
OBJ_ATTR_ENCRYPT
0x01
OBJ_ATTR_DECRYPT
0x01
OBJ_ATTR_WRAP
0x00
OBJ_ATTR_UNWRAP
0x00
OBJ_ATTR_SIGN
0x00
OBJ_ATTR_VERIFY
0x00
OBJ_ATTR_LOCAL
0x01
OBJ_ATTR_SENSITIVE
0x01
OBJ_ATTR_EXTRACTABLE
0x01
OBJ_ATTR_LABEL
3DES_shared
OBJ_ATTR_ID
IT-02
OBJ_ATTR_VALUE_LEN
0x00000018
OBJ_ATTR_KCV
0x59a46e
```

> **Tip**
>
> To use the keys you make with these examples for HMAC operations, you must set `OBJ_ATTR_SIGN` and `OBJ_ATTR_VERIFY` to `TRUE` after you generate the key. To set these values, use **setAttribute** in CMU. For more information, see setAttribute.

Parameters

-h

Displays help for the command.

Required: Yes

-t

Specifies the type of the symmetric key. Enter the constant that represents the key type. For example, to create an AES key, type `-t 31`.

Valid values:

- 16: GENERIC_SECRET. A *generic secret key* is a byte array that does not conform to any particular standard, such as the requirements for an AES key.
- 18: RC4. RC4 keys are not valid on FIPS-mode HSMs
- 21: Triple DES (3DES). In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.
- 31: AES

Required: Yes

-s

Specifies the key size in bytes. For example, to create a 192-bit key, type `24`.

Valid values for each key type:

- AES: 16 (128 bits), 24 (192 bits), 32 (256 bits)
- 3DES: 24 (192 bits)
- Generic Secret: <3584 (28672 bits)

Required: Yes

-l

Specifies a user-defined label for the key. Type a string.

You can use any phrase that helps you to identify the key. Because the label does not have to be unique, you can use it to group and categorize keys.

Required: Yes

-attest

Runs an integrity check that verifies that the firmware on which the cluster runs has not been tampered with.

Default: No attestation check.

Required: No

-id

Specifies a user-defined identifier for the key. Type a string that is unique in the cluster. The default is an empty string.

Default: No ID value.

Required: No

-min_srv

Specifies the minimum number of HSMs on which the key is synchronized before the value of the `-timeout` parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of `min_srv` to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-m_value

Specifies the number of users who must approve any cryptographic operation that uses the key. Type a value from `0` to `8`.

This parameter establishes a quorum authentication requirement for the key. The default value, `0`, disables the quorum authentication feature for the key. When quorum authentication is enabled, the specified number of users must sign a token to approve cryptographic operations that use the key, and operations that share or unshare the key.

To find the `m_value` of a key, use getKeyInfo.

This parameter is valid only when the `-u` parameter in the command shares the key with enough users to satisfy the `m_value` requirement.

Default: 0

Required: No

-nex

Makes the key nonextractable. The key that is generated cannot be exported from the HSM.

Default: The key is extractable.

Required: No

-sess

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use setAttribute.

Default: The key is persistent.

Required: No

-timeout

Specifies how long (in seconds) the command waits for a key to be synchronized to the number of HSMs specified by the `min_srv` parameter.

This parameter is valid only when the `min_srv` parameter is also used in the command.

Default: No timeout. The command waits indefinitely and returns only when the key is synchronized to the minimum number of servers.

Required: No

-u

Shares the key with the specified users. This parameter gives other HSM crypto users (CUs) permission to use this key in cryptographic operations.

Type a comma-separated list of HSM user IDs, such as -u `5,6`. Do not include the HSM user ID of the current user. To find HSM user IDs of CUs on the HSM, use listUsers. To share and unshare existing keys, use shareKey in the cloudhsm_mgmt_util.

Default: Only the current user can use the key.

Required: No

Related topics

- exSymKey
- genRSAKeyPair
- genDSAKeyPair
- genECCKeyPair
- setAttribute

Get an AWS CloudHSM key attribute using KMU

Use the **getAttribute** command in the AWS CloudHSM key_mgmt_util to write one or all of the attribute values for an AWS CloudHSM key to a file. If the attribute you specify does not exist for the key type, such as the modulus of an AES key, **getAttribute** returns an error.

Key attributes are properties of a key. They include characteristics, like the key type, class, label, and ID, and values that represent actions that you can perform with the key, like encrypt, decrypt, wrap, sign, and verify.

You can use **getAttribute** only on keys that you own and key that are shared with you. You can run this command or the getAttribute command in cloudhsm_mgmt_util, which gets one attribute value of a key from all HSMs in a cluster, and writes it to stdout or to a file.

To get a list of attributes and the constants that represent them, use the listAttributes command. To change the attribute values of existing keys, use setAttribute in key_mgmt_util and setAttribute in cloudhsm_mgmt_util. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
getAttribute -h

getAttribute -o <key handle>
             -a <attribute constant>
             -out <file>
```

Examples

These examples show how to use **getAttribute** to get the attributes of keys in your HSMs.

Example : Get the key type

This example gets the type of the key, such an AES, 3DES, or generic key, or an RSA or elliptic curve key pair.

The first command runs listAttributes, which gets the key attributes and the constants that represent them. The output shows that the constant for key type is 256. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

```
Command: listAttributes

Description
===========
The following are all of the possible attribute values for getAttributes.

      OBJ_ATTR_CLASS                  = 0
      OBJ_ATTR_TOKEN                  = 1
      OBJ_ATTR_PRIVATE                = 2
      OBJ_ATTR_LABEL                  = 3
      OBJ_ATTR_KEY_TYPE               = 256
      OBJ_ATTR_ID                     = 258
      OBJ_ATTR_SENSITIVE              = 259
      OBJ_ATTR_ENCRYPT                = 260
      OBJ_ATTR_DECRYPT                = 261
      OBJ_ATTR_WRAP                   = 262
```

```
        OBJ_ATTR_UNWRAP                 = 263
        OBJ_ATTR_SIGN                   = 264
        OBJ_ATTR_VERIFY                 = 266
        OBJ_ATTR_LOCAL                  = 355
        OBJ_ATTR_MODULUS                = 288
        OBJ_ATTR_MODULUS_BITS           = 289
        OBJ_ATTR_PUBLIC_EXPONENT        = 290
        OBJ_ATTR_VALUE_LEN              = 353
        OBJ_ATTR_EXTRACTABLE            = 354
        OBJ_ATTR_KCV                    = 371
```

The second command runs **getAttribute**. It requests the key type (attribute 256) for key handle 524296 and writes it to the `attribute.txt` file.

```
Command: getAttribute -o 524296 -a 256 -out attribute.txt
Attributes dumped into attribute.txt file
```

The final command gets the content of the key file. The output reveals that the key type is `0x15` or 21, which is a Triple DES (3DES) key. For definitions of the class and type values, see the Key Attribute Reference.

```
$  cat attribute.txt
OBJ_ATTR_KEY_TYPE
0x00000015
```

Example : Get all attributes of a key

This command gets all attributes of the key with key handle 6 and writes them to the `attr_6` file. It uses an attribute value of 512, which represents all attributes.

```
Command: getAttribute -o 6 -a 512 -out attr_6

got all attributes of size 444 attr cnt 17
Attributes dumped into attribute.txt file

        Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS>
```

This command shows the content of a sample attribute file with all attribute values. Among the values, it reports that key is a 256-bit AES key with an ID of `test_01` and a label of `aes256`.The key is extractable and persistent, that is, not a session-only key. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

```
$  cat attribute.txt

OBJ_ATTR_CLASS
0x04
OBJ_ATTR_KEY_TYPE
0x15
OBJ_ATTR_TOKEN
0x01
OBJ_ATTR_PRIVATE
0x01
OBJ_ATTR_ENCRYPT
0x01
OBJ_ATTR_DECRYPT
0x01
OBJ_ATTR_WRAP
0x01
OBJ_ATTR_UNWRAP
0x01
OBJ_ATTR_SIGN
0x00
OBJ_ATTR_VERIFY
0x00
OBJ_ATTR_LOCAL
0x01
OBJ_ATTR_SENSITIVE
0x01
OBJ_ATTR_EXTRACTABLE
0x01
OBJ_ATTR_LABEL
aes256
OBJ_ATTR_ID
test_01
OBJ_ATTR_VALUE_LEN
0x00000020
OBJ_ATTR_KCV
0x1a4b31
```

Parameters

-h

Displays help for the command.

Required: Yes

-o

Specifies the key handle of the target key. You can specify only one key in each command. To get the key handle of a key, use findKey.

Also, you must own the specified key or it must be shared with you. To find the users of a key, use getKeyInfo.

Required: Yes

-a

Identifies the attribute. Enter a constant that represents an attribute, or `512`, which represents all attributes. For example, to get the key type, type `256`, which is the constant for the `OBJ_ATTR_KEY_TYPE` attribute.

To list the attributes and their constants, use listAttributes. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

Required: Yes

-out

Writes the output to the specified file. Type a file path. You cannot write the output to `stdout`.

If the specified file exists, **getAttribute** overwrites the file without warning.

Required: Yes

Related topics

- getAttribute in cloudhsm_mgmt_util
- listAttributes
- setAttribute
- findKey
- Key Attribute Reference

Export an AWS CloudHSM key to fake PEM format using KMU

Use the **getCaviumPrivKey** command in the AWS CloudHSM key_mgmt_util to export a private key from a hardware security module (HSM) in fake PEM format. The fake PEM file, which does not contain the actual private key material but instead references the private key in the HSM, can then be used to establish SSL/TLS offloading from your web server to AWS CloudHSM. For more information, see SSL/TLS Offload on Linux using Tomcat or SSL/TLS Offload on Linux using NGINX or Apache.

Before you run any key_mgmt_util command, you must start key_mgmt_util and login to the HSM as a crypto user (CU).

Syntax

```
getCaviumPrivKey -h

getCaviumPrivKey -k <private-key-handle>
                 -out <fake-PEM-file>
```

Examples

This example shows how to use **getCaviumPrivKey** to export a private key in fake PEM format.

Example : Export a fake PEM file

This command creates and exports a fake PEM version of a private key with handle 15 and saves it to a file called `cavKey.pem`. When the command succeeds, **exportPrivateKey** returns a success message.

```
Command: getCaviumPrivKey -k 15 -out cavKey.pem

Private Key Handle is written to cavKey.pem in fake PEM format

        getCaviumPrivKey returned: 0x00 : HSM Return: SUCCESS
```

Parameters

This command takes the following parameters.

-h

Displays command line help for the command.

Required: Yes

-k

Specifies the key handle of the private key to be exported in fake PEM format.

Required: Yes

-out

Specifies the name of the file to which the fake PEM key will be written.

Required: Yes

Related topics

- importPrivateKey
- SSL/TLS Offload on Linux using Tomcat
- SSL/TLS Offload on Linux using NGINX or Apache

Get HSM partition certificates using AWS CloudHSM KMU

Use the **getCert** command in the AWS CloudHSM key_mgmt_util to retrieve a hardware security module's (HSM) partition certificates and saves them to a file. When you run the command, you designate the type of certificate to retrieve. To do that, you use one of the corresponding integers as described in the Parameters section that follows. To learn about the role of each of these certificates, see Verify HSM Identity.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
getCert -h

getCert -f <file-name>
        -t <certificate-type>
```

Example

This example shows how to use **getCert** to retrieve a cluster's customer root certificate and save it as a file.

Example : Retrieve a customer root certificate

This command exports a customer root certificate (represented by integer 4) and saves it to a file called `userRoot.crt`. When the command succeeds, **getCert** returns a success message.

```
Command: getCert -f userRoot.crt -s 4

Cfm3GetCert() returned 0 :HSM Return: SUCCESS
```

Parameters

This command takes the following parameters.

-h

Displays command line help for the command.

Required: Yes

-f

Specifies the name of the file to which the retrieved certificate will be saved.

Required: Yes

-s

An integer that specifies the type of partition certificate to retrieve. The integers and their corresponding certificate types are as follows:

- **1** – Manufacturer root certificate
- **2** – Manufacturer hardware certificate
- **4** – Customer root certificate
- **8** – Cluster certificate (signed by customer root certificate)
- **16** – Cluster certificate (chained to the manufacturer root certificate)

Required: Yes

Related topics

- Verify HSM Identity

Get the users of an AWS CloudHSM key using KMU

Use the **getKeyInfo** command in the AWS CloudHSM key_mgmt_util to return the hardware security module (HSM) user IDs of users who can use the key, including the owner and crypto users (CU) with whom the key is shared. When quorum authentication is enabled on a key, **getKeyInfo** also returns the number of users who must approve cryptographic operations that use the key. You can run **getKeyInfo** only on keys that you own and keys that are shared with you.

When you run **getKeyInfo** on public keys, **getKeyInfo** returns only the key owner, even though all users of the HSM can use the public key. To find the HSM user IDs of users in your HSMs, use listUsers. To find the keys for a particular user, use findKey -u.

You own the keys that you create. You can share a key with other users when you create it. Then, to share or unshare an existing key, use shareKey in cloudhsm_mgmt_util.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
getKeyInfo -h

getKeyInfo -k <key-handle>
```

Examples

These examples show how to use **getKeyInfo** to get information about the users of a key.

Example : Get the users for a symmetric key

This command gets the users who can use the AES (symmetric) key with key handle 9. The output shows that user 3 owns the key and has shared it with user 4.

```
Command:  getKeyInfo -k 9

       Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

       Owned by user 3
```

```
        also, shared to following 1 user(s):

                 4
```

Example : Get the users for an asymmetric key pair

These commands use **getKeyInfo** to get the users who can use the keys in an RSA (asymmetric) key pair. The public key has key handle 21. The private key has key handle 20.

When you run **getKeyInfo** on the private key (20), it returns the key owner (3) and crypto users (CUs) 4 and 5, with whom the key is shared.

```
Command:  getKeyInfo -k 20

        Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

        Owned by user 3

        also, shared to following 2 user(s):

                 4
                 5
```

When you run **getKeyInfo** on the public key (21), it returns only the key owner (3).

```
Command:  getKeyInfo -k 21

        Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS

        Owned by user 3
```

To confirm that user 4 can use the public key (and all public keys on the HSM), use the `-u` parameter of findKey.

The output shows that user 4 can use both the public (21) and private (20) key in the key pair. User 4 can also use all other public keys and any private keys that they have created or that have been shared with them.

```
Command:  findKey -u 4
Total number of keys present 8
```

```
 number of keys matched from start index 0::7
11, 12, 262159, 262161, 262162, 19, 20, 21

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Example : Get the quorum authentication value (m_value) for a key

This example shows how to get the `m_value` for a key, that is, the number of users in the quorum who must approve any cryptographic operations that use the key.

When quorum authentication is enabled on a key, a quorum of users must approve any cryptographic operations that use the key. To enable quorum authentication and set the quorum size, use the `-m_value` parameter when you create the key.

This command uses genRSAKeyPair to create an RSA key pair that is shared with user 4. It uses the `m_value` parameter to enable quorum authentication on the private key in the pair and set the quorum size to two users. The number of users must be large enough to provide the required approvals.

The output shows that the command created public key 27 and private key 28.

```
 Command:  genRSAKeyPair -m 2048 -e 195193 -l rsa_mofn -id rsa_mv2 -u 4 -m_value 2

        Cfm3GenerateKeyPair returned: 0x00 : HSM Return: SUCCESS

        Cfm3GenerateKeyPair:    public key handle: 27    private key handle: 28

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

This command uses **getKeyInfo** to get information about the users of the private key. The output shows that the key is owned by user 3 and shared with user 4. It also shows that a quorum of two users must approve every cryptographic operation that uses the key.

```
Command:  getKeyInfo -k 28

        Cfm3GetKey returned: 0x00 : HSM Return: SUCCESS
```

```
        Owned by user 3

        also, shared to following 1 user(s):

                 4
         2 Users need to approve to use/manage this key
```

Parameters

-h

Displays command line help for the command.

Required: Yes

-k

Specifies the key handle of one key in the HSM. Enter the key handle of a key that you own or share. This parameter is required.

To find key handles, use the findKey command.

Required: Yes

Related topics

- getKeyInfo in cloudhsm_mgmt_util
- listUsers
- findKey
- findAllKeys in cloudhsm_mgmt_util

Display help information for AWS CloudHSM KMU

Use the **help** command in the AWS CloudHSM key_mgmt_util to display information about all available key_mgmt_util commands.

Before you run **help**, you must start key_mgmt_util.

Syntax

```
help
```

Example

This example shows the output of the help command.

Example

```
Command:  help

Help Commands Available:

Syntax: <command> -h



   Command               Description
   =======               ===========

   exit                   Exits this application
   help                   Displays this information

        Configuration and Admin Commands
   getHSMInfo             Gets the HSM Information
   getPartitionInfo       Gets the Partition Information
   listUsers              Lists all users of a partition
   loginStatus            Gets the Login Information
   loginHSM               Login to the HSM
   logoutHSM              Logout from the HSM

        M of N commands
   getToken               Initiate an MxN service and get Token
   delToken               delete Token(s)
   approveToken           Approves an MxN service
   listTokens             List all Tokens in the current partition

        Key Generation Commands

        Asymmetric Keys:
   genRSAKeyPair          Generates an RSA Key Pair
   genDSAKeyPair          Generates a DSA Key Pair
   genECCKeyPair          Generates an ECC Key Pair

        Symmetric Keys:
   genPBEKey              Generates a PBE DES3 key
   genSymKey              Generates a Symmetric keys
```

```
     Key Import/Export Commands
createPublicKey        Creates an RSA public key
importPubKey           Imports RSA/DSA/EC Public key
exportPubKey           Exports RSA/DSA/EC Public key
importPrivateKey       Imports RSA/DSA/EC private key
exportPrivateKey       Exports RSA/DSA/EC private key
imSymKey               Imports a Symmetric key
exSymKey               Exports a Symmetric key
wrapKey                Wraps a key from from HSM using the specified handle
unWrapKey              UnWraps a key into HSM using the specified handle

     Key Management Commands
deleteKey              Delete Key
setAttribute           Sets an attribute of an object
getKeyInfo             Get Key Info about shared users/sessions
findKey                Find Key
findSingleKey          Find single Key
getAttribute           Reads an attribute from an object

     Certificate Setup Commands
getCert                Gets Partition Certificates stored on HSM

     Key Transfer Commands
insertMaskedObject     Inserts a masked object
extractMaskedObject    Extracts a masked object

     Management Crypto Commands
sign                   Generates a signature
verify                 Verifies a signature
aesWrapUnwrap          Does NIST AES Wrap/Unwrap

     Helper Commands
Error2String           Converts Error codes to Strings
                       save key handle in fake PEM format
getCaviumPrivKey       Saves an RSA private key handle
                       in fake PEM format
IsValidKeyHandlefile   Checks if private key file has
                       an HSM key handle or a real key
listAttributes         List all attributes for getAttributes
listECCCurveIds        List HSM supported ECC CurveIds
```

Parameters

There are no parameters for this command.

Related topics

- loginHSM and logoutHSM

Import a private key using AWS CloudHSM KMU

Use the **importPrivateKey** command in the AWS CloudHSM key_mgmt_util to import an asymmetric private key from a file to a hardware security module (HSM). The HSM does not allow direct import of keys in cleartext. The command encrypts the private key using an AES wrapping key you specify and unwraps the key inside the HSM. If you are trying to associate an AWS CloudHSM key with a certificate, refer to this topic.

> **Note**
>
> You cannot import a password-protected PEM key using a symmetric or private key.

You must specify an AES wrapping key that has `OBJ_ATTR_UNWRAP` and `OBJ_ATTR_ENCRYPT` attribute value 1. To find a key's attributes, use the **getAttribute** command.

> **Note**
>
> This command does not offer the option to mark the imported key as non-exportable.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
importPrivateKey -h

importPrivateKey -l <label>
                 -f <key-file>
                 -w <wrapping-key-handle>
                 [-sess]
                 [-id <key-id>]
                 [-m_value <0...8>]
                 [min_srv <minimum-number-of-servers>]
                 [-timeout <number-of-seconds>]
```

```
                    [-u <user-ids>]
                    [-wk <wrapping-key-file>]
                    [-attest]
```

Examples

This example shows how to use **importPrivateKey** to import a private key into an HSM.

Example : Import a private key

This command imports the private key from a file named `rsa2048.key` with the label `rsa2048-imported` and a wrapping key with handle 524299. When the command succeeds, **importPrivateKey** returns a key handle for the imported key and a success message.

```
Command: importPrivateKey -f rsa2048.key -l rsa2048-imported -w 524299

BER encoded key length is 1216

Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

Private Key Unwrapped.  Key Handle: 524301

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Parameters

This command takes the following parameters.

-h

Displays command line help for the command.

Required: Yes

-l

Specifies the user-defined private key label.

Required: Yes

-f

Specifies the file name of the key to import.

Required: Yes

-w

Specifies the key handle of the wrapping key. This parameter is required. To find key handles, use the **findKey** command.

To determine whether a key can be used as a wrapping key, use **getAttribute** to get the value of the `OBJ_ATTR_WRAP` attribute (262). To create a wrapping key, use **genSymKey** to create an AES key (type 31).

If you use the `-wk` parameter to specify an external unwrapping key, the `-w` wrapping key is used to wrap, but not unwrap, the key during import.

Required: Yes

-sess

Specifies the imported key as a session key.

Default: The imported key is held as a persistent (token) key in the cluster.

Required: No

-id

Specifies the ID of the key to be imported.

Default: No ID value.

Required: No

-m_value

Specifies the number of users who must approve any cryptographic operation that uses the imported key. Enter a value from **0** to **8**.

This parameter is valid only when the `-u` parameter in the command shares the key with enough users to satisfy the `m_value` requirement.

Default: 0

Required: No

-min_srv

Specifies the minimum number of HSMs on which the imported key is synchronized before the value of the `-timeout` parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of `min_srv` to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-timeout

Specifies the number of seconds to wait for the key to sync across HSMs when the `min-serv` parameter is included. If no number is specified, the polling continues forever.

Default: No limit

Required: No

-u

Specifies the list of users with whom to share the imported private key. This parameter gives other HSM crypto users (CUs) permission to use the imported key in cryptographic operations.

Enter a comma-separated list of HSM user IDs, such as `-u 5,6`. Do not include the HSM user ID of the current user. To find the HSM user IDs of CUs on the HSM, use listUsers.

Default: Only the current user can use the imported key.

Required: No

-wk

Specifies the key to be used to wrap the key that is being imported. Enter the path and name of a file that contains a plaintext AES key.

When you include this parameter, **importPrivateKey** uses the key in the `-wk` file to wrap the key being imported. It also uses the key specified by the `-w` parameter to unwrap it.

Default: Use the wrapping key specified in the `-w` parameter to both wrap and unwrap.

Required: No

-attest

Performs an attestation check on the firmware response to ensure that the firmware on which the cluster runs has not been compromised.

Required: No

Related topics

- wrapKey
- unWrapKey
- genSymKey
- exportPrivateKey

Import a public key using AWS CloudHSM KMU

Use the **importPubKey** command in the AWS CloudHSM key_mgmt_util to import a PEM format public key into a hardware security module (HSM). You can use it to import public keys that were generated outside of the HSM. You can also use the command to import keys that were exported from an HSM, such as those exported by the exportPubKey command.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
importPubKey -h

importPubKey -l <label>
             -f <key-file>
             [-sess]
             [-id <key-id>]
             [min_srv <minimum-number-of-servers>]
             [-timeout <number-of-seconds>]
```

Examples

This example shows how to use **importPubKey** to import a public key into an HSM.

Example : Import a public key

This command imports a public key from a file named `public.pem` with the label `importedPublicKey`. When the command succeeds, **importPubKey** returns a key handle for the imported key and a success message.

```
Command: importPubKey -l importedPublicKey -f public.pem

Cfm3CreatePublicKey returned: 0x00 : HSM Return: SUCCESS

Public Key Handle: 262230

        Cluster Error Status
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

Parameters

This command takes the following parameters.

`-h`

Displays command line help for the command.

Required: Yes

`-l`

Specifies the user-defined public key label.

Required: Yes

`-f`

Specifies the file name of the key to import.

Required: Yes

`-sess`

Designates the imported key as a session key.

Default: The imported key is held as a persistent (token) key in the cluster.

Required: No

-id

Specifies the ID of the key to be imported.

Default: No ID value.

Required: No

-min_srv

Specifies the minimum number of HSMs to which the imported key is synchronized before the value of the `-timeout` parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of `min_srv` to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-timeout

Specifies the number of seconds to wait for the key to sync across HSMs when the `min-serv` parameter is included. If no number is specified, the polling continues forever.

Default: No limit

Required: No

Related topics

- exportPubKey
- Generate Keys

Import a plaintext symmetric key using AWS CloudHSM KMU

Use the **imSymKey** command in the AWS CloudHSM key_mgmt_util tool to import a plaintext copy of a symmetric key from a file into the hardware security module (HSM). You can use it to import keys that you generate by any method outside of the HSM and keys that were exported from an HSM, such as the keys that the exSymKey, command writes to a file.

During the import process, **imSymKey** uses an AES key that you select (the *wrapping key*) to *wrap* (encrypt) and then *unwrap* (decrypt) the key to be imported. However, **imSymKey** works only on files that contain plaintext keys. To export and import encrypted keys, use the wrapKey and unWrapKey commands.

Also, the **imSymKey** command imports only symmetric keys. To import public keys, use importPubKey. To import private keys, use importPrivateKey or wrapKey.

> **Note**
>
> You cannot import a password-protected PEM key using a symmetric or private key.

Imported keys work very much like keys generated in the HSM. However, the value of the OBJ_ATTR_LOCAL attribute is zero, which indicates that it was not generated locally. You can use the following command to share a symmetric key as you import it. You can use the `shareKey` command in cloudhsm_mgmt_util to share the key after it is imported.

```
imSymKey -l aesShared -t 31 -f kms.key -w 3296 -u 5
```

After you import a key, be sure to mark or delete the key file. This command does not prevent you from importing the same key material multiple times. The result, multiple keys with distinct key handles and the same key material, make it difficult to track use of the key material and prevent it from exceeding its cryptographic limits.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
imSymKey -h

imSymKey -f <key-file>
         -w <wrapping-key-handle>
         -t <key-type>
         -l <label>
         [-id <key-ID>]
         [-sess]
```

```
          [-wk <wrapping-key-file> ]
          [-attest]
          [-min_srv <minimum-number-of-servers>]
          [-timeout <number-of-seconds> ]
          [-u <user-ids>]
```

Examples

These examples show how to use **imSymKey** to import symmetric keys into your HSMs.

Example : Import an AES symmetric key

This example uses **imSymKey** to import an AES symmetric key into the HSMs.

The first command uses OpenSSL to generate a random 256-bit AES symmetric key. It saves the key in the `aes256.key` file.

```
$  openssl rand -out aes256-forImport.key 32
```

The second command uses **imSymKey** to import the AES key from the `aes256.key` file into the HSMs. It uses key 20, an AES key in the HSM, as the wrapping key and it specifies a label of `imported`. Unlike the ID, the label does not need to be unique in the cluster. The value of the `-t` (type) parameter is 31, which represents AES.

The output shows that the key in the file was wrapped and unwrapped, then imported into the HSM, where it was assigned the key handle 262180.

```
Command:  imSymKey -f aes256.key -w 20 -t 31 -l imported

        Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Unwrapped.  Key Handle: 262180

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

The next command uses getAttribute to get the OBJ_ATTR_LOCAL attribute (attribute 355) of the newly imported key and writes it to the `attr_262180` file.

```
Command:  getAttribute -o 262180 -a 355 -out attributes/attr_262180
Attributes dumped into attributes/attr_262180_imported file

        Cfm3GetAttribute returned: 0x00 : HSM Return: SUCCESS
```

When you examine the attribute file, you can see that the value of the `OBJ_ATTR_LOCAL` attribute is zero, which indicates that the key material was not generated in the HSM.

```
$  cat attributes/attr_262180_local
OBJ_ATTR_LOCAL
0x00000000
```

Example : Move a symmetric key between clusters

This example shows how to use exSymKey and **imSymKey** to move a plaintext AES key between clusters. You might use a process like this one to create an AES wrapping that exists on the HSMs both clusters. Once the shared wrapping key is in place, you can use wrapKey and unWrapKey to move encrypted keys between the clusters.

The CU user who performs this operation must have permission to log in to the HSMs on both clusters.

The first command uses exSymKey to export key 14, a 32-bit AES key, from the cluster 1 into the `aes.key` file. It uses key 6, an AES key on the HSMs in cluster 1, as the wrapping key.

```
Command: exSymKey -k 14 -w 6 -out aes.key

        Cfm3WrapKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapHostKey returned: 0x00 : HSM Return: SUCCESS


Wrapped Symmetric Key written to file "aes.key"
```

The user then logs into key_mgmt_util in cluster 2 and runs an **imSymKey** command to import the key in the `aes.key` file into the HSMs in cluster 2. This command uses key 252152, an AES key on the HSMs in cluster 2, as the wrapping key.

Because the wrapping keys that exSymKey and **imSymKey** use wrap and immediately unwrap the target keys, the wrapping keys on the different clusters need not be the same.

The output shows that the key was successfully imported into cluster 2 and assigned a key handle of 21.

```
Command:  imSymKey -f aes.key -w 262152 -t 31 -l xcluster

        Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Unwrapped.  Key Handle: 21

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

To prove that key 14 of cluster 1 and key 21 in cluster 2 have the same key material, get the key check value (KCV) of each key. If the KCV values are the same, the key material is the same.

The following command uses getAttribute in cluster 1 to write the value of the KCV attribute (attribute 371) of key 14 to the `attr_14_kcv` file. Then, it uses a **cat** command to get the content of the `attr_14_kcv` file.

```
Command:  getAttribute -o 14 -a 371 -out attr_14_kcv
Attributes dumped into attr_14_kcv file

$  cat attr_14_kcv
OBJ_ATTR_KCV
0xc33cbd
```

This similar command uses getAttribute in cluster 2 to write the value of the KCV attribute (attribute 371) of key 21 to the `attr_21_kcv` file. Then, it uses a **cat** command to get the content of the `attr_21_kcv` file.

```
Command:  getAttribute -o 21 -a 371 -out attr_21_kcv
Attributes dumped into attr_21_kcv file
```

```
$  cat attr_21_kcv
OBJ_ATTR_KCV
0xc33cbd
```

The output shows that the KCV values of the two keys are the same, which proves that the key material is the same.

Because the same key material exists in the HSMs of both clusters, you can now share encrypted keys between the clusters without ever exposing the plaintext key. For example, you can use the `wrapKey` command with wrapping key 14 to export an encrypted key from cluster 1, and then use `unWrapKey` with wrapping key 21 to import the encrypted key into cluster 2.

Example : Import a session key

This command uses the `-sess` parameters of **imSymKey** to import a 192-bit Triple DES key that is valid only in the current session.

The command uses the `-f` parameter to specify he file that contains the key to import, the `-t` parameter to specify the key type, and the `-w` parameter to specify the wrapping key. It uses the `-l` parameter to specify a label that categorizes the key and the `-id` parameter to create a friendly, but unique, identifier for the key. It also uses the `-attest` parameter to verify the firmware that is importing the key.

The output shows that the key was successfully wrapped and unwrapped, imported into the HSM, and assigned the key handle 37. Also, the attestation check passed, which indicates that the firmware has not been tampered.

```
Command:  imSymKey -f 3des192.key -w 6 -t 21 -l temp -id test01 -sess -attest

        Cfm3WrapHostKey returned: 0x00 : HSM Return: SUCCESS

        Cfm3CreateUnwrapTemplate returned: 0x00 : HSM Return: SUCCESS

        Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Unwrapped.  Key Handle: 37

        Attestation Check : [PASS]

        Cluster Error Status
```

```
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Next, you can use the getAttribute or findKey commands to verify the attributes of the newly imported key. The following command uses **findKey** to verify that key 37 has the type, label, and ID specified by the command, and that it is a session key. A shown on line 5 of the output, **findKey** reports that the only key that matches all of the attributes is key 37.

```
Command:  findKey -t 21 -l temp -id test01 -sess 1
Total number of keys present 1

 number of keys matched from start index 0::0
37

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Parameters

-attest

Runs an integrity check that verifies that the firmware on which the cluster runs has not been tampered with.

Default: No attestation check.

Required: No

-f

Specifies the file that contains that key to import.

The file must contain a plaintext copy of an AES or Triple DES key of the specified length. RC4 and DES keys are not valid on FIPS-mode HSMs.

- **AES**: 16, 24 or 32 bytes
- **Triple DES (3DES)**: 24 bytes

Required: Yes

-h

Displays help for the command.

Required: Yes

-id

Specifies a user-defined identifier for the key. Type a string that is unique in the cluster. The default is an empty string.

Default: No ID value.

Required: No

-l

Specifies a user-defined label for the key. Type a string.

You can use any phrase that helps you to identify the key. Because the label does not have to be unique, you can use it to group and categorize keys.

Required: Yes

-min_srv

Specifies the minimum number of HSMs on which the key is synchronized before the value of the `-timeout` parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of `min_srv` to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-sess

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use setAttribute.

Default: The key is persistent.

Required: No

-timeout

Specifies how long (in seconds) the command waits for a key to be synchronized to the number of HSMs specified by the `min_srv` parameter.

This parameter is valid only when the `min_srv` parameter is also used in the command.

Default: No timeout. The command waits indefinitely and returns only when the key is synchronized to the minimum number of servers.

Required: No

-t

Specifies the type of the symmetric key. Enter the constant that represents the key type. For example, to create an AES key, enter `-t 31`.

Valid values:

- 21: Triple DES (3DES).
- 31: AES

Required: Yes

-u

Shares the key you are importing with specified users. This parameter gives other HSM crypto users (CUs) permission to use this key in cryptographic operations.

Type one ID or a comma-separated list of HSM user IDs, such as -u `5,6`. Do not include the HSM user ID of the current user. To find the an ID, you can use the listUsers command in the cloudhsm_mgmt_util command line tool or the listUsers command in the key_mgmt_util command line tool.

Required: No

-w

Specifies the key handle of the wrapping key. This parameter is required. To find key handles, use the findKey command.

A *wrapping key* is a key in the HSM that is used to encrypt ("wrap") and then decrypt ("unwrap) the key during the import process. Only AES keys can be used as wrapping keys.

You can use any AES key (of any size) as a wrapping key. Because the wrapping key wraps, and then immediately unwraps, the target key, you can use as session-only AES key as a wrapping key. To determine whether a key can be used as a wrapping key, use getAttribute to get the value of the `OBJ_ATTR_WRAP` attribute (262). To create a wrapping key, use genSymKey to create an AES key (type 31).

If you use the `-wk` parameter to specify an external wrapping key, the `-w` wrapping key is used to unwrap, but not to wrap, the key that is being imported.

> **Note**
>
> Key 4 is an unsupported internal key. We recommend that you use an AES key that you create and manage as the wrapping key.

Required: Yes

-wk

Use the AES key in the specified file to wrap the key that is being imported. Enter the path and name of a file that contains a plaintext AES key.

When you include this parameter. **imSymKey** uses the key in the `-wk` file to wrap the key being imported and it uses the key in the HSM that is specified by the `-w` parameter to unwrap it. The `-w` and `-wk` parameter values must resolve to the same plaintext key.

Default: Use the wrapping key on the HSM to unwrap.

Required: No

Related topics

- genSymKey
- exSymKey
- wrapKey
- unWrapKey

- exportPrivateKey
- exportPubKey

Insert a masked object using AWS CloudHSM KMU

Use the **insertMaskedObject** command in the AWS CloudHSM key_mgmt_util to insert a masked object from a file into a designated hardware security module (HSM). Masked objects are *cloned* objects that are extracted from an HSM by using the **extractMaskedObject** command. They can only be used after inserting them back into the original cluster. You can only insert a masked object into the same cluster from which it was generated, or a clone of that cluster. This includes any cloned versions of the original cluster generated by copying a backup across regions and using that backup to create a new cluster.

Masked objects are an efficient way to offload and synchronize keys, including nonextractable keys (that is, keys that have a `OBJ_ATTR_EXTRACTABLE` value of `0`). This way, keys can be securely synced across related clusters in different regions without the need to update the AWS CloudHSM configure file.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
insertMaskedObject -h

insertMaskedObject -f <filename>
                   [-min_srv <minimum-number-of-servers>]
                   [-timeout <number-of-seconds>]
```

Examples

This example shows how to use **insertMaskedObject** to insert a masked object file into an HSM.

Example : Insert a masked object

This command inserts a masked object into an HSM from a file named `maskedObj`. When the command succeeds, **insertMaskedObject** returns a key handle for the key decrypted from the masked object, and a success message.

```
Command: insertMaskedObject -f maskedObj
```

```
Cfm3InsertMaskedObject returned: 0x00 : HSM Return: SUCCESS
        New Key Handle: 262433

        Cluster Error Status
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

Parameters

This command takes the following parameters.

`-h`

Displays command line help for the command.

Required: Yes

`-f`

Specifies the file name of the masked object to insert.

Required: Yes

`-min_srv`

Specifies the minimum number of servers on which the inserted masked object is synchronized before the value of the `-timeout` parameter expires. If the object is not synchronized to the specified number of servers in the time allotted, it is not inserted.

Default: 1

Required: No

`-timeout`

Specifies the number of seconds to wait for the key to sync across servers when the `min-serv` parameter is included. If no number is specified, the polling continues forever.

Default: No limit

Required: No

Related topics

- extractMaskedObject
- syncKey
- Copying a Backup Across Regions
- Creating an AWS CloudHSM Cluster from a Previous Backup

Validate key file using AWS CloudHSM KMU

Use the **IsValidKeyHandlefile** command in the AWS CloudHSM key_mgmt_util to find out whether a key file contains a real private key or a fake RSA PEM key. A fake PEM file does not contain the actual private key material but instead references the private key in the HSM. Such a file can be used to establish SSL/TLS offloading from your web server to AWS CloudHSM. For more information, see SSL/TLS Offload on Linux using Tomcat or SSL/TLS Offload on Linux using NGINX or Apache.

> **Note**
>
> **IsValidKeyHandlefile** only works for RSA keys.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
IsValidKeyHandlefile -h

IsValidKeyHandlefile -f <rsa-private-key-file>
```

Examples

These examples show how to use **IsValidKeyHandlefile** to determine whether a given key file contains the real key material or fake PEM key material.

Example : Validate a real private key

This command confirms that the file called `privateKey.pem` contains real key material.

```
Command: IsValidKeyHandlefile -f privateKey.pem

Input key file has real private key
```

Example : Invalidate a fake PEM key

This command confirms that the file called `caviumKey.pem` contains fake PEM key material made from key handle 15.

```
Command: IsValidKeyHandlefile -f caviumKey.pem

Input file has invalid key handle: 15
```

Parameters

This command takes the following parameters.

-h

Displays command line help for the command.

Required: Yes

-f

Specifies the RSA private key file to be checked for valid key material.

Required: Yes

Related topics

- getCaviumPrivKey
- SSL/TLS Offload on Linux using Tomcat
- SSL/TLS Offload on Linux using NGINX or Apache

List the attributes of an AWS CloudHSM key using KMU

Use the **listAttributes** command in the AWS CloudHSM key_mgmt_util to list the attributes of an AWS CloudHSM key and the constants that represent them. You use these constants to identify the attributes in getAttribute and setAttribute commands. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

This command has no parameters.

```
listAttributes
```

Example

This command lists the key attributes that you can get and change in key_mgmt_util and the constants that represent them. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

To represent all attributes in the getAttribute command in key_mgmt_util, use 512.

```
Command: listAttributes

Following are the possible attribute values for getAttributes:

      OBJ_ATTR_CLASS                  = 0
      OBJ_ATTR_TOKEN                  = 1
      OBJ_ATTR_PRIVATE                = 2
      OBJ_ATTR_LABEL                  = 3
      OBJ_ATTR_KEY_TYPE               = 256
      OBJ_ATTR_ENCRYPT                = 260
      OBJ_ATTR_DECRYPT                = 261
      OBJ_ATTR_WRAP                   = 262
      OBJ_ATTR_UNWRAP                 = 263
      OBJ_ATTR_SIGN                   = 264
      OBJ_ATTR_VERIFY                 = 266
      OBJ_ATTR_LOCAL                  = 355
      OBJ_ATTR_MODULUS                = 288
      OBJ_ATTR_MODULUS_BITS           = 289
      OBJ_ATTR_PUBLIC_EXPONENT        = 290
      OBJ_ATTR_VALUE_LEN              = 353
      OBJ_ATTR_EXTRACTABLE            = 354
      OBJ_ATTR_KCV                    = 371
```

Related topics

- listAttributes in cloudhsm_mgmt_util
- getAttribute
- setAttribute
- Key Attribute Reference

List all AWS CloudHSM users using KMU

Use the **listUsers** command in the AWS CloudHSM key_mgmt_util to get the users in the hardware security modules (HSM), along with their user type and other attributes.

In key_mgmt_util, listUsers returns output that represents all HSMs in the cluster, even if they are not consistent. To get information about the users in each HSM, use the listUsers command in cloudhsm_mgmt_util.

The user commands in key_mgmt_util, **listUsers** and getKeyInfo, are read-only commands that crypto users (CUs) have permission to run. The remaining user management commands are part of cloudhsm_mgmt_util. They are run by crypto officers (CO) who have user management permissions.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
listUsers

listUsers -h
```

Example

This command lists the users of HSMs in the cluster and their attributes. You can use the `User ID` attribute to identify users in other commands, such as findKey, getAttribute, and getKeyInfo.

```
Command:  listUsers

        Number Of Users found 4
```

```
      Index        User ID     User Type        User Name             MofnPubKey    
LoginFailureCnt         2FA
        1                1     PCO              admin                       NO
   0               NO
        2                2     AU               app_user                    NO
   0               NO
        3                3     CU               alice                       YES
   0               NO
        4                4     CU               bob                         NO
   0               NO
        5                5     CU               trent                       YES
   0               NO

        Cfm3ListUsers returned: 0x00 : HSM Return: SUCCESS
```

The output includes the following user attributes:

- **User ID**: Identifies the user in key_mgmt_util and cloudhsm_mgmt_util commands.
- User type: Determines the operations that the user can perform on the HSM.
- **User Name**: Displays the user-defined friendly name for the user.
- **MofnPubKey**: Indicates whether the user has registered a key pair for signing quorum authentication tokens.
- **LoginFailureCnt**: Indicates the number of times the user has unsuccessfully logged in.
- **2FA**: Indicates that the user has enabled multi-factor authentication.

Parameters

-h

Displays help for the command.

Required: Yes

Related topics

- listUsers in cloudhsm_mgmt_util
- findKey

- getAttribute
- getKeyInfo

Log in and out of an HSM using AWS CloudHSM KMU

Use the **loginHSM** and **logoutHSM** commands in the AWS CloudHSM key_mgmt_util to log in and out of the hardware security modules (HSM) in a cluster. Once logged in to the HSMs, you can use key_mgmt_util to perform a variety of key management operations, including public and private key generation, synchronization, and wrapping.

Before you run any key_mgmt_util command, you must start key_mgmt_util. In order to manage keys with key_mgmt_util, you must log in to the HSMs as a crypto user (CU).

> **Note**
>
> If you exceed five incorrect login attempts, your account is locked out. If you created your cluster before February 2018, your account is locked out after 20 incorrect login attempts. To unlock the account, a cryptographic officer (CO) must reset your password using the changePswd command in cloudhsm_mgmt_util.
> If you have more than one HSM in your cluster, you may be allowed additional incorrect login attempts before your account is locked out. This is because the CloudHSM client balances load across various HSMs. Therefore, the login attempt may not begin on the same HSM each time. If you are testing this functionality, we recommend you do so on a cluster with only one active HSM.

Syntax

```
loginHSM -h

loginHSM -u <user type>
         { -p | -hpswd } <password>
         -s <username>
```

Example

This example shows how to log in and out of the HSMs in a cluster with the `loginHSM` and `logoutHSM` commands.

Example : Log in to the HSMs

This command logs you into the HSMs as a crypto user (CU) with the username `example_user` and password `aws`. The output shows that you have logged into all HSMs in the cluster.

```
Command:  loginHSM -u CU -s example_user -p aws

Cfm3LoginHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : Log in with a hidden password

This command is the same as the example above, except this time you specify that the system should hide the password.

```
Command:  loginHSM -u CU -s example_user -hpswd
```

The system prompts you for your password. You enter the password, the system hides the password, and the output shows that the command was successful and that the you have connected to the HSMs.

```
Enter password:

Cfm3LoginHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS

Command:
```

Example : Log out of the HSMs

This command logs you out of the HSMs. The output shows that you have logged out of all HSMs in the cluster.

```
Command: logoutHSM
```

```
Cfm3LogoutHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Parameters

-h

Displays help for this command.

-u

Specifies the login user type. In order to use key_mgmt_util, you must log in as a CU.

Required: Yes

-s

Specifies the login username.

Required: Yes

{ -p | -hpswd }

Specify the login password with `-p`. The password appears in plaintext when you type it. To hide your password, use the optional `-hpswd` parameter instead of `-p` and follow the prompt.

Required: Yes

Related topics

- exit

Set the attributes of AWS CloudHSM keys using KMU

Use the **setAttribute** command in the AWS CloudHSM key_mgmt_util to convert a key that is valid only in the current session to a persistent key that exists until you delete it. It does this by changing the value of the token attribute of the key (`OBJ_ATTR_TOKEN`) from false (`0`) to true (`1`). You can only change the attributes of keys that you own.

You can also use the **setAttribute** command in cloudhsm_mgmt_util to change the label, wrap, unwrap, encrypt, and decrypt attributes.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
setAttribute -h

setAttribute -o <object handle>
             -a 1
```

Example

This example shows how to convert a session key to a persistent key.

The first command uses the `-sess` parameter of genSymKey to create a 192-bit AES key that is valid only in the current session. The output shows that the key handle of the new session key is 262154.

```
Command: genSymKey -t 31 -s 24 -l tmpAES -sess

        Cfm3GenerateSymmetricKey returned: 0x00 : HSM Return: SUCCESS

        Symmetric Key Created.  Key Handle: 262154

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

This command uses findKey to find the session keys in the current session. The output verifies that key 262154 is a session key.

```
Command:  findKey -sess 1

Total number of keys present 1

 number of keys matched from start index 0::0
262154

        Cluster Error Status
```

```
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

This command uses **setAttribute** to convert key 262154 from a session key to a persistent key. To do so, it changes the value of the token attribute (OBJ_ATTR_TOKEN) of the key from 0 (false) to 1 (true). For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

The command uses the -o parameter to specify the key handle (262154) and the -a parameter to specify the constant that represents the token attribute (1). When you run the command, it prompts you for a value for the token attribute. The only valid value is 1 (true); the value for a persistent key.

```
Command: setAttribute -o 262154 -a 1
         This attribute is defined as a boolean value.
          Enter the boolean attribute value (0 or 1):1

        Cfm3SetAttribute returned: 0x00 : HSM Return: SUCCESS

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

To confirm that key 262154 is now persistent, this command uses **findKey** to search for session keys (-sess 1) and persistent keys (-sess 0). This time, the command does not find any session keys, but it returns 262154 in the list of persistent keys.

```
Command: findKey -sess 1

Total number of keys present 0

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

```
Command: findKey -sess 0

Total number of keys present 5

 number of keys matched from start index 0::4
6, 7, 524296, 9, 262154

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

Parameters

-h

Displays help for the command.

Required: Yes

-o

Specifies the key handle of the target key. You can specify only one key in each command. To get the key handle of a key, use findKey.

Required: Yes

-a

Specifies the constant that represents the attribute that you want to change. The only valid value is 1, which represents the token attribute, `OBJ_ATTR_TOKEN`.

To get the attributes and their integer values, use listAttributes.

Required: Yes

Related topics

- setAttribute in cloudhsm_mgmt_util
- getAttribute
- listAttributes
- Key Attribute Reference

Generate a signature using AWS CloudHSM KMU

Use the **sign** command in the AWS CloudHSM key_mgmt_util to use a chosen private key to generate a signature for a file.

In order to use **sign**, you must first have a private key in your HSM. You can generate a private key with the **genSymKey**, **genRSAKeyPair**, or **genECCKeyPair** commands. You can also import one with the **importPrivateKey** command. For more information, see Generate Keys.

The **sign** command uses a user-designated signing mechanism, represented by an integer, to sign a message file. For a list of possible signing mechanisms, see Parameters.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
sign -h

sign -f <file name>
     -k <private key handle>
     -m <signature mechanism>
     -out <signed file name>
```

Example

This example shows how to use **sign** to sign a file.

Example : Sign a file

This command signs a file named `messageFile` with a private key with handle `266309`. It uses the `SHA256_RSA_PKCS (1)` signing mechanism and saves the resulting signed file as `signedFile`.

```
Command: sign -f messageFile -k 266309 -m 1 -out signedFile

Cfm3Sign returned: 0x00 : HSM Return: SUCCESS

signature is written to file signedFile

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

```
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

Parameters

This command takes the following parameters.

-f

The name of the file to sign.

Required: Yes

-k

The handle of the private key to be used for signing.

Required: Yes

-m

An integer that represents the signing mechanism to be used for signing. The possible mechanisms correspond to the follow integers:

Signing Mechanism	Corresponding Integer
SHA1_RSA_PKCS	0
SHA256_RSA_PKCS	1
SHA384_RSA_PKCS	2
SHA512_RSA_PKCS	3
SHA224_RSA_PKCS	4
SHA1_RSA_PKCS_PSS	5
SHA256_RSA_PKCS_PSS	6
SHA384_RSA_PKCS_PSS	7
SHA512_RSA_PKCS_PSS	8

Signing Mechanism	Corresponding Integer
SHA224_RSA_PKCS_PSS	9
ECDSA_SHA1	15
ECDSA_SHA224	16
ECDSA_SHA256	17
ECDSA_SHA384	18
ECDSA_SHA512	19

Required: Yes

-out

The name of the file to which the signed file will be saved.

Required: Yes

Related topics

- verify
- importPrivateKey
- genRSAKeyPair
- genECCKeyPair
- genSymKey
- Generate Keys

Unwrap an AWS CloudHSM key using KMU

Use the **unWrapKey** command in the AWS CloudHSM key_mgmt_util tool to import a wrapped (encrypted) symmetric or private key from a file into the HSM. It is designed to import encrypted keys that were wrapped by the wrapKey command in key_mgmt_util, but it can also be used to unwrap keys that were wrapped with other tools. However, in those situations, we recommend using the PKCS#11 or JCE software libraries to unwrap the key.

Imported keys work like keys generated by AWS CloudHSM. However, the value of their OBJ_ATTR_LOCAL attribute is zero, which indicates that they were not generated locally.

After you import a key, ensure that you mark or delete the key file. This command does not prevent you from importing the same key material multiple times. The results—multiple keys with distinct key handles and the same key material—make it difficult to track use of the key materials and prevent them from exceeding their cryptographic limits.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
unWrapKey -h

unWrapKey -f <key-file-name>
          -w <wrapping-key-handle>
          [-sess]
          [-min_srv <minimum-number-of-HSMs>]
          [-timeout <number-of-seconds>]
          [-aad <additional authenticated data filename>]
          [-tag_size <tag size>]
          [-iv_file <IV file>]
          [-attest]
          [-m <wrapping-mechanism>]
          [-t <hash-type>]
          [-nex]
          [-u <user id list>]
          [-m_value <number of users needed for approval>]
          [-noheader]
          [-l <key-label>]
          [-id <key-id>]
          [-kt <key-type>]
          [-kc <key-class>]
          [-i <unwrapping-IV>]
```

Example

These examples show how to use **unWrapKey** to import a wrapped key from a file into the HSMs. In the first example, we unwrap a key that was wrapped with the **wrapKey** key_mgmt_util command, and thus has a header. In the second example, we unwrap a key that was wrapped outside of key_mgmt_util, and thus does not have a header.

Example : Unwrap a key (with header)

This command imports a wrapped copy of a 3DES symmetric key into an HSM. The key is unwrapped with an AES key with label 6, which is cryptographically identical to the one that was used to wrap the 3DES key. The output shows that the key in the file was unwrapped and imported, and that the imported key's handle is 29.

```
        Command:  unWrapKey -f 3DES.key -w 6 -m 4

        Cfm3UnWrapKey returned: 0x00 : HSM Return: SUCCESS

        Key Unwrapped.  Key Handle: 29

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Example : Unwrap a key (no header)

This command imports a wrapped copy of a 3DES symmetric key into an HSM. The key is unwrapped with an AES key with label 6, which is cryptographically identical to the one that was used to wrap the 3DES key. As this 3DES key was not wrapped with key_mgmt_util, the `noheader` parameter is specified, along with its required accompanying parameters: a key label (`unwrapped3DES`), key class (4), and key type (21). The output shows that the key in the file was unwrapped and imported, and that the imported key's handle is 8.

```
Command: unWrapKey -f 3DES.key -w 6 -noheader -l unwrapped3DES -kc 4 -kt 21 -m 4

      Cfm3CreateUnwrapTemplate2 returned: 0x00 : HSM Return: SUCCESS
       Cfm2UnWrapWithTemplate3 returned: 0x00 : HSM Return: SUCCESS

       Key Unwrapped. Key Handle: 8

       Cluster Error Status
       Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
       Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
```

Parameters

-h

Displays help for the command.

Required: Yes

-f

The path and name of the file that contains the wrapped key.

Required: Yes

-w

Specifies the wrapping key. Enter the key handle of an AES key or RSA key on the HSM. This parameter is required. To find key handles, use the findKey command.

To create a wrapping key, use genSymKey to generate an AES key (type 31) or genRSAKeyPair to generate an RSA key pair (type 0). If you are using an RSA key pair, be sure to wrap the key with one of the keys, and unwrap it with the other. To verify that a key can be used as a wrapping key, use getAttribute to get the value of the `OBJ_ATTR_WRAP` attribute, which is represented by constant `262`.

Required: Yes

-sess

Creates a key that exists only in the current session. The key cannot be recovered after the session ends.

Use this parameter when you need a key only briefly, such as a wrapping key that encrypts, and then quickly decrypts, another key. Do not use a session key to encrypt data that you might need to decrypt after the session ends.

To change a session key to a persistent (token) key, use setAttribute.

Default: The key is persistent.

Required: No

-min_srv

Specifies the minimum number of HSMs on which the key is synchronized before the value of the `-timeout` parameter expires. If the key is not synchronized to the specified number of servers in the time allotted, it is not created.

AWS CloudHSM automatically synchronizes every key to every HSM in the cluster. To speed up your process, set the value of `min_srv` to less than the number of HSMs in the cluster and set a low timeout value. Note, however, that some requests might not generate a key.

Default: 1

Required: No

-timeout

Specifies how long (in seconds) the command waits for a key to be synchronized to the number of HSMs specified by the `min_srv` parameter.

This parameter is valid only when the `min_srv` parameter is also used in the command.

Default: No timeout. The command waits indefinitely and returns only when the key is synchronized to the minimum number of servers.

Required: No

-attest

Runs an integrity check that verifies that the firmware on which the cluster runs has not been tampered with.

Default: No attestation check.

Required: No

-nex

Makes the key nonextractable. The key that is generated cannot be [exported from the HSM](#).

Default: The key is extractable.

Required: No

-m

The value representing the wrapping mechanism. CloudHSM supports the following mechanisms:

Mechanism	Value
AES_KEY_WRAP_PAD_PKCS5	4
NIST_AES_WRAP_NO_PAD	5
NIST_AES_WRAP_PAD	6
RSA_AES	7
RSA_OAEP (for maximum data size, see the note later in this section)	8
AES_GCM	10
CLOUDHSM_AES_GCM	11
RSA_PKCS (for maximum data size, see the note later in this section). See note 1 below for an upcoming change.	12

Required: Yes

Note

When using the RSA_OAEP wrapping mechanism, the maximum key size that you can wrap is determined by the modulus of the RSA key and the length of the specified hash as follows: Maximum key size = modulusLengthInBytes-(2*hashLengthInBytes)-2.

When using the RSA_PKCS wrapping mechanism, the maximum key size that you can wrap is determined by the modulus of the RSA key as follows: Maximum key size = (modulusLengthInBytes -11).

-t

Hash algorithm	Value
SHA1	2

Hash algorithm	Value
SHA256	3
SHA384	4
SHA512	5
SHA224 (valid for RSA_AES and RSA_OAEP mechanisms)	6

Required: No

-noheader

If you are unwrapping a key that was wrapped outside of key_mgmt_util, you must specify this parameter and all other associated parameters.

Required: No

> **Note**
>
> If you specify this parameter, you **must** also specify the following `-noheader` parameters:

- **-l**

  Specifies the label to be added to the unwrapped key.

  Required: Yes

- **-kc**

  Specifies the class of the key to be unwrapped. The following are acceptable values:

  3 = private key from a public-private key pair

  4 = secret (symmetric) key

  Required: Yes

- **-kt**

  Specifies the type of key to be unwrapped. The following are acceptable values:

  `0` = `RSA`

  `1` = `DSA`

  `3` = `ECC`

  `16` = `GENERIC_SECRET`

  `21` = `DES3`

  `31` = `AES`

  Required: Yes

You can also **optionally** specify the following `-noheader` parameters:

- **-id**

  The ID to be added to the unwrapped key.

  Required: No

- **-i**

  The unwrapping initialization vector (IV) to be used.

  Required: No

[1] In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.

Related topics

- wrapKey
- exSymKey
- imSymKey

Verify the signature of a file using AWS CloudHSMKMU

Use the **verify** command in the AWS CloudHSM key_mgmt_util to confirm whether or not a file has been signed by a given key. To do so, the **verify** command compares a signed file against a source file and analyzes whether they are cryptographically related based on a given public key and signing mechanism. Files can be signed in AWS CloudHSM with the **sign** operation.

Signing mechanisms are represented by the integers listed in the parameters section.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
verify -h

verify -f <message-file>
       -s <signature-file>
       -k <public-key-handle>
       -m <signature-mechanism>
```

Example

These examples show how to use **verify** to check whether a certain public key was used to sign a given file.

Example : Verify a file signature

This command attempts to verify whether a file named `hardwarCert.crt` was signed by public key 262276 using the SHA256_RSA_PKCS signing mechanism to produce the `hardwareCertSigned` signed file. Because the given parameters represent a true signing relationship, the command returns a success message.

```
Command: verify -f hardwareCert.crt -s hardwareCertSigned -k 262276 -m 1

Signature verification successful

Cfm3Verify returned: 0x00 : HSM Return: SUCCESS
```

Example : Prove false signing relationship

This command verifies whether a file named `hardwareCert.crt` was signed by public key `262276` using the `SHA256_RSA_PKCS` signing mechanism to produce the `userCertSigned` signed file. Because the given parameters do not make up a true signing relationship, the command returns an error message.

```
Command: verify -f hardwarecert.crt -s usercertsigned -k 262276 -m 1
Cfm3Verify returned: 0x1b

CSP Error: ERR_BAD_PKCS_DATA
```

Parameters

This command takes the following parameters.

-f

The name of the origin message file.

Required: Yes

-s

The name of the signed file.

Require: Yes

-k

The handle of the public key that is thought to be used to sign the file.

Required: Yes

-m

An integer that represents the proposed signing mechanism that is used to sign the file. The possible mechanisms correspond to the follow integers:

Signing Mechanism	Corresponding Integer
SHA1_RSA_PKCS	0

Signing Mechanism	Corresponding Integer
SHA256_RSA_PKCS	1
SHA384_RSA_PKCS	2
SHA512_RSA_PKCS	3
SHA224_RSA_PKCS	4
SHA1_RSA_PKCS_PSS	5
SHA256_RSA_PKCS_PSS	6
SHA384_RSA_PKCS_PSS	7
SHA512_RSA_PKCS_PSS	8
SHA224_RSA_PKCS_PSS	9
ECDSA_SHA1	15
ECDSA_SHA224	16
ECDSA_SHA256	17
ECDSA_SHA384	18
ECDSA_SHA512	19

Required: Yes

Related topics

- sign
- getCert
- Generate Keys

Export an AWS CloudHSM key using KMU

Use the **wrapKey** command in the AWS CloudHSM key_mgmt_util to export an encrypted copy of a symmetric or private key from the hardware security module (HSM) to a file. When you run **wrapKey**, you specify the key to export, a key on the HSM to encrypt (wrap) the key that you want to export, and the output file.

The `wrapKey` command writes the encrypted key to a file that you specify, but it does not remove the key from the HSM or prevent you from using it in cryptographic operations. You can export the same key multiple times.

Only the owner of a key, that is, the crypto user (CU) who created the key, can export it. Users who share the key can use it in cryptographic operations, but they cannot export it.

To import the encrypted key back into the HSM, use unWrapKey. To export a plaintext key from an HSM, use exSymKey or exportPrivateKey as appropriate. The aesWrapUnwrap command cannot decrypt (unwrap) keys that **wrapKey** encrypts.

Before you run any key_mgmt_util command, you must start key_mgmt_util and log in to the HSM as a crypto user (CU).

Syntax

```
wrapKey -h

wrapKey -k <exported-key-handle>
        -w <wrapping-key-handle>
        -out <output-file>
        [-m <wrapping-mechanism>]
        [-aad <additional authenticated data filename>]
        [-t <hash-type>]
        [-noheader]
        [-i <wrapping IV>]
        [-iv_file <IV file>]
        [-tag_size <num_tag_bytes>>]
```

Example

Example

This command exports a 192-bit Triple DES (3DES) symmetric key (key handle 7). It uses a 256-bit AES key in the HSM (key handle 14) to wrap key 7. Then, it writes the encrypted 3DES key to the `3DES-encrypted.key` file.

The output shows that key 7 (the 3DES key) was successfully wrapped and written to the specified file. The encrypted key is 307 bytes long.

```
        Command:  wrapKey -k 7 -w 14 -out 3DES-encrypted.key -m 4

        Key Wrapped.

        Wrapped Key written to file "3DES-encrypted.key length 307

        Cfm2WrapKey returned: 0x00 : HSM Return: SUCCESS
```

Parameters

-h

Displays help for the command.

Required: Yes

-k

The key handle of the key that you want to export. Enter the key handle of a symmetric or private key that you own. To find key handles, use the findKey command.

To verify that a key can be exported, use the getAttribute command to get the value of the `OBJ_ATTR_EXTRACTABLE` attribute, which is represented by constant `354`. For help interpreting the key attributes, see the AWS CloudHSM key attribute reference for KMU.

You can export only those keys that you own. To find the owner of a key, use the getKeyInfo command.

Required: Yes

-w

Specifies the wrapping key. Enter the key handle of an AES key or RSA key on the HSM. This parameter is required. To find key handles, use the findKey command.

To create a wrapping key, use genSymKey to generate an AES key (type 31) or genRSAKeyPair to generate an RSA key pair (type 0). If you are using an RSA key pair, be sure to wrap the key with one of the keys, and unwrap it with the other. To verify that a key can be used as a wrapping key, use getAttribute to get the value of the `OBJ_ATTR_WRAP` attribute, which is represented by constant `262`.

Required: Yes

-out

The path and name of the output file. When the command succeeds, this file contains an encrypted copy of the exported key. If the file already exists, the command overwrites it without warning.

Required: Yes

-m

The value representing the wrapping mechanism. CloudHSM supports the following mechanisms:

Mechanism	Value
`AES_KEY_WRAP_PAD_PKCS5`	4
`NIST_AES_WRAP_NO_PAD`	5
`NIST_AES_WRAP_PAD`	6
`RSA_AES`	7
`RSA_OAEP` (for maximum data size, see the note later in this section)	8
`AES_GCM`	10
`CLOUDHSM_AES_GCM`	11

Mechanism	Value
RSA_PKCS (for maximum data size, see the note later in this section). See note 1 below for an upcoming change.	12

Required: Yes

> **Note**
>
> When using the RSA_OAEP wrapping mechanism, the maximum key size that you can wrap is determined by the modulus of the RSA key and the length of the specified hash as follows: Maximum key size = (modulusLengthInBytes-2*hashLengthInBytes-2).
> When using the RSA_PKCS wrapping mechanism, the maximum key size that you can wrap is determined by the modulus of the RSA key as follows: Maximum key size = (modulusLengthInBytes -11).

-t

The value representing the hash algorithm. CloudHSM supports the following algorithms:

Hash algorithm	Value
SHA1	2
SHA256	3
SHA384	4
SHA512	5
SHA224 (valid for RSA_AES and RSA_OAEP mechanisms)	6

Required: No

-aad

The file name containing AAD.

> **Note**
>
> Valid only for AES_GCM and CLOUDHSM_AES_GCM mechanisms.

Required: No

-noheader

Omits the header that specifies CloudHSM-specific key attributes. Use this parameter *only* if you want to unwrap the key with tools outside of key_mgmt_util.

Required: No

-i

The initialization vector (IV) (hex value).

> **Note**
>
> Valid only when passed with the `-noheader` parameter for `CLOUDHSM_AES_KEY_WRAP`, and `NIST_AES_WRAP` mechanisms.

Required: No

-iv_file

The file in which you want to write the IV value obtained in response.

> **Note**
>
> Valid only when passed with the `-noheader` parameter for `AES_GCM` mechanism.

Required: No

-tag_size

The size of tag to be saved along with wrapped blob.

> **Note**
>
> Valid only when passed with the `-noheader` parameter for `AES_GCM` and `CLOUDHSM_AES_GCM` mechanisms. Minimum tag size is eight.

Required: No

[1] In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.

Related topics

- exSymKey
- imSymKey
- unWrapKey

AWS CloudHSM key attribute reference for KMU

The AWS CloudHSM key_mgmt_util commands use constants to represent the attributes of keys in a hardware security module (HSM). This topic can help you to identify the attributes, find the constants that represent them in commands, and understand their values.

You set the attributes of a key when you create it. To change the token attribute, which indicates whether a key is persistent or exists only in the session, use the setAttribute command in key_mgmt_util. To change the label, wrap, unwrap, encrypt, or decrypt attributes, use the `setAttribute` command in cloudhsm_mgmt_util.

To get a list of attributes and their constants, use listAttributes. To get the attribute values for a key, use getAttribute.

The following table lists the key attributes, their constants, and their valid values.

Attribute	Constant	Values
OBJ_ATTR_ALL	512	Represents all attributes.

Attribute	Constant	Values
OBJ_ATTR_ALWAYS_SE NSITIVE	357	**0**: False. **1**: True.
OBJ_ATTR_CLASS	0	**2**: Public key in a public–pr ivate key pair. **3**: Private key in a public–pr ivate key pair. **4**: Secret (symmetric) key.
OBJ_ATTR_DECRYPT	261	**0**: False. **1**: True. The key can be used to decrypt data.
OBJ_ATTR_DERIVE	268	**0**: False. **1**: True. The function derives the key.
OBJ_ATTR_DESTROYABLE	370	**0**: False. **1**: True.
OBJ_ATTR_ENCRYPT	260	**0**: False. **1**: True. The key can be used to encrypt data.
OBJ_ATTR_EXTRACTABLE	354	**0**: False. **1**: True. The key can be exported from the HSMs.
OBJ_ATTR_ID	258	User-defined string. Must be unique in the cluster. The default is an empty string.

Attribute	Constant	Values
OBJ_ATTR_KCV	371	Key check value of the key. For more information, see Additional Details.
OBJ_ATTR_KEY_TYPE	256	**0**: RSA. **1**: DSA. **3**: EC. **16**: Generic secret. **18**: RC4. **21**: Triple DES (3DES). **31**: AES.
OBJ_ATTR_LABEL	3	User-defined string. It does not have to be unique in the cluster.
OBJ_ATTR_LOCAL	355	**0**. False. The key was imported into the HSMs. **1**: True.
OBJ_ATTR_MODULUS	288	The modulus that was used to generate an RSA key pair. For EC keys, this value represent s the DER-encoding of ANSI X9.62 ECPoint value "Q" in a hexadecimal format. For other key types, this attribute does not exist.

Attribute	Constant	Values
OBJ_ATTR_MODULUS_BITS	289	The length of the modulus used to generate an RSA key pair. For EC keys this represents the ID of the elliptic curve used to generate the key. For other key types, this attribute does not exist.
OBJ_ATTR_NEVER_EXTRACTABLE	356	**0**: False. **1**: True. The key cannot be exported from the HSMs.
OBJ_ATTR_PUBLIC_EXPONENT	290	The public exponent used to generate an RSA key pair. For other key types, this attribute does not exist.
OBJ_ATTR_PRIVATE	2	**0**: False. **1**: True. This attribute indicates whether unauthenticated users can list the attributes of the key. Since the CloudHSM PKCS#11 provider currently does not support public sessions, all keys (including public keys in a public-private key pair) have this attribute set to 1.

Attribute	Constant	Values
OBJ_ATTR_SENSITIVE	259	**0**: False. Public key in a public–private key pair. **1**: True.
OBJ_ATTR_SIGN	264	**0**: False. **1**: True. The key can be used for signing (private keys).
OBJ_ATTR_TOKEN	1	**0**: False. Session key. **1**: True. Persistent key.
OBJ_ATTR_TRUSTED	134	**0**: False. **1**: True.
OBJ_ATTR_UNWRAP	263	**0**: False. **1**: True. The key can be used to decrypt keys.
OBJ_ATTR_UNWRAP_TE MPLATE	1073742354	Values should use the attribute template applied to any key unwrapped using this wrapping key.
OBJ_ATTR_VALUE_LEN	353	Key length in bytes.
OBJ_ATTR_VERIFY	266	**0**: False. **1**: True. The key can be used for verification (public keys).
OBJ_ATTR_WRAP	262	**0**: False. **1**: True. The key can be used to encrypt keys.

Attribute	Constant	Values
OBJ_ATTR_WRAP_TEMPLATE	1073742353	Values should use the attribute template to match the key wrapped using this wrapping key.
OBJ_ATTR_WRAP_WITH _TRUSTED	528	**0**: False. **1**: True.

Additional Details

Key check value (KCV)

The *key check value* (KCV) is a 3-byte hash or checksum of a key that is generated when the HSM imports or generates a key. You can also calculate a KCV outside of the HSM, such as after you export a key. You can then compare the KCV values to confirm the identity and integrity of the key. To get the KCV of a key, use getAttribute.

AWS CloudHSM uses the following standard method to generate a key check value:

- **Symmetric keys**: First 3 bytes of the result of encrypting a zero-block with the key.
- **Asymmetric key pairs**: First 3 bytes of the SHA-1 hash of the public key.
- **HMAC keys**: KCV for HMAC keys is not supported at this time.

Offload operations with AWS CloudHSM Client SDKs

Use a Client SDK to offload cryptographic operations from platform or language-based applications to hardware security modules (HSMs).

AWS CloudHSM offers two major versions, and Client SDK 5 is the latest. It offers a variety of advantages over Client SDK 3 (the previous series). For more information, see Benefits of Client SDK 5. For information about platform support, see AWS CloudHSM Client SDK 5 supported platforms.

The following topics describe how to work with AWS CloudHSM Client SDKs.

AWS CloudHSM supports the following components:

the section called "PKCS #11 library"

PKCS #11 is a standard for performing cryptographic operations on hardware security modules (HSMs). AWS CloudHSM offers implementations of the PKCS #11 library that are compliant with PKCS #11 version 2.40.

the section called "OpenSSL Dynamic Engine"

The AWS CloudHSM OpenSSL Dynamic Engine allows you to offload cryptographic operations to your CloudHSM cluster through the OpenSSL API.

the section called "JCE provider"

The AWS CloudHSM JCE provider is compliant with the Java Cryptographic Architecture (JCA). The provider allows you to perform cryptographic operations on the HSM.

the section called "Key storage provider (KSP)"

The AWS CloudHSM client for Windows includes CNG and KSP providers. Currently, only Client SDK 3 supports CNG and KSP providers.

Topics

- Using Client SDK 5 to work with AWS CloudHSM
- Using previous SDK version to work with AWS CloudHSM

Check your AWS CloudHSM Client SDK version

Use the following commands to verify the version of Client SDK that you're using with AWS CloudHSM.

Amazon Linux

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

Amazon Linux 2

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

CentOS 6

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

CentOS 7

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

CentOS 8

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

RHEL 6

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

RHEL 7

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

RHEL 8

Use the following command:

```
rpm -qa | grep ^cloudhsm
```

Ubuntu 16.04 LTS

Use the following command:

```
apt list --installed | grep ^cloudhsm
```

Ubuntu 18.04 LTS

Use the following command:

```
apt list --installed | grep ^cloudhsm
```

Ubuntu 20.04 LTS

Use the following command:

```
apt list --installed | grep ^cloudhsm
```

Windows Server

Use the following command:

```
wmic product get name,version
```

Compare AWS CloudHSM Client SDK component support

In addition to the command-line tools, Client SDK 3 contains components that enable off-loading cryptographic operations to the HSM from various platform or language-based applications. Client SDK 5 has parity with Client SDK 3, except it does not yet support CNG and KSP providers. The following table compares component availability in Client SDK 3 and Client SDK 5.

Component	Client SDK 5	Client SDK 3
PKCS #11 library	Yes	Yes
JCE provider	Yes	Yes
OpenSSL Dynamic Engine	Yes	Yes
Key Storage Provider (KSP)	Yes	Yes
CloudHSM Management Utility (CMU)[1]	Yes	Yes
Key Management Utility (KMU)[1]	Yes	Yes
Configure tool	Yes	Yes

[1] CMU and KMU components are included in CloudHSM CLI with Client SDK 5.

The following sections describe the components.

PKCS #11 library

PKCS #11 is a standard for performing cryptographic operations on hardware security modules (HSMs). AWS CloudHSM offers implementations of the PKCS #11 library that are compliant with PKCS #11 version 2.40.

- For Client SDK 3, the PKCS #11 library is a Linux only component that matches Linux base support. For more information, see the section called "Linux support for Client SDK 3".
- For Client SDK 5, the PKCS #11 library is a cross-platform component that matches Linux and Windows Client SDK 5 base support. For more information, see the section called "Linux support for Client SDK 5" and the section called "Windows support for Client SDK 5".

CloudHSM Management Utility (CMU)

The CloudHSM Management Utility (CMU) command line tool helps crypto officers manage users in the HSMs. It includes tools that create, delete, and list users, and change user passwords. For more information, see AWS CloudHSM Management Utility (CMU).

Key Management Utility (KMU)

The Key Management Utility (KMU) is a command line tool that helps crypto users (CU) manage keys on the hardware security modules (HSM). For more information, see AWS CloudHSM Key Management Utility (KMU).

JCE provider

The AWS CloudHSM JCE provider is compliant with the Java Cryptographic Architecture (JCA). The provider allows you to perform cryptographic operations on the HSM.

The JCE provider is a Linux only component that matches Linux base support. For more information, see the section called "Linux support for Client SDK 3".

- For Client SDK 3 requires OpenJDK 1.8

OpenSSL Dynamic Engine

The AWS CloudHSM OpenSSL Dynamic Engine allows you to offload cryptographic operations to your CloudHSM cluster through the OpenSSL API.

- For Client SDK 3, the OpenSSL Dynamic Engine is Linux only component that does *not* match Linux base support. See the exclusions below.
  - Requires OpenSSL 1.0.2[f+]

  Unsupported platforms:

  - CentOS 8
  - Red Hat Enterprise Linux (RHEL) 8
  - Ubuntu 18.04 LTS

  These platforms ship with a version of OpenSSL incompatible with OpenSSL Dynamic Engine for Client SDK 3. AWS CloudHSM supports these platforms with OpenSSL Dynamic Engine for Client SDK 5.

- For Client SDK 5, the OpenSSL Dynamic Engine is a Linux only component that requires OpenSSL 1.0.2, 1.1.1, or 3.x.

Key storage provider (KSP)

Key Storage Provider (KSP) is a cryptographic API specific to the Microsoft Windows operating system.

For Client SDK 3, the CNG and KSP providers is a Windows only component that matches Windows base support. For more information, see Windows support for AWS CloudHSM Client SDK 3.

For Client SDK 5, the Key Storage Provider (KSP) is a Windows only component that matches Windows base support. For more information, see Windows support for AWS CloudHSM Client SDK 5.

Migrating from AWS CloudHSM Client SDK 3 to Client SDK 5

For detailed instructions on migrating from Client SDK 3 to Client SDK 5, refer to the following topics.

For functionality or use cases that are not supported by CloudHSM CLI, please reach out to support.

- Migrate your AWS CloudHSM PKCS #11 library from Client SDK 3 to Client SDK 5
- Migrate your OpenSSL Dynamic Engine from AWS CloudHSM Client SDK 3 to Client SDK 5
- Migrate your Key Storage Provider (KSP) from AWS CloudHSM Client SDK 3 to Client SDK 5
- Migrate your JCE provider from AWS CloudHSM Client SDK 3 to Client SDK 5
- Migrate from AWS CloudHSM Client SDK 3 CMU and KMU to Client SDK 5 CloudHSM CLI

Migrate your AWS CloudHSM PKCS #11 library from Client SDK 3 to Client SDK 5

Use this topic to migrate your AWS CloudHSM PKCS #11 library from Client SDK 3 to Client SDK 5. For benefits on migrating, see Benefits of AWS CloudHSM Client SDK 5.

In AWS CloudHSM, customer applications perform cryptographic operations using the AWS CloudHSM Client Software Development Kit (SDK). Client SDK 5 is the primary SDK that continues to have new features and platform support added to it.

To review migration instructions for all providers, see Migrating from AWS CloudHSM Client SDK 3 to Client SDK 5.

Prepare by addressing breaking changes

Review these breaking changes and update your application in your development environment accordingly.

Wrap mechanisms have changed

Client SDK 3 mechanism	Equivalent Client SDK 5 mechanism
`CKM_AES_KEY_WRAP`	`CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD`
`CKM_AES_KEY_WRAP_PAD`	`CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD`
`CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD`	`CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD`
`CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD`	`CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD`
`CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD`	`CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD`

ECDH

In Client SDK 3, you can use ECDH and specify a KDF. This functionality is not currently available in Client SDK 5. If your application needs this functionality, please reach out to support.

Key handles are now session-specific

To successfully use key handles in Client SDK 5, you must obtain key handles each time you run an application. If you have existing applications that will use the same key handles across different sessions, you must modify your code to obtain the key handle each time you run the application. For information on retrieving key handles, see this AWS CloudHSM PKCS #11 example. This change is in compliance with the PKCS #11 2.40 specification.

Migrate to Client SDK 5

Follow the instructions in this section to migrate from Client SDK 3 to Client SDK 5.

> **Note**
>
> Amazon Linux, Ubuntu 16.04, Ubuntu 18.04, CentOS 6, CentOS 8, and RHEL 6 are not currently supported with Client SDK 5. If you are currently using one of these platforms with Client SDK 3, you will need to choose a different platform when migrating to Client SDK 5.

1. Uninstall the PKCS #11 library for Client SDK 3.

   Amazon Linux 2

   ```
   $ sudo yum remove cloudhsm-client-pkcs11
   ```

   CentOS 7

   ```
   $ sudo yum remove cloudhsm-client-pkcs11
   ```

   RHEL 7

   ```
   $ sudo yum remove cloudhsm-client-pkcs11
   ```

RHEL 8

```
$ sudo yum remove cloudhsm-client-pkcs11
```

2. Uninstall the Client Daemon for Client SDK 3.

   Amazon Linux 2

   ```
   $ sudo yum remove cloudhsm-client
   ```

   CentOS 7

   ```
   $ sudo yum remove cloudhsm-client
   ```

   RHEL 7

   ```
   $ sudo yum remove cloudhsm-client
   ```

   RHEL 8

   ```
   $ sudo yum remove cloudhsm-client
   ```

   Note

   Custom configurations need to be enabled again.

3. Install the Client SDK PKCS #11 library by following the steps in Install the PKCS #11 library for AWS CloudHSM Client SDK 5 .
4. Client SDK 5 introduces a new configuration file format and command-line bootstrapping tool. To bootstrap your Client SDK 5 PKCS #11 library, follow the instructions listed in the user guide under Bootstrap the Client SDK.
5. In your development environment, test your application. Make updates to your existing code to resolve your breaking changes before your final migration.

Related topics

- Best practices for AWS CloudHSM

Migrate your OpenSSL Dynamic Engine from AWS CloudHSM Client SDK 3 to Client SDK 5

Use this topic to migrate your OpenSSL Dynamic Engine from AWS CloudHSM Client SDK 3 to Client SDK 5. For benefits on migrating, see Benefits of AWS CloudHSM Client SDK 5.

In AWS CloudHSM, customer applications perform cryptographic operations using the AWS CloudHSM Client Software Development Kit (SDK). Client SDK 5 is the primary SDK that continues to have new features and platform support added to it.

> **Note**
>
> Random number generation is not currently supported in Client SDK 5 with OpenSSL Dynamic Engine.

To review migration instructions for all providers, see Migrating from AWS CloudHSM Client SDK 3 to Client SDK 5.

Migrate to Client SDK 5

Follow the instructions in this section to migrate from Client SDK 3 to Client SDK 5.

> **Note**
>
> Amazon Linux, Ubuntu 16.04, Ubuntu 18.04, CentOS 6, CentOS 8, and RHEL 6 are not currently supported with Client SDK 5. If you are currently using one of these platforms with Client SDK 3, you will need to choose a different platform when migrating to Client SDK 5.

1. Uninstall the OpenSSL Dynamic Engine for Client SDK 3.

Amazon Linux 2

```
$ sudo yum remove cloudhsm-client-dyn
```

CentOS 7

```
$ sudo yum remove cloudhsm-client-dyn
```

RHEL 7

```
$ sudo yum remove cloudhsm-client-dyn
```

RHEL 8

```
$ sudo yum remove cloudhsm-client-dyn
```

2. Uninstall the Client Daemon for Client SDK 3.

   Amazon Linux 2

```
$ sudo yum remove cloudhsm-client
```

   CentOS 7

```
$ sudo yum remove cloudhsm-client
```

   RHEL 7

```
$ sudo yum remove cloudhsm-client
```

   RHEL 8

```
$ sudo yum remove cloudhsm-client
```

> **Note**
>
> Custom configurations need to be enabled again.

3. Install the Client SDK OpenSSL Dynamic Engine by following the steps in Install the OpenSSL Dynamic Engine for AWS CloudHSM Client SDK 5.
4. Client SDK 5 introduces a new configuration file format and command-line bootstrapping tool. To bootstrap your Client SDK 5 OpenSSL Dynamic Engine, follow the instructions listed in the user guide under Bootstrap the Client SDK.
5. In your development environment, test your application. Make updates to your existing code to resolve your breaking changes before your final migration.

Related topics

- Best practices for AWS CloudHSM

Migrate your Key Storage Provider (KSP) from AWS CloudHSM Client SDK 3 to Client SDK 5

This topic explains how to migrate your Key Storage Provider (KSP) from AWS CloudHSM Client SDK 3 to Client SDK 5. For information about migration benefits, see Benefits of AWS CloudHSM Client SDK 5.

In AWS CloudHSM, you use the AWS CloudHSM Client Software Development Kit (SDK) to perform cryptographic operations. Client SDK 5 is the primary SDK that receives new features and platform support updates.

For migration instructions for all providers, see Migrating from AWS CloudHSM Client SDK 3 to Client SDK 5.

Migrate to Client SDK 5

1. Install the Client SDK 5 Key Storage Provider (KSP) on your Windows Server instance. For instructions, see Install the Key storage provider (KSP) for AWS CloudHSM Client SDK 5.
2. Configure your Client SDK 5 Key Storage Provider (KSP) using the new configuration file format and command-line bootstrapping tool. For instructions, see Bootstrap the Client SDK.

3. Key Storage Provider (KSP) for AWS CloudHSM Client SDK 5 includes SDK3 compatibility mode to support key reference files generated in SDK3. For more information, see SDK3 compatibility mode for Key Storage Provider (KSP) for AWS CloudHSM.

> **Note**
>
> You must enable SDK3 compatibility mode when using Client SDK 3 generated key reference files with Client SDK 5.

Migrate to new Windows Server instances

1. Complete all steps in Migrate to Client SDK 5 on your new Windows Server instances.
2. **Check for existing key reference files**

   On your original Windows Server instance, check for key reference files in `C:\Users\Default\AppData\Roaming\Microsoft\Crypto\CaviumKSP\GlobalPartition`.

   - If key reference files exist, copy all contents under `C:\Users\Default\AppData\Roaming\Microsoft\Crypto\CaviumKSP` including `GlobalPartition` to the same directory path on your new Windows Server instance. Create the directory if it doesn't exist.
   - If key reference files don't exist, use `cloudhsm-cli key generate-file --encoding ksp-key-reference` on your new Windows Server instance to create them. For instructions, see Generating KSP key references (Windows).
3. **Verify root certificate**

   Check your root certificate in the trusted root certification authorities:

```
PS C:\Users\Administrator\Desktop> certutil -store Root

Root "Trusted Root Certification Authorities"
================ Certificate 0 ================
Serial Number: certificate-serial-number
Issuer: CN=MYRootCA
 NotBefore: 2/5/2020 1:38 PM
 NotAfter: 2/5/2021 1:48 PM
 Issuer: CN=MYRootCA
Signature matches Public Key
Root Certificate: Subject matches Issuer
```

```
Cert Hash(sha1): cert-hash
No key provider information
Cannot find the certificate and private key for decryption.
CertUtil: -store command completed successfully.
```

Note

Note the certificate serial number for use in next step.

4. **Export root certificate**

   Export the root certificate to a file:

   ```
   certutil -store Root certificate-serial-number root-certificate-name.cer
   ```

5. **Verify HSM-backend certificate**

   Check your HSM-backend certificate in the Personal certificate store:

   ```
   PS C:\Users\Administrator\Desktop> certutil -store My

   my "Personal"
   ================ Certificate 0 ================
   Serial Number: certificate-serial-number
   Issuer: CN=MYRootCA
    NotBefore: 2/5/2020 1:38 PM
    NotAfter: 2/5/2021 1:48 PM
   Subject: CN=www.mydomain.com, OU=Certificate Management, O=Information Technology,
    L=Houston, S=Texas, C=US
   Non-root Certificate
   Cert Hash(sha1): cert-hash
     Key Container = key-container-name
     Provider = Cavium Key Storage Provider
   Private key is NOT exportable
   Encryption test passed
   CertUtil: -store command completed successfully.
   ```

   Note

   Note the certificate serial number for use in next step.

6. **Export HSM-backend certificate**

   Export the HSM-backend certificate to a file:

   ```
   certutil -store My certificate-serial-number signed-certificate-name.cer
   ```

7. **Import root certificate**

   On your new Windows instance:

   1. Copy the root CA file to your new Windows instance
   2. Import the certificate:

      ```
      certutil -addstore Root root-certificate-name.cer
      ```

8. **Verify root certificate installation**

   Confirm the root certificate is properly installed:

   ```
   PS C:\Users\Administrator\Desktop> certutil -store Root

   Root "Trusted Root Certification Authorities"
   ================ Certificate 0 ================
   Serial Number: certificate-serial-number
   Issuer: CN=MYRootCA
    NotBefore: 2/5/2020 1:38 PM
    NotAfter: 2/5/2021 1:48 PM
    Issuer: CN=MYRootCA
   Signature matches Public Key
   Root Certificate: Subject matches Issuer
   Cert Hash(sha1): cert-hash
   No key provider information
   Cannot find the certificate and private key for decryption.
   CertUtil: -store command completed successfully.
   ```

9. **Import HSM-backend certificate**

   On your new Windows instance:

   1. Copy the HSM-backend certificate to your new Windows instance
   2. Import the certificate:

```
certutil -addstore My signed-certificate-name.cer
```

10. **Verify HSM-backend certificate installation**

    Confirm the HSM-backend certificate is properly installed:

```
PS C:\Users\Administrator\Desktop> certutil -store My

my "Personal"
================ Certificate 0 ================
Serial Number: certificate-serial-number
Issuer: CN=MYRootCA
 NotBefore: 2/5/2020 1:38 PM
 NotAfter: 2/5/2021 1:48 PM
Subject: CN=www.mydomain.com, OU=Certificate Management, O=Information Technology,
 L=Houston, S=Texas, C=US
Non-root Certificate
Cert Hash(sha1): cert-hash
No key provider information
Cannot find the certificate and private key for decryption.
CertUtil: -store command completed successfully.
```

> **Note**
>
> Note the certificate serial number for use in subsequent steps.

11. **Create a key reference file (optional)**

    Complete this step only if you need to create new key reference file. Otherwise, proceed to the next step.

> **Note**
>
> This feature is only in SDK version 5.16.0 and later.

    1. Install OpenSSL and extract the modulus:

```
openssl x509 -in signed-certificate-name.cer -modulus -noout
```

> **Note**
>
> The OpenSSL command outputs the modulus in the format: `Modulus=`*`modulus-value`*. Note the *`modulus-value`* for use in the next command.

2. Create key reference file with CloudHSM CLI, see Generating KSP key references (Windows):

```
& "C:\Program Files\Amazon\CloudHSM\bin\cloudhsm-cli.exe" key generate-
file --encoding ksp-key-reference --filter attr.class=private-key
 attr.modulus=0xmodulus-value
```

> **Note**
>
> The *`modulus-value`* in CloudHSM CLI command arguments must be prefixed with `0x` to indicate hexadecimal format.
> Key reference files are created in `C:\Users\Default\AppData\Roaming\Microsoft\Crypto\CaviumKSP\GlobalPartition`.

12. **Create repair configuration**

Create a file named `repair.txt` with the following content:

```
[Properties]
11 = "" ; Add friendly name property
2 = "{text}" ; Add Key Provider Information property
_continue_="Container=key-container-name&"
_continue_="Provider=Cavium Key Storage Provider&"
_continue_="Flags=0&"
_continue_="KeySpec=2"
```

> **Note**
>
> Replace *`key-container-name`* with the key reference filename from `C:\Users\Default\AppData\Roaming\Microsoft\Crypto\CaviumKSP\GlobalPartition`.

13. **Repair certificate store**

Run the repair command:

```
certutil -repairstore My certificate-serial-number repair.txt
```

Note

The certificate serial number is obtained from the previous steps when verifying HSM-backend certificate installation.

14. **Verify certificate association**

    Confirm the certificate is properly associated:

```
PS C:\Users\Administrator\Desktop> certutil -store My

my "Personal"
================ Certificate 0 ================
Serial Number: certificate-serial-number
Issuer: CN=MYRootCA
 NotBefore: 2/5/2020 1:38 PM
 NotAfter: 2/5/2021 1:48 PM
Subject: CN=www.mydomain.com, OU=Certificate Management, O=Information Technology,
 L=Houston, S=Texas, C=US
Non-root Certificate
Cert Hash(sha1): cert-hash
  Key Container = key-container-name
  Provider = Cavium Key Storage Provider
Private key is NOT exportable
ERROR: Could not verify certificate public key against private key
CertUtil: -store command completed successfully.
```

    Verify the output shows:

    - The correct key container name
    - The Cavium Key Storage Provider
    - The `ERROR: Could not verify certificate public key against private key` is a known issue, see Issue: Verification of a certificate store fails

15. **Test your application**

Before completing the migration:

1. Test your application in your development environment
2. Update your code to resolve any breaking changes
3. For application-specific guidance, see Integrating third-party applications with AWS CloudHSM

Verify the migration

After completing the migration steps, verify that:

- Your certificates are properly installed in the correct certificate stores
- Key reference files are present in the correct location
- Your application can perform cryptographic operations using the migrated certificates

Troubleshooting

If you encounter issues during migration, verify:

- All certificates are properly exported from the source system
- Certificate serial numbers match between systems
- Key container names in the repair.txt file match your key reference files
- SDK3 compatibility mode is enabled if using SDK3-generated key reference files

Related topics

- Best practices for AWS CloudHSM

Migrate your JCE provider from AWS CloudHSM Client SDK 3 to Client SDK 5

Use this topic to migrate your JCE provider from AWS CloudHSM Client SDK 3 to Client SDK 5. For benefits on migrating, see Benefits of AWS CloudHSM Client SDK 5.

In AWS CloudHSM, customer applications perform cryptographic operations using the AWS CloudHSM Client Software Development Kit (SDK). Client SDK 5 is the primary SDK that continues to have new features and platform support added to it.

The Client SDK 3 JCE provider uses custom classes and APIs that are not part of the standard JCE specification. Client SDK 5 for the JCE provider is complaint with the JCE specification and is backwards incompatible with Client SDK 3 in certain areas. Customer applications may require changes as part of the migration to Client SDK 5. This section outlines the changes required for a successful migration.

To review migration instructions for all providers, see Migrating from AWS CloudHSM Client SDK 3 to Client SDK 5.

Topics

Prepare by addressing breaking changes

Review these breaking changes and update your application in your development environment accordingly.

The Provider class and name have changed

What has changed	What it was in Client SDK 3	What it is in Client SDK 5	Example
Provider class and name	The JCE provider class in Client SDK 3 is called `CaviumProvider` and has the Provider name `Cavium`.	In Client SDK 5, the Provider class is called `CloudHsmProvider` and has the Provider name `CloudHSM`.	An example of how to initialize the `CloudHsmProvider` object is available in the AWS CloudHSM GitHub sample repository.

Explicit login has changed, implicit has not

What has changed	What it was in Client SDK 3	What it is in Client SDK 5	Example
Explicit login	Client SDK 3 uses the `LoginManager` class for explicit login [1].	In Client SDK 5, the `CloudHSM` provider implements `AuthProvider` for explicit login. `AuthProvider` is a standard Java class and follows Java's idiomatic way to log in to a Provider. With improved login state management in Client SDK 5, applications no longer need to monitor and perform login during reconnections[2].	For an example on how to use explicit login with Client SDK 5, see the LoginRunner sample in the AWS CloudHSM GitHub sample repository.
Implicit login	No changes are required for implicit login. The same properties file and all environment variables will continue to work for the implicit login when migrating from Client SDK 3 to Client SDK 5.		For an example on how to use implicit login with Client SDK 5, see the LoginRunner sample in the AWS CloudHSM GitHub sample repository.

- [1] Client SDK 3 code snippet:

```
LoginManager lm = LoginManager.getInstance();
```

```
lm.login(partition, user, pass);
```

- [2] Client SDK 5 code snippet:

```
// Construct or get the existing provider object
AuthProvider provider = new CloudHsmProvider();

// Call login method on the CloudHsmProvider object
// Here loginHandler is a CallbackHandler
provider.login(null, loginHandler);
```

For an example on how to use explicit login with Client SDK 5, see the LoginRunner sample in the AWS CloudHSM GitHub sample repository.

Key generation has changed

What has changed	What it was in Client SDK 3	What it is in Client SDK 5	Example
Key generation	In Client SDK 3, `Cavium[Key-type]AlgorithmParameterSpec` is used to specify key generation parameters. For a code snippet, see footnote 1.	In Client SDK 5, `KeyAttributesMap` is used to specify key generation attributes. For a code snippet, see footnote 2.	For an example on how to use `KeyAttributesMap` to generate a symmetric key, see the SymmetricKeys sample in the AWS CloudHSM GitHub sample repository.
Key pair generation	In Client SDK 3, `Cavium[Key-type]AlgorithmparameterSpec` is used to specify key pair generation parameters. For	In Client SDK 5, `KeyPairAttributesMap` is used to specify these parameters. For a code snippet, see footnote 4.	For an example on how to use `KeyAttributesMap` to generate an asymmetric key, see the AsymmetricKeys sample in the AWS

What has changed	What it was in Client SDK 3	What it is in Client SDK 5	Example
	a code snippet, see footnote 3.		CloudHSM GitHub sample repository.

- [1] Client SDK 3 key generation code snippet:

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES", "Cavium");
CaviumAESKeyGenParameterSpec aesSpec = new CaviumAESKeyGenParameterSpec(
keySizeInBits,
keyLabel,
isExtractable,
isPersistent);
keyGen.init(aesSpec);
SecretKey aesKey = keyGen.generateKey();
```

- [2] Client SDK 5 key generation code snippet:

```
KeyGenerator keyGen = KeyGenerator.getInstance("AES",
CloudHsmProvider.PROVIDER_NAME);

final KeyAttributesMap aesSpec = new KeyAttributesMap();
aesSpec.put(KeyAttribute.LABEL, keyLabel);
aesSpec.put(KeyAttribute.SIZE, keySizeInBits);
aesSpec.put(KeyAttribute.EXTRACTABLE, isExtractable);
aesSpec.put(KeyAttribute.TOKEN, isPersistent);

keyGen.init(aesSpec);
SecretKey aesKey = keyGen.generateKey();
```

- [3] Client SDK 3 key pair generation code snippet::

```
KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("rsa", "Cavium");
CaviumRSAKeyGenParameterSpec spec = new CaviumRSAKeyGenParameterSpec(
keySizeInBits,
new BigInteger("65537"),
label + ":public",
label + ":private",
isExtractable,
isPersistent);
```

```
keyPairGen.initialize(spec);

keyPairGen.generateKeyPair();
```

- [4] Client SDK 5 key pair generation code snippet:

```
KeyPairGenerator keyPairGen =
KeyPairGenerator.getInstance("RSA", providerName);

// Set attributes for RSA public key
final KeyAttributesMap publicKeyAttrsMap = new KeyAttributesMap();
publicKeyAttrsMap.putAll(additionalPublicKeyAttributes);
publicKeyAttrsMap.put(KeyAttribute.LABEL, label + ":Public");
publicKeyAttrsMap.put(KeyAttribute.MODULUS_BITS, keySizeInBits);
publicKeyAttrsMap.put(KeyAttribute.PUBLIC_EXPONENT,
new BigInteger("65537").toByteArray());

// Set attributes for RSA private key
final KeyAttributesMap privateKeyAttrsMap = new KeyAttributesMap();
privateKeyAttrsMap.putAll(additionalPrivateKeyAttributes);
privateKeyAttrsMap.put(KeyAttribute.LABEL, label + ":Private");

// Create KeyPairAttributesMap and use that to initialize the
// keyPair generator
KeyPairAttributesMap keyPairSpec =
new KeyPairAttributesMapBuilder()
.withPublic(publicKeyAttrsMap)
.withPrivate(privateKeyAttrsMap)
.build();

keyPairGen.initialize(keyPairSpec);
keyPairGen.generateKeyPair();
```

Finding, deleting, and referencing keys have changed

Finding an already generated key with AWS CloudHSM entails using the KeyStore. Client SDK 3 has two KeyStore types: `Cavium` and `CloudHSM`. Client SDK 5 only has one KeyStore type: `CloudHSM`.

Moving from the `Cavium` KeyStore to `CloudHSM` KeyStore requires a change of KeyStore type. Additionally, Client SDK 3 uses key handles to reference keys, while Client SDK 5 uses key labels. The resulting behavior changes are listed below.

What has changed	What it was in Client SDK 3	What it is in Client SDK 5	Example
Key references	With Client SDK 3, applications use either key labels or key handles to reference keys in the HSM. They use labels with KeyStore to find a key, or they use handles and create `CaviumKey` objects.	In Client SDK 5, applications can use the AWS CloudHSM KeyStore Java class for Client SDK 5 to find keys by label. To find keys by handle, use the AWS CloudHSM `KeyStoreWithAttributes` with AWS CloudHSM `KeyReferenceSpec` .	
Finding multiple entries	When searching for a key using `getEntry`, `getKey`, or `getCertificate` in scenarios where multiple items with the same criteria exist in the `Cavium` KeyStore, only the first entry found will be returned.	With the AWS CloudHSM `KeyStore` and `KeyStoreWithAttributes` , this same scenario will result in an exception being thrown. To fix this problem, it is recommended to set unique labels for keys using the Set the attributes of keys with CloudHSM CLI command in CloudHSM CLI. Or use `KeyStoreWithAttrib`	

What has changed	What it was in Client SDK 3	What it is in Client SDK 5	Example
		`utes#getKeys` to return all keys that match the criteria.	
Find all keys	It is possible in Client SDK 3 to find all keys in the HSM using `Util.findAllKeys()` .	Client SDK 5 makes finding keys simpler and more efficient by using the `KeyStoreWithAttributes` class. When possible, cache your keys to minimize latency. For more information, see Effectively manage keys in your application. When you need to retrieve all keys from the HSM, use `KeyStoreWithAttributes#getKeys` with an empty `KeyAttributesMap` .	An example that uses the `KeyStoreWithAttributes` class to find a key is available in the AWS CloudHSM GitHub sample repository and a code snippet is shown in 1.
Key deletion	Client SDK 3 uses `Util.deleteKey()` to delete a key.	The `Key` object in Client SDK 5 implements the `Destroyable` interface which allows for keys to be deleted using the `destroy()` method of this interface.	An example code showing the delete key functionality can be found on the CloudHSM GitHub sample repository. A sample snippet for each SDK is shown in 2.

- [1] a snippet is shown below:

```
KeyAttributesMap findSpec = new KeyAttributesMap();
findSpec.put(KeyAttribute.LABEL, label);
findSpec.put(KeyAttribute.KEY_TYPE, keyType);
KeyStoreWithAttributes keyStore = KeyStoreWithAttributes.getInstance("CloudHSM");

keyStore.load(null, null);
keyStore.getKey(findSpec);
```

- [2] Deleting a key in Client SDK 3:

```
Util.deleteKey(key);
```

  Deleting a key in Client SDK 5:

```
((Destroyable) key).destroy();
```

Cipher unwrap operations have changed, other cipher operations have not

> **Note**
>
> No changes are required for Cipher encrypt/decrypt/wrap operations.

Unwrap operations require the Client SDK 3 `CaviumUnwrapParameterSpec` class to be replaced with one of the following classes specific to the cryptographic operations listed.

- `GCMUnwrapKeySpec` for `AES/GCM/NoPadding` unwrap
- `IvUnwrapKeySpec` for `AESWrap unwrap` and `AES/CBC/NoPadding unwrap`
- `OAEPUnwrapKeySpec` for `RSA OAEP unwrap`

Example snippet for `OAEPUnwrapkeySpec`:

```
OAEPParameterSpec oaepParameterSpec =
new OAEPParameterSpec(
        "SHA-256",
        "MGF1",
```

```
        MGF1ParameterSpec.SHA256,
        PSpecified.DEFAULT);

KeyAttributesMap keyAttributesMap =
        new KeyAttributesMap(KeyAttributePermissiveProfile.KEY_CREATION);
keyAttributesMap.put(KeyAttribute.TOKEN, true);
keyAttributesMap.put(KeyAttribute.EXTRACTABLE, false);

OAEPUnwrapKeySpec spec = new OAEPUnwrapKeySpec(oaepParameterSpec,
        keyAttributesMap);

Cipher hsmCipher =
        Cipher.getInstance(
                "RSA/ECB/OAEPPadding",
                CloudHsmProvider.PROVIDER_NAME);
hsmCipher.init(Cipher.UNWRAP_MODE, key, spec);
```

Signature operations have not changed

No changes are required for Signature operations.

Migrate to Client SDK 5

Follow the instructions in this section to migrate from Client SDK 3 to Client SDK 5.

> **Note**
>
> Amazon Linux, Ubuntu 16.04, Ubuntu 18.04 CentOS 6, CentOS 8, and RHEL 6 are not currently supported with Client SDK 5. If you are currently using one of these platforms with Client SDK 3, you will need to choose a different platform when migrating to Client SDK 5.

1. Uninstall the JCE provider for Client SDK 3.

   Amazon Linux 2

   ```
   $ sudo yum remove cloudhsm-client-jce
   ```

CentOS 7

```
$ sudo yum remove cloudhsm-client-jce
```

RHEL 7

```
$ sudo yum remove cloudhsm-client-jce
```

RHEL 8

```
$ sudo yum remove cloudhsm-client-jce
```

2. Uninstall the Client Daemon for Client SDK 3.

   Amazon Linux 2

```
$ sudo yum remove cloudhsm-client
```

   CentOS 7

```
$ sudo yum remove cloudhsm-client
```

   RHEL 7

```
$ sudo yum remove cloudhsm-client
```

   RHEL 8

```
$ sudo yum remove cloudhsm-client
```

> **Note**
>
> Custom configurations need to be enabled again.

3. Install the Client SDK JCE provider by following the steps in [Install the JCE provider for AWS CloudHSM Client SDK 5](#).

4. Client SDK 5 introduces a new configuration file format and command-line bootstrapping tool. To bootstrap your Client SDK 5 JCE provider, follow the instructions listed in the user guide under Bootstrap the Client SDK.
5. In your development environment, test your application. Make updates to your existing code to resolve your breaking changes before your final migration.

Related topics

- Best practices for AWS CloudHSM

Using Client SDK 5 to work with AWS CloudHSM

AWS CloudHSM includes two major Client SDK versions:

- Client SDK 5: This is our latest and default Client SDK. For information on the benefits and advantages it provides, see Benefits of AWS CloudHSM Client SDK 5.
- Client SDK 3: This is our older Client SDK. It includes a full set of components for platform and language-based applications compatibility and management tools.

For instructions on migrating from Client SDK 3 to Client SDK 5, see Migrating from AWS CloudHSM Client SDK 3 to Client SDK 5.

This topic describes Client SDK 5. To check what version of Client SDK you're using, see

Topics

Benefits of AWS CloudHSM Client SDK 5

Compared to AWS CloudHSM Client SDK 3, Client SDK 5 is easier to manage, offers superior configurability, and increased reliability. Client SDK 5 also provides some additional key advantages to Client SDK 3.

Designed for serverless architecture

Client SDK 5 does not require a client daemon, so you no longer need to manage a background service. This helps users in a few important ways:

- Simplifies the application startup process. All you need to do to get started with CloudHSM is configure the SDK before running your application.
- You don't need a constantly running process, which makes integration with serverless components like Lambda and Elastic Container Service (ECS) easier.

Better third party integrations and easier portability

Client SDK 5 follows the JCE specification closely and provides easier portability between different JCE providers and better third party integrations

Improved user experience and configurability

Client SDK 5 improves log message readability and provides clearer exceptions and error-handling mechanisms, all of which makes self-service triaging much easier for users. SDK 5 also offers a variety of configurations, which are listed in the Configure Tool page.

Broader platform support

Client SDK 5 offers more support for modern operating platforms. This includes support for ARM technologies and greater support for JCE, PKCS#11, and OpenSSL. For more information, refer to Supported platforms.

IPv6 connection support

Client SDK 5.14+ supports connections to dual-stack HSMs using IPv6.

Additional features and mechanisms

Client SDK 5 includes additional features and mechanisms that are not available in Client SDK 3, and Client SDK 5 will continue to add more mechanisms in the future.

AWS CloudHSM Client SDK 5 supported platforms

Base support is different for each version of the AWS CloudHSM Client SDK. Platform support for components in an SDK typically matches base support, but not always. To determine platform support for a given component, first make sure the platform you want appears in the base section for the SDK, then check for any exclusions or any other pertinent information in the component section.

AWS CloudHSM supports only 64-bit operating systems.

Platform support changes over time. Earlier versions of the CloudHSM Client SDK may not support all the operating systems listed here. Use release notes to determine the operating system support for previous versions of the CloudHSM Client SDK. For more information, see Downloads for AWS CloudHSM Client SDK.

For supported platforms for the previous Client SDK, see AWS CloudHSM Client SDK 3 supported platforms

Client SDK 5 does not require a client daemon.

Topics

- Linux support for AWS CloudHSM Client SDK 5
- Windows support for AWS CloudHSM Client SDK 5
- Serverless support for AWS CloudHSM Client SDK 5
- HSM compatibility for AWS CloudHSM Client SDK 5

Linux support for AWS CloudHSM Client SDK 5

AWS CloudHSM Client SDK 5 supports the following Linux operating systems and platforms.

Supported platforms	X86_64 Architecture	ARM architecture
Amazon Linux 2	Yes	Yes

Supported platforms	X86_64 Architecture	ARM architecture
Amazon Linux 2023	Yes	Yes
Red Hat Enterprise Linux 8 (8.3+)	Yes	No
Red Hat Enterprise Linux 9 (9.2+)	Yes	Yes
Ubuntu 22.04 LTS	Yes	Yes
Ubuntu 24.04 LTS	Yes	Yes

- SDK 5.16 was the last release to provide Ubuntu 20.04 LTS platform support. For more information, see the Ubuntu website.
- SDK 5.12 was the last release to provide CentOS 7 (7.8+) platform support. For more information, see the CentOS website.
- SDK 5.12 was the last release to provide Red Hat Enterprise Linux 7 (7.8+) platform support. For more information, see the Red Hat website.
- SDK 5.4.2 was the last release to provide CentOS 8 platform support. For more information, see the CentOS website.

Windows support for AWS CloudHSM Client SDK 5

AWS CloudHSM Client SDK 5 supports the following versions of Windows Server.

- Microsoft Windows Server 2016
- Microsoft Windows Server 2019
- Microsoft Windows Server 2022
- Microsoft Windows Server 2025

Serverless support for AWS CloudHSM Client SDK 5

AWS CloudHSM Client SDK 5 supports the following AWS serverless services.

- AWS Lambda
- Docker/ECS

HSM compatibility for AWS CloudHSM Client SDK 5

The following table describes AWS CloudHSM Client SDK 5 compatibility for HSMs.

hsm1.medium	hsm2m.medium
Compatible with Client SDK version 5.0.0 and later.	Compatible with Client SDK version 5.9.0 and later.

PKCS #11 library for AWS CloudHSM Client SDK 5

PKCS #11 is a standard for performing cryptographic operations on hardware security modules (HSMs). AWS CloudHSM offers implementations of the PKCS #11 library that are compliant with PKCS #11 version 2.40.

For information about bootstrapping, see Connecting to the cluster. For troubleshooting, see Known issues for the PKCS #11 library for AWS CloudHSM.

For information on using Client SDK 3, see Using previous SDK version to work with AWS CloudHSM.

Topics

- Install the PKCS #11 library for AWS CloudHSM Client SDK 5
- Authenticate to the PKCS #11 library for AWS CloudHSM Client SDK 5
- Supported key types for the PKCS #11 library for AWS CloudHSM Client SDK 5
- Supported mechanisms for the PKCS #11 library for AWS CloudHSM Client SDK 5
- Supported API operations for the PKCS #11 library for AWS CloudHSM Client SDK 5
- Key attributes in the PKCS #11 library for AWS CloudHSM Client SDK 5
- Code samples for the PKCS #11 library for AWS CloudHSM Client SDK 5
- Advanced configurations for PKCS #11 library for AWS CloudHSM
- Certificate storage with the PKCS #11 library

Install the PKCS #11 library for AWS CloudHSM Client SDK 5

This topic provides instructions for installing the latest version of the PKCS #11 library for the AWS CloudHSM Client SDK 5 version series. For more information about the Client SDK or PKCS #11 library, see Using the Client SDK and PKCS #11 library.

With Client SDK 5, you are not required to install or run a client daemon.

To run a single HSM cluster with Client SDK 5, you must first manage client key durability settings by setting `disable_key_availability_check` to `True`. For more information, see Key Synchronization and Client SDK 5 Configure Tool.

For more information about the PKCS #11 library in Client SDK 5, see PKCS #11 library.

> **Note**
>
> To run a single HSM cluster with Client SDK 5, you must first manage client key durability settings by setting `disable_key_availability_check` to `True`. For more information, see Key Synchronization and Client SDK 5 Configure Tool.

To install and configure the PKCS #11 library

1. Use the following commands to download and install the PKCS #11 library.

   Amazon Linux 2023

   Install the PKCS #11 library for Amazon Linux 2023 on X86_64 architecture:

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/cloudhsm-pkcs11-latest.amzn2023.x86_64.rpm
   ```

   ```
   $ sudo yum install ./cloudhsm-pkcs11-latest.amzn2023.x86_64.rpm
   ```

   Install the PKCS #11 library for Amazon Linux 2023 on ARM64 architecture:

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/cloudhsm-pkcs11-latest.amzn2023.aarch64.rpm
   ```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.amzn2023.aarch64.rpm
```

Amazon Linux 2

Install the PKCS #11 library for Amazon Linux 2 on X86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el7.x86_64.rpm
```

Install the PKCS #11 library for Amazon Linux 2 on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-pkcs11-latest.el7.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el7.aarch64.rpm
```

RHEL 9 (9.2+)

Install the PKCS #11 library for RHEL 9 on X86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-pkcs11-latest.el9.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el9.x86_64.rpm
```

Install the PKCS #11 library for RHEL 9 on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-pkcs11-latest.el9.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el9.aarch64.rpm
```

RHEL 8 (8.3+)

Install the PKCS #11 library for RHEL 8 on X86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-pkcs11-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-pkcs11-latest.el8.x86_64.rpm
```

Ubuntu 24.04 LTS

Install the PKCS #11 library for Ubuntu 24.04 LTS on X86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Noble/cloudhsm-pkcs11_latest_u24.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-pkcs11_latest_u24.04_amd64.deb
```

Install the PKCS #11 library for Ubuntu 24.04 LTS on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Noble/cloudhsm-pkcs11_latest_u24.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-pkcs11_latest_u24.04_arm64.deb
```

Ubuntu 22.04 LTS

Install the PKCS #11 library for Ubuntu 22.04 LTS on X86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/cloudhsm-pkcs11_latest_u22.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-pkcs11_latest_u22.04_amd64.deb
```

Install the PKCS #11 library for Ubuntu 22.04 LTS on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/
cloudhsm-pkcs11_latest_u22.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-pkcs11_latest_u22.04_arm64.deb
```

Ubuntu 20.04 LTS

Install the PKCS #11 library for Ubuntu 20.04 LTS on X86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Focal/
cloudhsm-pkcs11_latest_u20.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-pkcs11_latest_u20.04_amd64.deb
```

Windows Server

Install the PKCS #11 library for Windows Server on X86_64 architecture:

1. Download PKCS #11 library for Client SDK 5.
2. Run the PKCS #11 library installer (**AWSCloudHSMPKCS11-latest.msi**) with Windows administrative privilege.

2. Use the configure tool to specify the location of the issuing certificate. For instructions, see Specify the location of the issuing certificate.
3. To connect to your cluster, see Bootstrap the Client SDK.
4. You can find the PKCS #11 library files in the following locations:

   - Linux binaries, configuration scripts, and log files:

   ```
   /opt/cloudhsm
   ```

   Windows binaries:

   ```
   C:\Program Files\Amazon\CloudHSM
   ```

   Windows configuration scripts and log files:

```
C:\ProgramData\Amazon\CloudHSM
```

Authenticate to the PKCS #11 library for AWS CloudHSM Client SDK 5

When you use the PKCS #11 library, your application runs as a particular crypto user (CU) in your HSMs in AWS CloudHSM. Your application can view and manage only the keys that the CU owns and shares. You can use an existing CU in your HSMs or create a new CU for your application. For information on managing CUs, see Managing HSM users with CloudHSM CLI and Managing HSM users with CloudHSM Management Utility (CMU)

To specify the CU to PKCS #11 library, use the pin parameter of the PKCS #11 C_Login function. For AWS CloudHSM, the pin parameter has the following format:

```
<CU_user_name>:<password>
```

For example, the following command sets the PKCS #11 library pin to the CU with user name `CryptoUser` and password `CUPassword123!`.

```
CryptoUser:CUPassword123!
```

Supported key types for the PKCS #11 library for AWS CloudHSM Client SDK 5

The PKCS #11 library for AWS CloudHSM Client SDK 5supports the following key types.

Key Type	Description
AES	Generate 128, 192, and 256-bit AES keys.
Triple DES (3DES, DESede)	Generate 192-bit Triple DES keys. See note 1 below for an upcoming change.
EC	Generate keys with the secp224r1 (P-224), secp256r1 (P-256), secp256k1 (Blockchain), secp384r1 (P-384), and secp521r1 (P-521) curves.
GENERIC_SECRET	Generate 1 to 800 bytes generic secrets.

Key Type	Description
RSA	Generate 2048-bit to 4096-bit RSA keys, in increments of 256 bits.

[1] In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.

Supported mechanisms for the PKCS #11 library for AWS CloudHSM Client SDK 5

The PKCS #11 library is compliant with version 2.40 of the PKCS #11 specification. To invoke a cryptographic feature using PKCS #11, call a function with a given mechanism. The following sections summarize the combinations of functions and mechanisms supported by AWS CloudHSM Client SDK 5.

The PKCS #11 library supports the following algorithms:

- **Encryption and decryption** – AES-CBC, AES-CTR, AES-ECB, AES-GCM, DES3-CBC, DES3-ECB, RSA-OAEP, and RSA-PKCS
- **Sign and verify** – RSA, HMAC, and ECDSA; with and without hashing
- **Hash/digest** – SHA1, SHA224, SHA256, SHA384, and SHA512
- **Key wrap** – AES Key Wrap[1], AES-GCM, RSA-AES, and RSA-OAEP
- **Key derivation** – SP800-108 Counter KDF and ECDH with KDF (Supported KDF algorithms are X9.63 with SHA1, SHA224, SHA256, SHA384, SHA512)

Topics

- Generate key and key pair functions
- Sign and verify functions
- Sign recover and verify recover functions
- Digest functions
- Encrypt and decrypt functions
- Derive key functions
- Wrap and Unwrap functions

- Maximum data size for each mechanism
- Mechanism annotations

Generate key and key pair functions

The AWS CloudHSM software library for PKCS #11 library allows you to use the following mechanisms for Generate Key and Key Pair functions.

- `CKM_RSA_PKCS_KEY_PAIR_GEN`
- `CKM_RSA_X9_31_KEY_PAIR_GEN` – This mechanism is functionally identical to the `CKM_RSA_PKCS_KEY_PAIR_GEN` mechanism, but offers stronger guarantees for p and q generation.
- `CKM_EC_KEY_PAIR_GEN`
- `CKM_GENERIC_SECRET_KEY_GEN`
- `CKM_AES_KEY_GEN`
- `CKM_DES3_KEY_GEN` – upcoming change listed in footnote 5.

Sign and verify functions

The AWS CloudHSM software library for PKCS #11 library allows you to use the following mechanisms for Sign and Verify functions. With Client SDK 5, the data is hashed locally in software. This means there is no limit on the size of the data that can be hashed by the SDK.

With Client SDK 5 RSA and ECDSA hashing is done locally so there is no data limit. With HMAC, there is a data limit. See footnote 2 for more info.

RSA

- `CKM_RSA_X_509`
- `CKM_RSA_PKCS` – single-part operations only.
- `CKM_RSA_PKCS_PSS` – single-part operations only.
- `CKM_SHA1_RSA_PKCS`
- `CKM_SHA224_RSA_PKCS`
- `CKM_SHA256_RSA_PKCS`
- `CKM_SHA384_RSA_PKCS`

- CKM_SHA512_RSA_PKCS
- CKM_SHA512_RSA_PKCS
- CKM_SHA1_RSA_PKCS_PSS
- CKM_SHA224_RSA_PKCS_PSS
- CKM_SHA256_RSA_PKCS_PSS
- CKM_SHA384_RSA_PKCS_PSS
- CKM_SHA512_RSA_PKCS_PSS

ECDSA

- CKM_ECDSA – single-part operations only.
- CKM_ECDSA_SHA1
- CKM_ECDSA_SHA224
- CKM_ECDSA_SHA256
- CKM_ECDSA_SHA384
- CKM_ECDSA_SHA512

HMAC

- CKM_SHA_1_HMAC[2]
- CKM_SHA224_HMAC[2]
- CKM_SHA256_HMAC[2]
- CKM_SHA384_HMAC[2]
- CKM_SHA512_HMAC[2]

CMAC

- CKM_AES_CMAC

Sign recover and verify recover functions

Client SDK 5 does not support Sign Recover and Verify Recover functions.

Digest functions

The AWS CloudHSM software library for PKCS #11 library allows you to use the following mechanisms for Digest functions. With Client SDK 5, the data is hashed locally in software. This means there is no limit on the size of the data that can be hashed by the SDK.

- CKM_SHA_1
- CKM_SHA224
- CKM_SHA256
- CKM_SHA384
- CKM_SHA512

Encrypt and decrypt functions

The AWS CloudHSM software library for PKCS #11 library allows you to use the following mechanisms for Encrypt and Decrypt functions.

- CKM_RSA_X_509
- CKM_RSA_PKCS – single-part operations only. Upcoming change listed in footnote 5.
- CKM_RSA_PKCS_OAEP – single-part operations only.
- CKM_AES_ECB
- CKM_AES_CTR
- CKM_AES_CBC
- CKM_AES_CBC_PAD
- CKM_DES3_CBC – upcoming change listed in footnote 5.
- CKM_DES3_ECB – upcoming change listed in footnote 5.
- CKM_DES3_CBC_PAD – upcoming change listed in footnote 5.
- CKM_AES_GCM [1, 2]
- CKM_CLOUDHSM_AES_GCM[3]

Derive key functions

The AWS CloudHSM software library for PKCS #11 library supports the following key derivation mechanisms:

- CKM_SP800_108_COUNTER_KDF
- CKM_ECDH1_DERIVE - Supports ECDH key derivation with the following vendor-defined KDF types[6]:
  - CKD_CLOUDHSM_X963_SHA1_KDF - X9.63 KDF with SHA1[7]
  - CKD_CLOUDHSM_X963_SHA224_KDF - X9.63 KDF with SHA224[7]
  - CKD_CLOUDHSM_X963_SHA256_KDF - X9.63 KDF with SHA256[7]
  - CKD_CLOUDHSM_X963_SHA384_KDF - X9.63 KDF with SHA384[7]
  - CKD_CLOUDHSM_X963_SHA512_KDF - X9.63 KDF with SHA512[7]

Wrap and Unwrap functions

The AWS CloudHSM software library for PKCS #11 library allows you to use the following mechanisms for Wrap and Unwrap functions.

For additional information regarding AES key wrapping, see AES Key Wrapping.

- CKM_RSA_PKCS – single-part operations only. An upcoming change is listed in footnote 5.
- CKM_RSA_PKCS_OAEP[4]
- CKM_AES_GCM[1, 3]
- CKM_CLOUDHSM_AES_GCM[3]
- CKM_RSA_AES_KEY_WRAP
- CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD[3]
- CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD[3]
- CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD[3]

Maximum data size for each mechanism

The following table lists the maximum data size set for each mechanism:

Maximum data set size

Mechanism	Maximum data size in bytes
CKM_SHA_1_HMAC	16288
CKM_SHA224_HMAC	16256
CKM_SHA256_HMAC	16288
CKM_SHA384_HMAC	16224
CKM_SHA512_HMAC	16224
CKM_AES_CBC	16272
CKM_AES_GCM	16224
CKM_CLOUDHSM_AES_GCM	16224
CKM_DES3_CBC	16280

Mechanism annotations

- [1] When performing AES-GCM encryption, the HSM does not accept initialization vector (IV) data from the application. You must use an IV that it generates. The 12-byte IV provided by the HSM is written into the memory reference pointed to by the pIV element of the `CK_GCM_PARAMS` parameters structure that you supply. To prevent user confusion, PKCS #11 SDK in version 1.1.1 and later ensures that pIV points to a zeroized buffer when AES-GCM encryption is initialized.
- [2] When operating on data by using any of the following mechanisms, if the data buffer exceeds the maximum data size, the operation results in an error. For these mechanisms, all the data processing must occur inside the HSM. For information on maximum data size sets for each mechanism, refer to Maximum data size for each mechanism.
- [3] Vendor-defined mechanism. In order to use the CloudHSM vendor defined mechanisms, PKCS#11 applications must include `/opt/cloudhsm/include/pkcs11t.h` during compilation.

  `CKM_CLOUDHSM_AES_GCM`: This proprietary mechanism is a programmatically safer alternative to the standard `CKM_AES_GCM`. It prepends the IV generated by the HSM to the ciphertext instead of writing it back into the `CK_GCM_PARAMS` structure that is provided during cipher initialization. You can use this mechanism with `C_Encrypt`, `C_WrapKey`, `C_Decrypt`, and `C_UnwrapKey` functions. When using this mechanism, the pIV variable in the `CK_GCM_PARAMS` struct must be set to `NULL`. When using this mechanism with `C_Decrypt` and `C_UnwrapKey`, the IV is expected to be prepended to the ciphertext that is being unwrapped.

`CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD`: AES Key Wrap with PKCS #5 Padding.

`CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD`: AES Key Wrap with Zero Padding.

- [4] The following `CK_MECHANISM_TYPE` and `CK_RSA_PKCS_MGF_TYPE` are supported as `CK_RSA_PKCS_OAEP_PARAMS` for `CKM_RSA_PKCS_OAEP`:
  - `CKM_SHA_1` using `CKG_MGF1_SHA1`
  - `CKM_SHA224` using `CKG_MGF1_SHA224`
  - `CKM_SHA256` using `CKG_MGF1_SHA256`
  - `CKM_SHA384` using `CKM_MGF1_SHA384`
  - `CKM_SHA512` using `CKM_MGF1_SHA512`
- [5] In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.
- [6] Vendor defined types. In order to use CloudHSM vendor defined types, PKCS#11 applications must include `cloudhsm_pkcs11_vendor_defs.h` during compilation. This is found in `/opt/cloudhsm/include/pkcs11/cloudhsm_pkcs11_vendor_defs.h` for Linux based platforms and `C:\Program Files\Amazon\CloudHSM\include\pkcs11\cloudhsm_pkcs11_vendor_defs.h` for Windows based platforms
- [7] Key derivation functions (KDFs) are specified in RFC 8418, Section 2.1.

Supported API operations for the PKCS #11 library for AWS CloudHSM Client SDK 5

The PKCS #11 library supports the following PKCS #11 API operations for AWS CloudHSM Client SDK 5.

- `C_CloseAllSessions`
- `C_CloseSession`
- `C_CreateObject`
- `C_Decrypt`

- C_DecryptFinal
- C_DecryptInit
- C_DecryptUpdate
- C_DeriveKey
- C_DestroyObject
- C_Digest
- C_DigestFinal
- C_DigestInit
- C_DigestUpdate
- C_Encrypt
- C_EncryptFinal
- C_EncryptInit
- C_EncryptUpdate
- C_Finalize
- C_FindObjects
- C_FindObjectsFinal
- C_FindObjectsInit
- C_GenerateKey
- C_GenerateKeyPair
- C_GenerateRandom
- C_GetAttributeValue
- C_GetFunctionList
- C_GetInfo
- C_GetMechanismInfo
- C_GetMechanismList
- C_GetSessionInfo
- C_GetSlotInfo
- C_GetSlotList
- C_GetTokenInfo
- C_Initialize

- `C_Login`
- `C_Logout`
- `C_OpenSession`
- `C_Sign`
- `C_SignFinal`
- `C_SignInit`
- `C_SignUpdate`
- `C_UnWrapKey`
- `C_Verify`
- `C_VerifyFinal`
- `C_VerifyInit`
- `C_VerifyUpdate`
- `C_WrapKey`

Key attributes in the PKCS #11 library for AWS CloudHSM Client SDK 5

An AWS CloudHSM key object can be a public, private, or secret key. Actions permitted on a key object are specified through attributes. Attributes are defined when the key object is created. When you use the PKCS #11 library for AWS CloudHSM, we assign default values as specified by the PKCS #11 standard.

AWS CloudHSM does not support all attributes listed in the PKCS #11 specification. We are compliant with the specification for all attributes we support. These attributes are listed in the respective tables.

Cryptographic functions such as `C_CreateObject`, `C_GenerateKey`, `C_GenerateKeyPair`, `C_UnwrapKey`, and `C_DeriveKey` that create, modify, or copy objects take an attribute template as one of their parameters. For more information about passing an attribute template during object creation, see Generate keys through PKCS #11 library for examples.

The following topics provide more information about AWS CloudHSM key attributes.

Topics

- Interpreting PKCS #11 library error codes for AWS CloudHSM Client SDK 5

PKCS #11 library attributes tables for AWS CloudHSM Client SDK 5

The PKCS #11 library tables for AWS CloudHSM contain a list of attributes that differ by key types. It indicates whether a given attribute is supported for a particular key type when using a specific cryptographic function with AWS CloudHSM.

Legend:

- ✔ indicates that CloudHSM supports the attribute for the specific key type.
- ✖ indicates that CloudHSM does not support the attribute for the specific key type.
- R indicates that the attribute value is set to read-only for the specific key type.
- S indicates that the attribute cannot be read by the `GetAttributeValue` as it is sensitive.
- An empty cell in the Default Value column indicates that there is no specific default value assigned to the attribute.

GenerateKeyPair

Attribute	Key Type				Default Value
	EC private	EC public	RSA private	RSA public	
CKA_CLASS	✔	✔	✔	✔	
CKA_KEY_TYPE	✔	✔	✔	✔	
CKA_LABEL	✔	✔	✔	✔	
CKA_ID	✔	✔	✔	✔	
CKA_LOCAL	R	R	R	R	True

Attribute	Key Type					Default Value
CKA_TOKEN		✓	✓	✓	✓	False
CKA_PRIVATE		✓[1]	✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT		✖	✓	✖	✓	False
CKA_DECRYPT		✓	✖	✓	✖	False
CKA_DERIVE		✓	✓	✓	✓	False
CKA_MODIFIABLE		✓	✓	✓	✓	True
CKA_DESTROYABLE		✓	✓	✓	✓	True
CKA_SIGN		✓	✖	✓	✖	False
CKA_SIGN_RECOVER		✖	✖	✖	✖	
CKA_VERIFY		✖	✓	✖	✓	False
CKA_VERIFY_RECOVER		✖	✖	✖	✖	
CKA_WRAP		✖	✓	✖	✓	False

Attribute	Key Type					Default Value
CKA_WRAP_TEMPLATE		✖	✔	✖	✔	
CKA_TRUSTED		✖	✔	✖	✔	False
CKA_WRAP_WITH_TRUSTED		✔	✖	✔	✖	False
CKA_UNWRAP		✔	✖	✔	✖	False
CKA_UNWRAP_TEMPLATE		✔	✖	✔	✖	
CKA_SENSITIVE		✔[1]	✖	✔[1]	✖	True
CKA_ALWAYS_SENSITIVE		R	✖	R	✖	
CKA_EXTRACTABLE		✔	✖	✔	✖	True
CKA_NEVER_EXTRACTABLE		R	✖	R	✖	
CKA_MODULUS		✖	✖	✖	✖	
CKA_MODULUS_BITS		✖	✖	✖	✔[2]	

Attribute	Key Type					Default Value
CKA_PRIME_1		✖	✖	✖	✖	
CKA_PRIME_2		✖	✖	✖	✖	
CKA_COEFFICIENT		✖	✖	✖	✖	
CKA_EXPONENT_1		✖	✖	✖	✖	
CKA_EXPONENT_2		✖	✖	✖	✖	
CKA_PRIVATE_EXPONENT		✖	✖	✖	✖	
CKA_PUBLIC_EXPONENT		✖	✖	✖	✔[2]	
CKA_EC_PARAMS		✖	✔[2]	✖	✖	
CKA_EC_POINT		✖	✖	✖	✖	
CKA_VALUE		✖	✖	✖	✖	
CKA_VALUE_LEN		✖	✖	✖	✖	

Attribute	Key Type					Default Value
CKA_CHECK_VALUE		R	R	R	R	

GenerateKey

Attribute	Key Type				Default Value
		AES	DES3	Generic Secret	
CKA_CLASS		✓	✓	✓	
CKA_KEY_TYPE		✓	✓	✓	
CKA_LABEL		✓	✓	✓	
CKA_ID		✓	✓	✓	
CKA_LOCAL		R	R	R	True
CKA_TOKEN		✓	✓	✓	False
CKA_PRIVATE		✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT		✓	✓	✖	False
CKA_DECRYPT		✓	✓	✖	False
CKA_DERIVE		✓	✓	✓	False

Attribute	Key Type				Default Value
CKA_MODIFIABLE		✔	✔	✔	True
CKA_DESTROYABLE		✔	✔	✔	True
CKA_SIGN		✔	✔	✔	True
CKA_SIGN_RECOVER		✖	✖	✖	
CKA_VERIFY		✔	✔	✔	True
CKA_VERIFY_RECOVER		✖	✖	✖	
CKA_WRAP		✔	✔	✖	False
CKA_WRAP_TEMPLATE		✔	✔	✖	
CKA_TRUSTED		✔	✔	✖	False
CKA_WRAP_WITH_TRUSTED		✔	✔	✔	False
CKA_UNWRAP		✔	✔	✖	False
CKA_UNWRAP_TEMPLATE		✔	✔	✖	

Attribute	Key Type				Default Value
CKA_SENSITIVE		✔	✔	✔	True
CKA_ALWAYS_SENSITIVE		✖	✖	✖	
CKA_EXTRACTABLE		✔	✔	✔	True
CKA_NEVER_EXTRACTABLE		R	R	R	
CKA_MODULUS		✖	✖	✖	
CKA_MODULUS_BITS		✖	✖	✖	
CKA_PRIME_1		✖	✖	✖	
CKA_PRIME_2		✖	✖	✖	
CKA_COEFFICIENT		✖	✖	✖	
CKA_EXPONENT_1		✖	✖	✖	
CKA_EXPONENT_2		✖	✖	✖	

Attribute	Key Type				Default Value
CKA_PRIVATE_EXPONENT		✖	✖	✖	
CKA_PUBLIC_EXPONENT		✖	✖	✖	
CKA_EC_PARAMS		✖	✖	✖	
CKA_EC_POINT		✖	✖	✖	
CKA_VALUE		✖	✖	✖	
CKA_VALUE_LEN		✔[2]	✖	✔[2]	
CKA_CHECK_VALUE		R	R	R	

CreateObject

Attribute	Key Type								Default Value
		EC private	EC public	RSA private	RSA public	AES	DES3	Generic Secret	
CKA_CLASS		✔[2]	✔[2]	✔[2]	✔[2]	✔[2]	✔[2]	✔[2]	
CKA_KEY_TYPE		✔[2]	✔[2]	✔[2]	✔[2]	✔[2]	✔[2]	✔[2]	

Attribute	Key Type								Default Value
CKA_LABEL		✔	✔	✔	✔	✔	✔	✔	
CKA_ID		✔	✔	✔	✔	✔	✔	✔	
CKA_LOCAL		R	R	R	R	R	R	R	False
CKA_TOKEN		✔	✔	✔	✔	✔	✔	✔	False
CKA_PRIVATE		✔[1]	✔[1]	✔[1]	✔[1]	✔[1]	✔[1]	✔[1]	True
CKA_ENCRYPT		✖	✖	✖	✔	✔	✔	✖	False
CKA_DECRYPT		✖	✖	✔	✖	✔	✔	✖	False
CKA_DERIVE		✔	✔	✔	✔	✔	✔	✔	False
CKA_MODIFIABLE		✔	✔	✔	✔	✔	✔	✔	True
CKA_DESTROYABLE		✔	✔	✔	✔	✔	✔	✔	True
CKA_SIGN		✔	✖	✔	✖	✔	✔	✔	False
CKA_SIGN_RECOVER		✖	✖	✖	✖	✖	✖	✖	False
CKA_VERIFY		✖	✔	✖	✔	✔	✔	✔	False

Attribute	Key Type							Default Value
CKA_VERIFY_RECOVER	✖	✖	✖	✖	✖	✖	✖	
CKA_WRAP	✖	✖	✖	✔	✔	✔	✖	False
CKA_WRAP_TEMPLATE	✖	✔	✖	✔	✔	✔	✖	
CKA_TRUSTED	✖	✔	✖	✔	✔	✔	✖	False
CKA_WRAP_WITH_TRUSTED	✔	✖	✔	✖	✔	✔	✔	False
CKA_UNWRAP	✖	✖	✔	✖	✔	✔	✖	False
CKA_UNWRAP_TEMPLATE	✔	✖	✔	✖	✔	✔	✖	
CKA_SENSITIVE	✔	✖	✔	✖	✔	✔	✔	True
CKA_ALWAYS_SENSITIVE	R	✖	R	✖	R	R	R	
CKA_EXTRACTABLE	✔	✖	✔	✖	✔	✔	✔	True
CKA_NEVER_EXTRACTABLE	R	✖	R	✖	R	R	R	

Attribute	Key Type							Default Value
CKA_MODULUS	✖	✖	✔[2]	✔[2]	✖	✖	✖	
CKA_MODULUS_BITS	✖	✖	✖	✖	✖	✖	✖	
CKA_PRIME_1	✖	✖	✔	✖	✖	✖	✖	
CKA_PRIME_2	✖	✖	✔	✖	✖	✖	✖	
CKA_COEFFICIENT	✖	✖	✔	✖	✖	✖	✖	
CKA_EXPONENT_1	✖	✖	✔	✖	✖	✖	✖	
CKA_EXPONENT_2	✖	✖	✔	✖	✖	✖	✖	
CKA_PRIVATE_EXPONENT	✖	✖	✔[2]	✖	✖	✖	✖	
CKA_PUBLIC_EXPONENT	✖	✖	✔[2]	✔[2]	✖	✖	✖	
CKA_EC_PARAMS	✔[2]	✔[2]	✖	✖	✖	✖	✖	
CKA_EC_POINT	✖	✔[2]	✖	✖	✖	✖	✖	

Attribute	Key Type							Default Value
CKA_VALUE	✓[2]	✖	✖	✖	✓[2]	✓[2]	✓[2]	
CKA_VALUE_LEN	✖	✖	✖	✖	✖	✖	✖	
CKA_CHECK_VALUE	R	R	R	R	R	R	R	

UnwrapKey

Attribute	Key Type					Default Value
	EC private	RSA private	AES	DES3	Generic Secret	
CKA_CLASS	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_KEY_TYPE	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_LABEL	✓	✓	✓	✓	✓	
CKA_ID	✓	✓	✓	✓	✓	
CKA_LOCAL	R	R	R	R	R	False
CKA_TOKEN	✓	✓	✓	✓	✓	False

Attribute	Key Type						Default Value
CKA_PRIVATE		✔[1]	✔[1]	✔[1]	✔[1]	✔[1]	True
CKA_ENCRYPT		✖	✖	✔	✔	✖	False
CKA_DECRYPT		✖	✔	✔	✔	✖	False
CKA_DERIVE		✔	✔	✔	✔	✔	False
CKA_MODIFIABLE		✔	✔	✔	✔	✔	True
CKA_DESTROYABLE		✔	✔	✔	✔	✔	True
CKA_SIGN		✔	✔	✔	✔	✔	False
CKA_SIGN_RECOVER		✖	✖	✖	✖	✖	False
CKA_VERIFY		✖	✖	✔	✔	✔	False
CKA_VERIFY_RECOVER		✖	✖	✖	✖	✖	
CKA_WRAP		✖	✖	✔	✔	✖	False
CKA_UNWRAP		✖	✔	✔	✔	✖	False

Attribute	Key Type						Default Value
CKA_SENSITIVE		✓	✓	✓	✓	✓	True
CKA_EXTRACTABLE		✓	✓	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE		R	R	R	R	R	
CKA_ALWAYS_SENSITIVE		R	R	R	R	R	
CKA_MODULUS		✖	✖	✖	✖	✖	
CKA_MODULUS_BITS		✖	✖	✖	✖	✖	
CKA_PRIME_1		✖	✖	✖	✖	✖	
CKA_PRIME_2		✖	✖	✖	✖	✖	
CKA_COEFFICIENT		✖	✖	✖	✖	✖	
CKA_EXPONENT_1		✖	✖	✖	✖	✖	
CKA_EXPONENT_2		✖	✖	✖	✖	✖	

Attribute	Key Type						Default Value
CKA_PRIVATE_EXPONENT		✖	✖	✖	✖	✖	
CKA_PUBLIC_EXPONENT		✖	✖	✖	✖	✖	
CKA_EC_PARAMS		✖	✖	✖	✖	✖	
CKA_EC_POINT		✖	✖	✖	✖	✖	
CKA_VALUE		✖	✖	✖	✖	✖	
CKA_VALUE_LEN		✖	✖	✖	✖	✖	
CKA_CHECK_VALUE		R	R	R	R	R	

DeriveKey

Attribute	Key Type				Default Value
		AES	**DES3**	**Generic Secret**	
CKA_CLASS		✔[2]	✔[2]	✔[2]	
CKA_KEY_TYPE		✔[2]	✔[2]	✔[2]	

Attribute	Key Type				Default Value
CKA_LABEL		✓	✓	✓	
CKA_ID		✓	✓	✓	
CKA_LOCAL		R	R	R	True
CKA_TOKEN		✓	✓	✓	False
CKA_PRIVATE		✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT		✓	✓	✖	False
CKA_DECRYPT		✓	✓	✖	False
CKA_DERIVE		✓	✓	✓	False
CKA_MODIFIABLE		✓	✓	✓	True
CKA_DESTROYABLE		✓	✓	✓	True
CKA_SIGN		✓	✓	✓	False
CKA_SIGN_RECOVER		✖	✖	✖	
CKA_VERIFY		✓	✓	✓	False
CKA_VERIFY_RECOVER		✖	✖	✖	

Attribute	Key Type				Default Value
CKA_WRAP		✔	✔	✖	False
CKA_UNWRAP		✔	✔	✖	False
CKA_SENSITIVE		R	R	R	True
CKA_EXTRACTABLE		✔	✔	✔	True
CKA_NEVER_EXTRACTABLE		R	R	R	
CKA_ALWAYS_SENSITIVE		R	R	R	
CKA_MODULUS		✖	✖	✖	
CKA_MODULUS_BITS		✖	✖	✖	
CKA_PRIME_1		✖	✖	✖	
CKA_PRIME_2		✖	✖	✖	
CKA_COEFFICIENT		✖	✖	✖	
CKA_EXPONENT_1		✖	✖	✖	

Attribute	Key Type				Default Value
CKA_EXPONENT_2		✖	✖	✖	
CKA_PRIVATE_EXPONENT		✖	✖	✖	
CKA_PUBLIC_EXPONENT		✖	✖	✖	
CKA_EC_PARAMS		✖	✖	✖	
CKA_EC_POINT		✖	✖	✖	
CKA_VALUE		✖	✖	✖	
CKA_VALUE_LEN		✔[2]	✖	✔[2]	
CKA_CHECK_VALUE		R	R	R	

GetAttributeValue

Attribute	Key Type							
		EC private	EC public	RSA private	RSA public	AES	DES3	Generic Secret
CKA_CLASS		✔	✔	✔	✔	✔	✔	✔

Attribute	Key Type						
CKA_KEY_TYPE	✔	✔	✔	✔	✔	✔	✔
CKA_LABEL	✔	✔	✔	✔	✔	✔	✔
CKA_ID	✔	✔	✔	✔	✔	✔	✔
CKA_LOCAL	✔	✔	✔	✔	✔	✔	✔
CKA_TOKEN	✔	✔	✔	✔	✔	✔	✔
CKA_PRIVATE	✔[1]	✔[1]	✔[1]	✔[1]	✔[1]	✔[1]	✔[1]
CKA_ENCRYPT	✖	✖	✖	✔	✔	✔	✖
CKA_DECRYPT	✖	✖	✔	✖	✔	✔	✖
CKA_DERIVE	✔	✔	✔	✔	✔	✔	✔
CKA_MODIFIABLE	✔	✔	✔	✔	✔	✔	✔
CKA_DESTROYABLE	✔	✔	✔	✔	✔	✔	✔
CKA_SIGN	✔	✖	✔	✖	✔	✔	✔
CKA_SIGN_RECOVER	✖	✖	✔	✖	✖	✖	✖

Attribute	Key Type							
CKA_VERIFY		✖	✔	✖	✔	✔	✔	✔
CKA_VERIFY_RECOVER		✖	✖	✖	✔	✖	✖	✖
CKA_WRAP		✖	✖	✖	✔	✔	✔	✖
CKA_WRAP_TEMPLATE		✖	✔	✖	✔	✔	✔	✖
CKA_TRUSTED		✖	✔	✖	✔	✔	✔	✔
CKA_WRAP_WITH_TRUSTED		✔	✖	✔	✖	✔	✔	✔
CKA_UNWRAP		✖	✖	✔	✖	✔	✔	✖
CKA_UNWRAP_TEMPLATE		✔	✖	✔	✖	✔	✔	✖
CKA_SENSITIVE		✔	✖	✔	✖	✔	✔	✔
CKA_EXTRACTABLE		✔	✖	✔	✖	✔	✔	✔
CKA_NEVER_EXTRACTABLE		✔	✖	✔	✖	✔	✔	✔

Attribute	Key Type							
CKA_ALWAYS_SENSITIVE		R	R	R	R	R	R	R
CKA_MODULUS		✖	✖	✔	✔	✖	✖	✖
CKA_MODULUS_BITS		✖	✖	✖	✔	✖	✖	✖
CKA_PRIME_1		✖	✖	S	✖	✖	✖	✖
CKA_PRIME_2		✖	✖	S	✖	✖	✖	✖
CKA_COEFFICIENT		✖	✖	S	✖	✖	✖	✖
CKA_EXPONENT_1		✖	✖	S	✖	✖	✖	✖
CKA_EXPONENT_2		✖	✖	S	✖	✖	✖	✖
CKA_PRIVATE_EXPONENT		✖	✖	S	✖	✖	✖	✖
CKA_PUBLIC_EXPONENT		✖	✖	✔	✔	✖	✖	✖
CKA_EC_PARAMS		✔	✔	✖	✖	✖	✖	✖

Attribute	Key Type							
CKA_EC_POINT		✖	✔	✖	✖	✖	✖	✖
CKA_VALUE		S	✖	✖	✖	✔	✔	✔
CKA_VALUE_LEN		✖	✖	✖	✖	✔	✖	✔
CKA_CHECK_VALUE		✔	✔	✔	✔	✔	✔	✖

Attribute annotations

- [1] This attribute is partially supported by the firmware and must be explicitly set only to the default value.
- [2] Mandatory attribute.

Modifying PKCS #11 library attributes for AWS CloudHSM Client SDK 5

Some PKCS #11 library attributes for of an AWS CloudHSM object can be modified after the object has been created, whereas some cannot. To modify attributes, use the key set-attribute command from CloudHSM CLI. You can also derive a list of attributes by using the key list command from CloudHSM CLI.

The following list displays attributes that are allowed for modification after object creation:

- CKA_LABEL
- CKA_TOKEN

> **Note**
>
> Modification is allowed only for changing a session key to a token key. Use the key set-attribute command from CloudHSM CLI to change the attribute value.

- CKA_ENCRYPT

- CKA_DECRYPT
- CKA_SIGN
- CKA_VERIFY
- CKA_WRAP
- CKA_UNWRAP
- CKA_LABEL
- CKA_SENSITIVE
- CKA_DERIVE

> **Note**
>
> This attribute supports key derivation. It must be False for all public keys and cannot be set to True. For secret and EC private keys, it can be set to True or False.

- CKA_TRUSTED

> **Note**
>
> This attribute can be set to True or False by Crypto Officer (CO) only.

- CKA_WRAP_WITH_TRUSTED

> **Note**
>
> Apply this attribute to an exportable data key to specify that you can only wrap this key with keys marked as CKA_TRUSTED. Once you set CKA_WRAP_WITH_TRUSTED to true, the attribute becomes read-only and you cannot change or remove the attribute.

Interpreting PKCS #11 library error codes for AWS CloudHSM Client SDK 5

Specifying in the template a PKCS #11 library attribute that is not supported by a specific key results in an error. The following table contains error codes that are generated when you violate specifications:

Error Code	Description
CKR_TEMPLATE_INCONSISTENT	You receive this error when you specify an attribute in the attribute template, where the attribute complies with the PKCS #11 specification, but is not supported by CloudHSM.
CKR_ATTRIBUTE_TYPE_INVALID	You receive this error when you retrieve value for an attribute, which complies with the PKCS #11 specification, but is not supported by CloudHSM.
CKR_ATTRIBUTE_INCOMPLETE	You receive this error when you do not specify the mandatory attribute in the attribute template.
CKR_ATTRIBUTE_READ_ONLY	You receive this error when you specify a read-only attribute in the attribute template.

Code samples for the PKCS #11 library for AWS CloudHSM Client SDK 5

The code samples on GitHub show you how to accomplish basic tasks using the PKCS #11 library for AWS CloudHSM Client SDK 5.

Prerequisites

Before running samples, perform the following steps to set up your environment:

- Install and configure the PKCS #11 library for Client SDK 5.
- Set up a cryptographic user (CU). Your application uses this HSM account to run the code samples on the HSM.

Code samples

Code Samples for the AWS CloudHSM Software Library for PKCS#11 are available on GitHub. This repository includes examples on how to do common operations using PKCS#11 including encryption, decryption, signing and verifying.

- Generate keys (AES, RSA, EC)
- List key attributes
- Encrypt and decrypt data with AES GCM
- Encrypt and decrypt data with AES_CTR
- Encrypt and decrypt data with 3DES
- Sign and verify data with RSA
- Derive keys using HMAC KDF
- Wrap and unwrap keys with AES using PKCS #5 padding
- Wrap and unwrap keys with AES using no padding
- Wrap and unwrap keys with AES using zero padding
- Wrap and unwrap keys with AES-GCM
- Wrap and unwrap keys with RSA

Advanced configurations for PKCS #11 library for AWS CloudHSM

The AWS CloudHSM PKCS #11 provider includes the following advanced configuration, which is not part of the general configurations most customers utilize. These configurations provide additional capabilities.

- Connecting to multiple slots with PKCS #11
- Retry configuration for PKCS #11

Multiple slot configuration with PKCS #11 library for AWS CloudHSM

A single slot in Client SDK 5 PKCS #11 library represents a single connection to a cluster in AWS CloudHSM. With Client SDK 5, you can configure your PKCS11 library to allow multiple slots to connect users to multiple CloudHSM clusters from a single PKCS#11 application.

Use the instructions in this topic to make your application use multi-slot functionality to connect with multiple clusters.

Topics

- Multi-slot prerequisites for PKCS #11 library for AWS CloudHSM
- Configure the PKCS #11 library for multi-slot functionality for AWS CloudHSM

- Add a cluster with multi-slot functionality for AWS CloudHSM
- Remove a cluster with multi-slot functionality for AWS CloudHSM

Multi-slot prerequisites for PKCS #11 library for AWS CloudHSM

Before configuring for multiple slots for PKCS #11 library for AWS CloudHSM, complete the following prerequisites.

- Two or more AWS CloudHSM clusters to which you'd like to connect to, along with their cluster certificates.
- An EC2 instance with Security Groups correctly configured to connect to all of the clusters above. For more information about how to set up a cluster and the client instance, refer to Getting started with AWS CloudHSM.
- To set up multi-slot functionality, you must have already downloaded and installed the PKCS #11 library. If you have not already done this, refer to the instructions in ???.

Configure the PKCS #11 library for multi-slot functionality for AWS CloudHSM

To configure your PKCS #11 library for multi-slot functionality for AWS CloudHSM, follow these steps:

1. Identify the clusters you want to connect to using multi-slot functionality.
2. Add these clusters to your PKCS #11 configuration by following the instructions in ???
3. The next time your PKCS#11 application runs, it will have multi-slot functionality.

Add a cluster with multi-slot functionality for AWS CloudHSM

When connecting to multiple slots with PKCS #11 for AWS CloudHSM, use the **configure-pkcs11 add-cluster** command to add a cluster to your configuration.

Syntax

```
configure-pkcs11 add-cluster [OPTIONS]
        --cluster-id <CLUSTER ID>
        [--region <REGION>]
        [--endpoint <ENDPOINT>]
        [--hsm-ca-cert <HSM CA CERTIFICATE FILE>]
        [--client-cert-hsm-tls-file <CLIENT CERTIFICATE FILE>]
```

```
        [--client-key-hsm-tls-file <CLIENT KEY FILE>]
        [-h, --help]
```

Examples

Add a cluster using the `cluster-id` parameter

Example

Use the **configure-pkcs11 add-cluster** along with the `cluster-id` parameter to add a cluster (with the ID of `cluster-1234567`) to your configuration.

Linux

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 add-cluster --cluster-id <cluster-1234567>
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" add-cluster --
cluster-id <cluster-1234567>
```

> **Tip**
>
> If using **configure-pkcs11 add-cluster** with the `cluster-id` parameter doesn't result in the cluster being added, refer to the following example for a longer version of this command that also requires `--region` and `--endpoint` parameters to identify the cluster being added. If, for example, the region of the cluster is different than the one configured as your AWS CLI default, you should use the `--region` parameter to use the correct region. Additionally, you have the ability to specify the AWS CloudHSM API endpoint to use for the call, which may be necessary for various network setups, such as using VPC interface endpoints that don't use the default DNS hostname for AWS CloudHSM.

Add a cluster using `cluster-id`, `endpoint`, and `region` parameters

Example

Use the **configure-pkcs11 add-cluster** along with the `cluster-id`, `endpoint`, and `region` parameters to add a cluster (with the ID of `cluster-1234567`) to your configuration.

Linux

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 add-cluster --cluster-id <cluster-1234567>
 --region <us-east-1> --endpoint <https://cloudhsmv2.us-east-1.amazonaws.com>
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" add-cluster --
cluster-id <cluster-1234567>--region <us-east-1> --endpoint <https://cloudhsmv2.us-
east-1.amazonaws.com>
```

For more information about the `--cluster-id`, `--region`, and `--endpoint` parameters, see the section called “Parameters”.

Parameters

--cluster-id ***<Cluster ID>***

Makes a `DescribeClusters` call to find all of the HSM elastic network interface (ENI) IP addresses in the cluster associated with the cluster ID. The system adds the ENI IP addresses to the AWS CloudHSM configuration files.

> **Note**
>
> If you use the `--cluster-id` parameter from an EC2 instance within a VPC that does not have access to the public internet, then you must create an interface VPC endpoint to connect with AWS CloudHSM. For more information about VPC endpoints, see ???.

Required: Yes

--endpoint ***<Endpoint>***

Specify the AWS CloudHSM API endpoint used for making the `DescribeClusters` call. You must set this option in combination with `--cluster-id`.

Required: No

--hsm-ca-cert ***<HsmCA Certificate Filepath>***

Specifies the filepath to the HSM CA certificate.

Required: No

--region ***<Region>***

Specify the region of your cluster. You must set this option in combination with `--cluster-id`.

If you don't supply the `--region` parameter, the system chooses the region by attempting to read the `AWS_DEFAULT_REGION` or `AWS_REGION` environment variables. If those variables aren't set, then the system checks the region associated with your profile in your AWS config file (typically `~/.aws/config`) unless you specified a different file in the `AWS_CONFIG_FILE` environment variable. If none of the above are set, the system defaults to the `us-east-1` region.

Required: No

--client-cert-hsm-tls-file ***<client certificate hsm tls path>***

Path to the client certificate used for TLS client-HSM mutual authentication.

Only use this option if you have registered at least one trust anchor onto HSM with CloudHSM CLI. You must set this option in combination with `--client-key-hsm-tls-file`.

Required: No

--client-key-hsm-tls-file ***<client key hsm tls path>***

Path to the client key used for TLS client-HSM mutual authentication.

Only use this option if you have registered at least one trust anchor onto HSM with CloudHSM CLI. You must set this option in combination with `--client-cert-hsm-tls-file`.

Required: No

Remove a cluster with multi-slot functionality for AWS CloudHSM

When [connecting to multiple slots with PKCS#11](), use the **configure-pkcs11 remove-cluster** command to remove a cluster from available PKCS #11 slots.

Syntax

```
configure-pkcs11 remove-cluster [OPTIONS]
        --cluster-id <CLUSTER ID>
        [-h, --help]
```

Examples

Remove a cluster using the `cluster-id` parameter

Example

Use the **configure-pkcs11 remove-cluster** along with the `cluster-id` parameter to remove a cluster (with the ID of `cluster-1234567`) from your configuration.

Linux

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 remove-cluster --cluster-
id <cluster-1234567>
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" remove-cluster
 --cluster-id <cluster-1234567>
```

For more information about the `--cluster-id` parameter, see the section called “Parameters”.

Parameter

--cluster-id *<Cluster ID>*

The ID of the cluster to remove from the configuration

Required: Yes

Retry commands for PKCS #11 library for AWS CloudHSM

AWS CloudHSM Client SDK 5.8.0 and later have a built-in automatic retry strategy which will retry HSM-throttled operations from the client side. When an HSM throttles operations because it is too busy performing previous operations and cannot take more requests, client SDKs will attempt to retry throttled operations up to 3 times while exponentially backing off. This automatic retry strategy can be set to one of two modes: **off** and **standard**.

- **off**: The Client SDK will not perform any retry strategy for any throttled operations by the HSM.
- **standard**: This is the default mode for Client SDK 5.8.0 and later. In this mode, client SDKs will automatically retry throttled operations by exponentially backing off.

For more information, see HSM throttling.

Set retry commands to off mode

Linux

To set retry commands to off for Client SDK 5 on Linux

- You can use the following command to set retry configuration to **off** mode:

```
$ sudo /opt/cloudhsm/bin/configure-pkcs11 --default-retry-mode off
```

Windows

To set retry commands to off for Client SDK 5 on Windows

- You can use the following command to set retry configuration to **off** mode:

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" --default-retry-mode off
```

Certificate storage with the PKCS #11 library

The AWS CloudHSM PKCS #11 library supports storing public key certificates as "public objects" (as defined in PKCS #11 2.40) on hsm2m.medium clusters. This feature allows both public and private PKCS #11 sessions to create, retrieve, modify, and delete public key certificates.

To use certificate storage with the PKCS #11 library, you need to enable it in your client configuration. Once enabled, you can manage certificate objects from your PKCS #11 applications. Operations that apply to both certificate and key objects, such as C_FindObjects, will return results from both key and certificate storage.

Topics

- Enabling certificate storage
- Certificate storage API operations
- Certificate storage attributes
- Certificate storage audit logs

Enabling certificate storage

You can enable certificate storage on hsm2m.medium clusters using the PKCS #11 library configuration tool. This feature is available in SDK versions 5.13 and later. For a list of operations that support the certificate object type, see Certificate storage API operations.

To enable certificate storage, follow these steps for your operating system:

Linux

- **Enable certificate storage**

  Run the following command:

  ```
  $ sudo /opt/cloudhsm/bin/configure-pkcs11 --enable-certificate-storage
  ```

Windows

- **Enable certificate storage**

  Open a command prompt and run the following command:

  ```
  PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-pkcs11.exe" --enable-
  certificate-storage
  ```

Certificate storage API operations

The following PKCS #11 operations support the certificate object type (`CKO_CERTIFICATE`):

General certificate operations

`C_CreateObject`

Creates a new certificate object.

`C_DestroyObject`

Deletes an existing certificate object.

`C_GetAttributeValue`

Gets the value of one or more attributes of a certificate object.

`C_SetAttributeValue`

Updates the value of one or more attributes of a certificate object.

Certificate object search operations

`C_FindObjectsInit`

Starts a search for certificate objects.

`C_FindObjects`

Continues a search for certificate objects.

`C_FindObjectsFinal`

Ends a search for certificate objects.

Certificate storage attributes

The following table lists the supported certificate object attributes and their values:

Attribute	Default value	Description
`CKA_CLASS`	Required	Must be `CKO_CERTIFICATE` .
`CKA_TOKEN`	True	Must be `True`.

Attribute	Default value	Description
CKA_MODIFIABLE	True	Must be True.
CKA_PRIVATE	False	Must be False.
CKA_LABEL	Empty	Limit 127 characters.
CKA_COPYABLE	False	Must be False.
CKA_DESTROYABLE	True	Must be True.
CKA_CERTIFICATE_TYPE	Required	Must be CKC_X_509 .
CKA_TRUSTED	False	Must be False.
CKA_CERTIFICATE_CATEGORY	CK_CERTIFICATE_CATEGORY_UNSPECIFIED	Must be CK_CERTIFICATE_CATEGORY_UNSPECIFIED .
CKA_CHECK_VALUE	Derived from CKA_VALUE	Automatically set based on CKA_VALUE .
CKA_START_DATE	Empty	The certificate 'not before' date.
CKA_END_DATE	Empty	The certificate 'not after' date.
CKA_PUBLIC_KEY_INFO	Empty	Maximum size is 16 kilobytes.
CKA_SUBJECT	Required	The certificate subject.
CKA_ID	Empty	Maximum size is 128 bytes. Uniqueness isn't enforced.
CKA_ISSUER	Empty	The certificate issuer.

Attribute	Default value	Description
CKA_SERIAL_NUMBER	Empty	The certificate serial number.
CKA_VALUE	Required	Maximum size is 32 kilobytes.

Certificate storage audit logs

AWS CloudHSM writes audit logs for certificate storage operations that modify data to a separate Amazon CloudWatch Events log stream within your cluster's CloudWatch log group. This log stream is named for the cluster, not for a specific HSM within the cluster.

For information about accessing audit logs in CloudWatch, see Working with Amazon CloudWatch Logs and AWS CloudHSM Audit Logs.

Log entry fields

`object_handle`

The unique identifier of the certificate object.

`op_code`

The operation performed or attempted. Possible values:

- `CreateObject`
- `DestroyObject`
- `SetAttributeValues`

`response`

`OK` if the operation succeeded, or one of the following error types:

- `DuplicateAttribute`
- `InvalidAttributeValue`
- `ObjectNotFound`
- `MaxObjectsReached`
- `InternalFailure`

`attributes`

The attributes modified, if any.

`timestamp`

The time when the operation occurred, in milliseconds since the Unix epoch.

Audit log examples

CreateObject example

```
{
    "object_handle": 463180677312929947,
    "op_code": "CreateObject",
    "response": "OK",
    "attributes": null,
    "timestamp": 1725482483671
}
```

DestroyObject example

```
{
    "object_handle": 463180677312929947,
    "op_code": "DestroyObject",
    "response": "OK",
    "attributes": null,
    "timestamp": 1725482484559
}
```

SetAttributeValues example

```
{
    "object_handle": 463180678453346687,
    "op_code": "SetAttributeValues",
    "response": "OK",
    "attributes": [
        "Label"
    ],
    "timestamp": 1725482488004
}
```

Unsuccessful CreateObject example

```
{
    "object_handle": null,
    "op_code": "CreateObject",
    "response": "MaxObjectsReached",
    "attributes": null,
    "timestamp": 1726084937125
}
```

OpenSSL Dynamic Engine for AWS CloudHSM Client SDK 5

The AWS CloudHSM OpenSSL Dynamic Engine allows you to offload cryptographic operations to your CloudHSM cluster through the OpenSSL API.

AWS CloudHSM provides an OpenSSL Dynamic Engine, which you can read about in AWS CloudHSM SSL/TLS offload on Linux using Tomcat with JSSE or AWS CloudHSM SSL/TLS offload on Linux using NGINX or Apache with OpenSSL. For an example on using AWS CloudHSM with OpenSSL, refer to this AWS security blog. For information about platform support for SDKs, see the section called "Supported platforms". For troubleshooting, see Known issues for the OpenSSL Dynamic Engine for AWS CloudHSM.

Use the following sections to install and configure the AWS CloudHSM dynamic engine for OpenSSL, using Client SDK 5.

For information on using Client SDK 3, see Using previous SDK version to work with AWS CloudHSM.

Topics

Install the OpenSSL Dynamic Engine for AWS CloudHSM Client SDK 5

Use the following sections to install the OpenSSL Dynamic Engine for AWS CloudHSM Client SDK 5.

> **Note**
>
> To run a single HSM cluster with Client SDK 5, you must first manage client key durability settings by setting `disable_key_availability_check` to `True`. For more information, see Key Synchronization and Client SDK 5 Configure Tool.

To install and configure the OpenSSL Dynamic Engine

1. Use the following commands to download and install the OpenSSL engine.

   Amazon Linux 2023

   Install the OpenSSL Dynamic Engine for Amazon Linux 2023 on x86_64 architecture:

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/
   cloudhsm-dyn-latest.amzn2023.x86_64.rpm
   ```

   ```
   $ sudo yum install ./cloudhsm-dyn-latest.amzn2023.x86_64.rpm
   ```

   Install the OpenSSL Dynamic Engine for Amazon Linux 2023 on ARM64 architecture:

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/
   cloudhsm-dyn-latest.amzn2023.aarch64.rpm
   ```

   ```
   $ sudo yum install ./cloudhsm-dyn-latest.amzn2023.aarch64.rpm
   ```

   Amazon Linux 2

   Install the OpenSSL Dynamic Engine for Amazon Linux 2 on x86_64 architecture:

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-
   dyn-latest.el7.x86_64.rpm
   ```

   ```
   $ sudo yum install ./cloudhsm-dyn-latest.el7.x86_64.rpm
   ```

   Install the OpenSSL Dynamic Engine for Amazon Linux 2 on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-dyn-latest.el7.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.el7.aarch64.rpm
```

RHEL 9 (9.2+)

Install the OpenSSL Dynamic Engine for RHEL 9 on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-dyn-latest.el9.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.el9.x86_64.rpm
```

Install the OpenSSL Dynamic Engine for RHEL 9 on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-dyn-latest.el9.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.el9.aarch64.rpm
```

RHEL 8 (8.3+)

Install the OpenSSL Dynamic Engine for RHEL 8 on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-dyn-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-dyn-latest.el8.x86_64.rpm
```

Ubuntu 24.04 LTS

Install the OpenSSL Dynamic Engine for Ubuntu 24.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Noble/cloudhsm-dyn_latest_u24.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-dyn_latest_u24.04_amd64.deb
```

Install the OpenSSL Dynamic Engine for Ubuntu 24.04 LTS on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Noble/
cloudhsm-dyn_latest_u24.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-dyn_latest_u24.04_arm64.deb
```

Ubuntu 22.04 LTS

Install the OpenSSL Dynamic Engine for Ubuntu 22.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/
cloudhsm-dyn_latest_u22.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-dyn_latest_u22.04_amd64.deb
```

Install the OpenSSL Dynamic Engine for Ubuntu 22.04 LTS on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/
cloudhsm-dyn_latest_u22.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-dyn_latest_u22.04_arm64.deb
```

Ubuntu 20.04 LTS

Install the OpenSSL Dynamic Engine for Ubuntu 20.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Focal/
cloudhsm-dyn_latest_u20.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-dyn_latest_u20.04_amd64.deb
```

You have installed the shared library for the dynamic engine at `/opt/cloudhsm/lib/libcloudhsm_openssl_engine.so`.

2. Bootstrap Client SDK 5. For more information about bootstrapping, see Bootstrap the Client SDK.
3. Set an environment variable with the credentials of a crypto user (CU). For information about creating CUs, see User management with CMU.

```
$ export CLOUDHSM_PIN=<HSM user name>:<password>
```

> **Note**
>
> Client SDK 5 introduces the `CLOUDHSM_PIN` environment variable for storing the credentials of the CU. In Client SDK 3 you store the CU credentials in the `n3fips_password` environment variable. Client SDK 5 supports both environment variables, but we recommend using `CLOUDHSM_PIN`.

4. Connect your installation of OpenSSL Dynamic Engine to the cluster. For more information, see Connect to the Cluster.
5. Bootstrap the Client SDK 5. For more information, see the section called “Bootstrap the Client SDK”.

Verify the OpenSSL Dynamic Engine for Client SDK 5

Use the following command to verify your installation of OpenSSL Dynamic Engine.

```
$ openssl engine -t cloudhsm
```

The following output verifies your configuration:

```
(cloudhsm) CloudHSM OpenSSL Engine
     [ available ]
```

Supported key types for OpenSSL Dynamic Engine for AWS CloudHSM Client SDK 5

The AWS CloudHSM OpenSSL Dynamic Engine supports the following key types with Client SDK 5.

Key Type	Description
EC	ECDSA sign/verify for P-256, P-384, and secp256k1 key types. To generate EC keys that are interoperable with the OpenSSL engine, see Export an asymmetric key with CloudHSM CLI.
RSA	RSA key generation for 2048, 3072, and 4096-bit keys.RSA sign/verify. Verification is offloaded to OpenSSL software.

Supported mechanisms for OpenSSL Dynamic Engine for AWS CloudHSM Client SDK 5

The AWS CloudHSM OpenSSL Dynamic Engine supports the following mechanisms for Sign and Verify functions with Client SDK 5.

Sign and verify functions

With Client SDK 5, the data is hashed locally in software. This means there is no limit on the size of the data that can be hashed.

RSA Signature Types

- SHA1withRSA
- SHA224withRSA
- SHA256withRSA
- SHA384withRSA
- SHA512withRSA

ECDSA Signature Types

- SHA1withECDSA
- SHA224withECDSA
- SHA256withECDSA

- SHA384withECDSA
- SHA512withECDSA

Advanced configurations for OpenSSL for AWS CloudHSM

The AWS CloudHSM OpenSSL provider includes the following advanced configuration, which is not part of the general configurations most customers utilize. These configurations provide additional capabilities.

- Retry commands for OpenSSL

Retry commands for OpenSSL for AWS CloudHSM

AWS CloudHSM Client SDK 5.8.0 and later have a built-in automatic retry strategy which will retry HSM-throttled operations from the client side. When an HSM throttles operations because it is too busy performing previous operations and cannot take more requests, client SDKs will attempt to retry throttled operations up to 3 times while exponentially backing off. This automatic retry strategy can be set to one of two modes: **off** and **standard**.

- **off**: The Client SDK will not perform any retry strategy for any throttled operations by the HSM.
- **standard**: This is the default mode for Client SDK 5.8.0 and later. In this mode, client SDKs will automatically retry throttled operations by exponentially backing off.

For more information, see HSM throttling.

Set retry commands to off mode

You can use the following command to set retry commands to **off** mode:

```
$ sudo /opt/cloudhsm/bin/configure-dyn --default-retry-mode off
```

Key storage provider (KSP) for AWS CloudHSM Client SDK 5

Key Storage Provider (KSP) is a cryptographic API specific to the Microsoft Windows operating system. Key Storage Provider (KSP) enables developers to use cryptographic techniques to secure Windows-based applications.

For information about bootstrapping, see Connecting to the cluster.

For information on using Client SDK 3, see Using previous SDK version to work with AWS CloudHSM.

Topics

Install the Key storage provider (KSP) for AWS CloudHSM Client SDK 5

Use the following sections to install the Key storage provider (KSP) for AWS CloudHSM Client SDK 5.

> **Note**
>
> To run a single HSM cluster with Client SDK 5, you must first manage client key durability settings by setting `disable_key_availability_check` to `True`. For more information, see Key Synchronization and Client SDK 5 Configure Tool.

To install and configure the Key Storage Provider (KSP)

1. Install the Key Storage Provider (KSP) for Windows Server on x86_64 architecture, open PowerShell as an administrator and run the following command:

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Windows/AWSCloudHSMKSP-latest.msi -Outfile C:\AWSCloudHSMKSP-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMKSP-latest.msi /quiet /norestart /log C:\client-install.txt' -Wait
```

2. Use the configure tool to specify the location of the issuing certificate. For instructions, see Specify the location of the issuing certificate.
3. To connect to your cluster, see Bootstrap the Client SDK.
4. You can find the Key Storage Provider (KSP) files in the following locations:

- Windows binaries:

  ```
  C:\Program Files\Amazon\CloudHSM
  ```

  Windows configuration scripts and log files:

  ```
  C:\ProgramData\Amazon\CloudHSM
  ```

Authenticate to the Key storage provider (KSP) for AWS CloudHSM Client SDK 5

Before you use the Key storage provider (KSP) for AWS CloudHSM Client SDK 5, you must set the login credentials for the HSM on your system. You have two options:

- Windows Credentials Manager (recommended for better security)
- System environment variables (simpler setup)

Windows Credential Manager

You can set up credentials using either the `set_cloudhsm_credentials` utility or the Windows Credentials Manager interface.

- **Using the `set_cloudhsm_credentials` utility**:

  The Windows installer includes the `set_cloudhsm_credentials` utility. You can use this utility to conveniently pass HSM login credentials to Windows Credential Manager. If you want to compile this utility from source, you can use the Python code included in the installer.

  1. Navigate to `C:\Program Files\Amazon\CloudHSM\tools\`.
  2. Run the following command:

  ```
  set_cloudhsm_credentials.exe --username <CU USER> --password <CU PASSWORD>
  ```

- **Using the Credential Manager interface**:
  1. Open Credential Manager:
     - Enter `credential manager` in the taskbar search box
     - Select **Credential Manager**
  2. Select **Windows Credentials** to manage Windows credentials.

3. Select **Add a generic credential**
4. Enter the following details:
   - **Internet or Network Address**: `CLOUDHSM_PIN`.
   - **Username**: *<CU USER>*.
   - **Password**: *<CU PASSWORD>*.
5. Choose **OK**

System environment variables

You can set system environment variables to identify your HSM and crypto user (CU).

> **Warning**
>
> Setting credentials through system environment variables stores your password in plaintext on your system. For better security, use Windows Credential Manager instead.

You can set environment variables using:

- The **setx**.
- The Windows **System Properties** Control Panel (**Advanced** tab).
- set permanent system environment variables Programmatic methods.

To set the system environment variable:

CLOUDHSM_PIN=*<CU USERNAME>*:*<CU PASSWORD>*

Identifies a crypto user (CU) in the HSM and provides all required login information. Your application authenticates and runs as this CU. The application has the permissions of this CU and can view and manage only the keys that the CU owns and shares. To create a new CU, use the user create command in CloudHSM CLI. To find existing CUs, use the user list command in CloudHSM CLI.

For example:

```
setx /m CLOUDHSM_PIN test_user:password123
```

Supported key types for Key Storage Provider (KSP) for AWS CloudHSM Client SDK 5

The AWS CloudHSM Key Storage Provider (KSP) supports the following key types with Client SDK 5.

Key Type	Description
EC	Generate keys with the secp256r1 (P-256), secp384r1 (P-384), and secp521r1 (P-521) curves.
RSA	Generate 2048, 3072, and 4096-bit RSA keys.

Supported API operations Key storage provider (KSP) for AWS CloudHSM Client SDK 5

The parameters in the KSP are defined by Microsoft KSP. See the Microsoft documentation for more information.

The Key Storage Provider (KSP) supports the following KSP API operations for AWS CloudHSM Client SDK 5.

- `NCryptOpenStorageProvider`
- NCryptOpenKey
- NCryptCreatePersistedKey
- NCryptGetProperty
- NCryptSetProperty
- NCryptFinalizeKey
- NCryptDeleteKey
- NCryptFreeObject
- NCryptFreeBuffer
- NCryptIsAlgSupported
- NCryptEnumAlgorithms
- NCryptEnumKeys
- NCryptExportKey

- NCryptSignHash
- NCryptVerifySignature

NCryptOpenStorageProvider function with Key Storage Provider (KSP)

The `NCryptOpenStorageProvider` function loads and initializes the Key Storage Provider (KSP).

Parameters

`phProvider` [out]

A pointer to a `NCRYPT_PROV_HANDLE` variable that stores the provider handle.

`pszProviderName` [in]

A pointer to a null-terminated Unicode string identifying the key storage provider. AWS CloudHSM Key Storage Provider (KSP) supports the following values:

Value	Meaning
L"CloudHSM Key Storage Provider"	Identifies Client SDK 5 provider name. We recommend using this name by default.
L"Cavium Key Storage Provider"	Identifies the Client SDK 3 provider name. Supported for backward compatibility.

Note

Values are wide-character string literal, as indicated by L before the literal.

`dwFlags` [in]

Flags that modify the behavior of the function. No flags are defined for this function.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.
NTE_INVALID_PARAMETER	One or more parameters are not valid.
NTE_FAIL	The operation couldn't complete.

NCryptOpenKey with Key storage provider (KSP)

The `NCryptOpenKey` function opens a key that exists in the Key Storage Provider (KSP).

Parameters

`hProvider` [in]

The KSP handle that contains the key. Use `NCryptOpenStorageProvider` to get the handle.

`phKey` [out]

A pointer to a `NCRYPT_KEY_HANDLE` variable that stores the key handle.

`pszKeyName` [in]

A pointer to a null-terminated Unicode string containing the key name.

`dwLegacyKeySpec` [in, unused]

AWS CloudHSM Key Storage Provider (KSP) doesn't use this parameter.

`dwFlags` [in]

Flags that modify function's behavior. No flags are defined for this function.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.

Return code	Description
NTE_INVALID_PARAMETER	One or more parameters are not valid.
NTE_FAIL	The operation couldn't complete.
NTE_INVALID_HANDLE	The handle in `hProvider` is not valid.
NTE_BAD_KEYSET	The key name provided did not return unique result.

NCryptCreatePersistedKey with Key storage provider (KSP)

The `NCryptCreatePersistedKey` function creates a new key and stores it in the Key Storage Provider (KSP). You can use the [`NCryptSetProperty`] function to set its properties after creation. You must call [`NCryptFinalizeKey`] before you can use the key.

Parameters

`hProvider` [in]

The handle of the key storage provider where you will create the key. Use [`NCryptOpenStorageProvider`] to get this handle.

`phKey` [out]

The address of an `NCRYPT_KEY_HANDLE` variable that stores the key handle.

`pszAlgId` [in]

A pointer to a null-terminated Unicode string that specifies the cryptographic algorithm identifier for creating the key.

AWS CloudHSM Key Storage Provider (KSP) supports the following algorithms:

Constant/value	Description
BCRYPT_RSA_ALGORITHM "RSA"	The RSA public key algorithm.

Constant/value	Description
BCRYPT_ECDSA_P256_ALGORITHM "ECDSA_P256"	The 256-bit prime elliptic curve digital signature algorithm (FIPS 186-2).
BCRYPT_ECDSA_P384_ALGORITHM "ECDSA_P384"	The 384-bit prime elliptic curve digital signature algorithm (FIPS 186-2).
BCRYPT_ECDSA_P521_ALGORITHM "ECDSA_P521"	The 521-bit prime elliptic curve digital signature algorithm (FIPS 186-2).

`pszKeyName` [in, optional]

A pointer to a null-terminated Unicode string that contains the name of the key. If this parameter is NULL, this function will create an ephemeral key that is not persisted.

`dwLegacyKeySpec` [in, unused]

AWS CloudHSM Key Storage Provider (KSP) doesn't use this parameter.

`dwFlags` [in]

Flags to modify the function's behavior. Use zero or more of the following values:

Value	Meaning
NCRYPT_MACHINE_KEY_FLAG	This flag has no effect.
NCRYPT_SILENT_FLAG	This flag has no effect.
NCRYPT_OVERWRITE_KEY_FLAG	Specifying this flag overwrites any existing key with the same name in the HSM. Without this flag, the function returns NTE_EXISTS.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The function completed successfully.
NTE_INVALID_PARAMETER	One or more parameters are not valid.
NTE_FAIL	The operation couldn't complete.
NTE_BAD_FLAGS	The `dwFlags` parameter contains an invalid value.
NTE_NOT_SUPPORTED	The `pszAlgId` parameter contains an unsupported value.
NTE_EXISTS	A key with the specified name already exists and operation didn't use `NCRYPT_OVERWRITE_KEY_FLAG` .

NCryptGetProperty with Key storage provider (KSP)

The `NCryptGetProperty` function retrieves property values for a key storage object.

Parameters

`hObject` [in]

The handle of the object whose property you want to retrieve. You can use:

- A provider handle (`NCRYPT_PROV_HANDLE`)
- A key handle (`NCRYPT_KEY_HANDLE`)

`pszProperty` [in]

A pointer to a null-terminated Unicode string containing the property name to retrieve.

When using NCRYPT_PROV_HANDLE, AWS CloudHSM Key Storage Provider (KSP) supports the following KSP identifiers:

Identifier/Value	Description
NCRYPT_IMPL_TYPE_PROPERTY L"Impl Type"	A DWORD containing flags that define provider implementation details
NCRYPT_MAX_NAME_LENGTH_PROPERTY L"Max Name Length"	A DWORD containing the maximum length (in characters) for a persistent key name.
NCRYPT_NAME_PROPERTY L"Name"	A pointer to a null-terminated Unicode string containing the KSP name.
NCRYPT_VERSION_PROPERTY L"Version"	A DWORD containing the provider version (high word: major version, low word: minor version).
NCRYPT_USE_CONTEXT_PROPERTY L"Use Context"	A pointer to a null-terminated Unicode string describing the operation context.
NCRYPT_SECURITY_DESCR_SUPPORT_PROPERTY L"Security Descr Support"	Indicates if the provider supports security descriptors for keys.

When using NCRYPT_KEY_HANDLE, AWS CloudHSM Key Storage Provider (KSP) supports the following KSP identifiers:

Identifier/Value	Description
NCRYPT_ALGORITHM_PROPERTY L"Algorithm Name"	A null-terminated Unicode string containing the key's algorithm name.

Identifier/Value	Description
NCRYPT_BLOCK_LENGTH_PROPERTY L"Block Length"	A DWORD containing the encryption block length in bytes.
NCRYPT_EXPORT_POLICY_PROPERTY L"Export Policy"	A DWORD containing flags that specify the persisted key's export policy.
NCRYPT_KEY_USAGE_PROPERTY L"Key Usage"	A DWORD containing flags that define key usage details.
NCRYPT_KEY_TYPE_PROPERTY L"Key Type"	A DWORD containing flags that define the key type.
NCRYPT_LENGTH_PROPERTY L"Length"	A DWORD containing the key length in bits.
NCRYPT_LENGTHS_PROPERTY L"Lengths"	A pointer to an NCRYPT_SUPPORTED_LENGTHS structure containing supported key sizes.
NCRYPT_NAME_PROPERTY L"Name"	A pointer to a null-terminated Unicode string containing the key name.
NCRYPT_SECURITY_DESCR_PROPERTY L"Security Descr"	A pointer to a SECURITY_DESCRIPTOR structure containing key access control information.
NCRYPT_ALGORITHM_GROUP_PROPERTY L"Algorithm Group"	A null-terminated Unicode string containing the object's algorithm group name.
NCRYPT_UNIQUE_NAME_PROPERTY L"Unique Name"	A pointer to a null-terminated Unicode string containing the key's unique name.

> **Note**
>
> Values are wide-character string literal, as indicated by L before the literal.

`pbOutput` [out]

The address of a buffer to store the property value. Specify the buffer size using `cbOutput`.

To determine the required buffer size, set this parameter to NULL. The function stores the required size (in bytes) in the location pointed to by `pcbResult`.

`cbOutput` [in]

The size of the `pbOutput` buffer in bytes.

`pcbResult` [out]

A pointer to a DWORD variable that stores the number of bytes copied to the`pbOutput` buffer.

If the `pbOutput` is NULL, this stores the required size (in bytes).

`dwFlags` [in]

Flags to modify the function's behavior. You can use zero or:

Value	Meaning
NCRYPT_SILENT_FLAG	This flag has no effect.

When pszProperty is `NCRYPT_SECURITY_DESCR_PROPERTY`, use one or a combination of:

Value	Meaning
OWNER_SECURITY_INFORMATION	This flag has no effect.
GROUP_SECURITY_INFORMATION	This flag has no effect.
DACL_SECURITY_INFORMATION	This flag has no effect.
LABEL_SECURITY_INFORMATION	This flag has no effect.

Value	Meaning
SACL_SECURITY_INFORMATION	This flag has no effect.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.
NTE_INVALID_PARAMETER	One or more parameters are not valid.
NTE_FAIL	The operation couldn't complete.
NTE_BAD_FLAGS	The `dwFlags` parameter contains an invalid value.
NTE_NOT_SUPPORTED	The `pszAlgId` parameter contains a value that is not supported.
NTE_INVALID_HANDLE	The handle in `hObject` is not valid.
NTE_BUFFER_TOO_SMALL	The `cbOutput` parameter is too small for return values.

NCryptSetProperty with Key storage provider (KSP)

The `NCryptSetProperty` function sets property values for a key storage object.

Parameters

`hObject` [in]

The handle of the object whose property you want to set. You can use:

- A provider handle (`NCRYPT_PROV_HANDLE`)

- A key handle (`NCRYPT_KEY_HANDLE`)

`pszProperty` [in]

A pointer to a null-terminated Unicode string containing the property name to retrieve.

When using `NCRYPT_PROV_HANDLE`, AWS CloudHSM Key Storage Provider (KSP) supports the following KSP identifiers:

Identifier/Value	Description
NCRYPT_USE_CONTEXT_PROPERTY L"Use Context"	A pointer to a null-terminated Unicode string describing the operation context.

When using `NCRYPT_KEY_HANDLE`, AWS CloudHSM Key Storage Provider (KSP) supports the following KSP identifiers:

Identifier/Value	Description
NCRYPT_KEY_USAGE_PROPERTY L"Key Usage"	A DWORD containing a set of flags that define key usage details. This property only applies to keys. This can contain zero or a combination of one or more of the following values. NCRYPT_ALLOW_DECRYPT_FLAG (0x00000001) NCRYPT_ALLOW_SIGNING_FLAG (0x00000002)
NCRYPT_LENGTH_PROPERTY L"Length"	A DWORD containing the key length in bits.
NCRYPT_EXPORT_POLICY_PROPERTY L"Export Policy"	A DWORD containing flags that specify the persisted key's export policy. This can contain zero or a combination of one or more of the following values.

Identifier/Value	Description
	NCRYPT_ALLOW_EXPORT_FLAG (0x00000001)

> **Note**
>
> Values are wide-character string literal, as indicated by L before the literal.

`pbInput` [in]

The address of a buffer that contains the new property value. `cbInput` contains the size of the buffer.

`cbInput` [in]

The size of the `pbInput` buffer in bytes.

`dwFlags` [in]

Flags that modify function's behavior. No flags are defined for this function.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.
NTE_INVALID_PARAMETER	One or more parameters are not valid.
NTE_FAIL	The operation couldn't complete.
NTE_BAD_FLAGS	The `dwFlags` parameter contains an invalid value.

Return code	Description
NTE_NOT_SUPPORTED	The `pszProperty` parameter contains a value that is not supported.
NTE_INVALID_HANDLE	The handle in `hObject` is not valid.
NTE_BAD_DATA	The data pointed by `pbInput` and `cbInput` is not valid.

NCryptFinalizeKey with Key storage provider (KSP)

The `NCryptFinalizeKey` function completes a KSP key. You must call this function before you can use the key.

Parameters

`hKey` [in]

The handle of the key to complete. Get this handle by calling the NCryptCreatePersistedKey function.

`dwFlags` [in]

Flags to modify the function's behavior. You can use zero or these values:

Value	Meaning
NCRYPT_SILENT_FLAG	This flag has no effect.
NCRYPT_NO_KEY_VALIDATION	This flag has no effect.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.
NTE_FAIL	The operation couldn't complete.
NTE_INVALID_HANDLE	The handle in `hKey` is not valid.
NTE_NOT_SUPPORTED	The `dwFlags` parameter contains a value that is not supported.
NTE_BAD_FLAGS	The `dwFlags` parameter contains an invalid value.

NCryptDeleteKey with Key storage provider (KSP)

The `NCryptDeleteKey` function deletes a KSP key from the Key Storage Provider (KSP).

Parameters

`hKey` [in]

The handle of the key to delete.

`dwFlags` [in]

Flags to modify the function's behavior. You can use zero or more of the following values:

Value	Meaning
NCRYPT_SILENT_FLAG	This flag has no effect.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The function was successful.
NTE_INVALID_PARAMETER	One or more parameters are not valid.
NTE_BAD_FLAGS	The `dwFlags` parameter contains an invalid value.
NTE_FAIL	The operation couldn't complete.
NTE_INVALID_HANDLE	The handle in `hKey` is not valid.
NTE_INTERNAL_ERROR	A internal error happened when deleting key.

NCryptFreeObject with Key storage provider (KSP)

The `NCryptFreeObject` function releases provider or key handle from the Key Storage Provider (KSP).

Parameters

`hObject` [in]

The handle of the object to release. You can use:

- A provider handle (`NCRYPT_PROV_HANDLE`)
- A key handle (`NCRYPT_KEY_HANDLE`)

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.
NTE_INVALID_HANDLE	The handle in `hObject` is not valid.

NCryptFreeBuffer with Key storage provider (KSP)

The `NCryptFreeBuffer` function releases a block of memory that was allocated by the Key Storage Provider (KSP).

Parameters

`pvInput` [in]

The address of the memory to released.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.
NTE_FAIL	The operation couldn't complete.

NCryptIsAlgSupported with Key storage provider (KSP)

NCryptIsAlgSupported function determines if Key Storage Provider (KSP) supports a specific cryptographic algorithm.

Parameters

`hProvider` [in]

The handle of the key storage provider. Use [`NCryptOpenStorageProvider`] to get the handle.

`pszAlgId` [in]

A pointer to a null-terminated Unicode string that contains the identifier of the cryptographic algorithm to create the key. AWS CloudHSM Key Storage Provider (KSP) supports the following algorithms:

Constant/value	Description
BCRYPT_RSA_ALGORITHM "RSA"	The RSA public key algorithm.
BCRYPT_ECDSA_P256_ALGORITHM "ECDSA_P256"	The 256-bit prime elliptic curve digital signature algorithm (FIPS 186-2).
BCRYPT_ECDSA_P384_ALGORITHM "ECDSA_P384"	The 384-bit prime elliptic curve digital signature algorithm (FIPS 186-2).
BCRYPT_ECDSA_P521_ALGORITHM "ECDSA_P521"	The 521-bit prime elliptic curve digital signature algorithm (FIPS 186-2).

`dwFlags` [in]

Flags that modify function behavior. This can be zero or the following value:

Value	Meaning
NCRYPT_SILENT_FLAG	This flag has no effect.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.
NTE_INVALID_PARAMETER	One or more parameters are not valid.

Return code	Description
NTE_BAD_FLAGS	The `dwFlags` parameter contains an invalid value.
NTE_NOT_SUPPORTED	The `pszAlgId` parameter contains an unsupported value.
NTE_INVALID_HANDLE	The handle in `hProvider` is not valid.

NCryptEnumAlgorithms with Key storage provider (KSP)

The `NCryptEnumAlgorithms` function retrieves the names of algorithms that the Key Storage Provider (KSP) supports.

Parameters

`hProvider` [in]

The handle of the key storage provider for which to enumerate the algorithms. Use the [`NCryptOpenStorageProvider`] function to get this handle.

`dwAlgOperations` [in]

A set of values that specify which algorithm classes to enumerate. You can use zero to enumerate all algorithms, or combine one or more of these values:

Value	Meaning
NCRYPT_ASYMMETRIC_ENCRYPTION_OPERATION 0x00000004	List the asymmetric encryption algorithms.
NCRYPT_SIGNATURE_OPERATION 0x00000010	List the digital signature algorithms.

`pdwAlgCount` [out]

The address of a DWORD that stores the number of elements in the `ppAlgList` array.

`ppAlgList` [out]

The address of an `NCryptAlgorithmName` structure pointer that stores an array of registered algorithm names. The `pdwAlgCount` parameter indicates the number of elements in this array.

`dwFlags` [in]

Flags to modify the function's behavior. Use zero or the following value:

Value	Meaning
NCRYPT_SILENT_FLAG	This flag has no effect.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.
NTE_INVALID_PARAMETER	One or more parameters are not valid.
NTE_FAIL	The operation couldn't complete.
NTE_BAD_FLAGS	The `dwFlags` parameter contains an invalid value.
NTE_NOT_SUPPORTED	The `dwAlgOperations` parameter contains an unsupported value.

NCryptEnumKeys with Key storage provider (KSP)

NCryptEnumKeys function lists the keys stored in the Key Storage Provider (KSP).

Parameters

`hProvider` [in]

The key storage provider handle. Use `NCryptOpenStorageProvider` to get this handle.

`pszScope` [in, unused]

Set this parameter to NULL.

`ppKeyName` [out]

A pointer address to an `NCryptKeyName` structure that stores the key name. To free this memory after use, call `NCryptFreeBuffer`.

`ppEnumState` [in, out]

A VOID pointer address that tracks the enumeration progress. The key storage provider uses this information internally to manage the enumeration sequence. To start a new enumeration from the beginning, set this pointer to NULL.

To free this memory after completing the enumeration, pass this pointer to the `NCryptFreeBuffer`.

`dwFlags` [in]

Flags to modify the function's behavior. This function has no flags.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.
NTE_INVALID_PARAMETER	One or more parameters are not valid.
NTE_FAIL	The operation couldn't complete.
NTE_INVALID_HANDLE	The handle in `hProvider` is not valid.

Return code	Description
NTE_NO_MORE_ITEMS	The enumeration has listed all available keys.

NCryptExportKey with Key storage provider (KSP)

The `NCryptExportKey` function exports a KSP key to a memory BLOB. This function only supports exporting public keys.

Parameters

`hKey` [in]

The handle of the key to export.

`hExportKey` [in, unused]

AWS CloudHSM Key Storage Provider (KSP) doesn't use this parameter.

`pszBlobType` [in]

A null-terminated Unicode string that specifies the BLOB type to export. AWS CloudHSM Key Storage Provider (KSP) supports the following values:

Value	Meaning
BCRYPT_RSAPUBLIC_BLOB	Exports an RSA public key. The `pbOutput` buffer contains a `BCRYPT_RSAKEY_BLOB` structure followed by the key data.
BCRYPT_ECCPUBLIC_BLOB	Exports an ECC public key. The `pbOutput` buffer contains a `BCRYPT_ECCKEY_BLOB` structure followed by the key data.

`pParameterList` [in, unused]

AWS CloudHSM Key Storage Provider (KSP) doesn't use this parameter.

`pbOutput` [out, optional]

A buffer address to store the key BLOB. Specify the buffer size using `cbOutput`. If set to NULL, the function stores the required size (in bytes) in the DWORD pointed to by `pcbResult`.

`cbOutput` [in]

The size of the `pbOutput` buffer in bytes.

`pcbResult` [out]

A DWORD variable address that stores the number of bytes copied to the `pbOutput` buffer. If `pbOutput` is NULL, the function stores the required buffer size in bytes.

`dwFlags` [in]

Flags that modify how the function works. You can use zero or the following:

Value	Meaning
NCRYPT_SILENT_FLAG	This flag has no effect.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.
NTE_INVALID_PARAMETER	One or more parameters are not valid.
NTE_FAIL	The operation couldn't complete.
NTE_INVALID_HANDLE	The handle in `hProvider` is not valid.
NTE_BAD_FLAGS	The `dwFlags` parameter contains an invalid value.
NTE_BAD_KEY_STATE	The key state is not valid.

Return code	Description
NTE_NOT_SUPPORTED	The `pszBlobType` or `dwFlags` parameter contains an unsupported value.
STATUS_INTERNAL_ERROR	An internal error happened during the operation.

NCryptSignHash with Key storage provider (KSP)

The `NCryptSignHash` function creates a signature of a hash value.

Parameters

`hKey` [in]

The handle of the key to use to sign the hash.

`pPaddingInfo` [in, optional]

A pointer to a structure containing padding information. The structure type depends on the `dwFlags` value. Use this parameter only with asymmetric keys; set to NULL for other key types.

`pbHashValue` [in]

A pointer to a buffer containing the hash value to sign. Specify the buffer size using `cbHashValue`.

`cbHashValue` [in]

The size, in bytes, of the `pbHashValue` buffer to sign.

`pbSignature` [out]

The address of a buffer to store the signature. Specify the buffer size using `cbSignature`.

To determine the required buffer size, set this parameter to NULL. The function stores the required size (in bytes) in the location pointed to by `pcbResult`.

`cbSignature` [in]

The size of the `pbSignature` buffer in bytes. The function ignores this parameter if `pbSignature` is NULL.

`pcbResult` [out]

A pointer to a DWORD variable that stores the number of bytes copied to the `pbSignature` buffer.

If `pbSignature` is NULL, this stores the required buffer size, in bytes.

`dwFlags` [in]

Flags to modify the function's behavior. The allowed flags depend on your key type. Use one of these values:

Value	Meaning
BCRYPT_PAD_PKCS1	Uses the PKCS1 padding scheme. Set `pPaddingInfo` to point to a `BCRYPT_PKCS1_PADDING_INFO` structure.
BCRYPT_PAD_PSS	Uses the Probabilistic Signature Scheme (PSS) padding scheme. Set `pPaddingInfo` parameter to point to a `BCRYPT_PSS_PADDING_INFO` structure.
NCRYPT_SILENT_FLAG	This flag has no effect.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.
NTE_INVALID_PARAMETER	One or more parameters are not valid.
NTE_FAIL	The operation couldn't complete.
NTE_INVALID_HANDLE	The handle in `hKey` is not valid.

Return code	Description
NTE_BAD_FLAGS	The `dwFlags` parameter contains an invalid value.
NTE_BUFFER_TOO_SMALL	The `pcbOutput` parameter is too small for return values.
NTE_BAD_KEY_STATE	The key state is not valid.
NTE_INTERNAL_ERROR	An internal error happened when signing the hash.

NCryptVerifySignature with Key storage provider (KSP)

The `NCryptVerifySignature` function confirms whether a signature matches a specified hash.

Parameters

`hKey` [in]

The handle of the key to use to decrypt the signature. You must use the public key portion of the key pair that was used to sign the data with the `NCryptSignHash`.

`pPaddingInfo` [in, optional]

A pointer to a structure containing padding information. The structure type depends on the `dwFlags` value. Use this parameter only with asymmetric keys; set to NULL for other key types.

`pbHashValue` [in]

A pointer to a buffer containing the hash value to sign. Specify the buffer size using `cbHashValue`.

`cbHashValue` [in]

The size of the `pbHashValue` buffer in bytes.

`pbSignature` [out]

The address of a buffer containing the signed hash of the data. Use `NCryptSignHash` to create this signature. Specify the buffer size using `cbSignature`.

`cbSignature` [in]

The size of the `pbSignature` buffer in bytes. Use `NCryptSignHash` to create the signature.

`dwFlags` [in]

Flags to modify the function's behavior. The allowed flags depend on your key type. Use one of these values:

Value	Meaning
NCRYPT_PAD_PKCS1_FLAG	Indicates the signature used PKCS1 padding. Set `pPaddingInfo` to point to a `BCRYPT_PKCS1_PADDING_INFO` structure.
NCRYPT_PAD_PSS_FLAG	Indicates the signature used Probabilistic Signature Scheme (PSS) padding. Set `pPaddingInfo` to point to a `BCRYPT_PSS_PADDING_INFO` structure.
NCRYPT_SILENT_FLAG	This flag has no effect.

Return Value

The function returns a status code to indicate success or failure.

Common return codes include:

Return code	Description
ERROR_SUCCESS	The operation completed successfully.
NTE_INVALID_PARAMETER	One or more parameters are not valid.
NTE_FAIL	The operation couldn't complete.
NTE_INVALID_HANDLE	The handle in `hKey` is not valid.

Return code	Description
NTE_BAD_FLAGS	The `dwFlags` parameter contains an invalid value.
NTE_BAD_SIGNATURE	The signature was not verified.
NTE_BAD_KEY_STATE	The key state is not valid.
NTE_INTERNAL_ERROR	An internal error happened while verifying the signature.

Advanced configurations for KSP for AWS CloudHSM

The AWS CloudHSM Key Storage Provider (KSP) includes the following advanced configuration, which is not part of the general configurations most customers utilize. These configurations provide additional capabilities.

- SDK3 compatibility mode for KSP

SDK3 compatibility mode for Key Storage Provider (KSP) for AWS CloudHSM

Key Storage Provider (KSP) implements different approaches for HSM key interaction:

- Client SDK 5: Provides direct communication with keys stored in the HSM, eliminating the need for local reference files
- Client SDK 3: Maintains local files on the Windows server that act as references to keys stored in the HSM, using these files to facilitate key operations

For customers migrating from Client SDK 3 to Client SDK 5, enabling SDK3 compatibility mode option supports operations using existing key reference files while preserving the underlying HSM key storage architecture.

Enable SDK3 compatibility mode

Windows

To enable SDK3 compatibility mode for Key Storage Provider (KSP) for Client SDK 5 in Windows

- You can use the following command to enable SDK3 compatibility mode:

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" --enable-sdk3-compatibility-mode
```

Disable SDK3 compatibility mode

Windows

To disable SDK3 compatibility mode for Key Storage Provider (KSP) for Client SDK 5 in Windows

- You can use the following command to disable SDK3 compatibility mode:

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-ksp.exe" --disable-sdk3-compatibility-mode
```

JCE provider for AWS CloudHSM Client SDK 5

The AWS CloudHSM JCE provider is a provider implementation built from the Java Cryptographic Extension (JCE) provider framework. The JCE allows you to perform cryptographic operations using the Java Development Kit (JDK). In this guide, the AWS CloudHSM JCE provider is sometimes referred to as the JCE provider. Use the JCE provider and the JDK to offload cryptographic operations to the HSM. For troubleshooting, see Known issues for the JCE SDK for AWS CloudHSM.

For information on using Client SDK 3, see Using previous SDK version to work with AWS CloudHSM.

Topics

- Key management basics in the JCE provider for AWS CloudHSM Client SDK 5
- Supported mechanisms for JCE provider for AWS CloudHSM Client SDK 5
- Supported Java key attributes for AWS CloudHSM Client SDK 5
- Code samples for the AWS CloudHSM software library for Java for Client SDK 5
- AWS CloudHSM JCE provider Javadocs
- AWS CloudHSM KeyStore Java class for Client SDK 5
- Advanced configurations for AWS CloudHSM JCE for Client SDK 5

Install the JCE provider for AWS CloudHSM Client SDK 5

The JCE provider for AWS CloudHSM Client SDK 5 is compatible with OpenJDK 8, OpenJDK 11, OpenJDK 17, and OpenJDK 21. You can download both from the OpenJDK website.

Use the following sections to install and provide credentials to the provider.

> **Note**
>
> To run a single HSM cluster with Client SDK 5, you must first manage client key durability settings by setting `disable_key_availability_check` to `True`. For more information, see Key Synchronization and Client SDK 5 Configure Tool.

Topics

- Step 1: Install the JCE provider
- Step 2: Provide credentials to the JCE provider

Step 1: Install the JCE provider

1. Use the following commands to download and install the JCE provider.

   Amazon Linux 2023

   Install the JCE provider for Amazon Linux 2023 on x86_64 architecture:

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/
   cloudhsm-jce-latest.amzn2023.x86_64.rpm
   ```

```
$ sudo yum install ./cloudhsm-jce-latest.amzn2023.x86_64.rpm
```

Install the JCE provider for Amazon Linux 2023 on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Amzn2023/
cloudhsm-jce-latest.amzn2023.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.amzn2023.aarch64.rpm
```

Amazon Linux 2

Install the JCE provider for Amazon Linux 2 on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-
jce-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.el7.x86_64.rpm
```

Install the JCE provider for Amazon Linux 2 on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-
jce-latest.el7.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.el7.aarch64.rpm
```

RHEL 9 (9.2+)

Install the JCE provider for RHEL 9 (9.2+) on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-
jce-latest.el9.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.el9.x86_64.rpm
```

Install the JCE provider for RHEL 9 (9.2+) on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL9/cloudhsm-jce-latest.el9.aarch64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.el9.aarch64.rpm
```

RHEL 8 (8.3+)

Install the JCE provider for RHEL 8 on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-jce-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-jce-latest.el8.x86_64.rpm
```

Ubuntu 24.04 LTS

Install the JCE provider for Ubuntu 24.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Noble/cloudhsm-jce_latest_u24.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-jce_latest_u24.04_amd64.deb
```

Install the JCE provider for Ubuntu 24.04 LTS on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Noble/cloudhsm-jce_latest_u24.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-jce_latest_u24.04_arm64.deb
```

Ubuntu 22.04 LTS

Install the JCE provider for Ubuntu 22.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/cloudhsm-jce_latest_u22.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-jce_latest_u22.04_amd64.deb
```

Install the JCE provider for Ubuntu 22.04 LTS on ARM64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Jammy/
cloudhsm-jce_latest_u22.04_arm64.deb
```

```
$ sudo apt install ./cloudhsm-jce_latest_u22.04_arm64.deb
```

Ubuntu 20.04 LTS

Install the JCE provider for Ubuntu 20.04 LTS on x86_64 architecture:

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Focal/
cloudhsm-jce_latest_u20.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-jce_latest_u20.04_amd64.deb
```

Windows Server

Install the JCE provider for Windows Server on x86_64 architecture, open PowerShell as an administrator and run the following command:

```
PS C:\> wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/
Windows/AWSCloudHSMJCE-latest.msi -Outfile C:\AWSCloudHSMJCE-latest.msi
```

```
PS C:\> Start-Process msiexec.exe -ArgumentList '/i C:\AWSCloudHSMJCE-
latest.msi /quiet /norestart /log C:\client-install.txt' -Wait
```

2. Bootstrap Client SDK 5. For more information about bootstrapping, see Bootstrap the Client SDK.
3. Locate the following JCE provider files:

   Linux

   - `/opt/cloudhsm/java/cloudhsm-<version>.jar`
   - `/opt/cloudhsm/bin/configure-jce`

- `/opt/cloudhsm/bin/jce-info`

Windows

- `C:\Program Files\Amazon\CloudHSM\java\cloudhsm-<version>.jar>`
- `C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe`
- `C:\Program Files\Amazon\CloudHSM\bin\jce_info.exe`

Step 2: Provide credentials to the JCE provider

Before your Java application can use an HSM, the HSM needs to first authenticate the application. HSMs authenticate using either an explicit login or implicit login method.

Explicit login – This method lets you provide AWS CloudHSM credentials directly in the application. It uses the method from the `AuthProvider`, where you pass a CU username and password in the pin pattern. For more information, see Login to an HSM code example.

Implicit login – This method lets you set AWS CloudHSM credentials either in a new property file, system properties, or as environment variables.

- **System properties** – Set credentials through system properties when running your application. The following examples show two different ways that you can do this:

  Linux

  ```
  $ java -DHSM_USER=<HSM user name> -DHSM_PASSWORD=<password>
  ```

  ```
  System.setProperty("HSM_USER","<HSM user name>");
  System.setProperty("HSM_PASSWORD","<password>");
  ```

  Windows

  ```
  PS C:\> java -DHSM_USER=<HSM user name> -DHSM_PASSWORD=<password>
  ```

  ```
  System.setProperty("HSM_USER","<HSM user name>");
  System.setProperty("HSM_PASSWORD","<password>");
  ```

- **Environment variables** – Set credentials as environment variables.

Linux

```
$ export HSM_USER=<HSM user name>
$ export HSM_PASSWORD=<password>
```

Windows

```
PS C:\> $Env:HSM_USER="<HSM user name>"
PS C:\> $Env:HSM_PASSWORD="<password>"
```

Credentials might not be available if the application does not provide them or if you attempt an operation before the HSM authenticates session. In those cases, the CloudHSM software library for Java searches for the credentials in the following order:

1. System properties
2. Environment variables

Supported key types for JCE provider for AWS CloudHSM Client SDK 5

The AWS CloudHSM software library for Java enables you to generate the following key types.

Key Type	Description
AES	Generate 128, 192, and 256-bit AES keys.
Triple DES (3DES, DESede)	Generate a 192-bit Triple DES Key See footnote [1] for an upcoming change.
EC	Generate EC key pairs – NIST curves secp224r1 (P-224), secp256r1 (P-256), secp256k1 (Blockchain), secp384r1 (P-384), and secp521r1 (P-521).
GENERIC_SECRET	Generate 1 to 800 bytes generic secrets.
HMAC	Hash support for SHA1, SHA224, SHA256, SHA384, SHA512.

Key Type	Description
RSA	Generate 2048-bit to 4096-bit RSA keys, in increments of 256 bits.

[1] In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.

Key management basics in the JCE provider for AWS CloudHSM Client SDK 5

The basics on key management in the JCE provider involve importing keys, exporting keys, loading keys by handle, or deleting keys. For more information on managing keys, see the Manage keys code example.

You can also find more JCE provider code examples at Code samples.

Supported mechanisms for JCE provider for AWS CloudHSM Client SDK 5

This topic provides information about supported mechanisms for JCE provider with AWS CloudHSM Client SDK 5. For information about the Java Cryptography Architecture (JCA) interfaces and engine classes supported by AWS CloudHSM, see the following topics.

Topics

- Generate key and key pair functions
- Cipher functions
- Sign and verify functions
- Digest functions
- Hash-based message authentication code (HMAC) functions
- Cipher-based message authentication code (CMAC) functions
- Key Agreement Functions
- Convert keys to key specifications using key factories
- Mechanism annotations

Generate key and key pair functions

The AWS CloudHSM software library for Java allows you to use the following operations for generate key and key pair functions.

- `RSA`
- `EC`
- `AES`
- `DESede (Triple DES)` see note [1]
- `GenericSecret`

Cipher functions

The AWS CloudHSM software library for Java supports the following algorithm, mode, and padding combinations.

Algorithm	Mode	Padding	Notes

AES	CBC	`AES/CBC/NoPadding`
		
		`AES/CBC/PKCS5Padding`
			Implements `Cipher.ENCRYPT_MODE` and `Cipher.DECRYPT_MODE` .
			
			Implements `Cipher.UNWRAP_MODE for AES/CBC NoPadding`

AES	ECB	`AES/ECB/PKCS5Padding`
		
		`AES/ECB/NoPadding`
			Implements `Cipher.ENCRYPT_MODE` and `Cipher.DECRYPT_MODE` .

Algorithm	Mode	Padding	Notes
AES	CTR	AES/CTR/NoPadding	Implements Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE.
AES	GCM	AES/GCM/NoPadding	Implements Cipher.WRAP_MODE, Cipher.UNWRAP_MODE, Cipher.ENCRYPT_MODE, and Cipher.DECRYPT_MODE. When performing AES-GCM encryption, the HSM ignores the initialization vector (IV) in the request and uses an IV that it generates. When the operation completes, you must call Cipher.getIV() to get the IV.

Algorithm	Mode	Padding	Notes
AESWrap	ECB	AESWrap/ECB/NoPadding AESWrap/ECB/PKCS5Padding AESWrap/ECB/ZeroPadding	Implements Cipher.WRAP_MODE and Cipher.UNWRAP_MODE .
DESede (Triple DES)	CBC	DESede/CBC/PKCS5Padding DESede/CBC/NoPadding	Implements Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE . See note 1 below for an upcoming change.
DESede (Triple DES)	ECB	DESede/ECB/NoPadding DESede/ECB/PKCS5Padding	Implements Cipher.ENCRYPT_MODE and Cipher.DECRYPT_MODE . See note 1 below for an upcoming change.

Algorithm	Mode	Padding	Notes
RSA	ECB	RSA/ECB/PKCS1Padding **see note 1** RSA/ECB/OAEPPadding RSA/ECB/OAEPWithSHA-1ANDMGF1Padding RSA/ECB/OAEPWithSHA-224ANDMGF1Padding RSA/ECB/OAEPWithSHA-256ANDMGF1Padding RSA/ECB/OAEPWithSHA-384ANDMGF1Padding RSA/ECB/OAEPWithSHA-512ANDMGF1Padding	Implements Cipher.WRAP_MODE , Cipher.UNWRAP_MODE , Cipher.ENCRYPT_MODE , and Cipher.DECRYPT_MODE .

Algorithm	Mode	Padding	Notes
RSA	ECB	`RSA/ECB/NoPadding`	Implements `Cipher.ENCRYPT_MODE` and `Cipher.DECRYPT_MODE` .

RSAAESWrap	ECB	`RSAAESWrap/ECB/OAEPPadding`
		`RSAAESWrap/ECB/OAEPWithSHA-1ANDMGF1Padding`
		`RSAAESWrap/ECB/OAEPWithSHA-224ANDMGF1Padding`
		`RSAAESWrap/ECB/OAEPWithSHA-256ANDMGF1Padding`
		`RSAAESWrap/ECB/OAEPWithSHA-384ANDMGF1Padding`
		`RSAAESWrap/ECB/OAEPWithSHA-512ANDMGF1Padding`
			Implements `Cipher.WRAP_MODE` and `Cipher.UNWRAP_MODE` .

Sign and verify functions

The AWS CloudHSM software library for Java supports the following types of signature and verification. With Client SDK 5 and signature algorithms with hashing, the data is hashed locally in software before being sent to the HSM for the signature/verification. This means there is no limit on the size of the data that can be hashed by the SDK.

RSA Signature Types

- `NONEwithRSA`
- `RSASSA-PSS`
- `SHA1withRSA`
- `SHA1withRSA/PSS`
- `SHA1withRSAandMGF1`
- `SHA224withRSA`
- `SHA224withRSAandMGF1`
- `SHA224withRSA/PSS`
- `SHA256withRSA`
- `SHA256withRSAandMGF1`
- `SHA256withRSA/PSS`
- `SHA384withRSA`
- `SHA384withRSAandMGF1`
- `SHA384withRSA/PSS`
- `SHA512withRSA`
- `SHA512withRSAandMGF1`
- `SHA512withRSA/PSS`

ECDSA Signature Types

- `NONEwithECDSA`
- `SHA1withECDSA`
- `SHA224withECDSA`

- `SHA256withECDSA`
- `SHA384withECDSA`
- `SHA512withECDSA`

Digest functions

The AWS CloudHSM software library for Java supports the following message digests. With Client SDK 5, the data is hashed locally in software. This means there is no limit on the size of the data that can be hashed by the SDK.

- `SHA-1`
- `SHA-224`
- `SHA-256`
- `SHA-384`
- `SHA-512`

Hash-based message authentication code (HMAC) functions

The AWS CloudHSM software library for Java supports the following HMAC algorithms.

- `HmacSHA1` (Maximum data size in bytes: 16288)
- `HmacSHA224` (Maximum data size in bytes: 16256)
- `HmacSHA256` (Maximum data size in bytes: 16288)
- `HmacSHA384` (Maximum data size in bytes: 16224)
- `HmacSHA512` (Maximum data size in bytes: 16224)

Cipher-based message authentication code (CMAC) functions

CMACs (Cipher-based message authentication codes) create message authentication codes (MACs) using a block cipher and a secret key. They differ from HMACs in that they use a block symmetric key method for the MACs rather than a hashing method.

The AWS CloudHSM software library for Java supports the following CMAC algorithms.

- `AESCMAC`

Key Agreement Functions

The AWS CloudHSM software library for Java supports ECDH with Key Derivation Functions (KDF). The following KDF types are supported:

- `ECDHwithX963SHA1KDF` Supports X9.63 KDF SHA1 algorithm[2]
- `ECDHwithX963SHA224KDF` Supports X9.63 KDF SHA224 algorithm[2]
- `ECDHwithX963SHA256KDF` Supports X9.63 KDF SHA256 algorithm[2]
- `ECDHwithX963SHA384KDF` Supports X9.63 KDF SHA384 algorithm[2]
- `ECDHwithX963SHA512KDF` Supports X9.63 KDF SHA512 algorithm[2]

Convert keys to key specifications using key factories

You can use key factories to convert keys to key specifications. AWS CloudHSM has two types of key factories for JCE:

SecretKeyFactory: Used to import or derive symmetric keys. Using SecretKeyFactory, you can pass a supported Key or a supported KeySpec to import or derive symmetric keys into AWS CloudHSM. Following are the supported specs for KeyFactory:

- For SecretKeyFactory's `generateSecret` method following KeySpec classes are supported:
  - **KeyAttributesMap**can be used to import a key bytes with additional attributes as a CloudHSM Key. An example can be found here.
  - **SecretKeySpec**can be used to import a symmetric key spec as a CloudHSM Key.
  - **AesCmacKdfParameterSpec**can be used to derive symmetric keys using another CloudHSM AES Key.

> **Note**
>
> SecretKeyFactory's `translateKey` method takes any key that implements the key interface.

KeyFactory: Used for importing asymmetric keys. Using KeyFactory, you can pass a supported Key or supported KeySpec to import an asymmetric key into AWS CloudHSM. For more information, refer to the following resources:

- For KeyFactory's `generatePublic` method, following KeySpec classes are supported:
- CloudHSM KeyAttributesMap for both RSA and EC KeyTypes, including:
  - CloudHSM KeyAttributesMap for both RSA and EC public KeyTypes. An example can be found here
  - X509EncodedKeySpec for both RSA and EC Public Key
  - RSAPublicKeySpec for RSA Public Key
  - ECPublicKeySpec for EC Public Key
- For KeyFactory's `generatePrivate` method, following KeySpec classes are supported:
- CloudHSM KeyAttributesMap for both RSA and EC KeyTypes, including:
  - CloudHSM KeyAttributesMap for both RSA and EC public KeyTypes. An example can be found here
  - PKCS8EncodedKeySpec for both EC and RSA Private Key
  - RSAPrivateCrtKeySpec for RSA Private Key
  - ECPrivateKeySpec for EC Private Key

For KeyFactory's `translateKey` method, it takes in any Key that implements the Key Interface.

Mechanism annotations

[1] In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.

[2] Key derivation functions (KDFs) are specified in RFC 8418, Section 2.1.

Supported Java key attributes for AWS CloudHSM Client SDK 5

This topic provides information about supported Java key attributes for AWS CloudHSM Client SDK 5. This topic describes how to use a proprietary extension for the JCE provider to set key attributes. Use this extension to set supported key attributes and their values during these operations:

- Key generation
- Key import

For examples of how to use key attributes, see the section called “Code samples”.

Topics

- Understanding attributes
- Supported attributes
- Setting attributes for a key

Understanding attributes

Use key attributes to specify what actions are permitted on key objects, including public, private or secret keys. Key attributes and values are defined during key object creation operations.

The Java Cryptography Extension (JCE) does not specify how you should set values on key attributes, so most actions were permitted by default. In contrast, the PKCS# 11 standard defines a comprehensive set of attributes with more restrictive defaults. Starting with the JCE provider 3.1, AWS CloudHSM provides a proprietary extension that enables you to set more restrictive values for commonly used attributes.

Supported attributes

You can set values for the attributes listed in the following table. As a best practice, only set values for attributes you wish to make restrictive. If you don’t specify a value, AWS CloudHSM uses the default value specified in the table below. An empty cell in the Default Value columns indicates that there is no specific default value assigned to the attribute.

Attribute	Default Value			Notes
	Symmetric Key	Public Key in Key Pair	Private Key in Key Pair	
DECRYPT	TRUE		TRUE	True indicates you can use the key to decrypt any buffer. You generally set this to FALSE for a key whose

Attribute	Default Value			Notes
	Symmetric Key	Public Key in Key Pair	Private Key in Key Pair	
				WRAP is set to true.
DERIVE				Allows a key to be used to derive other keys.
ENCRYPT	TRUE	TRUE		True indicates you can use the key to encrypt any buffer.
EXTRACTABLE	TRUE		TRUE	True indicates you can export this key out of the HSM.
ID				A user-defined value used to identify the key.
KEY_TYPE				Used to identify the type of key (AES, DESede, generic secret, EC, or RSA).

Attribute	Default Value			Notes
	Symmetric Key	Public Key in Key Pair	Private Key in Key Pair	
LABEL				A user-defined string allowing you to conveniently identify keys on your HSM. To follow best practice, use a unique label for each key so it is easier to find later.
LOCAL				Indicates a key generated by the HSM.
OBJECT_CLASS				Used to identify the Object Class of a key (SecretKey, PublicKey or PrivateKey).

Attribute	Default Value			Notes
	Symmetric Key	Public Key in Key Pair	Private Key in Key Pair	
PRIVATE	TRUE	TRUE	TRUE	True indicates that a user may not access the key until the user is authenticated. For clarity, users cannot access any keys on AWS CloudHSM until they are authenticated, even if this attribute is set to FALSE.
SIGN	TRUE		TRUE	True indicates you can use the key to sign a message digest. This is generally set to FALSE for public keys and for private keys that you have archived.

Attribute	Default Value			Notes
	Symmetric Key	Public Key in Key Pair	Private Key in Key Pair	
SIZE				An attribute that defines the size of a key. For more details about supported key sizes, refer to Supported mechanisms for Client SDK 5.
TOKEN	FALSE	FALSE	FALSE	A permanent key which is replicated across all HSMs in the cluster and included in backups. TOKEN = FALSE implies an ephemeral key which is automatically erased when the connection to the HSM is broken or logged out.
UNWRAP	TRUE		TRUE	True indicates you can use the key to unwrap (import) another key.

Attribute	Default Value			Notes
	Symmetric Key	Public Key in Key Pair	Private Key in Key Pair	
`VERIFY`	`TRUE`	`TRUE`		True indicates you can use the key to verify a signature. This is generally set to FALSE for private keys.
`WRAP`	`TRUE`	`TRUE`		True indicates you can use the key to wrap another key. You will generally set this to FALSE for private keys.

Attribute	Default Value			Notes
	Symmetric Key	Public Key in Key Pair	Private Key in Key Pair	
WRAP_WITH _TRUSTED	FALSE		FALSE	True indicates a key can only be wrapped and unwrapped with keys that have the TRUSTED attribute set to true. Once a key has WRAP_WITH _TRUSTED set to true, that attribute is read-only and can't be set to false. To read about trust wrapping, see [Using trusted keys to control key unwraps]().

> **Note**
>
> You get broader support for attributes in the PKCS#11 library. For more information, see [Supported PKCS #11 Attributes]().

Setting attributes for a key

`KeyAttributesMap` is a Java Map-like object, which you can use to set attribute values for key objects. The methods for `KeyAttributesMap` function similar to the methods used for Java map manipulation.

To set custom values on attributes, you have two options:

- Use the methods listed in the following table
- Use builder patterns demonstrated later in this document

Attribute map objects support the following methods to set attributes:

Operation	Return Value	`KeyAttributesMap` method
Get the value of a key attribute for an existing key	Object (containing the value) or *null*	**get**(keyAttribute)
Populate the value of one key attribute	The previous value associate d with key attribute, or *null* if there was no mapping for a key attribute	**put**(keyAttribute, value)
Populate values for multiple key attributes	N/A	**putAll**(keyAttributesMap)
Remove a key-value pair from the attribute map	The previous value associate d with key attribute, or *null* if there was no mapping for a key attribute	**remove**(keyAttribute)

> **Note**
>
> Any attributes you do not explicitly specify are set to the defaults listed in the preceding table in the section called “Supported attributes”.

Setting attributes for a key pair

Use the Java class `KeyPairAttributesMap` to handle key attributes for a key pair. `KeyPairAttributesMap` encapsulates two `KeyAttributesMap` objects; one for a public key and one for a private key.

To set individual attributes for the public key and private key separately, you can use the `put()` method on corresponding `KeyAttributes` map object for that key. Use the `getPublic()` method to retrieve the attribute map for the public key, and use `getPrivate()` to retrieve the attribute map for the private key. Populate the value of multiple key attributes together for both public and private key pairs using the `putAll()` with a key pair attributes map as its argument.

Code samples for the AWS CloudHSM software library for Java for Client SDK 5

This topic provides resources and information on Java code samples for AWS CloudHSM Client SDK 5.

Prerequisites

Before running the samples, you must set up your environment:

- Install and configure the Java Cryptographic Extension (JCE) provider.
- Set up a valid HSM user name and password. Cryptographic user (CU) permissions are sufficient for these tasks. Your application uses these credentials to log in to the HSM in each example.
- Decide how to provide credentials to the JCE provider.

Code samples

The following code samples show you how to use the AWS CloudHSM JCE provider to perform basic tasks. More code samples are available on GitHub.

- Log in to an HSM
- Manage keys
- Generate Symmetric Keys
- Generate Asymmetric Keys
- Encrypt and decrypt with AES-GCM
- Encrypt and decrypt with AES-CTR
- Encrypt and decrypt with DESede-ECB see note [1]
- Sign and Verify with RSA Keys
- Sign and Verify with EC Keys
- Use supported key attributes
- Use the CloudHSM key store

[1] In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.

AWS CloudHSM JCE provider Javadocs

Use the JCE provider Javadocs to get usage information on Java types and methods defined in the AWS CloudHSM JCE SDK. To download the latest Javadocs for AWS CloudHSM, see the AWS CloudHSM latest Client SDK release section on the Downloads page.

You can import Javadocs into an integrated development environment (IDE) or view them in a web browser.

AWS CloudHSM KeyStore Java class for Client SDK 5

The AWS CloudHSM `KeyStore` class provides a special-purpose PKCS12 key store. This key store can store certificates along with your key data and correlate them to key data stored on AWS CloudHSM. The AWS CloudHSM `KeyStore` class implements the `KeyStore` Service Provider Interface (SPI) of the Java Cryptography Extension (JCE). For more information about using `KeyStore`, see Class KeyStore.

> **Note**
>
> Because certificates are public information, and to maximize storage capacity for cryptographic keys, AWS CloudHSM does not support storing certificates on HSMs.

Choose the appropriate key store for AWS CloudHSM Client SDK 5

The AWS CloudHSM Java Cryptographic Extension (JCE) provider offers a special-purpose AWS CloudHSM KeyStore. The AWS CloudHSM `KeyStore` class supports offloading key operations to the HSM, local storage of certificates and certificate-based operations.

Load the special-purpose CloudHSM KeyStore as follows:

```
KeyStore ks = KeyStore.getInstance("CloudHSM")
```

Initialize the AWS CloudHSM KeyStore Client SDK 5

Log into the AWS CloudHSM KeyStore the same way that you log into the JCE provider. You can use either environment variables or the system property file, and you should log in before you start using the CloudHSM KeyStore. For an example of logging into an HSM using the JCE provider, see Login to an HSM.

If desired, you can specify a password to encrypt the local PKCS12 file which holds key store data. When you create the AWS CloudHSM Keystore, you set the password and provide it when using the load, set and get methods.

Instantiate a new CloudHSM KeyStore object as follows:

```
ks.load(null, null);
```

Write keystore data to a file using the `store` method. From that point on, you can load the existing keystore using the `load` method with the source file and password as follows:

```
ks.load(inputStream, password);
```

Use AWS CloudHSM KeyStore or AWS CloudHSM Client SDK 5

AWS CloudHSM KeyStore complies with the JCE Class KeyStore specification and provides the following functions.

- `load`

  Loads the key store from the given input stream. If a password was set when saving the key store, this same password must be provided for the load to succeed. Set both parameters to null to initialize an new empty key store.

  ```
  KeyStore ks = KeyStore.getInstance("CloudHSM");
  ks.load(inputStream, password);
  ```

- `aliases`

  Returns an enumeration of the alias names of all entries in the given key store instance. Results include objects stored locally in the PKCS12 file and objects resident on the HSM.

  Sample code:

  ```
  KeyStore ks = KeyStore.getInstance("CloudHSM");
  for(Enumeration<String> entry = ks.aliases(); entry.hasMoreElements();) {
      String label = entry.nextElement();
  ```

```
    System.out.println(label);
}
```

- `containsalias`

  Returns true if the key store has access to at least one object with the specified alias. The key store checks objects stored locally in the PKCS12 file and objects resident on the HSM.

- `deleteEntry`

  Deletes a certificate entry from the local PKCS12 file. Deleting key data stored in an HSM is not supported using the AWS CloudHSM KeyStore. You can delete keys using the `destroy` method of the [Destroyable](https://docs.oracle.com/javase/8/docs/api/javax/security/auth/Destroyable.html) interface.

```
((Destroyable) key).destroy();
```

- `getCertificate`

  Returns the certificate associated with an alias if available. If the alias does not exist or references an object which is not a certificate, the function returns NULL.

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
Certificate cert = ks.getCertificate(alias);
```

- `getCertificateAlias`

  Returns the name (alias) of the first key store entry whose data matches the given certificate.

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
String alias = ks.getCertificateAlias(cert);
```

- `getCertificateChain`

  Returns the certificate chain associated with the given alias. If the alias does not exist or references an object which is not a certificate, the function returns NULL.

- `getCreationDate`

  Returns the creation date of the entry identified by the given alias. If a creation date is not available, the function returns the date on which the certificate became valid.

- `getKey`

GetKey is passed to the HSM and returns a key object corresponding to the given label. As `getKey` directly queries the HSM, it can be used for any key on the HSM regardless of whether it was generated by the KeyStore.

```
Key key = ks.getKey(keyLabel, null);
```

- `isCertificateEntry`

  Checks if the entry with the given alias represents a certificate entry.

- `isKeyEntry`

  Checks if the entry with the given alias represents a key entry. The action searches both the PKCS12 file and the HSM for the alias.

- `setCertificateEntry`

  Assigns the given certificate to the given alias. If the given alias is already being used to identify a key or certificate, a `KeyStoreException` is thrown. You can use JCE code to get the key object and then use the KeyStore `SetKeyEntry` method to associate the certificate to the key.

- `setKeyEntry` with `byte[]` key

  This API is currently unsupported with Client SDK 5.

- `setKeyEntry` with `Key` object

  Assigns the given key to the given alias and stores it inside the HSM. If the key does not already exist inside the HSM, it will be imported into the HSM as an extractable session key.

  If the `Key` object is of type `PrivateKey`, it must be accompanied by a corresponding certificate chain.

  If the alias already exists, the `SetKeyEntry` call throws a `KeyStoreException` and prevents the key from being overwritten. If the key must be overwritten, use KMU or JCE for that purpose.

- `engineSize`

  Returns the number of entries in the keystore.

- `store`

  Stores the key store to the given output stream as a PKCS12 file and secures it with the given password. In addition, it persists all loaded keys (which are set using `setKey` calls).

Advanced configurations for AWS CloudHSM JCE for Client SDK 5

The AWS CloudHSM JCE provider includes the following advanced configurations, which are not part of the general configurations most customers utilize.

- Connecting to multiple clusters
- Key extraction using JCE
- Retry configuration for JCE

Connecting to multiple AWS CloudHSM clusters with the JCE provider

This configuration allows a single client instance to communicate to multiple AWS CloudHSM clusters. Compared to having a single instance only communicate with a single cluster, this can be a cost-savings feature for some use cases. The `CloudHsmProvider` class is AWS CloudHSM's implementation of Java Security's Provider class. Each instance of this class represents a connection to your entire AWS CloudHSM cluster. You instantiate this class and add it to Java Security provider's list so that you can interact with it using standard JCE classes.

The following example instantiates this class and adds it to Java Security provider's list:

```
if (Security.getProvider(CloudHsmProvider.PROVIDER_NAME) == null) {
    Security.addProvider(new CloudHsmProvider());
}
```

`CloudHsmProvider` can be configured in two ways:

1. Configure with file (default configuration)
2. Configure using code

The following topics describe these configurations, and how to connect to multiple clusters.

Topics

- Configure the AWS CloudHSMCloudHsmProvider class with a file (Default configuration)
- Configure the AWS CloudHSMCloudHsmProvider class using code
- Connect to multiple AWS CloudHSM clusters

Configure the AWS CloudHSM`CloudHsmProvider` class with a file (Default configuration)

The default way to configure the AWS CloudHSM `CloudHsmProvider` class is with a file.

When you instantiate `CloudHsmProvider` using default constructor, by default it will look for configuration file in `/opt/cloudhsm/etc/cloudhsm-jce.cfg` path in Linux. This configuration file can be configured using the `configure-jce`.

An object created using the default constructor will use the default CloudHSM provider name `CloudHSM`. The provider name is useful to interact with JCE to let it know which provider to use for various operation. An example to use CloudHSM provider name for Cipher operation is as below:

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding", "CloudHSM");
```

Configure the AWS CloudHSM`CloudHsmProvider` class using code

As of Client SDK version 5.8.0, you can also configure the AWS CloudHSM `CloudHsmProvider` class using Java code. The way to do this is using an object of `CloudHsmProviderConfig` class. You can build this object using `CloudHsmProviderConfigBuilder`.

`CloudHsmProvider` has another constructor which takes the `CloudHsmProviderConfig` object, as the following sample shows.

Example

```
CloudHsmProviderConfig config = CloudHsmProviderConfig.builder()
                                    .withCluster(
                                        CloudHsmCluster.builder()
                                            .withHsmCAFilePath(hsmCAFilePath)

 .withClusterUniqueIdentifier("CloudHsmCluster1")
        .withServer(CloudHsmServer.builder().withHostIP(hostName).build())
                        .build())
        .build();
CloudHsmProvider provider = new CloudHsmProvider(config);
```

In this example, the name of the JCE provider is `CloudHsmCluster1`. This is the name that application can then use to interact with JCE:

Example

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding", "CloudHsmCluster1");
```

Alternatively, applications can also use the provider object created above to let JCE know to use that provider for the operation:

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding", provider);
```

If a unique identifier is not specified with the `withClusterUniqueIdentifier` method, a randomly generated provider name is created for you. To get this randomly generated identifier, applications can call `provider.getName()` to get the identifier.

Connect to multiple AWS CloudHSM clusters

Each `CloudHsmProvider` represents a connection to your AWS CloudHSM Cluster. If you want to talk to another cluster from the same application, you can create another object of `CloudHsmProvider` with configurations for your other cluster and you can interact with this other cluster either using the provider object or using the provider name, as shown in the following example.

Example

```
CloudHsmProviderConfig config = CloudHsmProviderConfig.builder()
                                    .withCluster(
                                        CloudHsmCluster.builder()
                                            .withHsmCAFilePath(hsmCAFilePath)

 .withClusterUniqueIdentifier("CloudHsmCluster1")
        .withServer(CloudHsmServer.builder().withHostIP(hostName).build())
                        .build())
        .build();
CloudHsmProvider provider1 = new CloudHsmProvider(config);

if (Security.getProvider(provider1.getName()) == null) {
    Security.addProvider(provider1);
}

CloudHsmProviderConfig config2 = CloudHsmProviderConfig.builder()
                                    .withCluster(
                                        CloudHsmCluster.builder()
```

```
                                            .withHsmCAFilePath(hsmCAFilePath2)

 .withClusterUniqueIdentifier("CloudHsmCluster2")
        .withServer(CloudHsmServer.builder().withHostIP(hostName2).build())
                        .build())
        .build();
CloudHsmProvider provider2 = new CloudHsmProvider(config2);

if (Security.getProvider(provider2.getName()) == null) {
    Security.addProvider(provider2);
}
```

Once you have configured both the providers (both the clusters) above, you can interact with them either using the provider object or using the provider name.

Expanding upon this example that shows how to talk to `cluster1`, you could use the following sample for a AES/GCM/NoPadding operation:

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding", provider1);
```

And in the same application to do "AES" Key generation on the second cluster using the provider name, you could also use the following sample:

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding", provider2.getName());
```

Key extraction using JCE for AWS CloudHSM

The Java Cryptography Extension (JCE) uses an architecture that allows different cryptography implementations to be plugged in. AWS CloudHSM ships one such JCE provider that offloads cryptographic operations to the HSM. For most other JCE providers to work with keys stored in AWS CloudHSM, they must extract the key bytes from your HSMs in clear text into your machine's memory for their use. HSMs typically only allow keys to be extracted as wrapped objects, not clear text. However, to support inter-provider integration use cases, AWS CloudHSM allows an opt-in configuration option to enable extraction of the key bytes in the clear.

> **Important**
>
> JCE offloads operations to AWS CloudHSM whenever the AWS CloudHSM provider is specified or an AWS CloudHSM key object is used. You do not need to extract keys in clear if you expect your operation to happen inside the HSM. Key extraction in clear text is

only needed when your application cannot use secure mechanisms such as wrapping and unwrapping a key due to restrictions from a third party library or JCE provider.

The AWS CloudHSM JCE Provider allows extraction of **public keys** to work with external JCE providers by default. The following methods are always allowed:

Class	Method	Format (getEncoded)
EcPublicKey	getEncoded()	X.509
	getW()	N/A
RSAPublicKey	getEncoded()	X.509
	getPublicExponent()	N/A
CloudHsmRsaPrivateCrtKey	getPublicExponent()	N/A

The AWS CloudHSM JCE Provider doesn't allow extraction of key bytes in clear for the **private** or **secret** keys by default. If your use case requires it, you can enable extraction of key bytes in clear for **private** or **secret** keys under the following conditions:

1. The `EXTRACTABLE` attribute for private and secret keys is set to **true**.
   - By default, the `EXTRACTABLE` attribute for private and secret keys is set to **true**. `EXTRACTABLE` keys are keys that are permitted to be exported out of the HSM. For more information see Supported Java attributes for Client SDK 5.
2. The `WRAP_WITH_TRUSTED` attribute for the private and secret keys is set to **false**.
   - `getEncoded`, `getPrivateExponent`, and `getS` cannot be used with private keys that cannot be exported in clear. `WRAP_WITH_TRUSTED` doesn't allow your private keys to exported out of the HSM in clear. For more information see Using trusted keys to control key unwraps.

Allow the JCE provider to extract private key secrets out of AWS CloudHSM

Use the following steps to allow AWS CloudHSM JCE provider to extract your private key secrets.

> **Important**
>
> This configuration change allows extraction of all `EXTRACTABLE` key bytes in clear from your HSM cluster. For better security, you should consider using key wrapping methods to extract the key out of the HSM securely. This prevents unintentional extraction of your key bytes from the HSM.

1. Use the following commands to enable your **private** or **secret** keys to be extracted in JCE:

   Linux

   ```
   $ /opt/cloudhsm/bin/configure-jce --enable-clear-key-extraction-in-software
   ```

   Windows

   ```
   PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" --enable-
   clear-key-extraction-in-software
   ```

2. Once you enable your clear key extraction, the following methods are enabled for extracting private keys into memory.

Class	Method	Format (getEncoded)
Key	getEncoded()	RAW
ECPrivateKey	getEncoded()	PKCS#8
	getS()	N/A
RSAPrivateCrtKey	getEncoded()	X.509
	getPrivateExponent()	N/A
	getPrimeP()	N/A
	getPrimeQ()	N/A

Class	Method	Format (getEncoded)
	getPrimeExponentP()	N/A
	getPrimeExponentQ()	N/A
	getCrtCoefficient()	N/A

If you want restore the default behavior and not allow JCE to export keys in clear, run the following command:

Linux

```
$ /opt/cloudhsm/bin/configure-jce --disable-clear-key-extraction-in-software
```

Windows

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" --disable-clear-key-extraction-in-software
```

Retry commands for JCE for AWS CloudHSM

AWS CloudHSM Client SDK 5.8.0 and later have a built-in automatic retry strategy which will retry HSM-throttled operations from the client side. When an HSM throttles operations because it is too busy performing previous operations and cannot take more requests, client SDKs will attempt to retry throttled operations up to 3 times while exponentially backing off. This automatic retry strategy can be set to one of two modes: **off** and **standard**.

- **off**: The Client SDK will not perform any retry strategy for any throttled operations by the HSM.
- **standard**: This is the default mode for Client SDK 5.8.0 and later. In this mode, client SDKs will automatically retry throttled operations by exponentially backing off.

For more information, see HSM throttling.

Set retry commands to off mode

Linux

To set retry commands to off for Client SDK 5 on Linux

- You can use the following command to set retry configuration to **off** mode:

```
$ sudo /opt/cloudhsm/bin/configure-jce --default-retry-mode off
```

Windows

To set retry commands to off for Client SDK 5 on Windows

- You can use the following command to set retry configuration to **off** mode:

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\configure-jce.exe" --default-retry-mode off
```

Using previous SDK version to work with AWS CloudHSM

SDK versions 5.8.0 and earlier have reached their end of support. After March 31, 2025 documentation for SDK versions 3.4.4 and earlier will no longer be available.

AWS CloudHSM includes two major Client SDK versions:

- Client SDK 5: This is our latest and default Client SDK. For information on the benefits and advantages it provides, see Benefits of AWS CloudHSM Client SDK 5.
- Client SDK 3: This is our older Client SDK. It includes a full set of components for platform and language-based applications compatibility and management tools.

For instructions on migrating from Client SDK 3 to Client SDK 5, see Migrating from AWS CloudHSM Client SDK 3 to Client SDK 5.

This topic describes Client SDK 3. To see what version of Client SDK you're using, see Check your AWS CloudHSM Client SDK version.

To download, see *Downloads*.

Topics

Upgrade AWS CloudHSM Client SDK 3 on Linux

SDK versions 5.8.0 and earlier have reached their end of support. After March 31, 2025 documentation for SDK versions 3.4.4 and earlier will no longer be available.

With AWS CloudHSM Client SDK 3.1 and higher, the version of the client daemon and any components you install must match to upgrade. For all Linux-based systems, you must use a single command to batch upgrade the client daemon with the same version of the PKCS #11 library, the Java Cryptographic Extension (JCE) provider, or the OpenSSL Dynamic Engine. This requirement does not apply to Windows-based systems because the binaries for the CNG and KSP providers are already included in the client daemon package.

To check the client daemon version

- On a Red Hat-based Linux system (including Amazon Linux and CentOS), use the following command:

```
rpm -qa | grep ^cloudhsm
```

- On an Debian-based Linux system, use the following command:

```
apt list --installed | grep ^cloudhsm
```

- On a Windows system, use the following command:

```
wmic product get name,version
```

Topics

- Prerequisites
- Step 1: Stop the client daemon
- Step 2: Upgrade the client SDK
- Step 3: Start the client daemon

Prerequisites

Download the latest version of AWS CloudHSM client daemon and choose your components.

> **Note**
>
> You do not have to install all the components. For every component you have installed, you must upgrade that component to match the version of the client daemon.

Latest Linux client daemon

Amazon Linux

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 7

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

CentOS 8

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-latest.el8.x86_64.rpm
```

RHEL 7

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-latest.el7.x86_64.rpm
```

RHEL 8

```
sudo yum install wget
```

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client_latest_u18.04_amd64.deb
```

Latest PKCS #11 library

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-pkcs11-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

CentOS 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

RHEL 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-pkcs11_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client-pkcs11_latest_u18.04_amd64.deb
```

Latest OpenSSL Dynamic Engine

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-dyn-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-dyn_latest_amd64.deb
```

Latest JCE provider

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-jce-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm
```

CentOS 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-jce-latest.el8.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm
```

RHEL 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-jce-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-jce_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client-jce_latest_u18.04_amd64.deb
```

Step 1: Stop the client daemon

Use the following command to stop the client daemon.

Amazon Linux

```
$ sudo stop cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client stop
```

CentOS 7

```
$ sudo service cloudhsm-client stop
```

CentOS 8

```
$ sudo service cloudhsm-client stop
```

RHEL 7

```
$ sudo service cloudhsm-client stop
```

RHEL 8

```
$ sudo service cloudhsm-client stop
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client stop
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client stop
```

Step 2: Upgrade the client SDK

The following command shows the syntax required to upgrade the client daemon and components. Before you run the command, remove any components you don't intend to upgrade.

Amazon Linux

```
$ sudo yum install ./cloudhsm-client-latest.el6.x86_64.rpm \
              <./cloudhsm-client-pkcs11-latest.el6.x86_64.rpm> \
              <./cloudhsm-client-dyn-latest.el6.x86_64.rpm> \
              <./cloudhsm-client-jce-latest.el6.x86_64.rpm>
```

Amazon Linux 2

```
$ sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm \
              <./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm> \
              <./cloudhsm-client-dyn-latest.el7.x86_64.rpm> \
              <./cloudhsm-client-jce-latest.el7.x86_64.rpm>
```

CentOS 7

```
$ sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm \
              <./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm> \
              <./cloudhsm-client-dyn-latest.el7.x86_64.rpm> \
              <./cloudhsm-client-jce-latest.el7.x86_64.rpm>
```

CentOS 8

```
$ sudo yum install ./cloudhsm-client-latest.el8.x86_64.rpm \
              <./cloudhsm-client-pkcs11-latest.el8.x86_64.rpm> \
              <./cloudhsm-client-jce-latest.el8.x86_64.rpm>
```

RHEL 7

```
$ sudo yum install ./cloudhsm-client-latest.el7.x86_64.rpm \
              <./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm> \
              <./cloudhsm-client-dyn-latest.el7.x86_64.rpm> \
              <./cloudhsm-client-jce-latest.el7.x86_64.rpm>
```

RHEL 8

```
$ sudo yum install ./cloudhsm-client-latest.el8.x86_64.rpm \
              <./cloudhsm-client-pkcs11-latest.el8.x86_64.rpm> \
              <./cloudhsm-client-jce-latest.el8.x86_64.rpm>
```

Ubuntu 16.04 LTS

```
$ sudo apt install ./cloudhsm-client_latest_amd64.deb \
              <cloudhsm-client-pkcs11_latest_amd64.deb> \
              <cloudhsm-client-dyn_latest_amd64.deb> \
              <cloudhsm-client-jce_latest_amd64.deb>
```

Ubuntu 18.04 LTS

```
$ sudo apt install ./cloudhsm-client_latest_u18.04_amd64.deb \
              <cloudhsm-client-pkcs11_latest_amd64.deb> \
              <cloudhsm-client-jce_latest_amd64.deb>
```

Step 3: Start the client daemon

Use the following command to start the client daemon.

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 20.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 22.04 LTS

Support for OpenSSL Dynamic Engine is not yet available.

AWS CloudHSM Client SDK 3 supported platforms

> SDK versions 5.8.0 and earlier have reached their end of support. After March 31, 2025 documentation for SDK versions 3.4.4 and earlier will no longer be available.

AWS CloudHSM Client SDK 3 requires a client daemon and offers command-line tools including, CloudHSM Management Utility (CMU), key management utility (KMU), and the configure tool.

Base support is different for each version of the AWS CloudHSM Client SDK. Typically platform support for components in an SDK matches base support, but not always. To determine platform support for a given component, first make sure the platform you want appears in the base section for the SDK, then check for any exclusions or any other pertinent information in the component section.

Platform support changes over time. Earlier versions of the CloudHSM Client SDK may not support all the operating systems listed here. Use release notes to determine the operating system support for previous versions of the CloudHSM Client SDK. For more information, see Downloads for AWS CloudHSM Client SDK.

AWS CloudHSM supports only 64-bit operating systems.

Topics

- Linux support for AWS CloudHSM Client SDK 3
- Windows support for AWS CloudHSM Client SDK 3
- HSM compatibility for AWS CloudHSM Client SDK 3

Linux support for AWS CloudHSM Client SDK 3

AWS CloudHSM Client SDK 3 supports the following Linux operating systems and platforms.

- Amazon Linux
- Amazon Linux 2
- CentOS 6.10+ [2]
- CentOS 7.3+
- CentOS 8 [1,4]
- Red Hat Enterprise Linux (RHEL) 6.10+ [2]
- Red Hat Enterprise Linux (RHEL) 7.3+
- Red Hat Enterprise Linux (RHEL) 8 [1]
- Ubuntu 16.04 LTS [3]
- Ubuntu 18.04 LTS [1]

[1] No support for OpenSSL Dynamic Engine. For more information, see OpenSSL Dynamic Engine.

[2] No support for Client SDK 3.3.0 and later.

[3] SDK 3.4 is the last supported release on Ubuntu 16.04.

[4] SDK 3.4 is the last supported release on CentOS 8.3+.

Windows support for AWS CloudHSM Client SDK 3

AWS CloudHSM Client SDK 3 supports the following versions of Windows Server.

- Microsoft Windows Server 2012
- Microsoft Windows Server 2012 R2
- Microsoft Windows Server 2016
- Microsoft Windows Server 2019

HSM compatibility for AWS CloudHSM Client SDK 3

The following table describes AWS CloudHSM Client SDK 3 compatibility for HSMs.

hsm1.medium	hsm2m.medium
Compatible with Client version SDK 3.1.0 and later.	Not supported.

PKCS #11 library for AWS CloudHSM Client SDK 3

PKCS #11 is a standard for performing cryptographic operations on hardware security modules (HSM) in AWS CloudHSM.

For information about bootstrapping, see Connecting to the cluster.

Topics

Install the PKCS #11 library for AWS CloudHSM Client SDK 3

This topic provides instructions for installing the PKCS #11 library for the AWS CloudHSM Client SDK 3 version series. For more information about the Client SDK or PKCS #11 library, see Using the Client SDK and PKCS #11 library.

Prerequisites for Client SDK 3

The PKCS #11 library requires the AWS CloudHSM client.

If you haven't installed and configured the AWS CloudHSM client, do that now by following the steps at Install the client (Linux). After you install and configure the client, use the following command to start it.

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo systemctl cloudhsm-client start
```

CentOS 7

```
$ sudo systemctl cloudhsm-client start
```

CentOS 8

```
$ sudo systemctl cloudhsm-client start
```

RHEL 7

```
$ sudo systemctl cloudhsm-client start
```

RHEL 8

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 20.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Install the PKCS #11 library for Client SDK 3

The following command downloads and installs the PKCS #11 library.

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-pkcs11-latest.el6.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

CentOS 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el7.x86_64.rpm
```

RHEL 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-pkcs11-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-pkcs11_latest_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-pkcs11_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client-pkcs11_latest_u18.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-pkcs11_latest_u18.04_amd64.deb
```

- If the EC2 instance on which you installed the PKCS #11 library has no other components from Client SDK 3 installed, you must bootstrap Client SDK 3. You only have to do this once on each instance with a component from Client SDK 3.
- You can find the PKCS #11 library files in the following locations:

  Linux binaries, configuration scripts, certificates, and log files:

  ```
  /opt/cloudhsm/lib
  ```

Authenticate to the PKCS #11 library for AWS CloudHSM Client SDK 3

When you use the PKCS #11 library, your application runs as a particular crypto user (CU) in your HSMs in AWS CloudHSM. Your application can view and manage only the keys that the CU owns and shares. You can use an existing CU in your HSMs or create a new CU. For information on managing CUs, see Managing HSM users with CloudHSM CLI and Managing HSM users with CloudHSM Management Utility (CMU).

To specify the CU to PKCS #11 library, use the pin parameter of the PKCS #11 C_Login function. For AWS CloudHSM, the pin parameter has the following format:

```
<CU_user_name>:<password>
```

For example, the following command sets the PKCS #11 library pin to the CU with user name `CryptoUser` and password `CUPassword123!`.

```
CryptoUser:CUPassword123!
```

Supported key types for PKCS #11 library for AWS CloudHSM Client SDK 3

The PKCS #11 library supports the following key types with AWS CloudHSM Client SDK 3.

Key Type	Description
RSA	Generate 2048-bit to 4096-bit RSA keys, in increments of 256 bits.
EC	Generate keys with the secp224r1 (P-224), secp256r1 (P-256), secp256k1 (Blockchain), secp384r1 (P-384), and secp521r1 (P-521) curves.
AES	Generate 128, 192, and 256-bit AES keys.
DES3 (Triple DES)	Generate 192-bit DES3 keys. See note 1 below for an upcoming change.
GENERIC_SECRET	Generate 1 to 64 bytes generic secrets.

- [1] In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.

Supported mechanisms for AWS CloudHSM Client SDK 3

The PKCS #11 library supports the following algorithms for AWS CloudHSM Client SDK 3:

- **Encryption and decryption –** AES-CBC, AES-CTR, AES-ECB, AES-GCM, DES3-CBC, DES3-ECB, RSA-OAEP, and RSA-PKCS
- **Sign and verify –** RSA, HMAC, and ECDSA; with and without hashing
- **Hash/digest –** SHA1, SHA224, SHA256, SHA384, and SHA512
- **Key wrap –** AES Key Wrap,[4] AES-GCM, RSA-AES, and RSA-OAEP
- **Key derivation –** ECDH,[5] SP800-108 CTR KDF

The PKCS #11 library mechanism-function table

The PKCS #11 library is compliant with version 2.40 of the PKCS #11 specification. To invoke a cryptographic feature using PKCS #11, call a function with a given mechanism. The following table summarizes the combinations of functions and mechanisms supported by AWS CloudHSM.

Interpreting the supported PKCS #11 mechanism-function table

A ✔ mark indicates that AWS CloudHSM supports the mechanism for the function. We do not support all possible functions listed in the PKCS #11 specification. A ✖ mark indicates that AWS CloudHSM does not yet support the mechanism for the given function, even though the PKCS #11 standard allows it. Empty cells indicate that PKCS #11 standard does not support the mechanism for the given function.

Supported PKCS #11 library mechanisms and functions

Mechanism	Functions						
		Generate Key or Key Pair	Sign & Verify	SR & VR	Digest	Encrypt & Decrypt	Derive Key	Wrap & UnWrap

Mechanism	Functions						
CKM_RSA_PKCS_KEY_PAIR_GEN	✔						
CKM_RSA_X9_31_KEY_PAIR_GEN	✔[2]						
CKM_RSA_X_509		✔			✔		
CKM_RSA_PKCS see note 8		✔[1]	✖		✔[1]		✔[1]
CKM_RSA_PKCS_OAEP					✔[1]		✔[6]
CKM_SHA1_RSA_PKCS		✔[3.2]					
CKM_SHA224_RSA_PKCS		✔[3.2]					
CKM_SHA256_RSA_PKCS		✔[3.2]					
CKM_SHA384_RSA_PKCS		✔[2,3.2]					
CKM_SHA512_RSA_PKCS		✔[3.2]					

Mechanism	Functions							
CKM_RSA_PKCS_PSS			✓[1]					
CKM_SHA1_RSA_PKCS_PSS			✓[3.2]					
CKM_SHA224_RSA_PKCS_PSS			✓[3.2]					
CKM_SHA256_RSA_PKCS_PSS			✓[3.2]					
CKM_SHA384_RSA_PKCS_PSS			✓[2,3.2]					
CKM_SHA512_RSA_PKCS_PSS			✓[3.2]					
CKM_EC_KEY_PAIR_GEN		✓						
CKM_ECDSA			✓[1]					
CKM_ECDSA_SHA1			✓[3.2]					
CKM_ECDSA_SHA224			✓[3.2]					

Mechanism	Functions							
CKM_ECDSA_SHA256			✓[3.2]					
CKM_ECDSA_SHA384			✓[3.2]					
CKM_ECDSA_SHA512			✓[3.2]					
CKM_ECDH1_DERIVE							✓[5]	
CKM_SP800_108_COUNTER_KDF							✓	
CKM_GENERIC_SECRET_KEY_GEN		✓						
CKM_AES_KEY_GEN		✓						
CKM_AES_ECB						✓		✖
CKM_AES_CTR						✓		✖
CKM_AES_CBC						✓[3.3]		✖
CKM_AES_CBC_PAD						✓		✖

Mechanism	Functions						
`CKM_DES3_KEY_GEN` **see note 8**	✔						
`CKM_DES3_CBC` **see note 8**					✔[3.3]		✖
`CKM_DES3_CBC_PAD` **see note 8**					✔		✖
`CKM_DES3_ECB` **see note 8**					✔		✖
`CKM_AES_GCM`					✔[3.3, 4]		✔[7.1]
`CKM_CLOUDHSM_AES_GCM`					✔[7.1]		✔[7.1]
`CKM_SHA_1`				✔[3.1]			
`CKM_SHA_1_HMAC`		✔[3.3]					
`CKM_SHA224`				✔[3.1]			
`CKM_SHA224_HMAC`		✔[3.3]					

Mechanism	Functions							
CKM_SHA256					✓[3.1]			
CKM_SHA256_HMAC			✓[3.3]					
CKM_SHA384					✓[3.1]			
CKM_SHA384_HMAC			✓[3.3]					
CKM_SHA512					✓[3.1]			
CKM_SHA512_HMAC			✓[3.3]					
CKM_RSA_AES_KEY_WRAP								✓
CKM_AES_KEY_WRAP								✓
CKM_AES_KEY_WRAP_PAD								✓
CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD								✓[7.1]

Mechanism	Functions							
CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD								✓[7.1]
CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD								✓[7.1]

Mechanism annotations

- [1] Single-part operations only.
- [2] Mechanism is functionally identical to the CKM_RSA_PKCS_KEY_PAIR_GEN mechanism, but offers stronger guarantees for p and q generation.
- [3.1] AWS CloudHSM approaches hashing differently based on the Client SDK. For Client SDK 3, where we do the hashing depends on data size and whether you're using single-part or multipart operations.

  Single-part operations in Client SDK 3

  Table 3.1 lists the maximum data set size for each mechanism for Client SDK 3. The entire hash is computed inside the HSM. No support for data sizes greater than 16KB.

  Table 3.1, Maximum data set size for single-part operations

Mechanism	Maximum Data Size
CKM_SHA_1	16296
CKM_SHA224	16264
CKM_SHA256	16296
CKM_SHA384	16232

Mechanism	Maximum Data Size
CKM_SHA512	16232

Multipart operations Client SDK 3

Support for data sizes greater than 16 KB, but data size determines where the hashing takes place. Data buffers less than 16 KB are hashed inside the HSM. Buffers between 16 KB and the maximum data size for your system are hashed locally in software. *Remember*: Hash functions do not require cryptographic secrets, so you can safely compute them outside of the HSM.

- [3.2] AWS CloudHSM approaches hashing differently based on the Client SDK. For Client SDK 3, where we do the hashing depends on data size and whether you're using single-part or multipart operations.

Single-part operations Client SDK 3

Table 3.2 lists the maximum data set size for each mechanism for Client SDK 3. No support for data sizes greater than 16KB.

Table 3.2, Maximum data set size for single-part operations

Mechanism	Maximum Data Size
CKM_SHA1_RSA_PKCS	16296
CKM_SHA224_RSA_PKCS	16264
CKM_SHA256_RSA_PKCS	16296
CKM_SHA384_RSA_PKCS	16232
CKM_SHA512_RSA_PKCS	16232
CKM_SHA1_RSA_PKCS_PSS	16296
CKM_SHA224_RSA_PKCS_PSS	16264
CKM_SHA256_RSA_PKCS_PSS	16296

Mechanism	Maximum Data Size
CKM_SHA384_RSA_PKCS_PSS	16232
CKM_SHA512_RSA_PKCS_PSS	16232
CKM_ECDSA_SHA1	16296
CKM_ECDSA_SHA224	16264
CKM_ECDSA_SHA256	16296
CKM_ECDSA_SHA384	16232
CKM_ECDSA_SHA512	16232

Multipart operations Client SDK 3

Support for data sizes greater than 16 KB, but data size determines where the hashing takes place. Data buffers less than 16 KB are hashed inside the HSM. Buffers between 16 KB and the maximum data size for your system are hashed locally in software. *Remember*: Hash functions do not require cryptographic secrets, so you can safely compute them outside of the HSM.

- [3.3] When operating on data by using any of the following mechanisms, if the data buffer exceeds the maximum data size, the operation results in an error. For these mechanisms, all the data processing must occur inside the HSM. The following table lists maximum data size set for each mechanism:

Table 3.3, Maximum data set size

Mechanism	Maximum Data Size
CKM_SHA_1_HMAC	16288
CKM_SHA224_HMAC	16256
CKM_SHA256_HMAC	16288
CKM_SHA384_HMAC	16224

Mechanism	Maximum Data Size
CKM_SHA512_HMAC	16224
CKM_AES_CBC	16272
CKM_AES_GCM	16224
CKM_CLOUDHSM_AES_GCM	16224
CKM_DES3_CBC	16280

- [4] When performing AES-GCM encryption, the HSM does not accept initialization vector (IV) data from the application. You must use an IV that it generates. The 12-byte IV provided by the HSM is written into the memory reference pointed to by the pIV element of the CK_GCM_PARAMS parameters structure that you supply. To prevent user confusion, PKCS #11 SDK in version 1.1.1 and later ensures that pIV points to a zeroized buffer when AES-GCM encryption is initialized.
- [5] **Client SDK 3 only**. Mechanism is implemented to support SSL/TLS offload cases and is executed only partially within the HSM. Before using this mechanism, see "Issue: ECDH key derivation is executed only partially within the HSM" in Known issues for the PKCS #11 library for AWS CloudHSM. CKM_ECDH1_DERIVE does not support the secp521r1 (P-521) curve.
- [6] The following CK_MECHANISM_TYPE and CK_RSA_PKCS_MGF_TYPE are supported as CK_RSA_PKCS_OAEP_PARAMS for CKM_RSA_PKCS_OAEP:
  - CKM_SHA_1 using CKG_MGF1_SHA1
  - CKM_SHA224 using CKG_MGF1_SHA224
  - CKM_SHA256 using CKG_MGF1_SHA256
  - CKM_SHA384 using CKM_MGF1_SHA384
  - CKM_SHA512 using CKM_MGF1_SHA512
- [7.1] Vendor-defined mechanism. In order to use the CloudHSM vendor defined mechanisms, PKCS#11 applications must include `/opt/cloudhsm/include/pkcs11t.h` during compilation.

  CKM_CLOUDHSM_AES_GCM: This proprietary mechanism is a programmatically safer alternative to the standard CKM_AES_GCM. It prepends the IV generated by the HSM to the ciphertext instead of writing it back into the CK_GCM_PARAMS structure that is provided during cipher initialization. You can use this mechanism with `C_Encrypt`, `C_WrapKey`, `C_Decrypt`, and `C_UnwrapKey` functions. When using this mechanism, the pIV variable in the CK_GCM_PARAMS struct must be set to NULL. When using this mechanism with C_Decrypt and C_UnwrapKey, the IV is expected to be prepended to the ciphertext that is being unwrapped.

CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD: AES Key Wrap with PKCS #5 Padding

CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD: AES Key Wrap with Zero Padding

For additional information regarding AES key wrapping, see AES Key Wrapping.

- [8] In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.

Supported API operations for AWS CloudHSM Client SDK 3

The PKCS #11 library supports the following PKCS #11 API operations for AWS CloudHSM Client SDK 3.

- C_CloseAllSessions
- C_CloseSession
- C_CreateObject
- C_Decrypt
- C_DecryptFinal
- C_DecryptInit
- C_DecryptUpdate
- C_DeriveKey
- C_DestroyObject
- C_Digest
- C_DigestFinal
- C_DigestInit
- C_DigestUpdate
- C_Encrypt
- C_EncryptFinal
- C_EncryptInit
- C_EncryptUpdate

- C_Finalize
- C_FindObjects
- C_FindObjectsFinal
- C_FindObjectsInit
- C_GenerateKey
- C_GenerateKeyPair
- C_GenerateRandom
- C_GetAttributeValue
- C_GetFunctionList
- C_GetInfo
- C_GetMechanismInfo
- C_GetMechanismList
- C_GetSessionInfo
- C_GetSlotInfo
- C_GetSlotList
- C_GetTokenInfo
- C_Initialize
- C_Login
- C_Logout
- C_OpenSession
- C_Sign
- C_SignFinal
- C_SignInit
- C_SignRecover (Client SDK 3 support only)
- C_SignRecoverInit (Client SDK 3 support only)
- C_SignUpdate
- C_UnWrapKey
- C_Verify
- C_VerifyFinal
- C_VerifyInit

- `C_VerifyRecover` (Client SDK 3 support only)
- `C_VerifyRecoverInit` (Client SDK 3 support only)
- `C_VerifyUpdate`
- `C_WrapKey`

Key attributes in the PKCS #11 library for AWS CloudHSM Client SDK 3

A key object can be a public, private, or secret key. Actions permitted on a key object are specified through attributes. Attributes are defined when the key object is created. When you use the PKCS #11 library for AWS CloudHSM, we assign default values as specified by the PKCS #11 standard.

AWS CloudHSM does not support all attributes listed in the PKCS #11 specification. We are compliant with the specification for all attributes we support. These attributes are listed in the respective tables.

Cryptographic functions such as `C_CreateObject`, `C_GenerateKey`, `C_GenerateKeyPair`, `C_UnwrapKey`, and `C_DeriveKey` that create, modify, or copy objects take an attribute template as one of their parameters. For more information about passing an attribute template during object creation, see Generate keys through PKCS #11 library sample.

The following topics provide more information about AWS CloudHSM key attributes for Client SDK 3.

Topics

PKCS #11 library attributes table for AWS CloudHSM Client SDK 3

The PKCS #11 library table for AWS CloudHSM Client SDK 3 contains a list of attributes that differ by key types. It indicates whether a given attribute is supported for a particular key type when using a specific cryptographic function with AWS CloudHSM.

Legend:

- ✔ indicates that CloudHSM supports the attribute for the specific key type.
- ✖ indicates that CloudHSM does not support the attribute for the specific key type.

- R indicates that the attribute value is set to read-only for the specific key type.
- S indicates that the attribute cannot be read by the `GetAttributeValue` as it is sensitive.
- An empty cell in the Default Value column indicates that there is no specific default value assigned to the attribute.

GenerateKeyPair

Attribute	Key Type					Default Value
		EC private	**EC public**	**RSA private**	**RSA public**	
`CKA_CLASS`		✔	✔	✔	✔	
`CKA_KEY_TYPE`		✔	✔	✔	✔	
`CKA_LABEL`		✔	✔	✔	✔	
`CKA_ID`		✔	✔	✔	✔	
`CKA_LOCAL`		R	R	R	R	True
`CKA_TOKEN`		✔	✔	✔	✔	False
`CKA_PRIVATE`		✔[1]	✔[1]	✔[1]	✔[1]	True
`CKA_ENCRYPT`		✖	✔	✖	✔	False
`CKA_DECRYPT`		✔	✖	✔	✖	False

Attribute	Key Type					Default Value
CKA_DERIVE		✔	✔	✔	✔	False
CKA_MODIFIABLE		✔[1]	✔[1]	✔[1]	✔[1]	True
CKA_DESTROYABLE		✔	✔	✔	✔	True
CKA_SIGN		✔	✖	✔	✖	False
CKA_SIGN_RECOVER		✖	✖	✔[3]	✖	
CKA_VERIFY		✖	✔	✖	✔	False
CKA_VERIFY_RECOVER		✖	✖	✖	✔[4]	
CKA_WRAP		✖	✔	✖	✔	False
CKA_WRAP_TEMPLATE		✖	✔	✖	✔	
CKA_TRUSTED		✖	✔	✖	✔	False
CKA_WRAP_WITH_TRUSTED		✔	✖	✔	✖	False
CKA_UNWRAP		✔	✖	✔	✖	False

Attribute	Key Type					Default Value
CKA_UNWRAP_TEMPLATE		✓	✖	✓	✖	
CKA_SENSITIVE		✓	✖	✓	✖	True
CKA_ALWAYS_SENSITIVE		R	✖	R	✖	
CKA_EXTRACTABLE		✓	✖	✓	✖	True
CKA_NEVER_EXTRACTABLE		R	✖	R	✖	
CKA_MODULUS		✖	✖	✖	✖	
CKA_MODULUS_BITS		✖	✖	✖	✓[2]	
CKA_PRIME_1		✖	✖	✖	✖	
CKA_PRIME_2		✖	✖	✖	✖	
CKA_COEFFICIENT		✖	✖	✖	✖	
CKA_EXPONENT_1		✖	✖	✖	✖	

Attribute	Key Type					Default Value
CKA_EXPONENT_2		✖	✖	✖	✖	
CKA_PRIVATE_EXPONENT		✖	✖	✖	✖	
CKA_PUBLIC_EXPONENT		✖	✖	✖	✔[2]	
CKA_EC_PARAMS		✖	✔[2]	✖	✖	
CKA_EC_POINT		✖	✖	✖	✖	
CKA_VALUE		✖	✖	✖	✖	
CKA_VALUE_LEN		✖	✖	✖	✖	
CKA_CHECK_VALUE		R	R	R	R	

GenerateKey

Attribute	Key Type				Default Value
		AES	DES3	Generic Secret	
CKA_CLASS		✔	✔	✔	

Attribute	Key Type				Default Value
CKA_KEY_TYPE		✓	✓	✓	
CKA_LABEL		✓	✓	✓	
CKA_ID		✓	✓	✓	
CKA_LOCAL		R	R	R	True
CKA_TOKEN		✓	✓	✓	False
CKA_PRIVATE		✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT		✓	✓	✖	False
CKA_DECRYPT		✓	✓	✖	False
CKA_DERIVE		✓	✓	✓	False
CKA_MODIFIABLE		✓[1]	✓[1]	✓[1]	True
CKA_DESTROYABLE		✓	✓	✓	True
CKA_SIGN		✓	✓	✓	True
CKA_SIGN_RECOVER		✖	✖	✖	
CKA_VERIFY		✓	✓	✓	True

Attribute	Key Type				Default Value
CKA_VERIFY_RECOVER		✖	✖	✖	
CKA_WRAP		✔	✔	✖	False
CKA_WRAP_TEMPLATE		✔	✔	✖	
CKA_TRUSTED		✔	✔	✖	False
CKA_WRAP_WITH_TRUSTED		✔	✔	✔	False
CKA_UNWRAP		✔	✔	✖	False
CKA_UNWRAP_TEMPLATE		✔	✔	✖	
CKA_SENSITIVE		✔	✔	✔	True
CKA_ALWAYS_SENSITIVE		✖	✖	✖	
CKA_EXTRACTABLE		✔	✔	✔	True
CKA_NEVER_EXTRACTABLE		R	R	R	

Attribute	Key Type				Default Value
CKA_MODULUS		✖	✖	✖	
CKA_MODULUS_BITS		✖	✖	✖	
CKA_PRIME_1		✖	✖	✖	
CKA_PRIME_2		✖	✖	✖	
CKA_COEFFICIENT		✖	✖	✖	
CKA_EXPONENT_1		✖	✖	✖	
CKA_EXPONENT_2		✖	✖	✖	
CKA_PRIVATE_EXPONENT		✖	✖	✖	
CKA_PUBLIC_EXPONENT		✖	✖	✖	
CKA_EC_PARAMS		✖	✖	✖	
CKA_EC_POINT		✖	✖	✖	
CKA_VALUE		✖	✖	✖	

Attribute	Key Type				Default Value
CKA_VALUE_LEN		✓[2]	✖	✓[2]	
CKA_CHECK_VALUE		R	R	R	

CreateObject

Attribute	Key Type								Default Value
		EC private	EC public	RSA private	RSA public	AES	DES3	Generic Secret	
CKA_CLASS		✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_KEY_TYPE		✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_LABEL		✓	✓	✓	✓	✓	✓	✓	
CKA_ID		✓	✓	✓	✓	✓	✓	✓	
CKA_LOCAL		R	R	R	R	R	R	R	False
CKA_TOKEN		✓	✓	✓	✓	✓	✓	✓	False
CKA_PRIVATE		✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	True

Attribute	Key Type								Default Value
CKA_ENCRYPT		✖	✖	✖	✔	✔	✔	✖	False
CKA_DECRYPT		✖	✖	✔	✖	✔	✔	✖	False
CKA_DERIVE		✔	✔	✔	✔	✔	✔	✔	False
CKA_MODIFIABLE		✔[1]	✔[1]	✔[1]	✔[1]	✔[1]	✔[1]	✔[1]	True
CKA_DESTROYABLE		✔	✔	✔	✔	✔	✔	✔	True
CKA_SIGN		✔	✖	✔	✖	✔	✔	✔	False
CKA_SIGN_RECOVER		✖	✖	✔[3]	✖	✖	✖	✖	False
CKA_VERIFY		✖	✔	✖	✔	✔	✔	✔	False
CKA_VERIFY_RECOVER		✖	✖	✖	✔[4]	✖	✖	✖	
CKA_WRAP		✖	✖	✖	✔	✔	✔	✖	False
CKA_WRAP_TEMPLATE		✖	✔	✖	✔	✔	✔	✖	
CKA_TRUSTED		✖	✔	✖	✔	✔	✔	✖	False

Attribute	Key Type								Default Value
CKA_WRAP_WITH_TRUSTED		✔	✖	✔	✖	✔	✔	✔	False
CKA_UNWRAP		✖	✖	✔	✖	✔	✔	✖	False
CKA_UNWRAP_TEMPLATE		✔	✖	✔	✖	✔	✔	✖	
CKA_SENSITIVE		✔	✖	✔	✖	✔	✔	✔	True
CKA_ALWAYS_SENSITIVE		R	✖	R	✖	R	R	R	
CKA_EXTRACTABLE		✔	✖	✔	✖	✔	✔	✔	True
CKA_NEVER_EXTRACTABLE		R	✖	R	✖	R	R	R	
CKA_MODULUS		✖	✖	✔[2]	✔[2]	✖	✖	✖	
CKA_MODULUS_BITS		✖	✖	✖	✖	✖	✖	✖	
CKA_PRIME_1		✖	✖	✔	✖	✖	✖	✖	
CKA_PRIME_2		✖	✖	✔	✖	✖	✖	✖	

Attribute	Key Type							Default Value
CKA_COEFFICIENT	✖	✖	✓	✖	✖	✖	✖	
CKA_EXPONENT_1	✖	✖	✓	✖	✖	✖	✖	
CKA_EXPONENT_2	✖	✖	✓	✖	✖	✖	✖	
CKA_PRIVATE_EXPONENT	✖	✖	✓[2]	✖	✖	✖	✖	
CKA_PUBLIC_EXPONENT	✖	✖	✓[2]	✓[2]	✖	✖	✖	
CKA_EC_PARAMS	✓[2]	✓[2]	✖	✖	✖	✖	✖	
CKA_EC_POINT	✖	✓[2]	✖	✖	✖	✖	✖	
CKA_VALUE	✓[2]	✖	✖	✖	✓[2]	✓[2]	✓[2]	
CKA_VALUE_LEN	✖	✖	✖	✖	✖	✖	✖	
CKA_CHECK_VALUE	R	R	R	R	R	R	R	

UnwrapKey

Attribute	Key Type						Default Value
		EC private	RSA private	AES	DES3	Generic Secret	
CKA_CLASS		✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_KEY_TYPE		✓[2]	✓[2]	✓[2]	✓[2]	✓[2]	
CKA_LABEL		✓	✓	✓	✓	✓	
CKA_ID		✓	✓	✓	✓	✓	
CKA_LOCAL		R	R	R	R	R	False
CKA_TOKEN		✓	✓	✓	✓	✓	False
CKA_PRIVATE		✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	True
CKA_ENCRYPT		✖	✖	✓	✓	✖	False
CKA_DECRYPT		✖	✓	✓	✓	✖	False
CKA_DERIVE		✓	✓	✓	✓	✓	False
CKA_MODIFIABLE		✓[1]	✓[1]	✓[1]	✓[1]	✓[1]	True

Attribute	Key Type						Default Value
CKA_DESTROYABLE		✓	✓	✓	✓	✓	True
CKA_SIGN		✓	✓	✓	✓	✓	False
CKA_SIGN_RECOVER		✖	✓[3]	✖	✖	✖	False
CKA_VERIFY		✖	✖	✓	✓	✓	False
CKA_VERIFY_RECOVER		✖	✖	✖	✖	✖	
CKA_WRAP		✖	✖	✓	✓	✖	False
CKA_UNWRAP		✖	✓	✓	✓	✖	False
CKA_SENSITIVE		✓	✓	✓	✓	✓	True
CKA_EXTRACTABLE		✓	✓	✓	✓	✓	True
CKA_NEVER_EXTRACTABLE		R	R	R	R	R	
CKA_ALWAYS_SENSITIVE		R	R	R	R	R	
CKA_MODULUS		✖	✖	✖	✖	✖	

Attribute	Key Type						Default Value
CKA_MODULUS_BITS		✖	✖	✖	✖	✖	
CKA_PRIME_1		✖	✖	✖	✖	✖	
CKA_PRIME_2		✖	✖	✖	✖	✖	
CKA_COEFFICIENT		✖	✖	✖	✖	✖	
CKA_EXPONENT_1		✖	✖	✖	✖	✖	
CKA_EXPONENT_2		✖	✖	✖	✖	✖	
CKA_PRIVATE_EXPONENT		✖	✖	✖	✖	✖	
CKA_PUBLIC_EXPONENT		✖	✖	✖	✖	✖	
CKA_EC_PARAMS		✖	✖	✖	✖	✖	
CKA_EC_POINT		✖	✖	✖	✖	✖	
CKA_VALUE		✖	✖	✖	✖	✖	

Attribute	Key Type						Default Value
CKA_VALUE_LEN		✖	✖	✖	✖	✖	
CKA_CHECK_VALUE		R	R	R	R	R	

DeriveKey

Attribute	Key Type				Default Value
		AES	**DES3**	**Generic Secret**	
CKA_CLASS		✔[2]	✔[2]	✔[2]	
CKA_KEY_TYPE		✔[2]	✔[2]	✔[2]	
CKA_LABEL		✔	✔	✔	
CKA_ID		✔	✔	✔	
CKA_LOCAL		R	R	R	True
CKA_TOKEN		✔	✔	✔	False
CKA_PRIVATE		✔[1]	✔[1]	✔[1]	True
CKA_ENCRYPT		✔	✔	✖	False
CKA_DECRYPT		✔	✔	✖	False

Attribute	Key Type				Default Value
CKA_DERIVE		✓	✓	✓	False
CKA_MODIFIABLE		✓[1]	✓[1]	✓[1]	True
CKA_DESTROYABLE		✓[1]	✓[1]	✓[1]	True
CKA_SIGN		✓	✓	✓	False
CKA_SIGN_RECOVER		✖	✖	✖	
CKA_VERIFY		✓	✓	✓	False
CKA_VERIFY_RECOVER		✖	✖	✖	
CKA_WRAP		✓	✓	✖	False
CKA_UNWRAP		✓	✓	✖	False
CKA_SENSITIVE		✓	✓	✓	True
CKA_EXTRACTABLE		✓	✓	✓	True
CKA_NEVER_EXTRACTABLE		R	R	R	

Attribute	Key Type				Default Value
CKA_ALWAYS_SENSITIVE		R	R	R	
CKA_MODULUS		✖	✖	✖	
CKA_MODULUS_BITS		✖	✖	✖	
CKA_PRIME_1		✖	✖	✖	
CKA_PRIME_2		✖	✖	✖	
CKA_COEFFICIENT		✖	✖	✖	
CKA_EXPONENT_1		✖	✖	✖	
CKA_EXPONENT_2		✖	✖	✖	
CKA_PRIVATE_EXPONENT		✖	✖	✖	
CKA_PUBLIC_EXPONENT		✖	✖	✖	
CKA_EC_PARAMS		✖	✖	✖	

Attribute	Key Type				Default Value
CKA_EC_POINT		✖	✖	✖	
CKA_VALUE		✖	✖	✖	
CKA_VALUE_LEN		✔[2]	✖	✔[2]	
CKA_CHECK_VALUE		R	R	R	

GetAttributeValue

Attribute	Key Type							
		EC private	EC public	RSA private	RSA public	AES	DES3	Generic Secret
CKA_CLASS		✔	✔	✔	✔	✔	✔	✔
CKA_KEY_TYPE		✔	✔	✔	✔	✔	✔	✔
CKA_LABEL		✔	✔	✔	✔	✔	✔	✔
CKA_ID		✔	✔	✔	✔	✔	✔	✔
CKA_LOCAL		✔	✔	✔	✔	✔	✔	✔
CKA_TOKEN		✔	✔	✔	✔	✔	✔	✔

Attribute	Key Type						
CKA_PRIVATE	✔[1]	✔[1]	✔[1]	✔[1]	✔[1]	✔[1]	✔[1]
CKA_ENCRYPT	✖	✖	✖	✔	✔	✔	✖
CKA_DECRYPT	✖	✖	✔	✖	✔	✔	✖
CKA_DERIVE	✔	✔	✔	✔	✔	✔	✔
CKA_MODIFIABLE	✔	✔	✔	✔	✔	✔	✔
CKA_DESTROYABLE	✔	✔	✔	✔	✔	✔	✔
CKA_SIGN	✔	✖	✔	✖	✔	✔	✔
CKA_SIGN_RECOVER	✖	✖	✔	✖	✖	✖	✖
CKA_VERIFY	✖	✔	✖	✔	✔	✔	✔
CKA_VERIFY_RECOVER	✖	✖	✖	✔	✖	✖	✖
CKA_WRAP	✖	✖	✖	✔	✔	✔	✖
CKA_WRAP_TEMPLATE	✖	✔	✖	✔	✔	✔	✖
CKA_TRUSTED	✖	✔	✖	✔	✔	✔	✔

Attribute	Key Type						
CKA_WRAP_WITH_TRUSTED	✔	✖	✔	✖	✔	✔	✔
CKA_UNWRAP	✖	✖	✔	✖	✔	✔	✖
CKA_UNWRAP_TEMPLATE	✔	✖	✔	✖	✔	✔	✖
CKA_SENSITIVE	✔	✖	✔	✖	✔	✔	✔
CKA_EXTRACTABLE	✔	✖	✔	✖	✔	✔	✔
CKA_NEVER_EXTRACTABLE	✔	✖	✔	✖	✔	✔	✔
CKA_ALWAYS_SENSITIVE	R	R;	R	R	R	R	R
CKA_MODULUS	✖	✖	✔	✔	✖	✖	✖
CKA_MODULUS_BITS	✖	✖	✖	✔	✖	✖	✖
CKA_PRIME_1	✖	✖	S	✖	✖	✖	✖
CKA_PRIME_2	✖	✖	S	✖	✖	✖	✖

Attribute	Key Type							
CKA_COEFFICIENT		✖	✖	S	✖	✖	✖	✖
CKA_EXPONENT_1		✖	✖	S	✖	✖	✖	✖
CKA_EXPONENT_2		✖	✖	S	✖	✖	✖	✖
CKA_PRIVATE_EXPONENT		✖	✖	S	✖	✖	✖	✖
CKA_PUBLIC_EXPONENT		✖	✖	✔	✔	✖	✖	✖
CKA_EC_PARAMS		✔	✔	✖	✖	✖	✖	✖
CKA_EC_POINT		✖	✔	✖	✖	✖	✖	✖
CKA_VALUE		S	✖	✖	✖	✔[2]	✔[2]	✔[2]
CKA_VALUE_LEN		✖	✖	✖	✖	✔	✖	✔
CKA_CHECK_VALUE		✔	✔	✔	✔	✔	✔	✖

Attribute annotations

- [1] This attribute is partially supported by the firmware and must be explicitly set only to the default value.

- [2] Mandatory attribute.
- [3] **Client SDK 3 only**. The `CKA_SIGN_RECOVER` attribute is derived from the `CKA_SIGN` attribute. If being set, it can only be set to the same value that is set for `CKA_SIGN`. If not set, it derives the default value of `CKA_SIGN`. Since CloudHSM only supports RSA-based recoverable signature mechanisms, this attribute is currently applicable to RSA public keys only.
- [4] **Client SDK 3 only**. The `CKA_VERIFY_RECOVER` attribute is derived from the `CKA_VERIFY` attribute. If being set, it can only be set to the same value that is set for `CKA_VERIFY`. If not set, it derives the default value of `CKA_VERIFY`. Since CloudHSM only supports RSA-based recoverable signature mechanisms, this attribute is currently applicable to RSA public keys only.

Modifying PKCS #11 library attributes for AWS CloudHSM Client SDK 3

Some attributes of an object can be modified after the object has been created, whereas some cannot. To modify attributes, use the setAttribute command from cloudhsm_mgmt_util. You can also derive a list of attributes and the constants that represent them by using the listAttribute command from cloudhsm_mgmt_util.

The following list displays attributes that are allowed for modification after object creation:

- `CKA_LABEL`
- `CKA_TOKEN`

> **Note**
>
> Modification is allowed only for changing a session key to a token key. Use the setAttribute command from key_mgmt_util to change the attribute value.

- `CKA_ENCRYPT`
- `CKA_DECRYPT`
- `CKA_SIGN`
- `CKA_VERIFY`
- `CKA_WRAP`
- `CKA_UNWRAP`
- `CKA_LABEL`
- `CKA_SENSITIVE`

- `CKA_DERIVE`

> **Note**
>
> This attribute supports key derivation. It must be `False` for all public keys and cannot be set to `True`. For secret and EC private keys, it can be set to `True` or `False`.

- `CKA_TRUSTED`

> **Note**
>
> This attribute can be set to `True` or `False` by Crypto Officer (CO) only.

- `CKA_WRAP_WITH_TRUSTED`

> **Note**
>
> Apply this attribute to an exportable data key to specify that you can only wrap this key with keys marked as `CKA_TRUSTED`. Once you set `CKA_WRAP_WITH_TRUSTED` to true, the attribute becomes read-only and you cannot change or remove the attribute.

Interpreting PKCS #11 library error codes for AWS CloudHSM Client SDK 3

Specifying in the template a PKCS #11 library attribute that is not supported by a specific key results in an error. The following table contains error codes that are generated when you violate specifications:

Error Code	Description
`CKR_TEMPLATE_INCONSISTENT`	You receive this error when you specify an attribute in the attribute template, where the attribute complies with the PKCS #11 specifica tion, but is not supported by CloudHSM.
`CKR_ATTRIBUTE_TYPE_INVALID`	You receive this error when you retrieve value for an attribute, which complies with the PKCS

Error Code	Description
	#11 specification, but is not supported by CloudHSM.
CKR_ATTRIBUTE_INCOMPLETE	You receive this error when you do not specify the mandatory attribute in the attribute template.
CKR_ATTRIBUTE_READ_ONLY	You receive this error when you specify a read-only attribute in the attribute template.

Code samples for the PKCS #11 library for AWS CloudHSM Client SDK 3

The code samples on GitHub show you how to accomplish basic tasks using the PKCS #11 library for AWS CloudHSM.

Sample code prerequisites

Before running the samples, perform the following steps to set up your environment:

- Install and configure the PKCS #11 library for Client SDK 3.
- Set up a cryptographic user (CU). Your application uses this HSM account to run the code samples on the HSM.

Code samples

Code Samples for the AWS CloudHSM Software Library for PKCS#11 are available on GitHub. This repository includes examples on how to do common operations using PKCS#11 including encryption, decryption, signing and verifying.

- Generate keys (AES, RSA, EC)
- List key attributes
- Encrypt and decrypt data with AES GCM
- Encrypt and decrypt data with AES_CTR
- Encrypt and decrypt data with 3DES
- Sign and verify data with RSA

- Derive keys using HMAC KDF
- Wrap and unwrap keys with AES using PKCS #5 padding
- Wrap and unwrap keys with AES using no padding
- Wrap and unwrap keys with AES using zero padding
- Wrap and unwrap keys with AES-GCM
- Wrap and unwrap keys with RSA

OpenSSL Dynamic Engine for AWS CloudHSM Client SDK 3

The AWS CloudHSM OpenSSL Dynamic Engine enables you to offload cryptographic operations to your CloudHSM cluster through the OpenSSL API.

AWS CloudHSM Client SDK 3 does require a client daemon to connect to the cluster. It supports:

- RSA key generation for 2048, 3072, and 4096-bit keys.
- RSA sign/verify.
- RSA encrypt/decrypt.
- Random number generation that is cryptographically secure and FIPS-validated.

  Use the following sections to install and configure the AWS CloudHSM dynamic engine for OpenSSL.

Topics

Prerequisites for OpenSSL Dynamic Engine with AWS CloudHSM Client SDK 3

For information about platform support, see AWS CloudHSM Client SDK 3 supported platforms.

Before you can use the AWS CloudHSM dynamic engine for OpenSSL with Client SDK 3, you need the AWS CloudHSM client.

The client is a daemon that establishes end-to-end encrypted communication with the HSMs in your cluster, and the OpenSSL engine communicates locally with the client. To install and configure the AWS CloudHSM client, see Install the client (Linux). Then use the following command to start it.

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo systemctl cloudhsm-client start
```

CentOS 6

```
$ sudo systemctl start cloudhsm-client
```

CentOS 7

```
$ sudo systemctl cloudhsm-client start
```

RHEL 6

```
$ sudo systemctl start cloudhsm-client
```

RHEL 7

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Install the OpenSSL Dynamic Engine for AWS CloudHSM Client SDK 3

The following steps describe how to install and configure the AWS CloudHSM dynamic engine for OpenSSL with Client SDK 3. For information about upgrading, see Upgrade Client SDK 3.

To install and configure the OpenSSL engine

1. Use the following commands to download and install the OpenSSL engine.

   Amazon Linux

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-dyn-latest.el6.x86_64.rpm
   ```

   ```
   $ sudo yum install ./cloudhsm-client-dyn-latest.el6.x86_64.rpm
   ```

   Amazon Linux 2

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-dyn-latest.el7.x86_64.rpm
   ```

   ```
   $ sudo yum install ./cloudhsm-client-dyn-latest.el7.x86_64.rpm
   ```

   CentOS 6

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-dyn-latest.el6.x86_64.rpm
   ```

   ```
   $ sudo yum install ./cloudhsm-client-dyn-latest.el6.x86_64.rpm
   ```

   CentOS 7

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-dyn-latest.el7.x86_64.rpm
   ```

   ```
   $ sudo yum install ./cloudhsm-client-dyn-latest.el7.x86_64.rpm
   ```

   RHEL 6

   ```
   $ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-dyn-latest.el6.x86_64.rpm
   ```

```
$ sudo yum install ./cloudhsm-client-dyn-latest.el6.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-dyn-latest.el7.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-dyn_latest_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-dyn_latest_amd64.deb
```

The OpenSSL engine is installed at `/opt/cloudhsm/lib/libcloudhsm_openssl.so`.

2. Use the following command to set an environment variable named `n3fips_password` that contains the credentials of a crypto user (CU).

```
$ export n3fips_password=<HSM user name>:<password>
```

Use the OpenSSL Dynamic Engine for AWS CloudHSM Client SDK 3

To use the AWS CloudHSM dynamic engine for OpenSSL from an OpenSSL-integrated application, ensure that your application uses the OpenSSL dynamic engine named `cloudhsm`. The shared library for the dynamic engine is located at `/opt/cloudhsm/lib/libcloudhsm_openssl.so`.

To use the AWS CloudHSM dynamic engine for OpenSSL from the OpenSSL command line, use the `-engine` option to specify the OpenSSL dynamic engine named `cloudhsm`. For example:

```
$ openssl s_server -cert <server.crt> -key <server.key> -engine cloudhsm
```

JCE provider for AWS CloudHSM Client SDK 3

The AWS CloudHSM JCE provider is a provider implementation built from the Java Cryptographic Extension (JCE) provider framework. The JCE allows you to perform cryptographic operations using the Java Development Kit (JDK). In this guide, the AWS CloudHSM JCE provider is sometimes referred to as the JCE provider. Use the JCE provider and the JDK to offload cryptographic operations to the HSM.

Topics

Install the JCE provider for AWS CloudHSM Client SDK 3

Before you can use the JCE provider, you need the AWS CloudHSM client.

The client is a daemon that establishes end-to-end encrypted communication with the HSMs in your cluster. The JCE provider communicates locally with the client. If you haven't installed and configured the AWS CloudHSM client package, do that now by following the steps at Install the client (Linux). After you install and configure the client, use the following command to start it.

Note that the JCE provider is supported only on Linux and compatible operating systems.

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo systemctl cloudhsm-client start
```

CentOS 7

```
$ sudo systemctl cloudhsm-client start
```

CentOS 8

```
$ sudo systemctl cloudhsm-client start
```

RHEL 7

```
$ sudo systemctl cloudhsm-client start
```

RHEL 8

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Ubuntu 20.04 LTS

```
$ sudo systemctl cloudhsm-client start
```

Use the following sections to install, validate, and provide credentials to the provider.

Topics

- Step 1: Install the JCE provider
- Step 2: Validate the installation
- Step 3: Provide credentials to the JCE provider

Step 1: Install the JCE provider

Use the following commands to download and install the JCE provider. This provider is supported only on Linux and compatible operating systems.

> **Note**
>
> For upgrading, see Upgrade Client SDK 3.

Amazon Linux

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL6/cloudhsm-client-jce-latest.el6.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-jce-latest.el6.x86_64.rpm
```

Amazon Linux 2

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-jce-latest.el7.x86_64.rpm
```

CentOS 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-jce-latest.el7.x86_64.rpm
```

CentOS 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-jce-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-jce-latest.el8.x86_64.rpm
```

RHEL 7

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL7/cloudhsm-client-jce-latest.el7.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-jce-latest.el7.x86_64.rpm
```

RHEL 8

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/EL8/cloudhsm-client-jce-latest.el8.x86_64.rpm
```

```
$ sudo yum install ./cloudhsm-client-jce-latest.el8.x86_64.rpm
```

Ubuntu 16.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Xenial/cloudhsm-client-jce_latest_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-jce_latest_amd64.deb
```

Ubuntu 18.04 LTS

```
$ wget https://s3.amazonaws.com/cloudhsmv2-software/CloudHsmClient/Bionic/cloudhsm-client-jce_latest_u18.04_amd64.deb
```

```
$ sudo apt install ./cloudhsm-client-jce_latest_u18.04_amd64.deb
```

After you run the preceding commands, you can find the following JCE provider files:

- `/opt/cloudhsm/java/cloudhsm-`*<version>*`.jar`
- `/opt/cloudhsm/java/cloudhsm-test-`*<version>*`.jar`
- `/opt/cloudhsm/java/hamcrest-all-1.3.jar`
- `/opt/cloudhsm/java/junit.jar`
- `/opt/cloudhsm/java/log4j-api-2.17.1.jar`
- `/opt/cloudhsm/java/log4j-core-2.17.1.jar`

- `/opt/cloudhsm/lib/libcaviumjca.so`

Step 2: Validate the installation

Perform basic operations on the HSM to validate the installation.

To validate JCE provider installation

1. (Optional) If you don't already have Java installed in your environment, use the following command to install it.

   Linux (and compatible libraries)

   ```
   $ sudo yum install java-1.8.0-openjdk
   ```

   Ubuntu

   ```
   $ sudo apt-get install openjdk-8-jre
   ```

2. Use the following commands to set the necessary environment variables. Replace *<HSM user name>* and *<password>* with the credentials of a crypto user (CU).

   ```
   $ export LD_LIBRARY_PATH=/opt/cloudhsm/lib
   ```

   ```
   $ export HSM_PARTITION=PARTITION_1
   ```

   ```
   $ export HSM_USER=<HSM user name>
   ```

   ```
   $ export HSM_PASSWORD=<password>
   ```

3. Use the following command to run the basic functionality test. If successful, the command's output should be similar to the one that follows.

   ```
   $ java8 -classpath "/opt/cloudhsm/java/*" org.junit.runner.JUnitCore
    TestBasicFunctionality

   JUnit version 4.11
   .2018-08-20 17:53:48,514 DEBUG [main] TestBasicFunctionality
    (TestBasicFunctionality.java:33) - Adding provider.
   ```

```
2018-08-20 17:53:48,612 DEBUG [main] TestBasicFunctionality
 (TestBasicFunctionality.java:42) - Logging in.
2018-08-20 17:53:48,612 INFO [main] cfm2.LoginManager (LoginManager.java:104) -
 Looking for credentials in HsmCredentials.properties
2018-08-20 17:53:48,612 INFO [main] cfm2.LoginManager (LoginManager.java:122) -
 Looking for credentials in System.properties
2018-08-20 17:53:48,613 INFO [main] cfm2.LoginManager (LoginManager.java:130) -
 Looking for credentials in System.env
 SDK Version: 2.03
2018-08-20 17:53:48,655 DEBUG [main] TestBasicFunctionality
 (TestBasicFunctionality.java:54) - Generating AES Key with key size 256.
2018-08-20 17:53:48,698 DEBUG [main] TestBasicFunctionality
 (TestBasicFunctionality.java:63) - Encrypting with AES Key.
2018-08-20 17:53:48,705 DEBUG [main] TestBasicFunctionality
 (TestBasicFunctionality.java:84) - Deleting AES Key.
2018-08-20 17:53:48,707 DEBUG [main] TestBasicFunctionality
 (TestBasicFunctionality.java:92) - Logging out.

Time: 0.205

OK (1 test)
```

Step 3: Provide credentials to the JCE provider

HSMs need to authenticate your Java application before the application can use them. Each application can use one session. HSMs authenticate a session by using either explicit login or implicit login method.

Explicit login – This method lets you provide CloudHSM credentials directly in the application. It uses the `LoginManager.login()` method, where you pass the CU user name, password, and the HSM partition ID. For more information about using the explicit login method, see the [Login to an HSM](#) code example.

Implicit login – This method lets you set CloudHSM credentials either in a new property file, system properties, or as environment variables.

- **New property file** – Create a new file named `HsmCredentials.properties` and add it to your application's `CLASSPATH`. The file should contain the following:

```
HSM_PARTITION = PARTITION_1
HSM_USER = <HSM user name>
```

```
HSM_PASSWORD = <password>
```

- **System properties** – Set credentials through system properties when running your application. The following examples show two different ways that you can do this:

```
$ java -DHSM_PARTITION=PARTITION_1 -DHSM_USER=<HSM user name> -
DHSM_PASSWORD=<password>
```

```
System.setProperty("HSM_PARTITION","PARTITION_1");
System.setProperty("HSM_USER","<HSM user name>");
System.setProperty("HSM_PASSWORD","<password>");
```

- **Environment variables** – Set credentials as environment variables.

```
$ export HSM_PARTITION=PARTITION_1
$ export HSM_USER=<HSM user name>
$ export HSM_PASSWORD=<password>
```

Credentials might not be available if the application does not provide them or if you attempt an operation before the HSM authenticates session. In those cases, the CloudHSM software library for Java searches for the credentials in the following order:

1. `HsmCredentials.properties`
2. System properties
3. Environment variables

Error handling

The error handling is easier with the explicit login than the implicit login method. When you use the `LoginManager` class, you have more control over how your application deals with failures. The implicit login method makes error handling difficult to understand when the credentials are invalid or the HSMs are having problems in authenticating session.

Key management basics in the JCE provider for AWS CloudHSM Client SDK 3

The basics on key management in the JCE provider involve importing keys, exporting keys, loading keys by handle, or deleting keys. For more information on managing keys, see the Manage keys code example.

You can also find more JCE provider code examples at Code samples.

Supported mechanisms for Client SDK 3 for AWS CloudHSM Client SDK 3

This topic provides information about supported mechanisms for JCE provider with AWS CloudHSM Client SDK 3. For information about the Java Cryptography Architecture (JCA) interfaces and engine classes supported by AWS CloudHSM, see the following topics.

Topics

Supported keys

The AWS CloudHSM software library for Java enables you to generate the following key types.

- AES – 128, 192, and 256-bit AES keys.
- DESede – 92 bit 3DES key. See note 1 below for an upcoming change.
- ECC key pairs for NIST curves secp256r1 (P-256), secp384r1 (P-384), and secp256k1 (Blockchain).
- RSA – 2048-bit to 4096-bit RSA keys, in increments of 256 bits.

In addition to standard parameters, we support the following parameters for each key that is generated.

- **Label**: A key label that you can use to search for keys.
- **isExtractable**: Indicates whether the key can be exported from the HSM.
- **isPersistent**: Indicates whether the key remains on the HSM when the current session ends.

> **Note**
>
> Java library version 3.1 provides the ability to specify parameters in greater detail. For more information, see Supported Java Attributes.

Supported ciphers

The AWS CloudHSM software library for Java supports the following algorithm, mode, and padding combinations.

Algorithm	Mode	Padding	Notes
AES	CBC	`AES/CBC/NoPadding` `AES/CBC/PKCS5Padding`	Implements `Cipher.ENCRYPT_MODE` and `Cipher.DECRYPT_MODE`.
AES	ECB	`AES/ECB/NoPadding` `AES/ECB/PKCS5Padding`	Implements `Cipher.ENCRYPT_MODE` and `Cipher.DECRYPT_MODE`. Use Transformation AES.
AES	CTR	`AES/CTR/NoPadding`	Implements `Cipher.ENCRYPT_MODE` and `Cipher.DECRYPT_MODE`.
AES	GCM	`AES/GCM/NoPadding`	Implements `Cipher.ENCRYPT_MODE` and `Cipher.DECRYPT_MODE`, `Cipher.WR`

Algorithm	Mode	Padding	Notes
			`AP_MODE` , and `Cipher.UNWRAP_MODE` . When performing AES-GCM encryption, the HSM ignores the initialization vector (IV) in the request and uses an IV that it generates. When the operation completes, you must call `Cipher.getIV()` to get the IV.

AESWrap	ECB	`AESWrap/ECB/ZeroPadding`
		
		`AESWrap/ECB/NoPadding`
		
		`AESWrap/ECB/PKCS5Padding`
			Implements `Cipher.WRAP_MODE` , and `Cipher.UNWRAP_MODE` . Use Transformation AES.

Algorithm	Mode	Padding	Notes

DESede (Triple DES)	CBC	`DESede/CBC/NoPadding`
		
		`DESede/CBC/PKCS5Padding`
			Implements `Cipher.ENCRYPT_MODE` and `Cipher.DECRYPT_MODE` .
			
			The key generation routines accept a size of 168 or 192 bits. However, internally, all DESede keys are 192 bits.
			
			See note 1 below for an upcoming change.

DESede (Triple DES)	ECB	`DESede/ECB/NoPadding`
		
		`DESede/ECB/PKCS5Padding`
			Implements `Cipher.ENCRYPT_MODE` and `Cipher.DECRYPT_MODE` .
			
			The key generation routines accept a size of 168 or 192 bits. However, internally, all DESede keys are 192 bits.
			
			See note 1 below for an upcoming change.

Algorithm	Mode	Padding	Notes

RSA	ECB	`RSA/ECB/NoPadding`
		
		`RSA/ECB/PKCS1Padding`
			Implements `Cipher.ENCRYPT_MODE` and `Cipher.DECRYPT_MODE` .
			
			See note 1 below for an upcoming change.

RSA	ECB	`RSA/ECB/OAEPPadding`
		
		`RSA/ECB/OAEPWithSHA-1ANDMGF1Padding`
		
		`RSA/ECB/OAEPWithSHA-224ANDMGF1Padding`
		
		`RSA/ECB/OAEPWithSHA-256ANDMGF1Padding`
		
		`RSA/ECB/OAEPWithSHA-384ANDMGF1Padding`
		
		`RSA/ECB/OAEPWithSHA-512ANDMGF1Padding`
			Implements `Cipher.ENCRYPT_MODE` , `Cipher.DECRYPT_MODE` , `Cipher.WRAP_MODE` , and `Cipher.UNWRAP_MODE` .
			
			`OAEPPadding` is OAEP with the `SHA-1` padding type.

Algorithm	Mode	Padding	Notes
RSAAESWrap	ECB	OAEPPADDING	Implements Cipher.WRAP_Mode and Cipher.UNWRAP_MODE .

Supported digests

The AWS CloudHSM software library for Java supports the following message digests.

- SHA-1
- SHA-224
- SHA-256
- SHA-384
- SHA-512

> **Note**
> Data under 16 KB in length are hashed on the HSM, while larger data are hashed locally in software.

Supported hash-based message authentication code (HMAC) algorithms

The AWS CloudHSM software library for Java supports the following HMAC algorithms.

- HmacSHA1
- HmacSHA224
- HmacSHA256
- HmacSHA384
- HmacSHA512

Supported sign/verify mechanisms

The AWS CloudHSM software library for Java supports the following types of signature and verification.

RSA Signature Types

- `NONEwithRSA`
- `SHA1withRSA`
- `SHA224withRSA`
- `SHA256withRSA`
- `SHA384withRSA`
- `SHA512withRSA`
- `SHA1withRSA/PSS`
- `SHA224withRSA/PSS`
- `SHA256withRSA/PSS`
- `SHA384withRSA/PSS`
- `SHA512withRSA/PSS`

ECDSA Signature Types

- `NONEwithECDSA`
- `SHA1withECDSA`
- `SHA224withECDSA`
- `SHA256withECDSA`
- `SHA384withECDSA`
- `SHA512withECDSA`

Mechanism annotations

[1] In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.

Supported Java key attributes for AWS CloudHSM Client SDK 3

This topic describes how to use a proprietary extension for the Java library version 3.1 to set key attributes for AWS CloudHSM Client SDK 3. Use this extension to set supported key attributes and their values during these operations:

- Key generation
- Key import
- Key unwrap

> **Note**
>
> The extension for setting custom key attributes is an optional feature. If you already have code that functions in Java library version 3.0, you do not need to modify that code. Keys you create will continue to contain the same attributes as before.

Topics

- Understanding attributes
- Supported attributes
- Setting attributes for a key
- Putting it all together

Understanding attributes

You use key attributes to specify what actions are permitted on key objects, including public, private or secret keys. You define key attributes and values during key object creation operations.

However, the Java Cryptography Extension (JCE) does not specify how you should set values on key attributes, so most actions were permitted by default. In contrast, the PKCS# 11 standard defines a comprehensive set of attributes with more restrictive defaults. Starting with the Java library version 3.1, CloudHSM provides a proprietary extension that enables you to set more restrictive values for commonly used attributes.

Supported attributes

You can set values for the attributes listed in the table below. As a best practice, only set values for attributes you wish to make restrictive. If you don't specify a value, CloudHSM uses the default value specified in the table below. An empty cell in the Default Value columns indicates that there is no specific default value assigned to the attribute.

Attribute	Default Value			Notes
	Symmetric Key	Public Key in Key Pair	Private Key in Key Pair	
CKA_TOKEN	FALSE	FALSE	FALSE	A permanent key which is replicated across all HSMs in the cluster and included in backups. CKA_TOKEN = FALSE implies a session key, which is only loaded onto one HSM and automatically erased when the connection to the HSM is broken.
CKA_LABEL				A user-defined string. It allows you to conveniently identify keys on your HSM.

Attribute	Default Value			Notes
CKA_EXTRACTABLE	TRUE		TRUE	True indicates you can export this key out of the HSM.
CKA_ENCRYPT	TRUE	TRUE		True indicates you can use the key to encrypt any buffer.
CKA_DECRYPT	TRUE		TRUE	True indicates you can use the key to decrypt any buffer. You generally set this to FALSE for a key whose CKA_WRAP is set to true.
CKA_WRAP	TRUE	TRUE		True indicates you can use the key to wrap another key. You will generally set this to FALSE for private keys.
CKA_UNWRAP	TRUE		TRUE	True indicates you can use the key to unwrap (import) another key.

Attribute	Default Value			Notes
`CKA_SIGN`	`TRUE`		`TRUE`	True indicates you can use the key to sign a message digest. This is generally set to FALSE for public keys and for private keys that you have archived.
`CKA_VERIFY`	`TRUE`	`TRUE`		True indicates you can use the key to verify a signature. This is generally set to FALSE for private keys.
`CKA_PRIVATE`	`TRUE`	`TRUE`	`TRUE`	True indicates that a user may not access the key until the user is authenticated. For clarity, users cannot access any keys on CloudHSM until they are authenticated, even if this attribute is set to FALSE.

> **Note**
>
> You get broader support for attributes in the PKCS#11 library. For more information, see Supported PKCS #11 Attributes.

Setting attributes for a key

`CloudHsmKeyAttributesMap` is a Java Map-like object, which you can use to set attribute values for key objects. The methods for `CloudHsmKeyAttributesMap` function similar to the methods used for Java map manipulation.

To set custom values on attributes, you have two options:

- Use the methods listed in the following table
- Use builder patterns demonstrated later in this document

Attribute map objects support the following methods to set attributes:

Operation	Return Value	`CloudHSMKeyAttributesMap` method
Get the value of a key attribute for an existing key	Object (containing the value) or *null*	**get**(keyAttribute)
Populate the value of one key attribute	The previous value associated with key attribute, or *null* if there was no mapping for a key attribute	**put**(keyAttribute, value)
Populate values for multiple key attributes	N/A	**putAll**(keyAttributesMap)
Remove a key-value pair from the attribute map	The previous value associated with key attribute, or *null* if there was no mapping for a key attribute	**remove**(keyAttribute)

Note

Any attributes you do not explicitly specify are set to the defaults listed in the preceding table in the section called “Supported attributes”.

Builder pattern example

Developers will generally find it more convenient to utilize classes through the Builder pattern. As examples:

```
import com.amazonaws.cloudhsm.CloudHsmKeyAttributes;
import com.amazonaws.cloudhsm.CloudHsmKeyAttributesMap;
import com.amazonaws.cloudhsm.CloudHsmKeyPairAttributesMap;

CloudHsmKeyAttributesMap keyAttributesSessionDecryptionKey =
   new CloudHsmKeyAttributesMap.Builder()
      .put(CloudHsmKeyAttributes.CKA_LABEL, "ExtractableSessionKeyEncryptDecrypt")
      .put(CloudHsmKeyAttributes.CKA_WRAP, false)
      .put(CloudHsmKeyAttributes.CKA_UNWRAP, false)
      .put(CloudHsmKeyAttributes.CKA_SIGN, false)
      .put(CloudHsmKeyAttributes.CKA_VERIFY, false)
      .build();

CloudHsmKeyAttributesMap keyAttributesTokenWrappingKey =
   new CloudHsmKeyAttributesMap.Builder()
      .put(CloudHsmKeyAttributes.CKA_LABEL, "TokenWrappingKey")
      .put(CloudHsmKeyAttributes.CKA_TOKEN, true)
      .put(CloudHsmKeyAttributes.CKA_ENCRYPT, false)
      .put(CloudHsmKeyAttributes.CKA_DECRYPT, false)
      .put(CloudHsmKeyAttributes.CKA_SIGN, false)
      .put(CloudHsmKeyAttributes.CKA_VERIFY, false)
      .build();
```

Developers may also utilize pre-defined attribute sets as a convenient way to enforce best practices in key templates. As an example:

```
//best practice template for wrapping keys

CloudHsmKeyAttributesMap commonKeyAttrs = new CloudHsmKeyAttributesMap.Builder()
    .put(CloudHsmKeyAttributes.CKA_EXTRACTABLE, false)
```

```
    .put(CloudHsmKeyAttributes.CKA_DECRYPT, false)
    .build();

// initialize a new instance of CloudHsmKeyAttributesMap by copying commonKeyAttrs
// but with an appropriate label

CloudHsmKeyAttributesMap firstKeyAttrs = new CloudHsmKeyAttributesMap(commonKeyAttrs);
firstKeyAttrs.put(CloudHsmKeyAttributes.CKA_LABEL, "key label");

// alternatively, putAll() will overwrite existing values to enforce conformance

CloudHsmKeyAttributesMap secondKeyAttrs = new CloudHsmKeyAttributesMap();
secondKeyAttrs.put(CloudHsmKeyAttributes.CKA_DECRYPT, true);
secondKeyAttrs.put(CloudHsmKeyAttributes.CKA_ENCRYPT, true);
secondKeyAttrs.put(CloudHsmKeyAttributes.CKA_LABEL, “safe wrapping key”);
secondKeyAttrs.putAll(commonKeyAttrs); // will overwrite CKA_DECRYPT to be FALSE
```

Setting attributes for a key pair

Use the Java class `CloudHsmKeyPairAttributesMap` to handle key attributes for a key pair. `CloudHsmKeyPairAttributesMap` encapsulates two `CloudHsmKeyAttributesMap` objects; one for a public key and one for a private key.

To set individual attributes for the public key and private key separately, you can use the `put()` method on corresponding `CloudHsmKeyAttributes` map object for that key. Use the `getPublic()` method to retrieve the attribute map for the public key, and use `getPrivate()` to retrieve the attribute map for the private key. Populate the value of multiple key attributes together for both public and private key pairs using the `putAll()` with a key pair attributes map as its argument.

Builder pattern example

Developers will generally find it more convenient to set key attributes through the Builder pattern. For example:

```
import com.amazonaws.cloudhsm.CloudHsmKeyAttributes;
import com.amazonaws.cloudhsm.CloudHsmKeyAttributesMap;
import com.amazonaws.cloudhsm.CloudHsmKeyPairAttributesMap;

//specify attributes up-front
CloudHsmKeyAttributesMap keyAttributes =
    new CloudHsmKeyAttributesMap.Builder()
```

```
        .put(CloudHsmKeyAttributes.CKA_SIGN, false)
        .put(CloudHsmKeyAttributes.CKA_LABEL, "PublicCertSerial12345")
        .build();

CloudHsmKeyPairAttributesMap keyPairAttributes =
    new CloudHsmKeyPairAttributesMap.Builder()
        .withPublic(keyAttributes)
        .withPrivate(
            new CloudHsmKeyAttributesMap.Builder() //or specify them inline
                .put(CloudHsmKeyAttributes.CKA_LABEL, "PrivateCertSerial12345")
                .put (CloudHSMKeyAttributes.CKA_WRAP, FALSE)
                .build()
        )
        .build();
```

Note

For more information about this proprietary extension, see the Javadoc archive and the sample on GitHub. To explore the Javadoc, download and expand the archive.

Putting it all together

To specify key attributes with your key operations, follow these steps:

1. Instantiate `CloudHsmKeyAttributesMap` for symmetric keys or `CloudHsmKeyPairAttributesMap` for key pairs.
2. Define the attributes object from step 1 with the required key attributes and values.
3. Instantiate a `Cavium*ParameterSpec` class, corresponding to your specific key type, and pass into its constructor this configured attributes object.
4. Pass this `Cavium*ParameterSpec` object into a corresponding crypto class or method.

For reference, the following table contains the `Cavium*ParameterSpec` classes and methods which support custom key attributes.

Key Type	Parameter Spec Class	Example Constructors
Base Class	`CaviumKeyGenAlgorithmParameterSpec`	`CaviumKeyGenAlgorithmParameterSpec(C`

Key Type	Parameter Spec Class	Example Constructors
		loudHsmKeyAttributesMap keyAttributesMap)
DES	CaviumDESKeyGenParameterSpec	CaviumDESKeyGenParameterSpec(int keySize, byte[] iv, CloudHsmKeyAttributesMap keyAttributesMap)
RSA	CaviumRSAKeyGenParameterSpec	CaviumRSAKeyGenParameterSpec(int keysize, BigInteger publicExponent, CloudHsmKeyPairAttributesMap keyPairAttributesMap)
Secret	CaviumGenericSecretKeyGenParameterSpec	CaviumGenericSecretKeyGenParameterSpec(int size, CloudHsmKeyAttributesMap keyAttributesMap)
AES	CaviumAESKeyGenParameterSpec	CaviumAESKeyGenParameterSpec(int keySize, byte[] iv, CloudHsmKeyAttributesMap keyAttributesMap)

Key Type	Parameter Spec Class	Example Constructors
EC	CaviumECGenParameterSpec	CaviumECGenParameterSpec(String stdName, CloudHsmKeyPairAttributesMap keyPairAttributesMap)

Sample code: Generate and wrap a key

These brief code samples demonstrate the steps for two different operations: Key Generation and Key Wrapping:

```
// Set up the desired key attributes

KeyGenerator keyGen = KeyGenerator.getInstance("AES", "Cavium");
CaviumAESKeyGenParameterSpec keyAttributes = new CaviumAESKeyGenParameterSpec(
    256,
    new CloudHsmKeyAttributesMap.Builder()
        .put(CloudHsmKeyAttributes.CKA_LABEL, "MyPersistentAESKey")
        .put(CloudHsmKeyAttributes.CKA_EXTRACTABLE, true)
        .put(CloudHsmKeyAttributes.CKA_TOKEN, true)
        .build()
);

// Assume we already have a handle to the myWrappingKey
// Assume we already have the wrappedBytes to unwrap

// Unwrap a key using Custom Key Attributes

CaviumUnwrapParameterSpec unwrapSpec = new
 CaviumUnwrapParameterSpec(myInitializationVector, keyAttributes);

Cipher unwrapCipher = Cipher.getInstance("AESWrap", "Cavium");
unwrapCipher.init(Cipher.UNWRAP_MODE, myWrappingKey, unwrapSpec);
Key unwrappedKey = unwrapCipher.unwrap(wrappedBytes, "AES", Cipher.SECRET_KEY);
```

Code samples for the AWS CloudHSM software library for Java for Client SDK 3

This topic provides resources and information on Java code samples for AWS CloudHSM Client SDK 3.

Prerequisites

Before running the samples, you must set up your environment:

- Install and configure the Java Cryptographic Extension (JCE) provider and the AWS CloudHSM client package.
- Set up a valid HSM user name and password. Cryptographic user (CU) permissions are sufficient for these tasks. Your application uses these credentials to log in to the HSM in each example.
- Decide how to provide credentials to the JCE provider.

Code samples

The following code samples show you how to use the AWS CloudHSM JCE provider to perform basic tasks. More code samples are available on GitHub.

- Log in to an HSM
- Manage keys
- Generate an AES key
- Encrypt and decrypt with AES-GCM
- Encrypt and decrypt with AES-CTR
- Encrypt and decrypt with D3DES-ECB **see note [1]**
- Wrap and unwrap keys with AES-GCM
- Wrap and unwrap keys with AES
- Wrap and unwrap keys with RSA
- Use supported key attributes
- Enumerate keys in the key store
- Use the CloudHSM key store
- Sign messages in a multi-threaded sample
- Sign and Verify with EC Keys

[1] In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.

AWS CloudHSM KeyStore Java class for Client SDK 3

The AWS CloudHSM `KeyStore` class provides a special-purpose PKCS12 key store that allows access to AWS CloudHSM keys through applications such as **keytool** and **jarsigner**. This key store can store certificates along with your key data and correlate them to key data stored on AWS CloudHSM.

> **Note**
>
> Because certificates are public information, and to maximize storage capacity for cryptographic keys, AWS CloudHSM does not support storing certificates on HSMs.

The AWS CloudHSM `KeyStore` class implements the `KeyStore` Service Provider Interface (SPI) of the Java Cryptography Extension (JCE). For more information about using `KeyStore`, see Class KeyStore.

Choose the appropriate key store for AWS CloudHSM Client SDK 3

The AWS CloudHSM Java Cryptographic Extension (JCE) provider comes with a default pass-through, read-only key store that passes all transactions to the HSM. This default key store is distinct from the special-purpose AWS CloudHSM KeyStore. In most situations, you will obtain better runtime performance and throughput by using the default. You should only use the AWS CloudHSM KeyStore for applications where you require support for certificates and certificate-based operations in addition to offloading key operations to the HSM.

Although both key stores use the JCE provider for operations, they are independent entities and do not exchange information with each other.

Load the default key store for your Java application as follows:

```
KeyStore ks = KeyStore.getInstance("Cavium");
```

Load the special-purpose CloudHSM KeyStore as follows:

```
KeyStore ks = KeyStore.getInstance("CloudHSM")
```

Initialize the AWS CloudHSM KeyStore for Client SDK 3

Log into the AWS CloudHSM KeyStore the same way that you log into the JCE provider. You can use either environment variables or the system property file, and you should log in before you start using the CloudHSM KeyStore. For an example of logging into an HSM using the JCE provider, see Login to an HSM.

If desired, you can specify a password to encrypt the local PKCS12 file which holds key store data. When you create the AWS CloudHSM Keystore, you set the password and provide it when using the load, set and get methods.

Instantiate a new CloudHSM KeyStore object as follows:

```
ks.load(null, null);
```

Write keystore data to a file using the `store` method. From that point on, you can load the existing keystore using the `load` method with the source file and password as follows:

```
ks.load(inputStream, password);
```

Use the AWS CloudHSM KeyStore for AWS CloudHSM Client SDK 3

A CloudHSM KeyStore object is generally used through a third-party application such as jarsigner or keytool. You can also access the object directly with code.

AWS CloudHSM KeyStore complies with the JCE Class KeyStore specification and provides the following functions.

- `load`

  Loads the key store from the given input stream. If a password was set when saving the key store, this same password must be provided for the load to succeed. Set both parameters to null to initialize an new empty key store.

  ```
  KeyStore ks = KeyStore.getInstance("CloudHSM");
  ks.load(inputStream, password);
  ```

- `aliases`

Returns an enumeration of the alias names of all entries in the given key store instance. Results include objects stored locally in the PKCS12 file and objects resident on the HSM.

Sample code:

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
for(Enumeration<String> entry = ks.aliases(); entry.hasMoreElements();)
 {
  String label = entry.nextElement();
  System.out.println(label);
 }
```

- `ContainsAlias`

  Returns true if the key store has access to at least one object with the specified alias. The key store checks objects stored locally in the PKCS12 file and objects resident on the HSM.

- `DeleteEntry`

  Deletes a certificate entry from the local PKCS12 file. Deleting key data stored in an HSM is not supported using the AWS CloudHSM KeyStore. You can delete keys with CloudHSM's key_mgmt_util tool.

- `GetCertificate`

  Returns the certificate associated with an alias if available. If the alias does not exist or references an object which is not a certificate, the function returns NULL.

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
Certificate cert = ks.getCertificate(alias)
```

- `GetCertificateAlias`

  Returns the name (alias) of the first key store entry whose data matches the given certificate.

```
KeyStore ks = KeyStore.getInstance("CloudHSM");
String alias = ks.getCertificateAlias(cert)
```

- `GetCertificateChain`

  Returns the certificate chain associated with the given alias. If the alias does not exist or references an object which is not a certificate, the function returns NULL.

- `GetCreationDate`

  Returns the creation date of the entry identified by the given alias. If a creation date is not available, the function returns the date on which the certificate became valid.

- `GetKey`

  GetKey is passed to the HSM and returns a key object corresponding to the given label. As `getKey` directly queries the HSM, it can be used for any key on the HSM regardless of whether it was generated by the KeyStore.

  ```
  Key key = ks.getKey(keyLabel, null);
  ```

- `IsCertificateEntry`

  Checks if the entry with the given alias represents a certificate entry.

- `IsKeyEntry`

  Checks if the entry with the given alias represents a key entry. The action searches both the PKCS12 file and the HSM for the alias.

- `SetCertificateEntry`

  Assigns the given certificate to the given alias. If the given alias is already being used to identify a key or certificate, a `KeyStoreException` is thrown. You can use JCE code to get the key object and then use the KeyStore `SetKeyEntry` method to associate the certificate to the key.

- `SetKeyEntry` with `byte[]` key

  This API is currently unsupported with Client SDK 3.

- `SetKeyEntry` with `Key` object

  Assigns the given key to the given alias and stores it inside the HSM. If the `Key` object is not of type `CaviumKey`, the key is imported into the HSM as an extractable session key.

  If the `Key` object is of type `PrivateKey`, it must be accompanied by a corresponding certificate chain.

  If the alias already exists, the `SetKeyEntry` call throws a `KeyStoreException` and prevents the key from being overwritten. If the key must be overwritten, use KMU or JCE for that purpose.

- `EngineSize`

Returns the number of entries in the keystore.

- `Store`

  Stores the key store to the given output stream as a PKCS12 file and secures it with the given password. In addition, it persists all loaded keys (which are set using `setKey` calls).

Cryptography API: Next Generation (CNG) and key storage providers (KSP) for AWS CloudHSM

The AWS CloudHSM client for Windows includes CNG and KSP providers.

Key storage providers (KSPs) enable key storage and retrieval. For example, if you add the Microsoft Active Directory Certificate Services (AD CS) role to your Windows server and choose to create a new private key for your certificate authority (CA), you can choose the KSP that will manage key storage. When you configure the AD CS role, you can choose this KSP. For more information, see Create Windows Server CA.

Cryptography API: Next Generation (CNG) is a cryptographic API specific to the Microsoft Windows operating system. CNG enables developers to use cryptographic techniques to secure Windows-based applications. At a high level, the AWS CloudHSM implementation of CNG provides the following functionality:

- **Cryptographic Primitives –** enable you to perform fundamental cryptographic operations.
- **Key Import and Export –** enables you to import and export asymmetric keys.
- **Data Protection API (CNG DPAPI) –** enables you to easily encrypt and decrypt data.
- **Key Storage and Retrieval** -–enables you to securely store and isolate the private key of an asymmetric key pair.

Topics

- Verify the KSP and CNG Providers for AWS CloudHSM
- Prerequisites for using the AWS CloudHSM Windows Client
- Associate an AWS CloudHSM key with a certificate
- Code sample for CNG provider for AWS CloudHSM

Verify the KSP and CNG Providers for AWS CloudHSM

The KSP and CNG providers are installed when you install the Windows AWS CloudHSM client. You can install the client by following the steps at Install the client (Windows).

Use the following sections to verify the installation of the providers.

Configure and run the Windows AWS CloudHSM client

To start the Windows CloudHSM client, you must first satisfy the Prerequisites. Then, update the configuration files that the providers use and start the client by completing the steps below. You need to do these steps the first time you use the KSP and CNG providers and after you add or remove HSMs in your cluster. This way, AWS CloudHSM is able to synchronize data and maintain consistency across all HSMs in the cluster.

Step 1: Stop the AWS CloudHSM client

Before you update the configuration files that the providers use, stop the AWS CloudHSM client. If the client is already stopped, running the stop command has no effect.

- For Windows client 1.1.2+:

  ```
  C:\Program Files\Amazon\CloudHSM>net.exe stop AWSCloudHSMClient
  ```

- For Windows clients 1.1.1 and older:

  Use **Ctrl+C** in the command window where you started the AWS CloudHSM client.

Step 2: Update the AWS CloudHSM configuration files

This step uses the `-a` parameter of the Configure tool to add the elastic network interface (ENI) IP address of one of the HSMs in the cluster to the configuration file.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\configure.exe" -a <HSM ENI IP>
```

To get the ENI IP address of an HSM in your cluster, navigate to the AWS CloudHSM console, choose **clusters**, and select the desired cluster. You can also use the DescribeClusters operation, the describe-clusters command, or the Get-HSM2Cluster PowerShell cmdlet. Type only one ENI IP address. It does not matter which ENI IP address you use.

Step 3: Start the AWS CloudHSM client

Next, start or restart the AWS CloudHSM client. When the AWS CloudHSM client starts, it uses the ENI IP address in its configuration file to query the cluster. Then it adds the ENI IP addresses of all HSMs in the cluster to the cluster information file.

- For Windows client 1.1.2+:

```
C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
```

- For Windows clients 1.1.1 and older:

```
C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe C:
\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

Checking the KSP and CNG providers

You can use either of the following commands to determine which providers are installed on your system. The commands list the registered KSP and CNG providers. The AWS CloudHSM client does not need to be running.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\ksp_config.exe" -enum
```

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\cng_config.exe" -enum
```

To verify that the KSP and CNG providers are installed on your Windows Server EC2 instance, you should see the following entries in the list:

```
Cavium CNG Provider
Cavium Key Storage Provider
```

If the CNG provider is missing, run the following command.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\cng_config.exe" -register
```

If the KSP provider is missing, run the following command.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\ksp_config.exe" -register
```

Prerequisites for using the AWS CloudHSM Windows Client

Before you can start the Windows AWS CloudHSM client and use the KSP and CNG providers, you must set the login credentials for the HSM on your system. You can set credentials through either Windows Credentials Manager or system environment variable. We recommend you use Windows Credential Manager for storing credentials. This option is available with AWS CloudHSM client version 2.0.4 and later. Using environment variable is easier to set up, but less secure than using Windows Credential Manager.

Windows Credential Manager

You can use either the `set_cloudhsm_credentials` utility or the Windows Credentials Manager interface.

- **Using the `set_cloudhsm_credentials` utility**:

  The `set_cloudhsm_credentials` utility is included in your Windows installer. You can use this utility to conveniently pass HSM login credentials to Windows Credential Manager. If you want to compile this utility from source, you can use the Python code that is included in the installer.

  1. Go to the `C:\Program Files\Amazon\CloudHSM\tools\` folder.
  2. Run the `set_cloudhsm_credentials.exe` file with the CU username and password parameters.

     ```
     set_cloudhsm_credentials.exe --username <CU USER> --password <CU PASSWORD>
     ```

- **Using the Credential Manager interface**:

  You can use the Credential Manager interface to manually manage your credentials.

  1. To open Credential Manager, type `credential manager` in the search box on the taskbar and select **Credential Manager**.
  2. Select **Windows Credentials** to manage Windows credentials.
  3. Select **Add a generic credential** and fill out the details as follows:
     - In **Internet or Network Address**, enter the target name as `cloudhsm_client`.
     - In **Username** and **Password** enter the CU credentials.
     - Click **OK**.

System environment variables

You can set system environment variables that identify an HSM and a crypto user (CU) for your Windows application. You can use the **setx** command to set system environment variables, or set permanent system environment variables programmatically or in the **Advanced** tab of the Windows **System Properties** Control Panel.

> **Warning**
>
> When you set credentials through system environment variables, the password is available in plaintext on a user's system. To overcome this problem, use Windows Credential Manager.

Set the following system environment variables:

`n3fips_password=`*`<CU USERNAME>`*`:`*`<CU PASSWORD>`*

Identifies a crypto user (CU) in the HSM and provides all required login information. Your application authenticates and runs as this CU. The application has the permissions of this CU and can view and manage only the keys that the CU owns and shares. To create a new CU, use createUser. To find existing CUs, use listUsers.

For example:

```
setx /m n3fips_password test_user:password123
```

Associate an AWS CloudHSM key with a certificate

Before you can use AWS CloudHSM keys with third-party tools, such as Microsoft's SignTool, you must import the key's metadata into the local certificate store and associate the metadata with a certificate. To import the key's metadata, use the import_key.exe utility which is included in CloudHSM version 3.0 and higher. The following steps provide additional information, and sample output.

Step 1: Import your certificate

On Windows, you should be able to double-click the certificate to import it to your local certificate store.

However, if double-clicking doesn't work, use the Microsoft Certreq tool to import the certificate into the certificate manager. For example:

```
certreq -accept <certificatename>
```

If this action fails and you receive the error, `Key not found`, continue to Step 2. If the certificate appears in your key store, you've completed the task and no further action is necessary.

Step 2: Gather certificate-identifying information

If the previous step wasn't successful, you'll need to associate your private key with a certificate. However, before you can create the association, you must first find the certificate's Unique Container Name and Serial Number. Use a utility, such as **certutil**, to display the needed certificate information. The following sample output from **certutil** shows the container name and the serial number.

```
================ Certificate 1 ================ Serial Number:
   72000000047f7f7a9d41851b4e000000000004Issuer: CN=Enterprise-CANotBefore: 10/8/2019
 11:50
   AM NotAfter: 11/8/2020 12:00 PMSubject: CN=www.example.com, OU=Certificate
 Management,
   O=Information Technology, L=Seattle, S=Washington, C=USNon-root CertificateCert
   Hash(sha1): 7f d8 5c 00 27 bf 37 74 3d 71 5b 54 4e c0 94 20 45 75 bc 65No key
 provider
   information Simple container name: CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c
 Unique
   container name: CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c
```

Step 3: Associate the AWS CloudHSM private key with the certificate

To associate the key with the certificate, first be sure to start the AWS CloudHSM client daemon. Then, use import_key.exe (which is included in CloudHSM version 3.0 and higher) to associate the private key with the certificate. When specifying the certificate, use its simple container name. The following example shows the command and the response. This action only copies the key's metadata; the key remains on the HSM.

```
$> import_key.exe –RSA CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c

Successfully opened Microsoft Software Key Storage Provider : 0NCryptOpenKey failed :
 80090016
```

Step 4: Update the certificate store

Be certain the AWS CloudHSM client daemon is still running. Then, use the **certutil** verb, **-repairstore**, to update the certificate serial number. The following sample shows the command and output. See the Microsoft documentation for information about the **-repairstore** verb.

```
C:\Program Files\Amazon\CloudHSM>certutil -f -csp "Cavium Key Storage Provider"-
repairstore my "72000000047f7f7a9d41851b4e000000000004"
my "Personal"
================ Certificate 1 ================
Serial Number: 72000000047f7f7a9d41851b4e000000000004
Issuer: CN=Enterprise-CA
NotBefore: 10/8/2019 11:50 AM
NotAfter: 11/8/2020 12:00 PM
Subject: CN=www.example.com, OU=Certificate Management, O=Information Technology,
 L=Seattle, S=Washington, C=US
Non-root CertificateCert Hash(sha1): 7f d8 5c 00 27 bf 37 74 3d 71 5b 54 4e c0 94 20 45
 75 bc 65
SDK Version: 3.0
Key Container = CertReq-39c04db0-6aa9-4310-93db-db0d9669f42c
Provider = "Cavium Key Storage Provider"
Private key is NOT exportableEncryption test passedCertUtil: -repairstore command
 completed successfully.
```

After updating the certificate serial number you can use this certificate and the corresponding AWS CloudHSM private key with any third-party signing tool on Windows.

Code sample for CNG provider for AWS CloudHSM

** Example code only – Not for production use **
This sample code is for illustrative purposes only. Do not run this code in production.

The following sample shows how to enumerate the registered cryptographic providers on your system to find the CNG provider installed with CloudHSM client for Windows. The sample also shows how to create an asymmetric key pair and how to use the key pair to sign data.

> **Important**
>
> Before you run this example, you must set up the HSM credentials as explained in the prerequisites. For details, see [Prerequisites for using the AWS CloudHSM Windows Client](#).

```
// CloudHsmCngExampleConsole.cpp : Console application that demonstrates CNG
 capabilities.
// This example contains the following functions.
//
//   VerifyProvider()          - Enumerate the registered providers and retrieve Cavium
 KSP and CNG providers.
//   GenerateKeyPair()         - Create an RSA key pair.
//   SignData()                - Sign and verify data.
//

#include "stdafx.h"
#include <Windows.h>

#ifndef NT_SUCCESS
#define NT_SUCCESS(Status) ((NTSTATUS)(Status) >= 0)
#endif

#define CAVIUM_CNG_PROVIDER L"Cavium CNG Provider"
#define CAVIUM_KEYSTORE_PROVIDER L"Cavium Key Storage Provider"

// Enumerate the registered providers and determine whether the Cavium CNG provider
// and the Cavium KSP provider exist.
//
bool VerifyProvider()
{
  NTSTATUS status;
  ULONG cbBuffer = 0;
  PCRYPT_PROVIDERS pBuffer = NULL;
  bool foundCng = false;
  bool foundKeystore = false;

  // Retrieve information about the registered providers.
  //   cbBuffer - the size, in bytes, of the buffer pointed to by pBuffer.
  //   pBuffer - pointer to a buffer that contains a CRYPT_PROVIDERS structure.
```

```
    status = BCryptEnumRegisteredProviders(&cbBuffer, &pBuffer);

    // If registered providers exist, enumerate them and determine whether the
    // Cavium CNG provider and Cavium KSP provider have been registered.
    if (NT_SUCCESS(status))
    {
      if (pBuffer != NULL)
      {
        for (ULONG i = 0; i < pBuffer->cProviders; i++)
        {
          // Determine whether the Cavium CNG provider exists.
          if (wcscmp(CAVIUM_CNG_PROVIDER, pBuffer->rgpszProviders[i]) == 0)
          {
            printf("Found %S\n", CAVIUM_CNG_PROVIDER);
            foundCng = true;
          }

          // Determine whether the Cavium KSP provider exists.
          else if (wcscmp(CAVIUM_KEYSTORE_PROVIDER, pBuffer->rgpszProviders[i]) == 0)
          {
            printf("Found %S\n", CAVIUM_KEYSTORE_PROVIDER);
            foundKeystore = true;
          }
        }
      }
    }
    else
    {
      printf("BCryptEnumRegisteredProviders failed with error code 0x%08x\n", status);
    }

    // Free memory allocated for the CRYPT_PROVIDERS structure.
    if (NULL != pBuffer)
    {
      BCryptFreeBuffer(pBuffer);
    }

    return foundCng == foundKeystore == true;
  }

  // Generate an asymmetric key pair. As used here, this example generates an RSA key
   pair
  // and returns a handle. The handle is used in subsequent operations that use the key
   pair.
```

```
// The key material is not available.
//
// The key pair is used in the SignData function.
//
NTSTATUS GenerateKeyPair(BCRYPT_ALG_HANDLE hAlgorithm, BCRYPT_KEY_HANDLE *hKey)
{
  NTSTATUS status;

  // Generate the key pair.
  status = BCryptGenerateKeyPair(hAlgorithm, hKey, 2048, 0);
  if (!NT_SUCCESS(status))
  {
    printf("BCryptGenerateKeyPair failed with code 0x%08x\n", status);
    return status;
  }

  // Finalize the key pair. The public/private key pair cannot be used until this
  // function is called.
  status = BCryptFinalizeKeyPair(*hKey, 0);
  if (!NT_SUCCESS(status))
  {
    printf("BCryptFinalizeKeyPair failed with code 0x%08x\n", status);
    return status;
  }

  return status;
}

// Sign and verify data using the RSA key pair. The data in this function is hardcoded
// and is for example purposes only.
//
NTSTATUS SignData(BCRYPT_KEY_HANDLE hKey)
{
  NTSTATUS status;
  PBYTE sig;
  ULONG sigLen;
  ULONG resLen;
  BCRYPT_PKCS1_PADDING_INFO pInfo;

  // Hardcode the data to be signed (for demonstration purposes only).
  PBYTE message = (PBYTE)"d83e7716bed8a20343d8dc6845e57447";
  ULONG messageLen = strlen((char*)message);

  // Retrieve the size of the buffer needed for the signature.
```

```
    status = BCryptSignHash(hKey, NULL, message, messageLen, NULL, 0, &sigLen, 0);
    if (!NT_SUCCESS(status))
    {
      printf("BCryptSignHash failed with code 0x%08x\n", status);
      return status;
    }

    // Allocate a buffer for the signature.
    sig = (PBYTE)HeapAlloc(GetProcessHeap(), 0, sigLen);
    if (sig == NULL)
    {
      return -1;
    }

    // Use the SHA256 algorithm to create padding information.
    pInfo.pszAlgId = BCRYPT_SHA256_ALGORITHM;

    // Create a signature.
    status = BCryptSignHash(hKey, &pInfo, message, messageLen, sig, sigLen, &resLen,
  BCRYPT_PAD_PKCS1);
    if (!NT_SUCCESS(status))
    {
      printf("BCryptSignHash failed with code 0x%08x\n", status);
      return status;
    }

    // Verify the signature.
    status = BCryptVerifySignature(hKey, &pInfo, message, messageLen, sig, sigLen,
  BCRYPT_PAD_PKCS1);
    if (!NT_SUCCESS(status))
    {
      printf("BCryptVerifySignature failed with code 0x%08x\n", status);
      return status;
    }

    // Free the memory allocated for the signature.
    if (sig != NULL)
    {
      HeapFree(GetProcessHeap(), 0, sig);
      sig = NULL;
    }

    return 0;
}
```

```
// Main function.
//
int main()
{
  NTSTATUS status;
  BCRYPT_ALG_HANDLE hRsaAlg;
  BCRYPT_KEY_HANDLE hKey = NULL;

  // Enumerate the registered providers.
  printf("Searching for Cavium providers...\n");
  if (VerifyProvider() == false) {
    printf("Could not find the CNG and Keystore providers\n");
    return 1;
  }

  // Get the RSA algorithm provider from the Cavium CNG provider.
  printf("Opening RSA algorithm\n");
  status = BCryptOpenAlgorithmProvider(&hRsaAlg, BCRYPT_RSA_ALGORITHM,
 CAVIUM_CNG_PROVIDER, 0);
  if (!NT_SUCCESS(status))
  {
    printf("BCryptOpenAlgorithmProvider RSA failed with code 0x%08x\n", status);
    return status;
  }

  // Generate an asymmetric key pair using the RSA algorithm.
  printf("Generating RSA Keypair\n");
  GenerateKeyPair(hRsaAlg, &hKey);
  if (hKey == NULL)
  {
    printf("Invalid key handle returned\n");
    return 0;
  }
  printf("Done!\n");

  // Sign and verify [hardcoded] data using the RSA key pair.
  printf("Sign/Verify data with key\n");
  SignData(hKey);
  printf("Done!\n");

  // Remove the key handle from memory.
  status = BCryptDestroyKey(hKey);
  if (!NT_SUCCESS(status))
```

```
  {
    printf("BCryptDestroyKey failed with code 0x%08x\n", status);
    return status;
  }

  // Close the RSA algorithm provider.
  status = BCryptCloseAlgorithmProvider(hRsaAlg, NULL);
  if (!NT_SUCCESS(status))
  {
    printf("BCryptCloseAlgorithmProvider RSA failed with code 0x%08x\n", status);
    return status;
  }

  return 0;
}
```

Integrating third-party applications with AWS CloudHSM

Some of the use cases for AWS CloudHSM involve integrating third-party software applications with the HSM in your AWS CloudHSM cluster. By integrating third-party software with AWS CloudHSM, you can accomplish a variety of security-related goals. The following topics describe how to accomplish some of these goals.

Topics

Improve your web server security with SSL/TLS offload in AWS CloudHSM

Web servers and their clients (web browsers) can use Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocols to confirm the identity of the web server and establish a secure connection that sends and receives webpages or other data over the internet. This is commonly known as HTTPS. The web server uses a public–private key pair and an SSL/TLS public key certificate to establish an HTTPS session with each client. This process involves a lot of computation for web servers, but you can offload some of this to your AWS CloudHSM cluster, which is referred to as SSL acceleration. Offloading reduces the computational burden on your web servers and provides extra security by storing servers' private keys in HSMs.

The following topics provide an overview of how SSL/TLS offload with AWS CloudHSM works and tutorials for setting up SSL/TLS offload with AWS CloudHSM on the following platforms.

For **Linux**, use OpenSSL Dynamic Engine on the NGINX or Apache HTTP Server web server software

For **Windows**, use the Internet Information Services (IIS) for Windows Server web server software

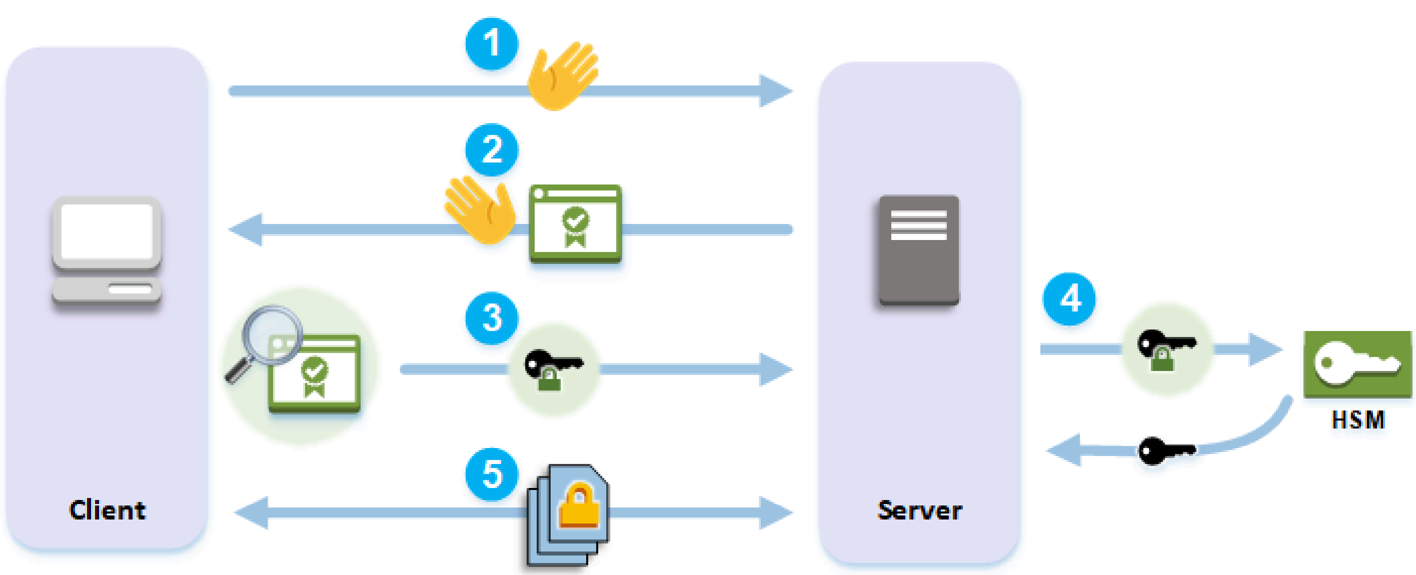
Topics

- How SSL/TLS offload with AWS CloudHSM works
- AWS CloudHSM SSL/TLS offload on Linux using NGINX or Apache with OpenSSL
- AWS CloudHSM SSL/TLS offload on Linux using Tomcat with JSSE
- AWS CloudHSM SSL/TLS offload on Windows using IIS with KSP
- Add a load balancer with Elastic Load Balancing for AWS CloudHSM(optional)

How SSL/TLS offload with AWS CloudHSM works

To establish an HTTPS connection, your web server performs a handshake process with clients. As part of this process, the server offloads some of the cryptographic processing to the HSMs in the AWS CloudHSM cluster, as shown in the following figure. Each step of the process is explained below the figure.

> **Note**
>
> The following image and process assumes that RSA is used for server verification and key exchange. The process is slightly different when Diffie–Hellman is used instead of RSA.



1. The client sends a hello message to the server.
2. The server responds with a hello message and sends the server's certificate.
3. The client performs the following actions:

a. Verifies that the SSL/TLS server certificate is signed by a root certificate that the client trusts.
b. Extracts the public key from the server certificate.
c. Generates a pre-master secret and encrypts it with the server's public key.
d. Sends the encrypted pre-master secret to the server.

4. To decrypt the client's pre-master secret, the server sends it to the HSM. The HSM uses the private key in the HSM to decrypt the pre-master secret and then it sends the pre-master secret to the server. Independently, the client and server each use the pre-master secret and some information from the hello messages to calculate a master secret.
5. The handshake process ends. For the rest of the session, all messages sent between the client and the server are encrypted with derivatives of the master secret.

To learn how to configure SSL/TLS offload with AWS CloudHSM, see one of the following topics:

- AWS CloudHSM SSL/TLS offload on Linux using NGINX or Apache with OpenSSL
- AWS CloudHSM SSL/TLS offload on Linux using Tomcat with JSSE
- AWS CloudHSM SSL/TLS offload on Windows using IIS with KSP

AWS CloudHSM SSL/TLS offload on Linux using NGINX or Apache with OpenSSL

This topic provides step-by-step instructions for setting up SSL/TLS offload with AWS CloudHSM on a Linux web server.

Topics

- Overview
- Step 1: Set up the prerequisites
- Step 2: Generate the private key and SSL/TLS certificate
- Step 3: Configure the web server
- Step 4: Enable HTTPS traffic and verify the certificate

Overview

On Linux, the NGINX and Apache HTTP Server web server software integrate with OpenSSL to support HTTPS. The AWS CloudHSM dynamic engine for OpenSSL provides an interface that enables the web server software to use the HSMs in your cluster for cryptographic offloading and key storage. The OpenSSL engine is the bridge that connects the web server to your AWS CloudHSM cluster.

To complete this tutorial, you must first choose whether to use the NGINX or Apache web server software on Linux. Then the tutorial shows you how to do the following:

- Install the web server software on an Amazon EC2 instance.
- Configure the web server software to support HTTPS with a private key stored in your AWS CloudHSM cluster.
- (Optional) Use Amazon EC2 to create a second web server instance and Elastic Load Balancing to create a load balancer. Using a load balancer can increase performance by distributing the load across multiple servers. It can also provide redundancy and higher availability if one or more servers fail.

When you're ready to get started, go to Step 1: Set up the prerequisites.

Step 1: Set up the prerequisites

Different platforms require different prerequisites. Use the prerequisites section below that matches your platform.

Prerequisites for Client SDK 5

To set up web server SSL/TLS offload with Client SDK 5, you need the following:

- An active AWS CloudHSM cluster with at least two hardware security modules (HSM)

> **Note**
> You can use a single HSM cluster, but you must first disable client key durability. For more information, see Manage Client Key Durability Settings and Client SDK 5 Configure Tool.

- An Amazon EC2 instance running a Linux operating system with the following software installed:
  - A web server (either NGINX or Apache)
  - The OpenSSL Dynamic Engine for Client SDK 5
- A crypto user (CU) to own and manage the web server's private key on the HSM.

To set up a Linux web server instance and create a CU on the HSM

1. Install and configure the OpenSSL Dynamic Engine for AWS CloudHSM. For more information about installing OpenSSL Dynamic Engine, see OpenSSL Dynamic Engine for Client SDK 5.
2. On an EC2 Linux instance that has access to your cluster, install either NGINX or Apache web server:

   Amazon Linux

   - NGINX

     ```
     $ sudo yum install nginx
     ```

   - Apache

     ```
     $ sudo yum install httpd24 mod24_ssl
     ```

   Amazon Linux 2

   - For information on how to download the latest version of NGINX on Amazon Linux 2, see the NGINX website.

     The latest version of NGINX available for Amazon Linux 2 uses a version of OpenSSL that is newer than the system version of OpenSSL. After installing NGINX, you need to create a symbolic link from the AWS CloudHSM OpenSSL Dynamic Engine library to the location that this version of OpenSSL expects

     ```
     $ sudo ln -sf /opt/cloudhsm/lib/libcloudhsm_openssl_engine.so /usr/lib64/
     engines-1.1/cloudhsm.so
     ```

   - Apache

     ```
     $ sudo yum install httpd mod_ssl
     ```

   Amazon Linux 2023

   - NGINX

```
$ sudo yum install nginx
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

CentOS 7

- For information on how to download the latest version of NGINX on CentOS 7, see the NGINX website.

  The latest version of NGINX available for CentOS 7 uses a version of OpenSSL that is newer than the system version of OpenSSL. After installing NGINX, you need to create a symbolic link from the AWS CloudHSM OpenSSL Dynamic Engine library to the location that this version of OpenSSL expects

```
$ sudo ln -sf /opt/cloudhsm/lib/libcloudhsm_openssl_engine.so /usr/lib64/
engines-1.1/cloudhsm.so
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

Red Hat 7

- For information on how to download the latest version of NGINX on Red Hat 7, see the NGINX website.

  The latest version of NGINX available for Red Hat 7 uses a version of OpenSSL that is newer than the system version of OpenSSL. After installing NGINX, you need to create a symbolic link from the AWS CloudHSM OpenSSL Dynamic Engine library to the location that this version of OpenSSL expects

```
$ sudo ln -sf /opt/cloudhsm/lib/libcloudhsm_openssl_engine.so /usr/lib64/
engines-1.1/cloudhsm.so
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

CentOS 8

- NGINX

```
$ sudo yum install nginx
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

Red Hat 8

- NGINX

```
$ sudo yum install nginx
```

- Apache

```
$ sudo yum install httpd mod_ssl
```

Ubuntu 18.04

- NGINX

```
$ sudo apt install nginx
```

- Apache

```
$ sudo apt install apache2
```

Ubuntu 20.04

- NGINX

```
$ sudo apt install nginx
```

- Apache

```
$ sudo apt install apache2
```

Ubuntu 22.04

- NGINX

```
$ sudo apt install nginx
```

- Apache

```
$ sudo apt install apache2
```

Ubuntu 24.04

- NGINX

```
$ sudo apt install nginx
```

- Apache

```
$ sudo apt install apache2
```

3. Use CloudHSM CLI to create a crypto user. For more information about managing HSM users, see Managing HSM users with CloudHSM CLI.

> **Tip**
>
> Keep track of the CU user name and password. You will need them later when you generate or import the HTTPS private key and certificate for your web server.

After you complete these steps, go to Step 2: Generate the private key and SSL/TLS certificate.

Notes

- To use Security-Enhanced Linux (SELinux) and web servers, you must allow outbound TCP connections on port 2223, which is the port Client SDK 5 uses to communicate with the HSM.
- To create and activate a cluster and give an EC2 instance access to the cluster, complete the steps in Getting Started with AWS CloudHSM. The getting started offers step-by-step instruction for creating an active cluster with one HSM and an Amazon EC2 client instance. You can use this client instance as your web server.
- To avoid disabling client key durability, add more than one HSM to your cluster. For more information, see Adding an HSM to an AWS CloudHSM cluster.
- To connect to your client instance, you can use SSH or PuTTY. For more information, see Connecting to Your Linux Instance Using SSH or Connecting to Your Linux Instance from Windows Using PuTTY in the Amazon EC2 documentation.

Step 2: Generate the private key and SSL/TLS certificate

To enable HTTPS, your web server application (NGINX or Apache) needs a private key and a corresponding SSL/TLS certificate. To use web server SSL/TLS offload with AWS CloudHSM, you must store the private key in an HSM in your AWS CloudHSM cluster. You will first generate a private key and use the key to create a certificate signing request (CSR). You then export a *fake PEM private key* from the HSM, which is a private key file in PEM format which contains a reference to the private key stored on the HSM (it's not the actual private key). Your web server uses the fake PEM private key file to identify the private key on the HSM during SSL/TLS offload.

Generate a private key and certificate

Generate a private key

This section shows you how to generate a keypair using the CloudHSM CLI. Once you have a key pair generated inside the HSM, you can export it as a fake PEM file and generate the corresponding certificate.

Install and configure the CloudHSM CLI

1. Install and Configure the CloudHSM CLI.
2. Use the following command to start the CloudHSM CLI.

```
$ /opt/cloudhsm/bin/cloudhsm-cli interactive
```

3. Run the following command to log in to the HSM. Replace *<user name>* with the user name of your crypto-user

```
Command: login --username <user name> --role crypto-user
```

Generate a Private Key

Depending on your use case, you can either generate an RSA or an EC key pair. Do one of the following:

- To generate an RSA private key on an HSM

  Use the [key generate-asymmetric-pair rsa](#) command to generate an RSA key pair. This example generates an RSA key pair with a modulus of 2048, a public exponent of 65537, public key label of *tls_rsa_pub*, and private key label of *tls_rsa_private*.

```
aws-cloudhsm > key generate-asymmetric-pair rsa \
--public-exponent 65537 \
--modulus-size-bits 2048 \
--public-label tls_rsa_pub \
--private-label tls_rsa_private
--private-attributes sign=true
{
  "error_code": 0,
  "data": {
    "public_key": {
      "key-reference": "0x0000000000280cc8",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
        "key-type": "rsa",
        "label": "tls_rsa_pub",
        "id": "",
```

```
        "check-value": "0x01fe6e",
        "class": "public-key",
        "encrypt": true,
        "decrypt": false,
        "token": true,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 512,
        "public-exponent": "0x010001",
        "modulus":
 "0xb1d27e857a876f4e9fd5de748a763c539b359f937eb4b4260e30d1435485a732c878cdad9c72538e2215351b1
73a80fdb457aa7b20cd61e486c326e2cfd5e124a7f6a996437437812b542e3caf85928aa866f0298580f7967ee6aa
f6e6296d6c116d5744c6d60d14d3bf3cb978fe6b75ac67b7089bafd50d8687213b31abc7dc1bad422780d29c851d5
133022653225bd129f8491101725e9ea33e1ded83fb57af35f847e532eb30cd7e726f23910d2671c6364092e83469
ac3160f0ca9725d38318b7",
        "modulus-size-bits": 2048
      }
    },
    "private_key": {
      "key-reference": "0x0000000000280cc7",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "full"
      },
      "attributes": {
```

```
        "key-type": "rsa",
        "label": "tls_rsa_private",
        "id": "",
        "check-value": "0x01fe6e",
        "class": "private-key",
        "encrypt": false,
        "decrypt": true,
        "token": true,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 1217,
        "public-exponent": "0x010001",
        "modulus":
 "0xb1d27e857a876f4e9fd5de748a763c539b359f937eb4b4260e30d1435485a732c878cdad9c72538e2215351b1
        "modulus-size-bits": 2048
      }
    }
  }
}
```

- To generate an EC private key on an HSM

  Use the [key generate-asymmetric-pair ec] command to generate an EC key pair. This example generates an EC key pair with the `prime256v1` curve (corresponding to the `NID_X9_62_prime256v1` curve), a public key label of *`tls_ec_pub`*, and a private key label of *`tls_ec_private`*.

```
aws-cloudhsm > key generate-asymmetric-pair ec \
    --curve prime256v1 \
    --public-label tls_ec_pub \
```

```
    --private-label tls_ec_private
    --private-attributes sign=true
{
  "error_code": 0,
  "data": {
    "public_key": {
      "key-reference": "0x000000000012000b",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "session"
      },
      "attributes": {
        "key-type": "ec",
        "label": "tls_ec_pub",
        "id": "",
        "check-value": "0xd7c1a7",
        "class": "public-key",
        "encrypt": false,
        "decrypt": false,
        "token": false,
        "always-sensitive": false,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": false,
        "sign": false,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 57,
        "ec-point":
 "0x047096513df542250a6b228fd9cb67fd0c903abc93488467681974d6f371083fce1d79da8ad1e9ede745fb9f3
```

```
        "curve": "secp224r1"
      }
    },
"private_key": {
      "key-reference": "0x000000000012000c",
      "key-info": {
        "key-owners": [
          {
            "username": "cu1",
            "key-coverage": "full"
          }
        ],
        "shared-users": [],
        "cluster-coverage": "session"
      },
      "attributes": {
        "key-type": "ec",
        "label": "tls_ec_private",
        "id": "",
        "check-value": "0xd7c1a7",
        "class": "private-key",
        "encrypt": false,
        "decrypt": false,
        "token": false,
        "always-sensitive": true,
        "derive": false,
        "destroyable": true,
        "extractable": true,
        "local": true,
        "modifiable": true,
        "never-extractable": false,
        "private": true,
        "sensitive": true,
        "sign": true,
        "trusted": false,
        "unwrap": false,
        "verify": false,
        "wrap": false,
        "wrap-with-trusted": false,
        "key-length-bytes": 122,
        "ec-point":
 "0x047096513df542250a6b228fd9cb67fd0c903abc93488467681974d6f371083fce1d79da8ad1e9ede745fb9f3
        "curve": "secp224r1"
      }
```

```
    }
  }
}
```

Export a fake PEM private key file

Once you have a private key on the HSM, you must export a fake PEM private key file. This file does not contain the actual key data, but it allows the OpenSSL Dynamic Engine to identify the private key on the HSM. You can then you use the private key to create a certificate signing request (CSR) and sign the CSR to create the certificate.

Use the `key generate-file` command to export the private key in fake PEM format and save it to a file. Replace the following values with your own.

- *<private_key_label>* – Label of the private key you generated in the previous step.
- *<web_server_fake_pem.key>* – Name of the file that your fake PEM key will be written to.

```
aws-cloudhsm > key generate-file --encoding reference-pem --
path <web_server_fake_pem.key> --filter attr.label=<private_key_label>
{
  "error_code": 0,
  "data": {
    "message": "Successfully generated key file"
  }
}
```

Exit the CloudHSM CLI

Run the following command to stop the CloudHSM CLI.

```
aws-cloudhsm > quit
```

You should now have a new file on your system, located at the path specified by *<web_server_fake_pem.key>* in the preceding command. This file is the fake PEM private key file.

Generate a self-signed certificate

Once you have generated a fake PEM private key, you can use this file to generate a certificate signing request (CSR) and certificate.

In a production environment, you typically use a certificate authority (CA) to create a certificate from a CSR. A CA is not necessary for a test environment. If you do use a CA, send the CSR file to them and use signed SSL/TLS certificate that they provide you in your web server for HTTPS.

As an alternative to using a CA, you can use the AWS CloudHSM OpenSSL Dynamic Engine to create a self-signed certificate. Self-signed certificates are not trusted by browsers and should not be used in production environments. They can be used in test environments.

> **Warning**
>
> Self-signed certificates should be used in a test environment only. For a production environment, use a more secure method such as a certificate authority to create a certificate.

Install and configure the OpenSSL Dynamic Engine

1. Connect to your client instance.
2. the section called “Install”

Generate a certificate

1. Obtain a copy of your fake PEM file generated in an earlier step.
2. Create a CSR

   Run the following command to use the AWS CloudHSM OpenSSL Dynamic Engine to create a certificate signing request (CSR). Replace *<web_server_fake_pem.key>* with the name of the file that contains your fake PEM private key. Replace *<web_server.csr>* with the name of the file that contains your CSR.

   The `req` command is interactive. Respond to each field. The field information is copied into your SSL/TLS certificate.

```
$ openssl req -engine cloudhsm -new -key <web_server_fake_pem.key> -
out <web_server.csr>
```

3. Create a self-signed certificate

   Run the following command to use the AWS CloudHSM OpenSSL Dynamic Engine to sign your CSR with your private key on your HSM. This creates a self-signed certificate. Replace the following values in the command with your own.

   - *<web_server.csr>* – Name of the file that contains the CSR.
   - *<web_server_fake_pem.key>* – Name of the file that contains the fake PEM private key.
   - *<web_server.crt>* – Name of the file that will contain your web server certificate.

```
$ openssl x509 -engine cloudhsm -req -days 365 -in <web_server.csr> -
signkey <web_server_fake_pem.key> -out <web_server.crt>
```

After you complete these steps, go to Step 3: Configure the web server.

Step 3: Configure the web server

Update your web server software's configuration to use the HTTPS certificate and corresponding fake PEM private key that you created in the previous step. Remember to backup your existing certificates and keys before you start. This will finish setting up your Linux web server software for SSL/TLS offload with AWS CloudHSM.

Complete the steps from one of the following sections.

Topics

- Configure NGINX web server
- Configure Apache web server

Configure NGINX web server

Use this section to configure NGINX on supported platforms.

To update the web server configuration for NGINX

1. Connect to your client instance.
2. Run the following command to create the required directories for the web server certificate and the fake PEM private key.

```
$ sudo mkdir -p /etc/pki/nginx/private
```

3. Run the following command to copy your web server certificate to the required location. Replace *<web_server.crt>* with the name of your web server certificate.

```
$ sudo cp <web_server.crt> /etc/pki/nginx/server.crt
```

4. Run the following command to copy your fake PEM private key to the required location. Replace *<web_server_fake_pem.key>* with the name of the file that contains your fake PEM private key.

```
$ sudo cp <web_server_example_pem.key> /etc/pki/nginx/private/server.key
```

5. Run the following command to change the file ownership so that the user named *nginx* can read them.

```
$ sudo chown nginx /etc/pki/nginx/server.crt /etc/pki/nginx/private/server.key
```

6. Run the following command to back up the `/etc/nginx/nginx.conf` file.

```
$ sudo cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.backup
```

7. Update the NGINX configuration.

> **Note**
>
> Each cluster can support a maximum of 1000 NGINX worker processes across all NGINX web servers.

Amazon Linux

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
server {
    listen       443 ssl http2 default_server;
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

Amazon Linux 2

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
server {
    listen       443 ssl http2 default_server;
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
```

```
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

Amazon Linux 2023

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
server {
    listen       443 ssl http2 default_server;
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;
```

```
    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

CentOS 7

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
server {
    listen       443 ssl http2 default_server;
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2;
```

```
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

CentOS 8

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
server {
    listen       443 ssl http2 default_server;
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
```

```
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

Red Hat 7

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
```

```
server {
    listen       443 ssl http2 default_server;
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

Red Hat 8

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
server {
    listen       443 ssl http2 default_server;
    listen       [::]:443 ssl http2 default_server;
    server_name  _;
    root         /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
    # It is *strongly* recommended to generate unique DH parameters
    # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem 2048
    #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
    ssl_session_cache shared:SSL:1m;
    ssl_session_timeout  10m;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
    ssl_prefer_server_ciphers on;

    # Load configuration files for the default server block.
    include /etc/nginx/default.d/*.conf;

    location / {
    }

    error_page 404 /404.html;
    location = /40x.html {
    }

    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
    }
}
```

Ubuntu 16.04 LTS

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
    env n3fips_password;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
    server {
        listen       443 ssl http2 default_server;
        listen       [::]:443 ssl http2 default_server;
        server_name  _;
        root         /usr/share/nginx/html;

        ssl_certificate "/etc/pki/nginx/server.crt";
        ssl_certificate_key "/etc/pki/nginx/private/server.key";
        # It is *strongly* recommended to generate unique DH parameters
        # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem
 2048
        #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
        ssl_session_cache shared:SSL:1m;
        ssl_session_timeout  10m;
        ssl_protocols TLSv1.2;
        ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-
SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-
RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-
GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
        ssl_prefer_server_ciphers on;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
        }
```

```
        error_page 404 /404.html;
        location = /40x.html {
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
```

Ubuntu 18.04 LTS

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
    env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
    server {
        listen       443 ssl http2 default_server;
        listen       [::]:443 ssl http2 default_server;
        server_name  _;
        root         /usr/share/nginx/html;

        ssl_certificate "/etc/pki/nginx/server.crt";
        ssl_certificate_key "/etc/pki/nginx/private/server.key";
        # It is *strongly* recommended to generate unique DH parameters
        # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem
 2048
        #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
        ssl_session_cache shared:SSL:1m;
        ssl_session_timeout  10m;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-
SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-
RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-
GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
```

```
        ssl_prefer_server_ciphers on;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
        }

        error_page 404 /404.html;
        location = /40x.html {
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
```

Ubuntu 20.04 LTS

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
    env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
    server {
        listen       443 ssl http2 default_server;
        listen       [::]:443 ssl http2 default_server;
        server_name  _;
        root         /usr/share/nginx/html;

        ssl_certificate "/etc/pki/nginx/server.crt";
        ssl_certificate_key "/etc/pki/nginx/private/server.key";
        # It is *strongly* recommended to generate unique DH parameters
        # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem
 2048
        #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
        ssl_session_cache shared:SSL:1m;
```

```
        ssl_session_timeout  10m;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-
SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-
RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-
GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
        ssl_prefer_server_ciphers on;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
        }

        error_page 404 /404.html;
        location = /40x.html {
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
```

Ubuntu 22.04 LTS

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
  env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
    server {
        listen       443 ssl http2 default_server;
        listen       [::]:443 ssl http2 default_server;
        server_name  _;
        root         /usr/share/nginx/html;
```

```
        ssl_certificate "/etc/pki/nginx/server.crt";
        ssl_certificate_key "/etc/pki/nginx/private/server.key";
        # It is *strongly* recommended to generate unique DH parameters
        # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem
 2048
        #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
        ssl_session_cache shared:SSL:1m;
        ssl_session_timeout  10m;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-
SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-
RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-
GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
        ssl_prefer_server_ciphers on;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
        }

        error_page 404 /404.html;
        location = /40x.html {
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
```

Ubuntu 24.04 LTS

Use a text editor to edit the `/etc/nginx/nginx.conf` file. This requires Linux root permissions. At the top of the file, add the following lines:

```
ssl_engine cloudhsm;
  env CLOUDHSM_PIN;
```

Then add the following to the TLS section of the file:

```
# Settings for a TLS enabled server.
    server {
        listen       443 ssl http2 default_server;
        listen       [::]:443 ssl http2 default_server;
        server_name  _;
        root         /usr/share/nginx/html;

        ssl_certificate "/etc/pki/nginx/server.crt";
        ssl_certificate_key "/etc/pki/nginx/private/server.key";
        # It is *strongly* recommended to generate unique DH parameters
        # Generate them with: openssl dhparam -out /etc/pki/nginx/dhparams.pem
 2048
        #ssl_dhparam "/etc/pki/nginx/dhparams.pem";
        ssl_session_cache shared:SSL:1m;
        ssl_session_timeout  10m;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers "ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-
SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-
RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-
GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA";
        ssl_prefer_server_ciphers on;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
        }

        error_page 404 /404.html;
        location = /40x.html {
        }

        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
        }
    }
```

Save the file.

8. Back up the `systemd` configuration file, and then set the `EnvironmentFile` path.

   Amazon Linux

   No action required.

   Amazon Linux 2

   1. Back up the `nginx.service` file.

      ```
      $ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/nginx.service.backup
      ```

   2. Open the `/lib/systemd/system/nginx.service` file in a text editor, and then under the [Service] section, add the following path:

      ```
      EnvironmentFile=/etc/sysconfig/nginx
      ```

   Amazon Linux 2023

   1. Back up the `nginx.service` file.

      ```
      $ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/nginx.service.backup
      ```

   2. Open `/lib/systemd/system/nginx.service` in a text editor. Under the [Service] section, add:

      ```
      EnvironmentFile=/etc/sysconfig/nginx
      ```

   CentOS 7

   No action required.

   CentOS 8

   1. Back up the `nginx.service` file.

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. Open the `/lib/systemd/system/nginx.service` file in a text editor, and then under the [Service] section, add the following path:

```
EnvironmentFile=/etc/sysconfig/nginx
```

Red Hat 7

No action required.

Red Hat 8

1. Back up the `nginx.service` file.

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. Open the `/lib/systemd/system/nginx.service` file in a text editor, and then under the [Service] section, add the following path:

```
EnvironmentFile=/etc/sysconfig/nginx
```

Ubuntu 16.04

1. Back up the `nginx.service` file.

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. Open the `/lib/systemd/system/nginx.service` file in a text editor, and then under the [Service] section, add the following path:

```
EnvironmentFile=/etc/sysconfig/nginx
```

Ubuntu 18.04

1. Back up the `nginx.service` file.

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. Open the `/lib/systemd/system/nginx.service` file in a text editor, and then under the [Service] section, add the following path:

```
EnvironmentFile=/etc/sysconfig/nginx
```

Ubuntu 20.04 LTS

1. Back up the `nginx.service` file.

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. Open the `/lib/systemd/system/nginx.service` file in a text editor, and then under the [Service] section, add the following path:

```
EnvironmentFile=/etc/sysconfig/nginx
```

Ubuntu 22.04 LTS

1. Back up the `nginx.service` file.

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. Open the `/lib/systemd/system/nginx.service` file in a text editor, and then under the [Service] section, add the following path:

```
EnvironmentFile=/etc/sysconfig/nginx
```

Ubuntu 24.04 LTS

1. Back up the `nginx.service` file.

```
$ sudo cp /lib/systemd/system/nginx.service /lib/systemd/system/
nginx.service.backup
```

2. Open the `/lib/systemd/system/nginx.service` file in a text editor, and then under the [Service] section, add the following path:

```
EnvironmentFile=/etc/sysconfig/nginx
```

9. Check if the `/etc/sysconfig/nginx` file exists, and then do one of the following:

   - If the file exists, back up the file by running the following command:

   ```
   $ sudo cp /etc/sysconfig/nginx /etc/sysconfig/nginx.backup
   ```

   - If the file doesn't exist, open a text editor, and then create a file named `nginx` in the `/etc/sysconfig/` folder.

10. Configure the NGINX environment.

> **Note**
>
> Client SDK 5 introduces the `CLOUDHSM_PIN` environment variable for storing the credentials of the CU.

Amazon Linux

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions. Add the Cryptography User (CU) credentials:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Amazon Linux 2

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions. Add the Cryptography User (CU) credentials:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Amazon Linux 2023

As the Linux root user, open `/etc/sysconfig/nginx` file in a text editor. For example,

```
sudo vi /etc/sysconfig/nginx
```

Add the Cryptography User (CU) credentials:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

CentOS 7

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions. Add the Cryptography User (CU) credentials:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

CentOS 8

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions. Add the Cryptography User (CU) credentials:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Red Hat 7

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions. Add the Cryptography User (CU) credentials:

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Red Hat 8

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions. Add the Cryptography User (CU) credentials:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Ubuntu 16.04 LTS

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions. Add the Cryptography User (CU) credentials:

```
n3fips_password=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Ubuntu 18.04 LTS

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions. Add the Cryptography User (CU) credentials:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Ubuntu 20.04 LTS

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions. Add the Cryptography User (CU) credentials:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Ubuntu 22.04 LTS

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions. Add the Cryptography User (CU) credentials:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

Ubuntu 24.04 LTS

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions. Add the Cryptography User (CU) credentials:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Save the file.

11. Start the NGINX web server.

Amazon Linux

Open the `/etc/sysconfig/nginx` file in a text editor. This requires Linux root permissions. Add the Cryptography User (CU) credentials:

```
$ sudo service nginx start
```

Amazon Linux 2

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

Amazon Linux 2023

Stop all NGINX processes

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start NGINX

```
$ sudo systemctl start nginx
```

CentOS 7

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

CentOS 8

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

Red Hat 7

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

Red Hat 8

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

Ubuntu 16.04 LTS

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

Ubuntu 18.04 LTS

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

Ubuntu 20.04 LTS

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

Ubuntu 22.04 LTS

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

Ubuntu 24.04 LTS

Stop any running NGINX process

```
$ sudo systemctl stop nginx
```

Reload the `systemd` configuration to pick up the latest changes

```
$ sudo systemctl daemon-reload
```

Start the NGINX process

```
$ sudo systemctl start nginx
```

12. (Optional) Configure your platform to start NGINX at start-up.

Amazon Linux

```
$ sudo chkconfig nginx on
```

Amazon Linux 2

```
$ sudo systemctl enable nginx
```

Amazon Linux 2023

```
$ sudo systemctl enable nginx
```

CentOS 7

No action required.

CentOS 8

```
$ sudo systemctl enable nginx
```

Red Hat 7

No action required.

Red Hat 8

```
$ sudo systemctl enable nginx
```

Ubuntu 16.04 LTS

```
$ sudo systemctl enable nginx
```

Ubuntu 18.04 LTS

```
$ sudo systemctl enable nginx
```

Ubuntu 20.04 LTS

```
$ sudo systemctl enable nginx
```

Ubuntu 22.04 LTS

```
$ sudo systemctl enable nginx
```

Ubuntu 24.04 LTS

```
$ sudo systemctl enable nginx
```

After you update your web server configuration, go to Step 4: Enable HTTPS traffic and verify the certificate.

Configure Apache web server

Use this section to configure Apache on supported platforms.

To update the web server configuration for Apache

1. Connect to your Amazon EC2 client instance.
2. Define default locations for certificates and private keys for your platform.

   Amazon Linux

   In the `/etc/httpd/conf.d/ssl.conf` file, ensure these values exist:

   ```
   SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
   SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
   ```

   Amazon Linux 2

   In the `/etc/httpd/conf.d/ssl.conf` file, ensure these values exist:

   ```
   SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
   SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
   ```

   Amazon Linux 2023

   Open `/etc/httpd/conf.d/ssl.conf` file. Add these values if they don't already exist:

   ```
   SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
   SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
   ```

   CentOS 7

   In the `/etc/httpd/conf.d/ssl.conf` file, ensure these values exist:

   ```
   SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
   SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
   ```

   CentOS 8

   In the `/etc/httpd/conf.d/ssl.conf` file, ensure these values exist:

   ```
   SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
   SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
   ```

Red Hat 7

In the `/etc/httpd/conf.d/ssl.conf` file, ensure these values exist:

```
SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
```

Red Hat 8

In the `/etc/httpd/conf.d/ssl.conf` file, ensure these values exist:

```
SSLCertificateFile      /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile   /etc/pki/tls/private/localhost.key
```

Ubuntu 16.04 LTS

In the `/etc/apache2/sites-available/default-ssl.conf` file, ensure these values exist:

```
SSLCertificateFile      /etc/ssl/certs/localhost.crt
SSLCertificateKeyFile   /etc/ssl/private/localhost.key
```

Ubuntu 18.04 LTS

In the `/etc/apache2/sites-available/default-ssl.conf` file, ensure these values exist:

```
SSLCertificateFile      /etc/ssl/certs/localhost.crt
SSLCertificateKeyFile   /etc/ssl/private/localhost.key
```

Ubuntu 20.04 LTS

In the `/etc/apache2/sites-available/default-ssl.conf` file, ensure these values exist:

```
SSLCertificateFile      /etc/ssl/certs/localhost.crt
SSLCertificateKeyFile   /etc/ssl/private/localhost.key
```

Ubuntu 22.04 LTS

In the `/etc/apache2/sites-available/default-ssl.conf` file, ensure these values exist:

```
SSLCertificateFile      /etc/ssl/certs/localhost.crt
SSLCertificateKeyFile   /etc/ssl/private/localhost.key
```

Ubuntu 24.04 LTS

In the `/etc/apache2/sites-available/default-ssl.conf` file, ensure these values exist:

```
SSLCertificateFile      /etc/ssl/certs/localhost.crt
SSLCertificateKeyFile   /etc/ssl/private/localhost.key
```

3. Copy your web server certificate to the required location for your platform.

Amazon Linux

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Amazon Linux 2

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Amazon Linux 2023

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

CentOS 7

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

CentOS 8

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Red Hat 7

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Red Hat 8

```
$ sudo cp <web_server.crt> /etc/pki/tls/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Ubuntu 16.04 LTS

```
$ sudo cp <web_server.crt> /etc/ssl/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Ubuntu 18.04 LTS

```
$ sudo cp <web_server.crt> /etc/ssl/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Ubuntu 20.04 LTS

```
$ sudo cp <web_server.crt> /etc/ssl/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Ubuntu 22.04 LTS

```
$ sudo cp <web_server.crt> /etc/ssl/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

Ubuntu 24.04 LTS

```
$ sudo cp <web_server.crt> /etc/ssl/certs/localhost.crt
```

Replace *<web_server.crt>* with the name of your web server certificate.

4. Copy your fake PEM private key to the required location for your platform.

Amazon Linux

```
$ sudo cp <web_server_example_pem.key> /etc/pki/tls/private/localhost.key
```

Replace *<web_server_example_pem.key>* with the name of the file that contains your fake PEM private key.

Amazon Linux 2

```
$ sudo cp <web_server_example_pem.key> /etc/pki/tls/private/localhost.key
```

Replace *<web_server_example_pem.key>* with the name of the file that contains your fake PEM private key.

Amazon Linux 2023

```
$ sudo cp <web_server_example_pem.key> /etc/pki/tls/private/localhost.key
```

Replace *<web_server_example_pem.key>* with the name of the file that contains your fake PEM private key.

CentOS 7

```
$ sudo cp <web_server_example_pem.key> /etc/pki/tls/private/localhost.key
```

Replace *<web_server_example_pem.key>* with the name of the file that contains your fake PEM private key.

CentOS 8

```
$ sudo cp <web_server_example_pem.key> /etc/pki/tls/private/localhost.key
```

Replace *<web_server_example_pem.key>* with the name of the file that contains your fake PEM private key.

Red Hat 7

```
$ sudo cp <web_server_example_pem.key> /etc/pki/tls/private/localhost.key
```

Replace *<web_server_example_pem.key>* with the name of the file that contains your fake PEM private key.

Red Hat 8

```
$ sudo cp <web_server_example_pem.key> /etc/pki/tls/private/localhost.key
```

Replace *<web_server_example_pem.key>* with the name of the file that contains your fake PEM private key.

Ubuntu 16.04 LTS

```
$ sudo cp <web_server_example_pem.key> /etc/ssl/private/localhost.key
```

Replace *<web_server_example_pem.key>* with the name of the file that contains your fake PEM private key.

Ubuntu 18.04 LTS

```
$ sudo cp <web_server_example_pem.key> /etc/ssl/private/localhost.key
```

Replace *<web_server_example_pem.key>* with the name of the file that contains your fake PEM private key.

Ubuntu 20.04 LTS

```
$ sudo cp <web_server_example_pem.key> /etc/ssl/private/localhost.key
```

Replace *<web_server_example_pem.key>* with the name of the file that contains your fake PEM private key.

Ubuntu 22.04 LTS

```
$ sudo cp <web_server_example_pem.key> /etc/ssl/private/localhost.key
```

Replace *<web_server_example_pem.key>* with the name of the file that contains your fake PEM private key.

Ubuntu 24.04 LTS

```
$ sudo cp <web_server_example_pem.key> /etc/ssl/private/localhost.key
```

Replace *<web_server_example_pem.key>* with the name of the file that contains your fake PEM private key.

5. Change ownership of these files if required by your platform.

   Amazon Linux

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

   Provides read permission to the user named *apache*.

   Amazon Linux 2

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

   Provides read permission to the user named *apache*.

   Amazon Linux 2023

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

   Provides read permission to the user named *apache*.

   CentOS 7

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

Provides read permission to the user named *apache*.

CentOS 8

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

Provides read permission to the user named *apache*.

Red Hat 7

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

Provides read permission to the user named *apache*.

Red Hat 8

```
$ sudo chown apache /etc/pki/tls/certs/localhost.crt /etc/pki/tls/private/
localhost.key
```

Provides read permission to the user named *apache*.

Ubuntu 16.04 LTS

No action required.

Ubuntu 18.04 LTS

No action required.

Ubuntu 20.04 LTS

No action required.

Ubuntu 22.04 LTS

No action required.

Ubuntu 24.04 LTS

No action required.

6. Configure Apache directives for your platform.

Amazon Linux

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

Save the file.

Amazon Linux 2

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
```

```
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

Save the file.

Amazon Linux 2023

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

The Apache configuration file defines server behavior. Edit this file with root permissions.

Update or add the following directives:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

Save the file.

CentOS 7

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
```

```
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

Save the file.

CentOS 8

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLProtocol TLSv1.2 TLSv1.3
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-
RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
SSLProxyCipherSuite HIGH:!aNULL
```

Save the file.

Red Hat 7

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

Save the file.

Red Hat 8

Locate the SSL file for this platform:

```
/etc/httpd/conf.d/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLProtocol TLSv1.2 TLSv1.3
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
SSLProxyCipherSuite HIGH:!aNULL
```

Save the file.

Ubuntu 16.04 LTS

Locate the SSL file for this platform:

```
/etc/apache2/mods-available/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

Save the file.

Enable the SSL module and default SSL site configuration:

```
$ sudo a2enmod ssl
$ sudo a2ensite default-ssl
```

Ubuntu 18.04 LTS

Locate the SSL file for this platform:

```
/etc/apache2/mods-available/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
```

```
SSLProtocol TLSv1.2 TLSv1.3
```

Save the file.

Enable the SSL module and default SSL site configuration:

```
$ sudo a2enmod ssl
$ sudo a2ensite default-ssl
```

Ubuntu 20.04 LTS

Locate the SSL file for this platform:

```
/etc/apache2/mods-available/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
SSLProtocol TLSv1.2 TLSv1.3
```

Save the file.

Enable the SSL module and default SSL site configuration:

```
$ sudo a2enmod ssl
$ sudo a2ensite default-ssl
```

Ubuntu 22.04 LTS

Locate the SSL file for this platform:

```
/etc/apache2/mods-available/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
SSLProtocol TLSv1.2 TLSv1.3
```

Save the file.

Enable the SSL module and default SSL site configuration:

```
$ sudo a2enmod ssl
$ sudo a2ensite default-ssl
```

Ubuntu 24.04 LTS

Locate the SSL file for this platform:

```
/etc/apache2/mods-available/ssl.conf
```

This file contains Apache directives which define how your server should run. Directives appear on the left, followed by a value. Use a text editor to edit this file. This requires Linux root permissions.

Update or enter the following directives with these values:

```
SSLCryptoDevice cloudhsm
SSLCipherSuite ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-
```

```
RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES256-
SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES256-GCM-
SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-
AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA
SSLProtocol TLSv1.2 TLSv1.3
```

Save the file.

Enable the SSL module and default SSL site configuration:

```
$ sudo a2enmod ssl
$ sudo a2ensite default-ssl
```

7. Configure an environment-values file for your platform.

Amazon Linux

No action required. Environment values go in `/etc/sysconfig/httpd`

Amazon Linux 2

Open the httpd service file:

```
/lib/systemd/system/httpd.service
```

Under the `[Service]` section, add the following:

```
EnvironmentFile=/etc/sysconfig/httpd
```

Amazon Linux 2023

Open `/lib/systemd/system/httpd.service`

Under the [Service] section, add:

```
EnvironmentFile=/etc/sysconfig/httpd
```

CentOS 7

Open the httpd service file:

```
/lib/systemd/system/httpd.service
```

Under the `[Service]` section, add the following:

```
EnvironmentFile=/etc/sysconfig/httpd
```

CentOS 8

Open the httpd service file:

```
/lib/systemd/system/httpd.service
```

Under the `[Service]` section, add the following:

```
EnvironmentFile=/etc/sysconfig/httpd
```

Red Hat 7

Open the httpd service file:

```
/lib/systemd/system/httpd.service
```

Under the `[Service]` section, add the following:

```
EnvironmentFile=/etc/sysconfig/httpd
```

Red Hat 8

Open the httpd service file:

```
/lib/systemd/system/httpd.service
```

Under the `[Service]` section, add the following:

```
EnvironmentFile=/etc/sysconfig/httpd
```

Ubuntu 16.04 LTS

No action required. Environment values go in `/etc/sysconfig/httpd`

Ubuntu 18.04 LTS

No action required. Environment values go in `/etc/sysconfig/httpd`

Ubuntu 20.04 LTS

No action required. Environment values go in `/etc/sysconfig/httpd`

Ubuntu 22.04 LTS

No action required. Environment values go in `/etc/sysconfig/httpd`

Ubuntu 24.04 LTS

No action required. Environment values go in `/etc/sysconfig/httpd`

8. In the file that stores environment variables for your platform, set an environment variable that contains the credentials of the cryptographic user (CU):

Amazon Linux

Use a text editor to edit the `/etc/sysconfig/httpd`.

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Replace *`<CU user name>`* and *`<password>`* with the CU credentials.

Amazon Linux 2

Use a text editor to edit the `/etc/sysconfig/httpd`.

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Replace *`<CU user name>`* and *`<password>`* with the CU credentials.

Amazon Linux 2023

Open `/etc/sysconfig/httpd`, add:

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

CentOS 7

Use a text editor to edit the `/etc/sysconfig/httpd`.

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Replace *<CU user name>* and *<password>* with the CU credentials.

CentOS 8

Use a text editor to edit the `/etc/sysconfig/httpd`.

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Red Hat 7

Use a text editor to edit the `/etc/sysconfig/httpd`.

```
ssl_engine cloudhsm;
env CLOUDHSM_PIN;
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Red Hat 8

Use a text editor to edit the `/etc/sysconfig/httpd`.

```
CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

> **Note**
>
> Client SDK 5 introduces the `CLOUDHSM_PIN` environment variable for storing the credentials of the CU.

Ubuntu 16.04 LTS

Use a text editor to edit the `/etc/apache2/envvars`.

```
export n3fips_password=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

Ubuntu 18.04 LTS

Use a text editor to edit the `/etc/apache2/envvars`.

```
export CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

> **Note**
>
> Client SDK 5 introduces the `CLOUDHSM_PIN` environment variable for storing the credentials of the CU. In Client SDK 3 you stored the CU credentials in the `n3fips_password` environment variable. Client SDK 5 supports both environment variables, but we recommend using `CLOUDHSM_PIN`.

Ubuntu 20.04 LTS

Use a text editor to edit the `/etc/apache2/envvars`.

```
export CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

> **Note**
>
> Client SDK 5 introduces the `CLOUDHSM_PIN` environment variable for storing the credentials of the CU. In Client SDK 3 you stored the CU credentials in the `n3fips_password` environment variable. Client SDK 5 supports both environment variables, but we recommend using `CLOUDHSM_PIN`.

Ubuntu 22.04 LTS

Use a text editor to edit the `/etc/apache2/envvars`.

```
export CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

> **Note**
>
> Client SDK 5 introduces the `CLOUDHSM_PIN` environment variable for storing the credentials of the CU. In Client SDK 3 you stored the CU credentials in the `n3fips_password` environment variable. Client SDK 5 supports both environment variables, but we recommend using `CLOUDHSM_PIN`.

Ubuntu 24.04 LTS

Use a text editor to edit the `/etc/apache2/envvars`.

```
export CLOUDHSM_PIN=<CU user name>:<password>
```

Replace *<CU user name>* and *<password>* with the CU credentials.

> **Note**
>
> Client SDK 5 introduces the `CLOUDHSM_PIN` environment variable for storing the credentials of the CU. In Client SDK 3 you stored the CU credentials in the

n3fips_password environment variable. Client SDK 5 supports both environment variables, but we recommend using CLOUDHSM_PIN.

9. Start the Apache web server.

Amazon Linux

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Amazon Linux 2

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Amazon Linux 2023

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

CentOS 7

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

CentOS 8

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Red Hat 7

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Red Hat 8

```
$ sudo systemctl daemon-reload
$ sudo service httpd start
```

Ubuntu 16.04 LTS

```
$ sudo service apache2 start
```

Ubuntu 18.04 LTS

```
$ sudo service apache2 start
```

Ubuntu 20.04 LTS

```
$ sudo service apache2 start
```

Ubuntu 22.04 LTS

```
$ sudo service apache2 start
```

Ubuntu 24.04 LTS

```
$ sudo service apache2 start
```

10. (Optional) Configure your platform to start Apache at start-up.

Amazon Linux

```
$ sudo chkconfig httpd on
```

Amazon Linux 2

```
$ sudo chkconfig httpd on
```

Amazon Linux 2023

```
$ sudo chkconfig httpd on
```

CentOS 7

```
$ sudo chkconfig httpd on
```

CentOS 8

```
$ systemctl enable httpd
```

Red Hat 7

```
$ sudo chkconfig httpd on
```

Red Hat 8

```
$ systemctl enable httpd
```

Ubuntu 16.04 LTS

```
$ sudo systemctl enable apache2
```

Ubuntu 18.04 LTS

```
$ sudo systemctl enable apache2
```

Ubuntu 20.04 LTS

```
$ sudo systemctl enable apache2
```

Ubuntu 22.04 LTS

```
$ sudo systemctl enable apache2
```

Ubuntu 24.04 LTS

```
$ sudo systemctl enable apache2
```

After you update your web server configuration, go to Step 4: Enable HTTPS traffic and verify the certificate.

Step 4: Enable HTTPS traffic and verify the certificate

After you configure your web server for SSL/TLS offload with AWS CloudHSM, add your web server instance to a security group that allows inbound HTTPS traffic. This allows clients, such as web browsers, to establish an HTTPS connection with your web server. Then make an HTTPS connection to your web server and verify that it's using the certificate that you configured for SSL/TLS offload with AWS CloudHSM.

Topics

Enable inbound HTTPS connections

To connect to your web server from a client (such as a web browser), create a security group that allows inbound HTTPS connections. Specifically, it should allow inbound TCP connections on port 443. Assign this security group to your web server.

To create a security group for HTTPS and assign it to your web server

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Security groups** in the navigation pane.
3. Choose **Create security group**.
4. For **Create Security Group**, do the following:

    a. For **Security group name**, type a name for the security group that you are creating.

    b. (Optional) Type a description of the security group that you are creating.

    c. For **VPC**, choose the VPC that contains your web server Amazon EC2 instance.

    d. Select **Add Rule**.

    e. For **Type**, select **HTTPS** from the drop-down window.

    f. For **Source**, enter a source location.

    g. Choose **Create security group**.
5. In the navigation pane, choose **Instances**.
6. Select the check box next to your web server instance.

7. Select the **Actions** drop-down menu at the top of the page. Select **Security** and then **Change Security Groups**.
8. For **Associated security groups**, select the search box and choose the security group that you created for HTTPS. Then choose **Add Security Groups**.
9. Select **Save**.

Verify that HTTPS uses the certificate that you configured

After you add the web server to a security group, you can verify that SSL/TLS offload is using your self-signed certificate. You can do this with a web browser or with a tool such as OpenSSL s_client.

To verify SSL/TLS offload with a web browser

1. Use a web browser to connect to your web server using the public DNS name or IP address of the server. Ensure that the URL in the address bar begins with https://. For example, **`https://ec2-52-14-212-67.us-east-2.compute.amazonaws.com/`**.

   > **Tip**
   >
   > You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the *Amazon Route 53 Developer Guide* or in the documentation for your DNS service.

2. Use your web browser to view the web server certificate. For more information, see the following:

   - For Mozilla Firefox, see View a Certificate on the Mozilla Support website.
   - For Google Chrome, see Understand Security Issues on the Google Tools for Web Developers website.

   Other web browsers might have similar features that you can use to view the web server certificate.

3. Ensure that the SSL/TLS certificate is the one that you configured your web server to use.

To verify SSL/TLS offload with OpenSSL s_client

1. Run the following OpenSSL command to connect to your web server using HTTPS. Replace *<server name>* with the public DNS name or IP address of your web server.

   ```
   openssl s_client -connect <server name>:443
   ```

   > **Tip**
   >
   > You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the *Amazon Route 53 Developer Guide* or in the documentation for your DNS service.

2. Ensure that the SSL/TLS certificate is the one that you configured your web server to use.

You now have a website that is secured with HTTPS. The private key for the web server is stored in an HSM in your AWS CloudHSM cluster.

To add a load balancer, see Add a load balancer with Elastic Load Balancing for AWS CloudHSM(optional).

AWS CloudHSM SSL/TLS offload on Linux using Tomcat with JSSE

This topic provides step-by-step instructions for setting up SSL/TLS offload using Java Secure Socket Extension (JSSE) with the AWS CloudHSM JCE SDK.

Topics

Overview

In AWS CloudHSM, Tomcat web servers work on Linux to support HTTPS. The AWS CloudHSM JCE SDK provides an interface that can be used with JSSE (Java Secure Socket Extension) to enable use of HSMs for such web servers. AWS CloudHSM JCE is the bridge that connects JSSE to your AWS CloudHSM cluster. JSSE is a Java API for Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols.

Step 1: Set up the prerequisites

Follow these prerequisites to use a Tomcat web server with AWS CloudHSM for SSL/TLS offload on Linux. These prerequisites must be met to set up web server SSL/TLS offload with Client SDK 5 and a Tomcat web server.

> **Note**
>
> Different platforms require different prerequisites. Always follow the correct installation steps for your platform.

Prerequisites

- An Amazon EC2 instance running a Linux operating system with A tomcat web server installed.
- A crypto user (CU) to own and manage the web server's private key on the HSM.
- An active AWS CloudHSM cluster with at least two hardware security modules (HSMs) that have JCE for Client SDK 5 installed and configured.

  > **Note**
  >
  > You can use a single HSM cluster, but you must first disable client key durability. For more information, see Manage Client Key Durability Settings and Client SDK 5 Configure Tool.

How to meet the prerequisites

1. Install and configure the JCE for AWS CloudHSM on an active AWS CloudHSM cluster with at least two hardware security modules (HSMs). For more information about installation, see JCE for Client SDK 5.

2. On an EC2 Linux instance that has access to your AWS CloudHSM cluster, follow the Apache Tomcat instructions to download and install the Tomcat web server.

3. Use CloudHSM CLI to create a crypto user (CU). For more information about managing HSM users, see Managing HSM users with CloudHSM CLI.

> **Tip**
>
> Keep track of the CU user name and password. You will need them later when you generate or import the HTTPS private key and certificate for your web server.

4. To setup JCE with Java Keytool, follow the instructions in Use Client SDK 5 to integrate AWS CloudHSM with Java Keytool and Jarsigner.

After you complete these steps, go to Step 2: Generate or import a private key and SSL/TLS certificate.

Notes

- To use Security-Enhanced Linux (SELinux) and web servers, you must allow outbound TCP connections on port 2223, which is the port Client SDK 5 uses to communicate with the HSM.
- To create and activate a cluster and give an EC2 instance access to the cluster, complete the steps in Getting Started with AWS CloudHSM. This section offers step-by-step instructions for creating an active cluster with one HSM and an Amazon EC2 client instance. You can use this client instance as your web server.
- To avoid disabling client key durability, add more than one HSM to your cluster. For more information, see Adding an HSM to an AWS CloudHSM cluster.
- To connect to your client instance, you can use SSH or PuTTY. For more information, see Connecting to Your Linux Instance Using SSH or Connecting to Your Linux Instance from Windows Using PuTTY in the Amazon EC2 documentation.

Step 2: Generate or import a private key and SSL/TLS certificate

To enable HTTPS, your Tomcat web server application needs a private key and a corresponding SSL/TLS certificate. To use web server SSL/TLS offload with AWS CloudHSM, you must store the private key in an HSM in your AWS CloudHSM cluster.

> **Note**
>
> If you don't yet have a private key and a corresponding certificate, generate a private key in an HSM. You use the private key to create a certificate signing request (CSR), which you use to create the SSL/TLS certificate.

You create a local AWS CloudHSM KeyStore file that contains a reference to your private key on the HSM and the associated certificate. Your web server uses the AWS CloudHSM KeyStore file to identify the private key on the HSM during SSL/TLS offload.

Topics

- Generate a private key
- Generate a self-signed certificate

Generate a private key

This section shows you how to generate a key pair using the KeyTool from JDK. Once you have a key pair generated inside the HSM, you can export it as a KeyStore file, and generate the corresponding certificate.

Depending on your use case, you can either generate an RSA or an EC key pair. The following steps show how to generate an RSA key pair.

Use the `genkeypair` command in KeyTool to generate an RSA key pair

1. After replacing the *<VARIABLES>* below with your specific data, use the following command to generate a keystore file named `jsse_keystore.keystore`, which will have a reference of your private key on the HSM.

```
$ keytool -genkeypair -alias <UNIQUE ALIAS FOR KEYS> -keyalg <KEY ALGORITHM> -
keysize <KEY SIZE> -sigalg <SIGN ALGORITHM> \
        -keystore <PATH>/<JSSE KEYSTORE NAME>.keystore -storetype CLOUDHSM \
        -dname CERT_DOMAIN_NAME \
        -J-classpath '-J'$JAVA_LIB'/*:/opt/cloudhsm/java/*:./*' \
        -provider "com.amazonaws.cloudhsm.jce.provider.CloudHsmProvider" \
        -providerpath "$CLOUDHSM_JCE_LOCATION" \
        -keypass <KEY PASSWORD> -storepass <KEYSTORE PASSWORD>
```

- ***<PATH>***: The path that you want to generate your keystore file.
- ***<UNIQUE ALIAS FOR KEYS>***: This is used to uniquely identify your key on the HSM. This alias will be set as the LABEL attribute for the key.
- ***<KEY PASSWORD>***: We store reference to your key in the local keystore file, and this password protects that local reference.
- ***<KEYSTORE PASSWORD>***: This is the password for your local keystore file.
- ***<JSSE KEYSTORE NAME>***: Name of the Keystore file.
- ***<CERT DOMAIN NAME>***: X.500 Distinguished name.
- ***<KEY ALGORITHM>***: Key algorithm to generate key pair (For example, RSA and EC).
- ***<KEY SIZE>***: Key size to generate key pair (for example, 2048, 3072, and 4096).
- ***<SIGN ALGORITHM>***: Key size to generate key pair (for example, SHA1withRSA, SHA224withRSA, SHA256withRSA, SHA384withRSA, and SHA512withRSA).

2. To confirm the command was successful, enter the following command and verify you have successfully generated an RSA key pair.

```
$ ls <PATH>/<JSSE KEYSTORE NAME>.keystore
```

Generate a self-signed certificate

Once you have generated a private key along with the keystore file, you can use this file to generate a certificate signing request (CSR) and certificate.

In a production environment, you typically use a certificate authority (CA) to create a certificate from a CSR. A CA is not necessary for a test environment. If you do use a CA, send the CSR file to them and use signed SSL/TLS certificate that they provide you in your web server for HTTPS.

As an alternative to using a CA, you can use the KeyTool to create a self-signed certificate. Self-signed certificates are not trusted by browsers and should not be used in production environments. They can be used in test environments.

> **Warning**
>
> Self-signed certificates should be used in a test environment only. For a production environment, use a more secure method, such as a certificate authority to create a certificate.

Generate a certificate

1. Obtain a copy of your keystore file generated in an earlier step.
2. Run the following command to use the KeyTool to create a certificate signing request (CSR).

```
$ keytool -certreq -keyalg RSA -alias unique_alias_for_key -file certreq.csr \
        -keystore <JSSE KEYSTORE NAME>.keystore -storetype CLOUDHSM \
        -J-classpath '-J$JAVA_LIB/*:/opt/cloudhsm/java/*:./*' \
        -keypass <KEY PASSWORD> -storepass <KEYSTORE PASSWORD>
```

> **Note**
>
> The output file of the certificate signing request is `certreq.csr`.

Sign a certificate

- After replacing the *<VARIABLES>* below with your specific data, run the following command to sign your CSR with your private key on your HSM. This creates a self-signed certificate.

```
$ keytool -gencert -infile certreq.csr -outfile certificate.crt \
    -alias <UNIQUE ALIAS FOR KEYS> -keypass <KEY_PASSWORD> -
storepass <KEYSTORE_PASSWORD> -sigalg SIG_ALG \
    -storetype CLOUDHSM -J-classpath '-J$JAVA_LIB/*:/opt/cloudhsm/java/*:./*' \
    -keystore jsse_keystore.keystore
```

> **Note**
>
> `certificate.crt` is the signed certificate that uses the alias's private key.

Import a certificate in Keystore

- After replacing the *<VARIABLES>* below with your specific data, run the following command to import a signed certificate as a trusted certificate. This step will store the certificate in the keystore entry identified by alias.

```
$ keytool -import -alias <UNIQUE ALIAS FOR KEYS> -keystore jsse_keystore.keystore \
    -file certificate.crt -storetype CLOUDHSM \
    -v -J-classpath '-J$JAVA_LIB/*:/opt/cloudhsm/java/*:./*' \
    -keypass <KEY PASSWORD> -storepass <KEYSTORE_PASSWORD>
```

Convert a certificate to a PEM

- Run following command to convert the signed certificate file (`.crt`) to a PEM. The PEM file will be used to send the request from the http client.

```
$ openssl x509 -inform der -in certificate.crt -out certificate.pem
```

After you complete these steps, go to Step 3: Configure the web server.

Step 3: Configure the Tomcat web server

Update your web server software's configuration to use the HTTPS certificate and corresponding PEM file that you created in the previous step. Remember to backup your existing certificates and keys before you start. This will finish setting up your Linux web server software for SSL/TLS offload with AWS CloudHSM. For more information, refer to the Apache Tomcat 9 Configuration Reference.

Stop the server

- After replacing the *<VARIABLES>* below with your specific data, run following command to stop Tomcat Server before updating configuration

```
$ /<TOMCAT DIRECTORY>/bin/shutdown.sh
```

  - ***<TOMCAT DIRECTORY>***: Your Tomcat installation directory.

Update the Tomcat classpath

1. Connect to your client instance.
2. Locate the Tomcat installation folder.
3. After replacing the *<VARIABLES>* below with your specific data, use the following command to add Java library and AWS CloudHSM Java path in Tomcat **classpath**, located in Tomcat/bin/catalina.sh file.

```
$ sed -i 's@CLASSPATH="$CLASSPATH""$CATALINA_HOME"\/bin\/
bootstrap.jar@CLASSPATH="$CLASSPATH""$CATALINA_HOME"\/bin\/bootstrap.jar:'"
        <JAVA LIBRARY>"'\/*:\/opt\/cloudhsm\/java\/*:.\/*@' <TOMCAT PATH> /bin/
catalina.sh
```

   - ***<JAVA LIBRARY>***: Java JRE Library location.
   - ***<TOMCAT PATH>***: Tomcat installation folder.

Add an HTTPS connector in the server configuration.

1. Go to the Tomcat installation folder.
2. After replacing the *<VARIABLES>* below with your specific data, use the following command to add an HTTPS connector to use certificates generated in prerequisites:

```
$ sed -i '/<Connector port="8080"/i <Connector port=\"443\" maxThreads=\"200\"
 scheme=\"https\" secure=\"true\" SSLEnabled=\"true\" keystoreType=\"CLOUDHSM\"
 keystoreFile=\"
        <CUSTOM DIRECTORY>/<JSSE KEYSTORE NAME>.keystore\" keystorePass=\"<KEYSTORE
 PASSWORD>\" keyPass=\"<KEY PASSWORD>
        \" keyAlias=\"<UNIQUE ALIAS FOR KEYS>" clientAuth=\"false\" sslProtocol=
\"TLS\"/>' <TOMCAT PATH>/conf/server.xml
```

   - ***<CUSTOM DIRECTORY>***: Directory where keystore file is located.
   - ***<JSSE KEYSTORE NAME>***: Name of the Keystore file.
   - ***<KEYSTORE PASSWORD>***: This is the password for your local keystore file.
   - ***<KEY PASSWORD>***: We store reference to your key in the local keystore file, and this password protects that local reference.
   - ***<UNIQUE ALIAS FOR KEYS>***: This is used to uniquely identify your key on the HSM. This alias will be set as the LABEL attribute for the key.

- ***<TOMCAT PATH>***: The path to your Tomcat folder.

Start Server

- After replacing the *<VARIABLES>* below with your specific data, use the following command to start Tomcat Server:

```
$ /<TOMCAT DIRECTORY>/bin/startup.sh
```

> **Note**
>
> ***<TOMCAT DIRECTORY>*** is the name of your Tomcat installation directory.

After you update your web server configuration, go to Step 4: Enable HTTPS traffic and verify the certificate.

Step 4: Enable HTTPS traffic and verify the certificate

After you configure your web server for SSL/TLS offload with AWS CloudHSM, add your web server instance to a security group that allows inbound HTTPS traffic. This allows clients, such as web browsers, to establish an HTTPS connection with your web server. Then make an HTTPS connection to your web server and verify that it's using the certificate that you configured for SSL/TLS offload with AWS CloudHSM.

Topics

Enable inbound HTTPS connections

To connect to your web server from a client (such as a web browser), create a security group that allows inbound HTTPS connections. Specifically, it should allow inbound TCP connections on port 443. Assign this security group to your web server.

To create a security group for HTTPS and assign it to your web server

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.

2. Choose **Security groups** in the navigation pane.
3. Choose **Create security group**.
4. For **Create Security Group**, do the following:

    a. For **Security group name**, type a name for the security group that you are creating.

    b. (Optional) Type a description of the security group that you are creating.

    c. For **VPC**, choose the VPC that contains your web server Amazon EC2 instance.

    d. Select **Add Rule**.

    e. For **Type**, select **HTTPS** from the drop-down window.

    f. For **Source**, enter a source location.

    g. Choose **Create security group**.
5. In the navigation pane, choose **Instances**.
6. Select the check box next to your web server instance.
7. Select the **Actions** drop-down menu at the top of the page. Select **Security** and then **Change Security Groups**.
8. For **Associated security groups**, select the search box and choose the security group that you created for HTTPS. Then choose **Add Security Groups**.
9. Select **Save**.

Verify that HTTPS uses the certificate that you configured

After you add the web server to a security group, you can verify that SSL/TLS offload is using your self-signed certificate. You can do this with a web browser or with a tool such as OpenSSL s_client.

To verify SSL/TLS offload with a web browser

1. Use a web browser to connect to your web server using the public DNS name or IP address of the server. Ensure that the URL in the address bar begins with https://. For example, **`https://ec2-52-14-212-67.us-east-2.compute.amazonaws.com/`**.

    > **Tip**
    >
    > You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, https://www.example.com/) to your web server. For more

information, see Routing Traffic to an Amazon EC2 Instance in the *Amazon Route 53 Developer Guide* or in the documentation for your DNS service.

2. Use your web browser to view the web server certificate. For more information, see the following:

   - For Mozilla Firefox, see View a Certificate on the Mozilla Support website.
   - For Google Chrome, see Understand Security Issues on the Google Tools for Web Developers website.

   Other web browsers might have similar features that you can use to view the web server certificate.

3. Ensure that the SSL/TLS certificate is the one that you configured your web server to use.

To verify SSL/TLS offload with OpenSSL s_client

1. Run the following OpenSSL command to connect to your web server using HTTPS. Replace *<server name>* with the public DNS name or IP address of your web server.

```
openssl s_client -connect <server name>:443
```

> **Tip**
>
> You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the *Amazon Route 53 Developer Guide* or in the documentation for your DNS service.

2. Ensure that the SSL/TLS certificate is the one that you configured your web server to use.

You now have a website that is secured with HTTPS. The private key for the web server is stored in an HSM in your AWS CloudHSM cluster.

To add a load balancer, see Add a load balancer with Elastic Load Balancing for AWS CloudHSM(optional).

AWS CloudHSM SSL/TLS offload on Windows using IIS with KSP

This tutorial provides step-by-step instructions for setting up SSL/TLS offload with AWS CloudHSM on a Windows web server.

Topics

Overview

On Windows, the Internet Information Services (IIS) for Windows Server web server application natively supports HTTPS. The AWS CloudHSM key storage provider (KSP) for Microsoft's Cryptography API: Next Generation (CNG) provides the interface that allows IIS to use the HSMs in your cluster for cryptographic offloading and key storage. The AWS CloudHSM KSP is the bridge that connects IIS to your AWS CloudHSM cluster.

This tutorial shows you how to do the following:

- Install the web server software on an Amazon EC2 instance.
- Configure the web server software to support HTTPS with a private key stored in your AWS CloudHSM cluster.
- (Optional) Use Amazon EC2 to create a second web server instance and Elastic Load Balancing to create a load balancer. Using a load balancer can increase performance by distributing the load across multiple servers. It can also provide redundancy and higher availability if one or more servers fail.

When you're ready to get started, go to Step 1: Set up the prerequisites.

Step 1: Set up the prerequisites

Different platforms require different prerequisites. Use the prerequisites section below that matches your platform.

Topics

Prerequisites for Client SDK 5

To set up web server SSL/TLS offload with AWS CloudHSM, you need the following:

- An active AWS CloudHSM cluster with at least one HSM.
- An Amazon EC2 instance running a Windows operating system with the following software installed:
  - The AWS CloudHSM client software for Windows.
  - Internet Information Services (IIS) for Windows Server.
- A crypto user (CU) to own and manage the web server's private key on the HSM.

> **Note**
>
> This tutorial uses Microsoft Windows Server 2019. Microsoft Windows Server 2016 and 2022 is also supported.

To set up a Windows Server instance and create a CU on the HSM

1. Complete the steps in Getting started. When you launch the Amazon EC2 client, choose a Windows Server 2019 AMI. When you complete these steps, you have an active cluster with at least one HSM. You also have an Amazon EC2 client instance running Windows Server with the AWS CloudHSM client software for Windows installed.
2. (Optional) Add more HSMs to your cluster. For more information, see Adding an HSM to an AWS CloudHSM cluster.
3. Connect to your Windows server. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.
4. Use CloudHSM CLI to create a crypto user (CU). Keep track of the CU user name and password. You will need them to complete the next step.

> **Note**
>
> For information on creating a user, see Managing HSM users with CloudHSM CLI.

5. Set the login credentials for the HSM, using the CU user name and password that you created in the previous step.
6. In step 5, if you used Windows Credentials Manager to set HSM credentials, download `psexec.exe` from SysInternals to run the following command as *NT Authority\SYSTEM*:

```
psexec.exe -s "C:\Program Files\Amazon\CloudHsm\tools\set_cloudhsm_credentials.exe"
 --username <USERNAME> --password <PASSWORD>
```

   Replace *<USERNAME>* and *<PASSWORD>* with the HSM credentials.

To install IIS on your Windows Server

1. If you haven't already done so, connect to your Windows server. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.
2. On your Windows server, start **Server Manager**.
3. In the **Server Manager** dashboard, choose **Add roles and features**.
4. Read the **Before you begin** information, and then choose **Next**.
5. For **Installation Type**, choose **Role-based or feature-based installation**. Then choose **Next**.
6. For **Server Selection**, choose **Select a server from the server pool**. Then choose **Next**.
7. For **Server Roles**, do the following:
   a. Select **Web Server (IIS)**.
   b. For **Add features that are required for Web Server (IIS)**, choose **Add Features**.
   c. Choose **Next** to finish selecting server roles.
8. For **Features**, accept the defaults. Then choose **Next**.
9. Read the **Web Server Role (IIS)** information. Then choose **Next**.
10. For **Select role services**, accept the defaults or change the settings as preferred. Then choose **Next**.
11. For **Confirmation**, read the confirmation information. Then choose **Install**.

12. After the installation is complete, choose **Close**.

After you complete these steps, go to Step 2: Create a certificate signing request (CSR) and certificate.

Prerequisites for Client SDK 3

To set up web server SSL/TLS offload with AWS CloudHSM, you need the following:

- An active AWS CloudHSM cluster with at least one HSM.
- An Amazon EC2 instance running a Windows operating system with the following software installed:
  - The AWS CloudHSM client software for Windows.
  - Internet Information Services (IIS) for Windows Server.
- A crypto user (CU) to own and manage the web server's private key on the HSM.

> **Note**
>
> This tutorial uses Microsoft Windows Server 2016. Microsoft Windows Server 2012 is also supported, but Microsoft Windows Server 2012 R2 is not.

To set up a Windows Server instance and create a CU on the HSM

1. Complete the steps in Getting started. When you launch the Amazon EC2 client, choose a Windows Server 2016 or Windows Server 2012 AMI. When you complete these steps, you have an active cluster with at least one HSM. You also have an Amazon EC2 client instance running Windows Server with the AWS CloudHSM client software for Windows installed.
2. (Optional) Add more HSMs to your cluster. For more information, see Adding an HSM to an AWS CloudHSM cluster.
3. Connect to your Windows server. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.
4. Use CloudHSM CLI to create a crypto user (CU). Keep track of the CU user name and password. You will need them to complete the next step.

> **Note**
>
> For information on creating a user, see Managing HSM users with CloudHSM CLI.

5. Set the login credentials for the HSM, using the CU user name and password that you created in the previous step.
6. In step 5, if you used Windows Credentials Manager to set HSM credentials, download `psexec.exe` from SysInternals to run the following command as *NT Authority\SYSTEM*:

   ```
   psexec.exe -s "C:\Program Files\Amazon\CloudHsm\tools\set_cloudhsm_credentials.exe"
    --username <USERNAME> --password <PASSWORD>
   ```

   Replace *<USERNAME>* and *<PASSWORD>* with the HSM credentials.

To install IIS on your Windows Server

1. If you haven't already done so, connect to your Windows server. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.
2. On your Windows server, start **Server Manager**.
3. In the **Server Manager** dashboard, choose **Add roles and features**.
4. Read the **Before you begin** information, and then choose **Next**.
5. For **Installation Type**, choose **Role-based or feature-based installation**. Then choose **Next**.
6. For **Server Selection**, choose **Select a server from the server pool**. Then choose **Next**.
7. For **Server Roles**, do the following:

   a. Select **Web Server (IIS)**.
   b. For **Add features that are required for Web Server (IIS)**, choose **Add Features**.
   c. Choose **Next** to finish selecting server roles.
8. For **Features**, accept the defaults. Then choose **Next**.
9. Read the **Web Server Role (IIS)** information. Then choose **Next**.
10. For **Select role services**, accept the defaults or change the settings as preferred. Then choose **Next**.
11. For **Confirmation**, read the confirmation information. Then choose **Install**.

12. After the installation is complete, choose **Close**.

After you complete these steps, go to Step 2: Create a certificate signing request (CSR) and certificate.

Step 2: Create a certificate signing request (CSR) and certificate

To enable HTTPS, your web server needs an SSL/TLS certificate and a corresponding private key. To use SSL/TLS offload with AWS CloudHSM, you store the private key in the HSM in your AWS CloudHSM cluster. To do this, you use the AWS CloudHSM key storage provider (KSP) for Microsoft's Cryptography API: Next Generation (CNG) to create a certificate signing request (CSR). Then you give the CSR to a certificate authority (CA), which signs the CSR to produce a certificate.

Topics

Create a CSR with Client SDK 5

1. On your Windows Server, use a text editor to create a certificate request file named `IISCertRequest.inf`. The following shows the contents of an example `IISCertRequest.inf` file. For more information about the sections, keys, and values that you can specify in the file, see Microsoft's documentation. Do not change the `ProviderName` value.

   ```
   [Version]
   Signature = "$Windows NT$"
   [NewRequest]
   Subject = "CN=example.com,C=US,ST=Washington,L=Seattle,O=ExampleOrg,OU=WebServer"
   HashAlgorithm = SHA256
   KeyAlgorithm = RSA
   KeyLength = 2048
   ProviderName = "CloudHSM Key Storage Provider"
   KeyUsage = 0xf0
   MachineKeySet = True
   [EnhancedKeyUsageExtension]
   OID=1.3.6.1.5.5.7.3.1
   ```

2. Use the [Windows **certreq** command] to create a CSR from the `IISCertRequest.inf` file that you created in the previous step. The following example saves the CSR to a file named `IISCertRequest.csr`. If you used a different file name for your certificate request file, replace *`IISCertRequest.inf`* with the appropriate file name. You can optionally replace *`IISCertRequest.csr`* with a different file name for your CSR file.

```
C:\>certreq -new IISCertRequest.inf IISCertRequest.csr

CertReq: Request Created
```

   The `IISCertRequest.csr` file contains your CSR. You need this CSR to get a signed certificate.

Create a CSR with Client SDK 3

1. If you haven't already done so, connect to your Windows server. For more information, see [Connect to Your Instance] in the *Amazon EC2 User Guide*.
2. Use the following command to start the AWS CloudHSM client daemon.

   Amazon Linux

```
$ sudo start cloudhsm-client
```

   Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

   CentOS 7

```
$ sudo service cloudhsm-client start
```

   CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- For Windows client 1.1.2+:

```
C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
```

- For Windows clients 1.1.1 and older:

```
C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe
 C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

3. On your Windows Server, use a text editor to create a certificate request file named `IISCertRequest.inf`. The following shows the contents of an example `IISCertRequest.inf` file. For more information about the sections, keys, and values that you can specify in the file, see Microsoft's documentation. Do not change the `ProviderName` value.

```
[Version]
Signature = "$Windows NT$"
[NewRequest]
Subject = "CN=example.com,C=US,ST=Washington,L=Seattle,O=ExampleOrg,OU=WebServer"
HashAlgorithm = SHA256
KeyAlgorithm = RSA
```

```
KeyLength = 2048
ProviderName = "Cavium Key Storage Provider"
KeyUsage = 0xf0
MachineKeySet = True
[EnhancedKeyUsageExtension]
OID=1.3.6.1.5.5.7.3.1
```

4. Use the Windows **certreq** command to create a CSR from the `IISCertRequest.inf` file that you created in the previous step. The following example saves the CSR to a file named `IISCertRequest.csr`. If you used a different file name for your certificate request file, replace *`IISCertRequest.inf`* with the appropriate file name. You can optionally replace *`IISCertRequest.csr`* with a different file name for your CSR file.

   ```
   C:\>certreq -new IISCertRequest.inf IISCertRequest.csr
           SDK Version: 2.03

   CertReq: Request Created
   ```

   The `IISCertRequest.csr` file contains your CSR. You need this CSR to get a signed certificate.

Get a signed certificate and import it

In a production environment, you typically use a certificate authority (CA) to create a certificate from a CSR. A CA is not necessary for a test environment. If you do use a CA, send the CSR file (`IISCertRequest.csr`) to it and use the CA to create a signed SSL/TLS certificate.

As an alternative to using a CA, you can use a tool like OpenSSL to create a self-signed certificate.

> **Warning**
>
> Self-signed certificates are not trusted by browsers and should not be used in production environments. They can be used in test environments.

The following procedures show how to create a self-signed certificate and use it to sign your web server's CSR.

To create a self-signed certificate

1. Use the following OpenSSL command to create a private key. You can optionally replace *SelfSignedCA.key* with the file name to contain your private key.

```
openssl genrsa -aes256 -out SelfSignedCA.key 2048
Generating RSA private key, 2048 bit long modulus
......................................................................+++
.........................................+++
e is 65537 (0x10001)
Enter pass phrase for SelfSignedCA.key:
Verifying - Enter pass phrase for SelfSignedCA.key:
```

2. Use the following OpenSSL command to create a self-signed certificate using the private key that you created in the previous step. This is an interactive command. Read the on-screen instructions and follow the prompts. Replace *SelfSignedCA.key* with the name of the file that contains your private key (if different). You can optionally replace *SelfSignedCA.crt* with the file name to contain your self-signed certificate.

```
openssl req -new -x509 -days 365 -key SelfSignedCA.key -out SelfSignedCA.crt
Enter pass phrase for SelfSignedCA.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:
Email Address []:
```

To use your self-signed certificate to sign your web server's CSR

- Use the following OpenSSL command to use your private key and self-signed certificate to sign the CSR. Replace the following with the names of the files that contain the corresponding data (if different).
  - *IISCertRequest.csr* – The name of the file that contains your web server's CSR
  - *SelfSignedCA.crt* – The name of the file that contains your self-signed certificate
  - *SelfSignedCA.key* – The name of the file that contains your private key
  - *IISCert.crt* – The name of the file to contain your web server's signed certificate

```
openssl x509 -req -days 365 -in IISCertRequest.csr \
                            -CA SelfSignedCA.crt \
                            -CAkey SelfSignedCA.key \
                            -CAcreateserial \
                            -out IISCert.crt
Signature ok
subject=/ST=IIS-HSM/L=IIS-HSM/OU=IIS-HSM/O=IIS-HSM/CN=IIS-HSM/C=IIS-HSM
Getting CA Private Key
Enter pass phrase for SelfSignedCA.key:
```

After you complete the previous step, you have a signed certificate for your web server (`IISCert.crt`) and a self-signed certificate (`SelfSignedCA.crt`). When you have these files, go to Step 3: Configure the web server.

Step 3: Configure the web server

Update your IIS website's configuration to use the HTTPS certificate that you created at the end of the previous step. This will finish setting up your Windows web server software (IIS) for SSL/TLS offload with AWS CloudHSM.

If you used a self-signed certificate to sign your CSR, you must first import the self-signed certificate into the Windows Trusted Root Certification Authorities.

To import your self-signed certificate into the Windows Trusted Root Certification Authorities

1. If you haven't already done so, connect to your Windows server. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.

2. Copy your self-signed certificate to your Windows server.
3. On your Windows Server, open the **Control Panel**.
4. For **Search Control Panel**, type **`certificates`**. Then choose **Manage computer certificates**.
5. In the **Certificates - Local Computer** window, double-click **Trusted Root Certification Authorities**.
6. Right-click on **Certificates** and then choose **All Tasks**, **Import**.
7. In the **Certificate Import Wizard**, choose **Next**.
8. Choose **Browse**, then find and select your self-signed certificate. If you created your self-signed certificate by following the instructions in the previous step of this tutorial, your self-signed certificate is named `SelfSignedCA.crt`. Choose **Open**.
9. Choose **Next**.
10. For **Certificate Store**, choose **Place all certificates in the following store**. Then ensure that **Trusted Root Certification Authorities** is selected for **Certificate store**.
11. Choose **Next** and then choose **Finish**.

To update the IIS website's configuration

1. If you haven't already done so, connect to your Windows server. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.
2. Start the AWS CloudHSM client daemon.
3. Copy your web server's signed certificate—the one that you created at the end of this tutorial's previous step—to your Windows server.
4. On your Windows Server, use the Windows **certreq** command to accept the signed certificate, as in the following example. Replace *`IISCert.crt`* with the name of the file that contains your web server's signed certificate.

```
C:\>certreq -accept IISCert.crt
        SDK Version: 2.03
```

5. On your Windows server, start **Server Manager**.
6. In the **Server Manager** dashboard, in the top right corner, choose **Tools**, **Internet Information Services (IIS) Manager**.
7. In the **Internet Information Services (IIS) Manager** window, double-click your server name. Then double-click **Sites**. Select your website.

8. Select **SSL Settings**. Then, on the right side of the window, choose **Bindings**.
9. In the **Site Bindings** window, choose **Add**.
10. For **Type**, choose **https**. For **SSL certificate**, choose the HTTPS certificate that you created at the end of this tutorial's previous step.

> **Note**
>
> If you encounter an error during this certificate binding, restart your server and retry this step.

11. Choose **OK**.

After you update your website's configuration, go to Step 4: Enable HTTPS traffic and verify the certificate.

Step 4: Enable HTTPS traffic and verify the certificate

After you configure your web server for SSL/TLS offload with AWS CloudHSM, add your web server instance to a security group that allows inbound HTTPS traffic. This allows clients, such as web browsers, to establish an HTTPS connection with your web server. Then make an HTTPS connection to your web server and verify that it's using the certificate that you configured for SSL/TLS offload with AWS CloudHSM.

Topics

Enable inbound HTTPS connections

To connect to your web server from a client (such as a web browser), create a security group that allows inbound HTTPS connections. Specifically, it should allow inbound TCP connections on port 443. Assign this security group to your web server.

To create a security group for HTTPS and assign it to your web server

1. Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
2. Choose **Security groups** in the navigation pane.

3. Choose **Create security group**.
4. For **Create Security Group**, do the following:

   a. For **Security group name**, type a name for the security group that you are creating.
   b. (Optional) Type a description of the security group that you are creating.
   c. For **VPC**, choose the VPC that contains your web server Amazon EC2 instance.
   d. Select **Add Rule**.
   e. For **Type**, select **HTTPS** from the drop-down window.
   f. For **Source**, enter a source location.
   g. Choose **Create security group**.
5. In the navigation pane, choose **Instances**.
6. Select the check box next to your web server instance.
7. Select the **Actions** drop-down menu at the top of the page. Select **Security** and then **Change Security Groups**.
8. For **Associated security groups**, select the search box and choose the security group that you created for HTTPS. Then choose **Add Security Groups**.
9. Select **Save**.

Verify that HTTPS uses the certificate that you configured

After you add the web server to a security group, you can verify that SSL/TLS offload is using your self-signed certificate. You can do this with a web browser or with a tool such as OpenSSL s_client.

To verify SSL/TLS offload with a web browser

1. Use a web browser to connect to your web server using the public DNS name or IP address of the server. Ensure that the URL in the address bar begins with https://. For example, **`https://ec2-52-14-212-67.us-east-2.compute.amazonaws.com/`**.

   > **Tip**
   >
   > You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the *Amazon Route 53 Developer Guide* or in the documentation for your DNS service.

2. Use your web browser to view the web server certificate. For more information, see the following:

   - For Mozilla Firefox, see View a Certificate on the Mozilla Support website.
   - For Google Chrome, see Understand Security Issues on the Google Tools for Web Developers website.

   Other web browsers might have similar features that you can use to view the web server certificate.

3. Ensure that the SSL/TLS certificate is the one that you configured your web server to use.

To verify SSL/TLS offload with OpenSSL s_client

1. Run the following OpenSSL command to connect to your web server using HTTPS. Replace *<server name>* with the public DNS name or IP address of your web server.

```
openssl s_client -connect <server name>:443
```

> **Tip**
>
> You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the *Amazon Route 53 Developer Guide* or in the documentation for your DNS service.

2. Ensure that the SSL/TLS certificate is the one that you configured your web server to use.

You now have a website that is secured with HTTPS. The private key for the web server is stored in an HSM in your AWS CloudHSM cluster.

To add a load balancer, see Add a load balancer with Elastic Load Balancing for AWS CloudHSM(optional).

Add a load balancer with Elastic Load Balancing for AWS CloudHSM(optional)

After you set up SSL/TLS offload with one web server, you can create more web servers and an Elastic Load Balancing load balancer that routes HTTPS traffic to the web servers. A load balancer can reduce the load on your individual web servers by balancing traffic across two or more servers. It can also increase the availability of your website because the load balancer monitors the health of your web servers and only routes traffic to healthy servers. If a web server fails, the load balancer automatically stops routing traffic to it.

Topics

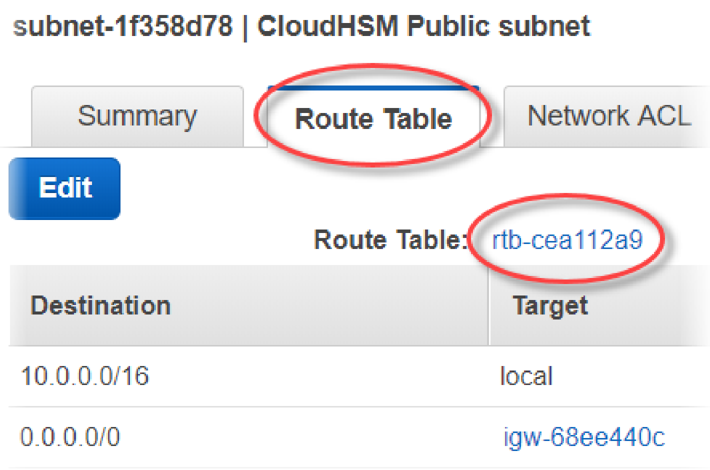
Step 1. Create a subnet for a second web server

Before you can create another web server, you need to create a new subnet in the same VPC that contains your existing web server and AWS CloudHSM cluster.

To create a new subnet

1. Open the **Subnets** section of the Amazon VPC console.
2. Choose **Create Subnet**.
3. In the **Create Subnet** dialog box, do the following:
    a. For **Name tag**, type a name for your subnet.
    b. For **VPC**, choose the AWS CloudHSM VPC that contains your existing web server and AWS CloudHSM cluster.
    c. For **Availability Zone**, choose an Availability Zone that is different from the one that contains your existing web server.
    d. For **IPv4 CIDR block**, type the CIDR block to use for the subnet. For example, type **`10.0.10.0/24`**.
    e. Choose **Yes, Create**.
4. Select the check box next to the public subnet that contains your existing web server. This is different from the public subnet that you created in the previous step.

5. In the content pane, choose the **Route Table** tab. Then choose the link for the route table.



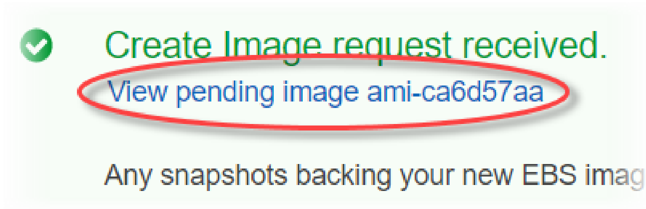
6. Select the check box next to the route table.
7. Choose the **Subnet Associations** tab. Then choose **Edit**.
8. Select the check box next to the public subnet that you created earlier in this procedure. Then choose **Save**.

Step 2. Create the second web server

Complete the following steps to create a second web server with the same configuration as your existing web server.

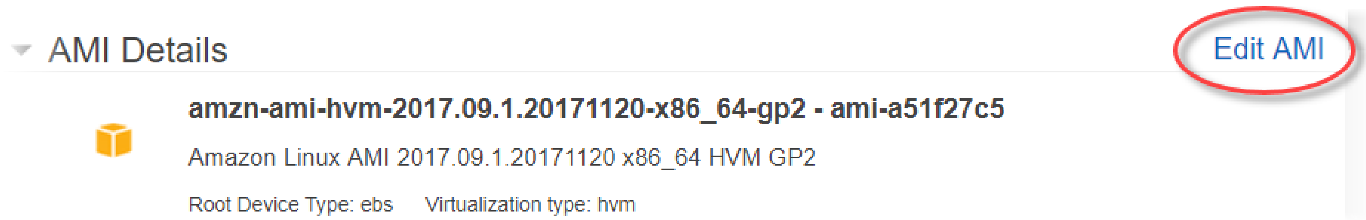
To create a second web server

1. Open the Instances section of the Amazon EC2 console at.
2. Select the check box next to your existing web server instance.
3. Choose **Actions**, **Image**, and then **Create Image**.
4. In the **Create Image** dialog box, do the following:
   a. For **Image name**, type a name for the image.
   b. For **Image description**, type a description for the image.
   c. Choose **Create Image**. This action reboots your existing web server.
   d. Choose the **View pending image ami-*<AMI ID>*** link.



In the **Status** column, note your image status. When your image status is **available** (this might take several minutes), go to the next step.

5. In the navigation pane, choose **Instances**.
6. Select the check box next to your existing web server.
7. Choose **Actions** and choose **Launch More Like This**.
8. Choose **Edit AMI**.



9. In the left navigation pane, choose **My AMIs**. Then clear the text in the search box.
10. Next to your web server image, choose **Select**.
11. Choose **Yes, I want to continue with this AMI (*<image name>* - ami-*<AMI ID>*)**.
12. Choose **Next**.
13. Select an instance type, and then choose **Next: Configure Instance Details**.
14. For **Step 3: Configure Instance Details**, do the following:

    a. For **Network**, choose the VPC that contains your existing web server.

    b. For **Subnet**, choose the public subnet that you created for the second web server.

    c. For **Auto-assign Public IP**, choose**Enable**.

    d. Change the remaining instance details as preferred. Then choose **Next: Add Storage**.
15. Change the storage settings as preferred. Then choose **Next: Add Tags**.
16. Add or edit tags as preferred. Then choose **Next: Configure Security Group**.
17. For **Step 6: Configure Security Group**, do the following:

    a. For **Assign a security group**, choose **Select an existing security group**.

b. Select the check box next to the security group named **cloudhsm-*<cluster ID>*-sg**. AWS CloudHSM created this security group on your behalf when you [created the cluster](). You must choose this security group to allow the web server instance to connect to the HSMs in the cluster.

c. Select the check box next to the security group that allows inbound HTTPS traffic. You [created this security group previously]().

d. (Optional) Select the check box next to a security group that allows inbound SSH (for Linux) or RDP (for Windows) traffic from your network. That is, the security group must allow inbound TCP traffic on port 22 (for SSH on Linux) or port 3389 (for RDP on Windows). Otherwise, you cannot connect to your client instance. If you don't have a security group like this, you must create one and then assign it to your client instance later.

Choose **Review and Launch**.

18. Review your instance details, and then choose **Launch**.

19. Choose whether to launch your instance with an existing key pair, create a new key pair, or launch your instance without a key pair.

- To use an existing key pair, do the following:
  1. Choose **Choose an existing key pair**.
  2. For **Select a key pair**, choose the key pair to use.
  3. Select the check box next to **I acknowledge that I have access to the selected private key file (*<private key file name>*.pem), and that without this file, I won't be able to log into my instance.**
- To create a new key pair, do the following:
  1. Choose **Create a new key pair**.
  2. For **Key pair name**, type a key pair name.
  3. Choose **Download Key Pair** and save the private key file in a secure and accessible location.

> **Warning**
>
> You cannot download the private key file again after this point. If you do not download the private key file now, you will be unable to access the client instance.

- To launch your instance without a key pair, do the following:
  1. Choose **Proceed without a key pair**.
  2. Select the check box next to **I acknowledge that I will not be able to connect to this instance unless I already know the password built into this AMI.**

Choose **Launch Instances**.

Step 3. Create the load balancer

Complete the following steps to create an Elastic Load Balancing load balancer that routes HTTPS traffic to your web servers.

To create a load balancer

1. Open the [Load balancers] section of the Amazon EC2 console.
2. Choose **Create Load Balancer**.
3. In the **Network Load Balancer** section, choose **Create**.
4. For **Step 1: Configure Load Balancer**, do the following:
   a. For **Name**, type a name for the load balancer that you are creating.
   b. In the **Listeners** section, for **Load Balancer Port**, change the value to **443**.
   c. In the **Availability Zones** section, for **VPC**, choose the VPC that contains your web servers.
   d. In the **Availability Zones** section, choose the subnets that contain your web servers.
   e. Choose **Next: Configure Routing**.
5. For **Step 2: Configure Routing**, do the following:
   a. For **Name**, type a name for the target group that you are creating.
   b. For **Port**, change the value to **443**.
   c. Choose **Next: Register Targets**.
6. For **Step 3: Register Targets**, do the following:
   a. In the **Instances** section, select the check boxes next to your web server instances. Then choose **Add to registered**.
   b. Choose **Next: Review**.
7. Review your load balancer details, then choose **Create**.

8. When the load balancer has been successfully created, choose **Close**.

After you complete the preceding steps, the Amazon EC2 console shows your Elastic Load Balancing load balancer.

When your load balancer's state is active, you can verify that the load balancer is working. That is, you can verify that it's sending HTTPS traffic to your web servers with SSL/TLS offload with AWS CloudHSM. You can do this with a web browser or a tool such as OpenSSL s_client.

To verify that your load balancer is working with a web browser

1. In the Amazon EC2 console, find the **DNS name** for the load balancer that you just created. Then select the DNS name and copy it.
2. Use a web browser such as Mozilla Firefox or Google Chrome to connect to your load balancer using the load balancer's DNS name. Ensure that the URL in the address bar begins with https://.

> **Tip**
>
> You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the *Amazon Route 53 Developer Guide* or in the documentation for your DNS service.

3. Use your web browser to view the web server certificate. For more information, see the following:

   - For Mozilla Firefox, see View a Certificate on the Mozilla Support website.
   - For Google Chrome, see Understand Security Issues on the Google Tools for Web Developers website.

   Other web browsers might have similar features that you can use to view the web server certificate.
4. Ensure that the certificate is the one that you configured the web server to use.

To verify that your load balancer is working with OpenSSL s_client

1. Use the following OpenSSL command to connect to your load balancer using HTTPS. Replace *<DNS name>* with the DNS name of your load balancer.

   ```
   openssl s_client -connect <DNS name>:443
   ```

   > **Tip**
   >
   > You can use a DNS service such as Amazon Route 53 to route your website's domain name (for example, https://www.example.com/) to your web server. For more information, see Routing Traffic to an Amazon EC2 Instance in the *Amazon Route 53 Developer Guide* or in the documentation for your DNS service.

2. Ensure that the certificate is the one that you configured the web server to use.

You now have a website that is secured with HTTPS, with the web server's private key stored in an HSM in your AWS CloudHSM cluster. Your website has two web servers and a load balancer to help improve efficiency and availability.

Configure Windows Server as a certificate authority (CA) with AWS CloudHSM

AWS CloudHSM offers support to configure Windows Server as a certificate authority (CA) through Client SDK 3 and Client SDK 5. The steps to use these tools will vary depending on the version of the client SDK in which you currently have downloaded. The following sections provide information to each SDK.

Topics

- Configure Windows Server as a certificate authority (CA) with Client SDK 5
- Configure Windows Server as a certificate authority (CA) with Client SDK 3

Configure Windows Server as a certificate authority (CA) with Client SDK 5

In a public key infrastructure (PKI), a certificate authority (CA) is a trusted entity that issues digital certificates. These digital certificates bind a public key to an identity (a person or organization) by means of public key cryptography and digital signatures. To operate a CA, you must maintain trust by protecting the private key that signs the certificates issued by your CA. You can store the private key in the HSM in your AWS CloudHSM cluster, and use the HSM to perform the cryptographic signing operations.

In this tutorial, you use Windows Server and AWS CloudHSM to configure a CA. You install the AWS CloudHSM client software for Windows on your Windows server, then add the Active Directory Certificate Services (AD CS) role to your Windows Server. When you configure this role, you use an AWS CloudHSM key storage provider (KSP) to create and store the CA's private key on your AWS CloudHSM cluster. The KSP is the bridge that connects your Windows server to your AWS CloudHSM cluster. In the last step, you sign a certificate signing request (CSR) with your Windows Server CA.

For more information, see the following topics:

Topics

Step 1: Set up the prerequisites

To set up Windows Server as a certificate authority (CA) with AWS CloudHSM, you need the following:

- An active AWS CloudHSM cluster with at least one HSM.
- An Amazon EC2 instance running a Windows Server operating system with the AWS CloudHSM client software for Windows installed. This tutorial uses Microsoft Windows Server 2016.
- A cryptographic user (CU) to own and manage the CA's private key on the HSM.

To set up the prerequisites for a Windows Server CA with AWS CloudHSM

1. Complete the steps in Getting started. When you launch the Amazon EC2 client, choose a Windows Server AMI. This tutorial uses Microsoft Windows Server 2016. When you complete these steps, you have an active cluster with at least one HSM. You also have an Amazon EC2 client instance running Windows Server with the AWS CloudHSM client software for Windows installed.
2. (Optional) Add more HSMs to your cluster. For more information, see Adding an HSM to an AWS CloudHSM cluster.
3. Connect to your client instance. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.
4. Create a crypto user (CU) using Managing HSM users with CloudHSM CLI or Managing HSM users with CloudHSM Management Utility (CMU). Keep track of the CU user name and password. You will need them to complete the next step.
5. Set the login credentials for the HSM, using the CU user name and password that you created in the previous step.
6. In step 5, if you used Windows Credentials Manager to set HSM credentials, download `psexec.exe` from SysInternals to run the following command as *NT Authority\SYSTEM*:

   ```
   psexec.exe -s "C:\Program Files\Amazon\CloudHsm\tools\set_cloudhsm_credentials.exe"
    --username <USERNAME> --password <PASSWORD>
   ```

   Replace *<USERNAME>* and *<PASSWORD>* with the HSM credentials.

To create a Windows Server CA with AWS CloudHSM, go to Create Windows Server CA.

Step 2: Create a Windows Server CA with AWS CloudHSM

To create a Windows Server CA, you add the Active Directory Certificate Services (AD CS) role to your Windows Server. When you add this role, you use an AWS CloudHSM key storage provider (KSP) to create and store the CA's private key on your AWS CloudHSM cluster.

Note

When you create your Windows Server CA, you can choose to create a root CA or a subordinate CA. You typically make this decision based on the design of your public key infrastructure and the security policies of your organization. This tutorial explains how to create a root CA for simplicity.

To add the AD CS role to your Windows Server and create the CA's private key

1. If you haven't already done so, connect to your Windows server. For more information, see [Connect to Your Instance](#) in the *Amazon EC2 User Guide*.
2. On your Windows server, start **Server Manager**.
3. In the **Server Manager** dashboard, choose **Add roles and features**.
4. Read the **Before you begin** information, and then choose **Next**.
5. For **Installation Type**, choose **Role-based or feature-based installation**. Then choose **Next**.
6. For **Server Selection**, choose **Select a server from the server pool**. Then choose **Next**.
7. For **Server Roles**, do the following:

    a. Select **Active Directory Certificate Services**.

    b. For **Add features that are required for Active Directory Certificate Services**, choose **Add Features**.

    c. Choose **Next** to finish selecting server roles.
8. For **Features**, accept the defaults, and then choose **Next**.
9. For **AD CS**, do the following:

    a. Choose **Next**.

    b. Select **Certification Authority**, and then choose **Next**.
10. For **Confirmation**, read the confirmation information, and then choose **Install**. Do not close the window.
11. Choose the highlighted **Configure Active Directory Certificate Services on the destination server** link.
12. For **Credentials**, verify or change the credentials displayed. Then choose **Next**.
13. For **Role Services**, select **Certification Authority**. Then choose **Next**.
14. For **Setup Type**, select **Standalone CA**. Then choose **Next**.
15. For **CA Type**, select **Root CA**. Then choose **Next**.

> **Note**
>
> You can choose to create a root CA or a subordinate CA based on the design of your public key infrastructure and the security policies of your organization. This tutorial explains how to create a root CA for simplicity.

16. For **Private Key**, select **Create a new private key**. Then choose **Next**.
17. For **Cryptography**, do the following:

    a. For **Select a cryptographic provider**, choose one of the **CloudHSM Key Storage Provider** options from the menu. These are the AWS CloudHSM key storage providers. For example, you can choose **RSA#CloudHSM Key Storage Provider**.

    b. For **Key length**, choose one of the key length options.

    c. For **Select the hash algorithm for signing certificates issued by this CA**, choose one of the hash algorithm options.

    Choose **Next**.
18. For **CA Name**, do the following:

    a. (Optional) Edit the common name.

    b. (Optional) Type a distinguished name suffix.

    Choose **Next**.
19. For **Validity Period**, specify a time period in years, months, weeks, or days. Then choose **Next**.
20. For **Certificate Database**, you can accept the default values, or optionally change the location for the database and the database log. Then choose **Next**.
21. For **Confirmation**, review the information about your CA; Then choose **Configure**.
22. Choose **Close**, and then choose **Close** again.

You now have a Windows Server CA with AWS CloudHSM. To learn how to sign a certificate signing request (CSR) with your CA, go to Sign a CSR.

Step 3: Sign a certificate signing request (CSR) with your Windows Server CA with AWS CloudHSM

You can use your Windows Server CA with AWS CloudHSM to sign a certificate signing request (CSR). To complete these steps, you need a valid CSR. You can create a CSR in several ways, including the following:

- Using OpenSSL
- Using the Windows Server Internet Information Services (IIS) Manager
- Using the certificates snap-in in the Microsoft Management Console
- Using the **certreq** command line utility on Windows

The steps for creating a CSR are outside the scope of this tutorial. When you have a CSR, you can sign it with your Windows Server CA.

To sign a CSR with your Windows Server CA

1. If you haven't already done so, connect to your Windows server. For more information, see [Connect to Your Instance](#) in the *Amazon EC2 User Guide*.
2. On your Windows server, start **Server Manager**.
3. In the **Server Manager** dashboard, in the top right corner, choose **Tools**, **Certification Authority**.
4. In the **Certification Authority** window, choose your computer name.
5. From the **Action** menu, choose **All Tasks**, **Submit new request**.
6. Select your CSR file, and then choose **Open**.
7. In the **Certification Authority** window, double-click **Pending Requests**.
8. Select the pending request. Then, from the **Action** menu, choose **All Tasks**, **Issue**.
9. In the **Certification Authority** window, double-click **Issued Requests** to view the signed certificate.
10. (Optional) To export the signed certificate to a file, complete the following steps:

    a. In the **Certification Authority** window, double-click the certificate.

    b. Choose the **Details** tab, and then choose **Copy to File**.

    c. Follow the instructions in the **Certificate Export Wizard**.

You now have a Windows Server CA with AWS CloudHSM, and a valid certificate signed by the Windows Server CA.

Configure Windows Server as a certificate authority (CA) with Client SDK 3

In a public key infrastructure (PKI), a certificate authority (CA) is a trusted entity that issues digital certificates. These digital certificates bind a public key to an identity (a person or organization) by means of public key cryptography and digital signatures. To operate a CA, you must maintain trust by protecting the private key that signs the certificates issued by your CA. You can store the private key in the HSM in your AWS CloudHSM cluster, and use the HSM to perform the cryptographic signing operations.

In this tutorial, you use Windows Server and AWS CloudHSM to configure a CA. You install the AWS CloudHSM client software for Windows on your Windows server, then add the Active Directory Certificate Services (AD CS) role to your Windows Server. When you configure this role, you use an AWS CloudHSM key storage provider (KSP) to create and store the CA's private key on your AWS CloudHSM cluster. The KSP is the bridge that connects your Windows server to your AWS CloudHSM cluster. In the last step, you sign a certificate signing request (CSR) with your Windows Server CA.

For more information, see the following topics:

Topics

Step 1: Set up the prerequisites

To set up Windows Server as a certificate authority (CA) with AWS CloudHSM, you need the following:

- An active AWS CloudHSM cluster with at least one HSM.
- An Amazon EC2 instance running a Windows Server operating system with the AWS CloudHSM client software for Windows installed. This tutorial uses Microsoft Windows Server 2016.

- A cryptographic user (CU) to own and manage the CA's private key on the HSM.

To set up the prerequisites for a Windows Server CA with AWS CloudHSM

1. Complete the steps in Getting started. When you launch the Amazon EC2 client, choose a Windows Server AMI. This tutorial uses Microsoft Windows Server 2016. When you complete these steps, you have an active cluster with at least one HSM. You also have an Amazon EC2 client instance running Windows Server with the AWS CloudHSM client software for Windows installed.
2. (Optional) Add more HSMs to your cluster. For more information, see Adding an HSM to an AWS CloudHSM cluster.
3. Connect to your client instance. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.
4. Create a crypto user (CU) using Managing HSM users with CloudHSM CLI or Managing HSM users with CloudHSM Management Utility (CMU). Keep track of the CU user name and password. You will need them to complete the next step.
5. Set the login credentials for the HSM, using the CU user name and password that you created in the previous step.
6. In step 5, if you used Windows Credentials Manager to set HSM credentials, download `psexec.exe` from SysInternals to run the following command as *NT Authority\SYSTEM*:

```
psexec.exe -s "C:\Program Files\Amazon\CloudHsm\tools
\set_cloudhsm_credentials.exe" --username <USERNAME> --password <PASSWORD>
```

   Replace *<USERNAME>* and *<PASSWORD>* with the HSM credentials.

To create a Windows Server CA with AWS CloudHSM, go to Create Windows Server CA.

Step 2: Create a Windows Server CA with AWS CloudHSM

To create a Windows Server CA, you add the Active Directory Certificate Services (AD CS) role to your Windows Server. When you add this role, you use an AWS CloudHSM key storage provider (KSP) to create and store the CA's private key on your AWS CloudHSM cluster.

> **Note**
>
> When you create your Windows Server CA, you can choose to create a root CA or a subordinate CA. You typically make this decision based on the design of your public key infrastructure and the security policies of your organization. This tutorial explains how to create a root CA for simplicity.

To add the AD CS role to your Windows Server and create the CA's private key

1. If you haven't already done so, connect to your Windows server. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.
2. On your Windows server, start **Server Manager**.
3. In the **Server Manager** dashboard, choose **Add roles and features**.
4. Read the **Before you begin** information, and then choose **Next**.
5. For **Installation Type**, choose **Role-based or feature-based installation**. Then choose **Next**.
6. For **Server Selection**, choose **Select a server from the server pool**. Then choose **Next**.
7. For **Server Roles**, do the following:
   a. Select **Active Directory Certificate Services**.
   b. For **Add features that are required for Active Directory Certificate Services**, choose **Add Features**.
   c. Choose **Next** to finish selecting server roles.
8. For **Features**, accept the defaults, and then choose **Next**.
9. For **AD CS**, do the following:
   a. Choose **Next**.
   b. Select **Certification Authority**, and then choose **Next**.
10. For **Confirmation**, read the confirmation information, and then choose **Install**. Do not close the window.
11. Choose the highlighted **Configure Active Directory Certificate Services on the destination server** link.
12. For **Credentials**, verify or change the credentials displayed. Then choose **Next**.
13. For **Role Services**, select **Certification Authority**. Then choose **Next**.

14. For **Setup Type**, select **Standalone CA**. Then choose **Next**.
15. For **CA Type**, select **Root CA**. Then choose **Next**.

> **Note**
>
> You can choose to create a root CA or a subordinate CA based on the design of your public key infrastructure and the security policies of your organization. This tutorial explains how to create a root CA for simplicity.

16. For **Private Key**, select **Create a new private key**. Then choose **Next**.
17. For **Cryptography**, do the following:

    a. For **Select a cryptographic provider**, choose one of the **Cavium Key Storage Provider** options from the menu. These are the AWS CloudHSM key storage providers. For example, you can choose **RSA#Cavium Key Storage Provider**.

    b. For **Key length**, choose one of the key length options.

    c. For **Select the hash algorithm for signing certificates issued by this CA**, choose one of the hash algorithm options.

    Choose **Next**.
18. For **CA Name**, do the following:

    a. (Optional) Edit the common name.

    b. (Optional) Type a distinguished name suffix.

    Choose **Next**.
19. For **Validity Period**, specify a time period in years, months, weeks, or days. Then choose **Next**.
20. For **Certificate Database**, you can accept the default values, or optionally change the location for the database and the database log. Then choose **Next**.
21. For **Confirmation**, review the information about your CA; Then choose **Configure**.
22. Choose **Close**, and then choose **Close** again.

You now have a Windows Server CA with AWS CloudHSM. To learn how to sign a certificate signing request (CSR) with your CA, go to Sign a CSR.

Step 3: Sign a certificate signing request (CSR) with your Windows Server CA with AWS CloudHSM

You can use your Windows Server CA with AWS CloudHSM to sign a certificate signing request (CSR). To complete these steps, you need a valid CSR. You can create a CSR in several ways, including the following:

- Using OpenSSL
- Using the Windows Server Internet Information Services (IIS) Manager
- Using the certificates snap-in in the Microsoft Management Console
- Using the **certreq** command line utility on Windows

The steps for creating a CSR are outside the scope of this tutorial. When you have a CSR, you can sign it with your Windows Server CA.

To sign a CSR with your Windows Server CA

1. If you haven't already done so, connect to your Windows server. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.
2. On your Windows server, start **Server Manager**.
3. In the **Server Manager** dashboard, in the top right corner, choose **Tools**, **Certification Authority**.
4. In the **Certification Authority** window, choose your computer name.
5. From the **Action** menu, choose **All Tasks**, **Submit new request**.
6. Select your CSR file, and then choose **Open**.
7. In the **Certification Authority** window, double-click **Pending Requests**.
8. Select the pending request. Then, from the **Action** menu, choose **All Tasks**, **Issue**.
9. In the **Certification Authority** window, double-click **Issued Requests** to view the signed certificate.
10. (Optional) To export the signed certificate to a file, complete the following steps:

    a. In the **Certification Authority** window, double-click the certificate.

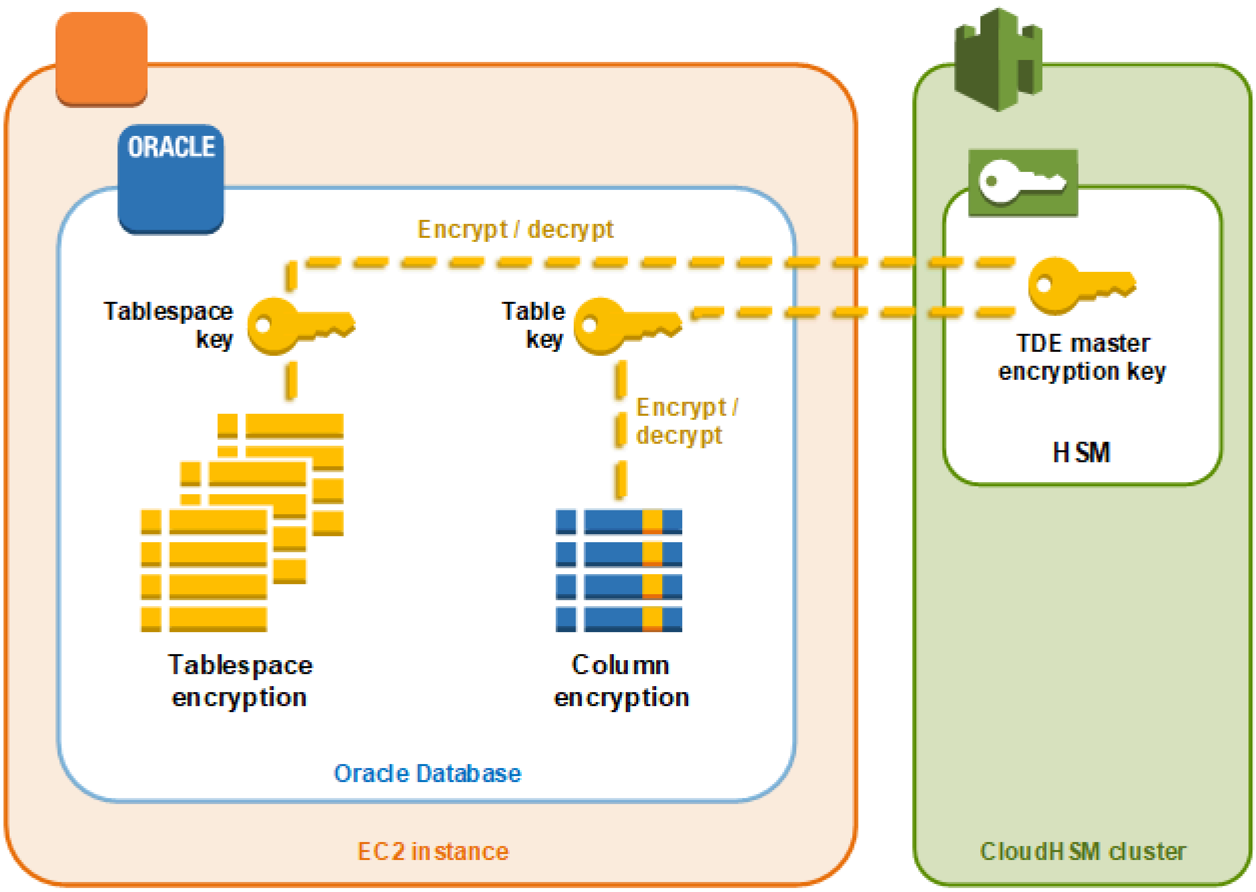
    b. Choose the **Details** tab, and then choose **Copy to File**.

    c. Follow the instructions in the **Certificate Export Wizard**.

You now have a Windows Server CA with AWS CloudHSM, and a valid certificate signed by the Windows Server CA.

Oracle database transparent data encryption (TDE) with AWS CloudHSM

Transparent Data Encryption (TDE) is used to encrypt database files. Using TDE, database software encrypts data before storing it on disk. The data in the database's table columns or tablespaces are encrypted with a table key or tablespace key. Some versions of Oracle's database software offer TDE. In Oracle TDE, these keys are encrypted with a TDE master encryption key. You can achieve greater security by storing the TDE master encryption key in the HSMs in your AWS CloudHSM cluster.



In this solution, you use Oracle Database installed on an Amazon EC2 instance. Oracle Database integrates with the [AWS CloudHSM software library for PKCS #11](#) to store the TDE master key in the HSMs in your cluster.

> **Important**
>
> - We recommend installing Oracle Database on an Amazon EC2 instance.

Complete the following steps to accomplish Oracle TDE integration with AWS CloudHSM.

Topics

Step 1. Set up the prerequisites

To accomplish Oracle TDE integration with AWS CloudHSM, you need the following:

- An active AWS CloudHSM cluster with at least one HSM.
- An Amazon EC2 instance running the Amazon Linux operating system with the following software installed:
  - The AWS CloudHSM client and command line tools.
  - The AWS CloudHSM software library for PKCS #11.
  - Oracle Database. AWS CloudHSM supports Oracle TDE integration. Client SDK 5.6 and higher support Oracle TDE for Oracle Database 19c. Client SDK 3 supports Oracle TDE for Oracle Database versions 11g and 12c.
- A cryptographic user (CU) to own and manage the TDE master encryption key on the HSMs in your cluster.

Complete the following steps to set up all of the prerequisites.

To set up the prerequisites for Oracle TDE integration with AWS CloudHSM

1. Complete the steps in Getting started. After you do so, you'll have an active cluster with one HSM. You will also have an Amazon EC2 instance running the Amazon Linux operating system. The AWS CloudHSM client and command line tools will also be installed and configured.
2. (Optional) Add more HSMs to your cluster. For more information, see Adding an HSM to an AWS CloudHSM cluster.

3. Connect to your Amazon EC2 client instance and do the following:

   a. Install the AWS CloudHSM software library for PKCS #11.

   b. Install Oracle Database. For more information, see the Oracle Database documentation. Client SDK 5.6 and higher support Oracle TDE for Oracle Database 19c. Client SDK 3 supports Oracle TDE for Oracle Database versions 11g and 12c.

   c. Use the cloudhsm_mgmt_util command line tool to create a cryptographic user (CU) on your cluster. For more information about creating a CU, see How to Manage HSM Users with CMU and HSM users.

Step 3: Generate the Oracle TDE master encryption key

To generate the Oracle TDE master key on the HSMs in your cluster, complete the steps in the following procedure.

To generate the master key

1. Use the following command to open Oracle SQL*Plus. When prompted, type the system password that you set when you installed Oracle Database.

```
sqlplus / as sysdba
```

> **Note**
>
> For Client SDK 3, you must set the `CLOUDHSM_IGNORE_CKA_MODIFIABLE_FALSE` environment variable each time you generate a master key. This variable is only needed for master key generation. For more information, see "Issue: Oracle sets the PCKS #11 attribute `CKA_MODIFIABLE` during master key generation, but the HSM does not support it" in Known Issues for Integrating Third-Party Applications.

2. Run the SQL statement that creates the master encryption key, as shown in the following examples. Use the statement that corresponds to your version of Oracle Database. Replace *<CU user name>* with the user name of the cryptographic user (CU). Replace *<password>* with the CU password.

> **Important**
>
> Run the following command only once. Each time the command is run, it creates a new master encryption key.

- For Oracle Database version 11, run the following SQL statement.

  ```
  SQL> alter system set encryption key identified by "<CU user name>:<password>";
  ```

- For Oracle Database version 12 and version 19c, run the following SQL statement.

  ```
  SQL> administer key management set key identified by "<CU user
   name>:<password>";
  ```

If the response is `System altered` or `keystore altered`, then you successfully generated and set the master key for Oracle TDE.

3. (Optional) Run the following command to verify the status of the *Oracle wallet*.

```
SQL> select * from v$encryption_wallet;
```

If the wallet is not open, use one of the following commands to open it. Replace *<CU user name>* with the name of the cryptographic user (CU). Replace *<password>* with the CU password.

- For Oracle 11, run the following command to open the wallet.

  ```
  SQL> alter system set encryption wallet open identified by "<CU user
   name>:<password>";
  ```

  To manually close the wallet, run the following command.

  ```
  SQL> alter system set encryption wallet close identified by "<CU user
   name>:<password>";
  ```

- For Oracle 12 and Oracle 19c, run the following command to open the wallet.

```
SQL> administer key management set keystore open identified by "<CU user
 name>:<password>";
```

To manually close the wallet, run the following command.

```
SQL> administer key management set keystore close identified by "<CU user
 name>:<password>";
```

Use Microsoft SignTool with AWS CloudHSM to sign files

AWS CloudHSM offers support to use Microsoft Signtool to sign file through Client SDK 3 and Client SDK 5. The steps to use these tools will vary depending on the version of the client SDK in which you currently have downloaded. The following sections provide information to each SDK.

Topics

Use Microsoft SignTool with Client SDK 5 to sign files

In cryptography and public key infrastructure (PKI), digital signatures are used to confirm that data has been sent by a trusted entity. Signatures also indicate that the data has not been tampered with in transit. A signature is an encrypted hash that is generated with the sender's private key. The receiver can verify the data integrity by decrypting its hash signature with the sender's public key. In turn, it is the sender's responsibility to maintain a digital certificate. The digital certificate demonstrates the sender's ownership of the private key and provides the recipient with the public key that is needed for decryption. As long as the private key is owned by the sender, the signature can be trusted. AWS CloudHSM provides secure FIPS 140-2 level 3 validated hardware for you to secure these keys with exclusive single-tenant access.

Many organizations use Microsoft SignTool, a command line tool that signs, verifies, and timestamps files to simplify the code signing process. You can use AWS CloudHSM to securely store your key pairs until they are needed by SignTool, thus creating an easily automatable workflow for signing data.

The following topics provide an overview of how to use SignTool with AWS CloudHSM.

Topics

- Step 1: Set up the prerequisites
- Step 2: Create a signing certificate
- Step 3: Sign a file

Step 1: Set up the prerequisites

To use Microsoft SignTool with AWS CloudHSM, you need the following:

- An Amazon EC2 client instance running a Windows operating system.
- A certificate authority (CA), either self-maintained or established by a third-party provider.
- An active AWS CloudHSM cluster in the same virtual public cloud (VPC) as your EC2 instance. The cluster must contain at least one HSM.
- A crypto user (CU) to own and manage keys in the AWS CloudHSM cluster.
- An unsigned file or executable.
- The Microsoft Windows Software Development Kit (SDK).

To set up the prerequisites for using AWS CloudHSM with Windows SignTool

1. Follow the instructions in the Getting Started section of this guide to launch a Windows EC2 instance and an AWS CloudHSM cluster.
2. If you would like to host your own Windows Server CA, follow steps 1 and 2 in Configuring Windows Server as a Certificate Authority with AWS CloudHSM. Otherwise, continue to use your publicly trusted third-party CA.
3. Download and install one of the following versions of the Microsoft Windows SDK on your Windows EC2 instance:

   - Microsoft Windows SDK 10
   - Microsoft Windows SDK 8.1
   - Microsoft Windows SDK 7

   The `SignTool` executable is part of the Windows SDK Signing Tools for Desktop Apps installation feature. You can omit the other features to be installed if you don't need them. The default installation location is:

```
C:\Program Files (x86)\Windows Kits\<SDK version>\bin\<version number>\<CPU
 architecture>\signtool.exe
```

You can now use the Microsoft Windows SDK, your AWS CloudHSM cluster, and your CA to Create a Signing Certificate.

Step 2: Create a signing certificate

Now that you've downloaded the Windows SDK on to your EC2 instance, you can use it to generate a certificate signing request (CSR). The CSR is an unsigned certificate that is eventually passed to your CA for signing. In this example, we use the `certreq` executable that's included with the Windows SDK to generate the CSR.

To generate a CSR using the `certreq` executable

1. If you haven't already done so, connect to your Windows EC2 instance. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.
2. Create a file called `request.inf` that contains the lines below. Replace the `Subject` information with that of your organization. For an explanation of each parameter, see Microsoft's documentation.

```
[Version]
Signature= $Windows NT$
[NewRequest]
Subject = "C=<Country>,CN=<www.website.com>,O=<Organization>,OU=<Organizational-
Unit>,L=<City>,S=<State>"
RequestType=PKCS10
HashAlgorithm = SHA256
KeyAlgorithm = RSA
KeyLength = 2048
ProviderName = "CloudHSM Key Storage Provider"
KeyUsage = "CERT_DIGITAL_SIGNATURE_KEY_USAGE"
MachineKeySet = True
Exportable = False
```

3. Run `certreq.exe`. For this example, we save the CSR as `request.csr`.

```
certreq.exe -new request.inf request.csr
```

Internally, a new key pair is generated on your AWS CloudHSM cluster, and the pair's private key is used to create the CSR.

4. Submit the CSR to your CA. If you are using a Windows Server CA, follow these steps:

    a. Enter the following command to open the CA tool:

    ```
    certsrv.msc
    ```

    b. In the new window, right-click the CA server's name. Choose **All Tasks**, and then choose **Submit new request**.

    c. Navigate to `request.csr`'s location and choose **Open**.

    d. Navigate to the **Pending Requests** folder by expanding the **Server CA** menu. Right-click on the request you just created, and under **All Tasks** choose **Issue**.

    e. Now navigate to the **Issued Certificates** folder (above the **Pending Requests** folder).

    f. Choose **Open** to view the certificate, and then choose the **Details** tab.

    g. Choose **Copy to File** to start the Certificate Export Wizard. Save the DER-encoded X.509 file to a secure location as `signedCertificate.cer`.

    h. Exit the CA tool and use the following command, which moves the certificate file to the Personal Certificate Store in Windows. It can then be used by other applications.

    ```
    certreq.exe -accept signedCertificate.cer
    ```

You can now use your imported certificate to Sign a File .

Step 3: Sign a file

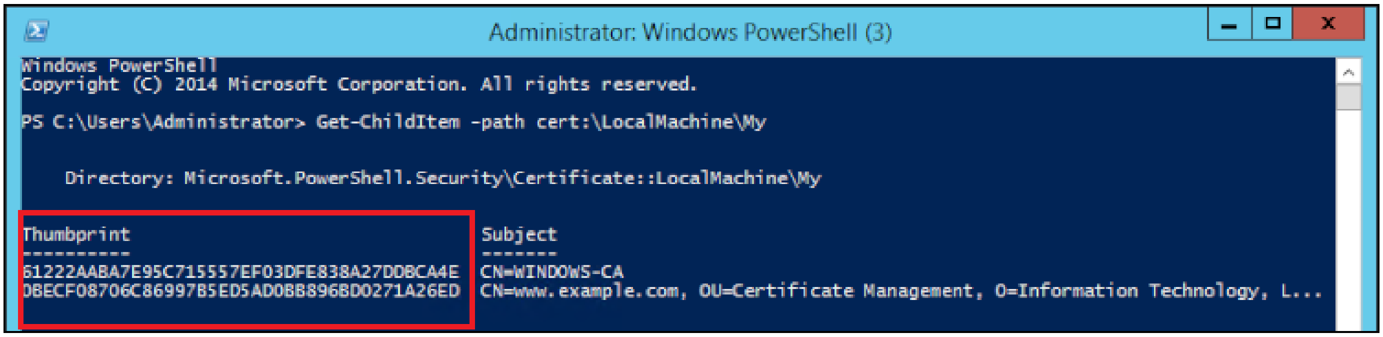
You are now ready to use SignTool and your imported certificate to sign your example file. In order to do so, you need to know the certificate's SHA-1 hash, or *thumbprint*. The thumbprint is used to ensure that SignTool only uses certificates that are verified by AWS CloudHSM. In this example, we use PowerShell to get the certificate's hash. You can also use the CA's GUI or the Windows SDK's `certutil` executable.

To obtain a certificate's thumbprint and use it to sign a file

1. Open PowerShell as an administrator and run the following command:

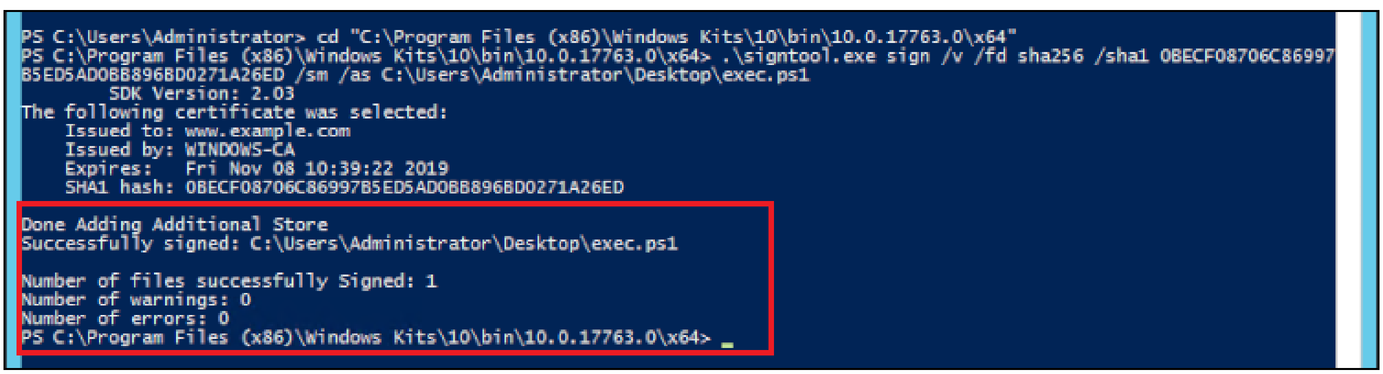
```
Get-ChildItem -path cert:\LocalMachine\My
```

Copy the `Thumbprint` that is returned.



2. Navigate to the directory within PowerShell that contains `SignTool.exe`. The default location is `C:\Program Files (x86)\Windows Kits\10\bin\10.0.17763.0\x64`.
3. Finally, sign your file by running the following command. If the command is successful, PowerShell returns a success message.

```
signtool.exe sign /v /fd sha256 /sha1 <thumbprint> /sm C:\Users\Administrator
\Desktop\<test>.ps1
```



4. (Optional) To verify the signature on the file, use the following command:

```
signtool.exe verify /v /pa C:\Users\Administrator\Desktop\<test>.ps1
```

Use Microsoft SignTool with Client SDK 3 to sign files

In cryptography and public key infrastructure (PKI), digital signatures are used to confirm that data has been sent by a trusted entity. Signatures also indicate that the data has not been tampered with in transit. A signature is an encrypted hash that is generated with the sender's private key. The receiver can verify the data integrity by decrypting its hash signature with the sender's public key. In turn, it is the sender's responsibility to maintain a digital certificate. The digital certificate demonstrates the sender's ownership of the private key and provides the recipient with the public key that is needed for decryption. As long as the private key is owned by the sender, the signature can be trusted. AWS CloudHSM provides secure FIPS 140-2 level 3 validated hardware for you to secure these keys with exclusive single-tenant access.

Many organizations use Microsoft SignTool, a command line tool that signs, verifies, and timestamps files to simplify the code signing process. You can use AWS CloudHSM to securely store your key pairs until they are needed by SignTool, thus creating an easily automatable workflow for signing data.

The following topics provide an overview of how to use SignTool with AWS CloudHSM.

Topics

Step 1: Set up the prerequisites

To use Microsoft SignTool with AWS CloudHSM, you need the following:

- An Amazon EC2 client instance running a Windows operating system.
- A certificate authority (CA), either self-maintained or established by a third-party provider.
- An active AWS CloudHSM cluster in the same virtual public cloud (VPC) as your EC2 instance. The cluster must contain at least one HSM.
- A crypto user (CU) to own and manage keys in the AWS CloudHSM cluster.
- An unsigned file or executable.
- The Microsoft Windows Software Development Kit (SDK).

To set up the prerequisites for using AWS CloudHSM with Windows SignTool

1. Follow the instructions in the Getting Started section of this guide to launch a Windows EC2 instance and an AWS CloudHSM cluster.

2. If you would like to host your own Windows Server CA, follow steps 1 and 2 in Configuring Windows Server as a Certificate Authority with AWS CloudHSM. Otherwise, continue to use your publicly trusted third-party CA.

3. Download and install one of the following versions of the Microsoft Windows SDK on your Windows EC2 instance:

   - Microsoft Windows SDK 10
   - Microsoft Windows SDK 8.1
   - Microsoft Windows SDK 7

   The `SignTool` executable is part of the Windows SDK Signing Tools for Desktop Apps installation feature. You can omit the other features to be installed if you don't need them. The default installation location is:

   ```
   C:\Program Files (x86)\Windows Kits\<SDK version>\bin\<version number>\<CPU architecture>\signtool.exe
   ```

You can now use the Microsoft Windows SDK, your AWS CloudHSM cluster, and your CA to Create a Signing Certificate.

Step 2: Create a signing certificate

Now that you've downloaded the Windows SDK on to your EC2 instance, you can use it to generate a certificate signing request (CSR). The CSR is an unsigned certificate that is eventually passed to your CA for signing. In this example, we use the `certreq` executable that's included with the Windows SDK to generate the CSR.

To generate a CSR using the `certreq` executable

1. If you haven't already done so, connect to your Windows EC2 instance. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.

2. Create a file called `request.inf` that contains the lines below. Replace the `Subject` information with that of your organization. For an explanation of each parameter, see Microsoft's documentation.

   ```
   [Version]
   Signature= $Windows NT$
   ```

```
[NewRequest]
Subject = "C=<Country>,CN=<www.website.com>,O=<Organization>,OU=<Organizational-Unit>,L=<City>,S=<State>"
RequestType=PKCS10
HashAlgorithm = SHA256
KeyAlgorithm = RSA
KeyLength = 2048
ProviderName = "Cavium Key Storage Provider"
KeyUsage = "CERT_DIGITAL_SIGNATURE_KEY_USAGE"
MachineKeySet = True
Exportable = False
```

3. Run `certreq.exe`. For this example, we save the CSR as `request.csr`.

```
certreq.exe -new request.inf request.csr
```

Internally, a new key pair is generated on your AWS CloudHSM cluster, and the pair's private key is used to create the CSR.

4. Submit the CSR to your CA. If you are using a Windows Server CA, follow these steps:

a. Enter the following command to open the CA tool:

```
certsrv.msc
```

b. In the new window, right-click the CA server's name. Choose **All Tasks**, and then choose **Submit new request**.

c. Navigate to `request.csr`'s location and choose **Open**.

d. Navigate to the **Pending Requests** folder by expanding the **Server CA** menu. Right-click on the request you just created, and under **All Tasks** choose **Issue**.

e. Now navigate to the **Issued Certificates** folder (above the **Pending Requests** folder).

f. Choose **Open** to view the certificate, and then choose the **Details** tab.

g. Choose **Copy to File** to start the Certificate Export Wizard. Save the DER-encoded X.509 file to a secure location as `signedCertificate.cer`.

h. Exit the CA tool and use the following command, which moves the certificate file to the Personal Certificate Store in Windows. It can then be used by other applications.

```
certreq.exe -accept signedCertificate.cer
```

You can now use your imported certificate to Sign a File .

Step 3: Sign a file

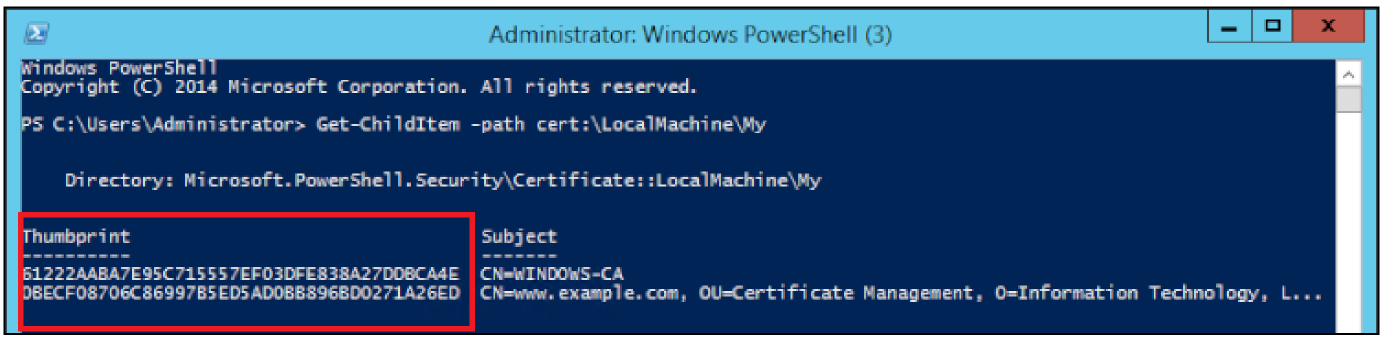
You are now ready to use SignTool and your imported certificate to sign your example file. In order to do so, you need to know the certificate's SHA-1 hash, or *thumbprint*. The thumbprint is used to ensure that SignTool only uses certificates that are verified by AWS CloudHSM. In this example, we use PowerShell to get the certificate's hash. You can also use the CA's GUI or the Windows SDK's `certutil` executable.

To obtain a certificate's thumbprint and use it to sign a file

1. Open PowerShell as an administrator and run the following command:

```
Get-ChildItem -path cert:\LocalMachine\My
```

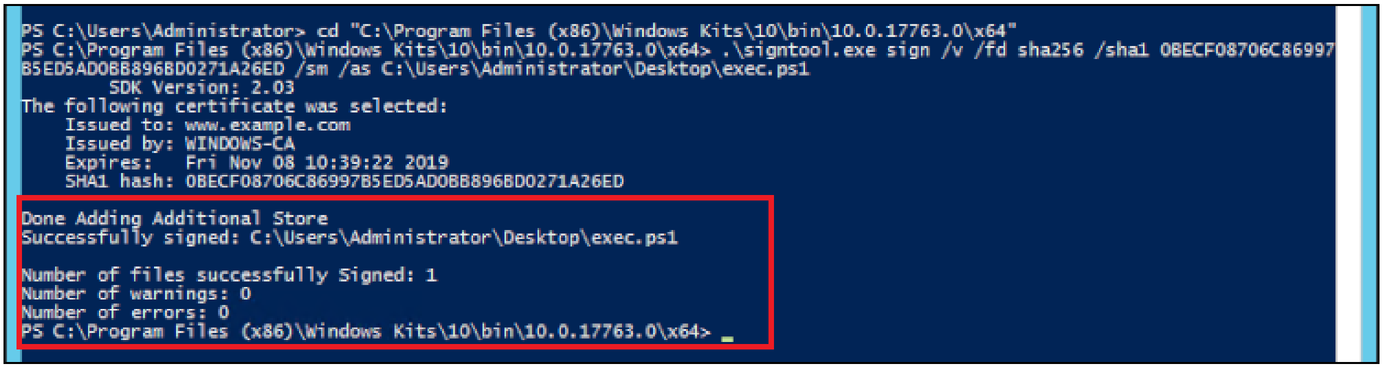
   Copy the `Thumbprint` that is returned.



2. Navigate to the directory within PowerShell that contains `SignTool.exe`. The default location is `C:\Program Files (x86)\Windows Kits\10\bin\10.0.17763.0\x64`.

3. Finally, sign your file by running the following command. If the command is successful, PowerShell returns a success message.

```
signtool.exe sign /v /fd sha256 /sha1 <thumbprint> /sm C:\Users\Administrator
\Desktop\<test>.ps1
```



4. (Optional) To verify the signature on the file, use the following command:

```
signtool.exe verify /v /pa C:\Users\Administrator\Desktop\<test>.ps1
```

Java Keytool and Jarsigner integration with AWS CloudHSM

AWS CloudHSM offers integration with the Java Keytool and Jarsigner utilities through Client SDK 3 and Client SDK 5. The steps to use these tools will vary depending on the version of the client SDK in which you currently have downloaded. The following sections provide information to each SDK.

Topics

- Use Client SDK 5 to integrate AWS CloudHSM with Java Keytool and Jarsigner
- Use Client SDK 3 to integrate AWS CloudHSM with Java Keytool and Jarsigner

Use Client SDK 5 to integrate AWS CloudHSM with Java Keytool and Jarsigner

AWS CloudHSM key store is a special-purpose JCE key store that utilizes certificates associated with keys on your hardware security module (HSM) through third-party tools such as `keytool` and `jarsigner`. AWS CloudHSM does not store certificates on the HSM, as certificates are public, non-confidential data. The AWS CloudHSM key store stores the certificates in a local file and maps the certificates to corresponding keys on your HSM.

When you use the AWS CloudHSM key store to generate new keys, no entries are generated in the local key store file – the keys are created on the HSM. Similarly, when you use the AWS CloudHSM key store to search for keys, the search is passed on to the HSM. When you store certificates in the

AWS CloudHSM key store, the provider verifies that a key pair with the corresponding alias exists on the HSM, and then associates the certificate provided with the corresponding key pair.

Topics

- Prerequisites for integrating AWS CloudHSM with Java Keytool and Jarsigner using Client SDK 5
- Use AWS CloudHSM key store with keytool using Client SDK 5
- Use AWS CloudHSM key store with Jarsigner using Client SDK 5
- Known issues for AWS CloudHSM integration Java Keytool and Jarsigner using Client SDK 5

Prerequisites for integrating AWS CloudHSM with Java Keytool and Jarsigner using Client SDK 5

To use the AWS CloudHSM key store, you must first initialize and configure the AWS CloudHSM JCE SDK. Use the following steps to do so.

Step 1: Install the JCE

To install the JCE, including the AWS CloudHSM client prerequisites, follow the steps for installing the Java library.

Step 2: Add HSM login credentials to environment variables

Set up environment variables to contain your HSM login credentials.

Linux

```
$ export HSM_USER=<HSM user name>
```

```
$ export HSM_PASSWORD=<HSM password>
```

Windows

```
PS C:\> $Env:HSM_USER=<HSM user name>
```

```
PS C:\> $Env:HSM_PASSWORD=<HSM password>
```

> **Note**
>
> The AWS CloudHSM JCE offers various login options. To use the AWS CloudHSM key store with third-party applications, you must use implicit login with environment variables. If you want to use explicit login through application code, you must build your own application using the AWS CloudHSM key store. For additional information, see the article on Using AWS CloudHSM Key Store.

Step 3: Registering the JCE provider

To register the JCE provider in the Java CloudProvider configuration, follow these steps:

1. Open the `java.security` configuration file in your Java installation for editing.
2. In the `java.security` configuration file, add `com.amazonaws.cloudhsm.jce.provider.CloudHsmProvider` as the last provider. For example, if there are nine providers in the `java.security` file, add the following provider as the last provider in the section:

```
security.provider.10=com.amazonaws.cloudhsm.jce.provider.CloudHsmProvider
```

> **Note**
>
> Adding the AWS CloudHSM provider as a higher priority may negatively impact your system's performance because the AWS CloudHSM provider will be prioritized for operations that may be safely offloaded to software. As a best practice, **always** specify the provider you wish to use for an operation, whether it is the AWS CloudHSM or a software-based provider.

> **Note**
>
> Specifying `-providerName`, `-providerclass`, and `-providerpath` command line options when generating keys using **keytool** with the AWS CloudHSM key store may cause errors.

Use AWS CloudHSM key store with keytool using Client SDK 5

Keytool is a popular command line utility for common key and certificate tasks. A complete tutorial on keytool is out of scope for AWS CloudHSM documentation. This article explains the specific parameters you should use with various keytool functions when utilizing AWS CloudHSM as the root of trust through the AWS CloudHSM key store.

When using keytool with the AWS CloudHSM key store, specify the following arguments to any keytool command:

Linux

```
-storetype CLOUDHSM -J-classpath< '-J/opt/cloudhsm/java/*'>
```

Windows

```
-storetype CLOUDHSM -J-classpath<'-J"C:\Program Files\Amazon\CloudHSM\java\*"'>
```

If you want to create a new key store file using AWS CloudHSM key store, see Use the AWS CloudHSM KeyStore for AWS CloudHSM Client SDK 3. To use an existing key store, specify its name (including path) using the –keystore argument to keytool. If you specify a non-existent key store file in a keytool command, the AWS CloudHSM key store creates a new key store file.

Create new AWS CloudHSM keys with keytool

You can use keytool to generate RSA, AES, and DESede type of key supported by the AWS CloudHSM JCE SDK.

> **Important**
>
> A key generated through keytool is generated in software, and then imported into AWS CloudHSM as an extractable, persistent key.

We strongly recommend generating non-exportable keys outside of keytool, and then importing corresponding certificates to the key store. If you use extractable RSA or EC keys through keytool and Jarsigner, the providers export keys from the AWS CloudHSM and then use the key locally for signing operations.

If you have multiple client instances connected to your AWS CloudHSM cluster, be aware that importing a certificate on one client instance's key store won't automatically make the certificates available on other client instances. To register the key and associated certificates on each client instance you need to run a Java application as described in the section called "Generate a CSR using keytool". Alternatively, you can make the necessary changes on one client and copy the resulting key store file to every other client instance.

Example 1: To generate a symmetric AES-256 key and save it in a key store file named, "example_keystore.store", in the working directory. Replace *<secret label>* with a unique label.

Linux

```
$ keytool -genseckey -alias <secret label> -keyalg aes \
 -keysize 256 -keystore example_keystore.store \
 -storetype CloudHSM -J-classpath '-J/opt/cloudhsm/java/*' \
```

Windows

```
PS C:\> keytool -genseckey -alias <secret label> -keyalg aes `
 -keysize 256 -keystore example_keystore.store `
 -storetype CloudHSM -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

Example 2: To generate an RSA 2048 key pair and save it in a key store file named, "example_keystore.store" in the working directory. Replace *<RSA key pair label>* with a unique label.

Linux

```
$ keytool -genkeypair -alias <RSA key pair label> \
 -keyalg rsa -keysize 2048 \
 -sigalg sha512withrsa \
 -keystore example_keystore.store \
 -storetype CLOUDHSM \
 -J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -genkeypair -alias <RSA key pair label> `
 -keyalg rsa -keysize 2048 `
 -sigalg sha512withrsa `
```

```
 -keystore example_keystore.store `
 -storetype CLOUDHSM `
 -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

You can find a list of supported signature algorithms in the Java library.

Delete an AWS CloudHSM key using keytool

The AWS CloudHSM key store doesn't support deleting keys. You can delete keys using the destroy method of the Destroyable interface.

```
((Destroyable) key).destroy();
```

Generate an AWS CloudHSM CSR using keytool

You receive the greatest flexibility in generating a certificate signing request (CSR) if you use the OpenSSL Dynamic Engine for AWS CloudHSM Client SDK 5. The following command uses keytool to generate a CSR for a key pair with the alias, `example-key-pair`.

Linux

```
$ keytool -certreq -alias <key pair label> \
 -file my_csr.csr \
 -keystore example_keystore.store \
 -storetype CLOUDHSM \
 -J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -certreq -alias <key pair label> `
 -file my_csr.csr `
 -keystore example_keystore.store `
 -storetype CLOUDHSM `
 -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

> **Note**
>
> To use a key pair from keytool, that key pair must have an entry in the specified key store file. If you want to use a key pair that was generated outside of keytool, you must import

the key and certificate metadata into the key store. For instructions on importing the keystore data see the section called "Use keytool to import certificates into key store ".

Use keytool to import intermediate and root certificates into AWS CloudHSM key store

To import a CA certificate in AWS CloudHSM, you must enable verification of a full certificate chain on a newly imported certificate. The following command shows an example.

Linux

```
$ keytool -import -trustcacerts -alias rootCAcert \
 -file rootCAcert.cert -keystore example_keystore.store \
 -storetype CLOUDHSM \
 -J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -import -trustcacerts -alias rootCAcert `
 -file rootCAcert.cert -keystore example_keystore.store `
 -storetype CLOUDHSM `
 -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

If you connect multiple client instances to your AWS CloudHSM cluster, importing a certificate on one client instance's key store won't automatically make the certificate available on other client instances. You must import the certificate on each client instance.

Use keytool to delete certificates from AWS CloudHSM key store

The following command shows an example of how to delete a AWS CloudHSM certificate from a Java keytool key store.

Linux

```
$ keytool -delete -alias mydomain \
 -keystore example_keystore.store \
 -storetype CLOUDHSM \
 -J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -delete -alias mydomain `
 -keystore example_keystore.store `
 -storetype CLOUDHSM `
 -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

If you connect multiple client instances to your AWS CloudHSM cluster, deleting a certificate on one client instance’s key store won't automatically remove the certificate from other client instances. You must delete the certificate on each client instance.

Import a working certificate into AWS CloudHSM key store using keytool

Once a certificate signing request (CSR) is signed, you can import it into the AWS CloudHSM key store and associate it with the appropriate key pair. The following command provides an example.

Linux

```
$ keytool -importcert -noprompt -alias <key pair label> \
 -file my_certificate.crt \
 -keystore example_keystore.store \
 -storetype CLOUDHSM \
 -J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -importcert -noprompt -alias <key pair label> `
 -file my_certificate.crt `
 -keystore example_keystore.store `
 -storetype CLOUDHSM `
 -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

The alias should be a key pair with an associated certificate in the key store. If the key is generated outside of keytool, or is generated on a different client instance, you must first import the key and certificate metadata into the key store.

The certificate chain must be verifiable. If you can't verify the certificate, you might need to import the signing (certificate authority) certificate into the key store so the chain can be verified.

Export a certificate from AWS CloudHSM using keytool

The following example generates a certificate in binary X.509 format. To export a human readable certificate from AWS CloudHSM, add `-rfc` to the `-exportcert` command.

Linux

```
$ keytool -exportcert -alias <key pair label> \
 -file my_exported_certificate.crt \
 -keystore example_keystore.store \
 -storetype CLOUDHSM \
 -J-classpath '-J/opt/cloudhsm/java/*'
```

Windows

```
PS C:\> keytool -exportcert -alias <key pair label> `
 -file my_exported_certificate.crt `
 -keystore example_keystore.store `
 -storetype CLOUDHSM `
 -J-classpath '-J"C:\Program Files\Amazon\CloudHSM\java\*"'
```

Use AWS CloudHSM key store with Jarsigner using Client SDK 5

Jarsigner is a popular command line utility for signing JAR files using a key securely stored on a hardware security module (HSM). A complete tutorial on Jarsigner is out of scope for the AWS CloudHSM documentation. This section explains the Jarsigner parameters you should use to sign and verify signatures with AWS CloudHSM as the root of trust through the AWS CloudHSM key store.

Set up AWS CloudHSM keys and certificates with Jarsigner

Before you can sign AWS CloudHSM JAR files with Jarsigner, make sure you have set up or completed the following steps:

1. Follow the guidance in the AWS CloudHSM key store prerequisites .
2. Set up your signing keys and the associated certificates and certificate chain which should be stored in the AWS CloudHSM key store of the current server or client instance. Create the keys on the AWS CloudHSM and then import associated metadata into your AWS CloudHSM key store. If you want to use keytool to set up the keys and certificates, see the section called "Create new keys with keytool". If you use multiple client instances to sign your JARs, create

the key and import the certificate chain. Then copy the resulting key store file to each client instance. If you frequently generate new keys, you may find it easier to individually import certificates to each client instance.

3. The entire certificate chain should be verifiable. For the certificate chain to be verifiable, you may need to add the CA certificate and intermediate certificates to the AWS CloudHSM key store. See the code snippet in the section called "Sign a JAR file" for instruction on using Java code to verify the certificate chain. If you prefer, you can use keytool to import certificates. For instructions on using keytool, see the section called "Use keytool to import certificates into key store ".

Sign a JAR file using AWS CloudHSM and Jarsigner

Use the following command to sign a JAR file using AWS CloudHSM and Jarsigner:

Linux;

For OpenJDK 8

```
jarsigner -keystore example_keystore.store \
 -signedjar signthisclass_signed.jar \
 -sigalg sha512withrsa \
 -storetype CloudHSM \
 -J-classpath '-J/opt/cloudhsm/java/*:/usr/lib/jvm/java-1.8.0/lib/tools.jar' \
 -J-Djava.library.path=/opt/cloudhsm/lib \
 signthisclass.jar <key pair label>
```

For OpenJDK 11, OpenJDK 17, and OpenJDK 21

```
jarsigner -keystore example_keystore.store \
 -signedjar signthisclass_signed.jar \
 -sigalg sha512withrsa \
 -storetype CloudHSM \
 -J-classpath '-J/opt/cloudhsm/java/*' \
 -J-Djava.library.path=/opt/cloudhsm/lib \
 signthisclass.jar <key pair label>
```

Windows

For OpenJDK8

```
jarsigner -keystore example_keystore.store `
 -signedjar signthisclass_signed.jar `
 -sigalg sha512withrsa `
 -storetype CloudHSM `
 -J-classpath '-JC:\Program Files\Amazon\CloudHSM\java\*;C:\Program Files\Java
\jdk1.8.0_331\lib\tools.jar' `
  "-J-Djava.library.path='C:\Program Files\Amazon\CloudHSM\lib\'" `
 signthisclass.jar <key pair label>
```

For OpenJDK 11, OpenJDK 17, and OpenJDK 21

```
jarsigner -keystore example_keystore.store `
 -signedjar signthisclass_signed.jar `
 -sigalg sha512withrsa `
 -storetype CloudHSM `
 -J-classpath '-JC:\Program Files\Amazon\CloudHSM\java\*'`
  "-J-Djava.library.path='C:\Program Files\Amazon\CloudHSM\lib\'" `
 signthisclass.jar <key pair label>
```

Use the following command to verify a signed JAR:

Linux

For OpenJDK8

```
jarsigner -verify \
 -keystore example_keystore.store \
 -sigalg sha512withrsa \
 -storetype CloudHSM \
 -J-classpath '-J/opt/cloudhsm/java/*:/usr/lib/jvm/java-1.8.0/lib/tools.jar' \
 -J-Djava.library.path=/opt/cloudhsm/lib \
 signthisclass_signed.jar <key pair label>
```

For OpenJDK 11, OpenJDK 17, and OpenJDK 21

```
jarsigner -verify \
 -keystore example_keystore.store \
```

```
 -sigalg sha512withrsa \
 -storetype CloudHSM \
 -J-classpath '-J/opt/cloudhsm/java/*' \
 -J-Djava.library.path=/opt/cloudhsm/lib \
 signthisclass_signed.jar <key pair label>
```

Windows

For OpenJDK 8

```
jarsigner -verify `
 -keystore example_keystore.store `
 -sigalg sha512withrsa `
 -storetype CloudHSM `
 -J-classpath '-JC:\Program Files\Amazon\CloudHSM\java\*;C:\Program Files\Java
\jdk1.8.0_331\lib\tools.jar' `
 "-J-Djava.library.path='C:\Program Files\Amazon\CloudHSM\lib\'" `
 signthisclass_signed.jar <key pair label>
```

For OpenJDK 11, OpenJDK 17, and OpenJDK 21

```
jarsigner -verify `
 -keystore example_keystore.store `
 -sigalg sha512withrsa `
 -storetype CloudHSM `
 -J-classpath '-JC:\Program Files\Amazon\CloudHSM\java\*`
 "-J-Djava.library.path='C:\Program Files\Amazon\CloudHSM\lib\'" `
 signthisclass_signed.jar <key pair label>
```

Known issues for AWS CloudHSM integration Java Keytool and Jarsigner using Client SDK 5

The following list provides the current list of known issues for integrations with AWS CloudHSM and Java Keytool and Jarsigner using Client SDK 5.

1. We do not support EC keys with Keytool and Jarsigner.

Use Client SDK 3 to integrate AWS CloudHSM with Java Keytool and Jarsigner

AWS CloudHSM key store is a special-purpose JCE key store that utilizes certificates associated with keys on your hardware security module (HSM) through third-party tools such as `keytool` and `jarsigner`. AWS CloudHSM does not store certificates on the HSM, as certificates are public, non-confidential data. The AWS CloudHSM key store stores the certificates in a local file and maps the certificates to corresponding keys on your HSM.

When you use the AWS CloudHSM key store to generate new keys, no entries are generated in the local key store file – the keys are created on the HSM. Similarly, when you use the AWS CloudHSM key store to search for keys, the search is passed on to the HSM. When you store certificates in the AWS CloudHSM key store, the provider verifies that a key pair with the corresponding alias exists on the HSM, and then associates the certificate provided with the corresponding key pair.

Topics

Prerequisites for integrating AWS CloudHSM with Java Keytool and Jarsigner using Client SDK 3

To use the AWS CloudHSM key store, you must first initialize and configure the AWS CloudHSM JCE SDK. Use the following steps to do so.

Step 1: Install the JCE

To install the JCE, including the AWS CloudHSM client prerequisites, follow the steps for installing the Java library.

Step 2: Add HSM login credentials to environment variables

Set up environment variables to contain your HSM login credentials.

```
export HSM_PARTITION=PARTITION_1
export HSM_USER=<HSM user name>
export HSM_PASSWORD=<HSM password>
```

> **Note**
>
> The CloudHSM JCE offers various login options. To use the AWS CloudHSM key store with third-party applications, you must use implicit login with environment variables. If you want to use explicit login through application code, you must build your own application using the AWS CloudHSM key store. For additional information, see the article on [Using AWS CloudHSM Key Store](#).

Step 3: Register the JCE provider

To register the JCE provider, in the Java CloudProvider configuration.

1. Open the java.security configuration file in your Java installation, for editing.
2. In the java.security configuration file, add `com.cavium.provider.CaviumProvider` as the last provider. For example, if there are nine providers in the java.security file, add the following provider as the last provider in the section. Adding the Cavium provider as a higher priority may negatively impact your system's performance.

   ```
   security.provider.10=com.cavium.provider.CaviumProvider
   ```

   > **Note**
   >
   > Power users may be accustomed to specifying `-providerName`, `-providerclass`, and `-providerpath` command line options when using keytool, instead of updating the security configuration file. If you attempt to specify command line options when generating keys with AWS CloudHSM key store, it will cause errors.

Use AWS CloudHSM key store with keytool using Client SDK 3

[Keytool](#) is a popular command line utility for common key and certificate tasks on Linux systems. A complete tutorial on keytool is out of scope for AWS CloudHSM documentation. This article explains the specific parameters you should use with various keytool functions when utilizing AWS CloudHSM as the root of trust through the AWS CloudHSM key store.

When using keytool with the AWS CloudHSM key store, specify the following arguments to any keytool command:

```
-storetype CLOUDHSM \
  -J-classpath '-J/opt/cloudhsm/java/*' \
  -J-Djava.library.path=/opt/cloudhsm/lib
```

If you want to create a new key store file using AWS CloudHSM key store, see Use the AWS CloudHSM KeyStore for AWS CloudHSM Client SDK 3. To use an existing key store, specify its name (including path) using the –keystore argument to keytool. If you specify a non-existent key store file in a keytool command, the AWS CloudHSM key store creates a new key store file.

Create new AWS CloudHSM keys with keytool

You can use keytool to generate any type of key supported by the AWS CloudHSM JCE SDK. See a full list of keys and lengths in the Supported Keys article in the Java Library.

> **Important**
>
> A key generated through keytool is generated in software, and then imported into AWS CloudHSM as an extractable, persistent key.

Instructions for creating non-extractable keys directly on the hardware security module (HSM), and then using them with keytool or Jarsigner, are shown in the code sample in Registering Pre-existing Keys with AWS CloudHSM Key Store. We strongly recommend generating non-exportable keys outside of keytool, and then importing corresponding certificates to the key store. If you use extractable RSA or EC keys through keytool and jarsigner, the providers export keys from the AWS CloudHSM and then use the key locally for signing operations.

If you have multiple client instances connected to your CloudHSM cluster, be aware that importing a certificate on one client instance’s key store won't automatically make the certificates available on other client instances. To register the key and associated certificates on each client instance you need to run a Java application as described in Generate a CSR using Keytool. Alternatively, you can make the necessary changes on one client and copy the resulting key store file to every other client instance.

Example 1: To generate a symmetric AES-256 key and save it in a key store file named, "example_keystore.store", in the working directory. Replace *<secret label>* with a unique label.

```
keytool -genseckey -alias <secret label> -keyalg aes \
  -keysize 256 -keystore example_keystore.store \
  -storetype CloudHSM -J-classpath '-J/opt/cloudhsm/java/*' \
  -J-Djava.library.path=/opt/cloudhsm/lib/
```

Example 2: To generate an RSA 2048 key pair and save it in a key store file named, "example_keystore.store" in the working directory. Replace *<RSA key pair label>* with a unique label.

```
keytool -genkeypair -alias <RSA key pair label> \
        -keyalg rsa -keysize 2048 \
        -sigalg sha512withrsa \
        -keystore example_keystore.store \
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

Example 3: To generate a p256 ED key and save it in a key store file named, "example_keystore.store" in the working directory. Replace *<ec key pair label>* with a unique label.

```
keytool -genkeypair -alias <ec key pair label> \
        -keyalg ec -keysize 256 \
        -sigalg SHA512withECDSA \
        -keystore example_keystore.store \
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

You can find a list of supported signature algorithms in the Java library.

Delete an AWS CloudHSM key using keytool

The AWS CloudHSM key store doesn't support deleting keys. To delete key, you must use the `deleteKey` function of AWS CloudHSM's command line tool, Delete an AWS CloudHSM key using KMU.

Generate an AWS CloudHSM CSR using keytool

You receive the greatest flexibility in generating a certificate signing request (CSR) if you use the OpenSSL Dynamic Engine for AWS CloudHSM Client SDK 5. The following command uses keytool to generate a CSR for a key pair with the alias, `example-key-pair`.

```
keytool -certreq -alias <key pair label> \
        -file example_csr.csr \
        -keystore example_keystore.store \
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

> **Note**
>
> To use a key pair from keytool, that key pair must have an entry in the specified key store file. If you want to use a key pair that was generated outside of keytool, you must import the key and certificate metadata into the key store. For instructions on importing the keystore data see Importing Intermediate and root certificates into AWS CloudHSM Key Store using Keytool.

Use keytool to import intermediate and root certificates into AWS CloudHSM key store

To import a CA certificate into AWS CloudHSM, you must enable verification of a full certificate chain on a newly imported certificate. The following command shows an example.

```
keytool -import -trustcacerts -alias rootCAcert \
        -file rootCAcert.cert -keystore example_keystore.store \
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

If you connect multiple client instances to your AWS CloudHSM cluster, importing a certificate on one client instance's key store won't automatically make the certificate available on other client instances. You must import the certificate on each client instance.

Use keytool to delete certificates from AWS CloudHSM key store

The following command shows an example of how to delete an AWS CloudHSM certificate from a Java keytool key store.

```
keytool -delete -alias mydomain -keystore \
        -keystore example_keystore.store \
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

If you connect multiple client instances to your AWS CloudHSM cluster, deleting a certificate on one client instance's key store won't automatically remove the certificate from other client instances. You must delete the certificate on each client instance.

Import a working certificate into AWS CloudHSM key store using keytool

Once a certificate signing request (CSR) is signed, you can import it into the AWS CloudHSM key store and associate it with the appropriate key pair. The following command provides an example.

```
keytool -importcert -noprompt -alias <key pair label> \
        -file example_certificate.crt \
        -keystore example_keystore.store
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

The alias should be a key pair with an associated certificate in the key store. If the key is generated outside of keytool, or is generated on a different client instance, you must first import the key and certificate metadata into the key store. For instructions on importing the certificate metadata, see the code sample in Registering Pre-existing Keys with AWS CloudHSM Key Store.

The certificate chain must be verifiable. If you can't verify the certificate, you might need to import the signing (certificate authority) certificate into the key store so the chain can be verified.

Export a certificate from AWS CloudHSM using keytool

The following example generates a certificate in binary X.509 format. To export a human readable certificate from AWS CloudHSM, add `-rfc` to the `-exportcert` command.

```
keytool -exportcert -alias <key pair label> \
        -file example_exported_certificate.crt \
```

```
        -keystore example_keystore.store \
        -storetype CLOUDHSM \
        -J-classpath '-J/opt/cloudhsm/java/*' \
        -J-Djava.library.path=/opt/cloudhsm/lib/
```

Use AWS CloudHSM key store with Jarsigner using Client SDK 3

Jarsigner is a popular command line utility for signing JAR files using a key securely stored on a hardware security module (HSM). A complete tutorial on Jarsigner is out of scope for the AWS CloudHSM documentation. This section explains the Jarsigner parameters you should use to sign and verify signatures with AWS CloudHSM as the root of trust through the AWS CloudHSM key store.

Set up AWS CloudHSM keys and certificates with Jarsigner

Before you can sign AWS CloudHSM JAR files with Jarsigner, make sure you have set up or completed the following steps:

1. Follow the guidance in the AWS CloudHSM Key store prerequisites .
2. Set up your signing keys and the associated certificates and certificate chain which should be stored in the AWS CloudHSM key store of the current server or client instance. Create the keys on the AWS CloudHSM and then import associated metadata into your AWS CloudHSM key store. Use the code sample in Registering Pre-existing Keys with AWS CloudHSM Key Store to import metadata into the key store. If you want to use keytool to set up the keys and certificates, see Create new AWS CloudHSM keys with keytool. If you use multiple client instances to sign your JARs, create the key and import the certificate chain. Then copy the resulting key store file to each client instance. If you frequently generate new keys, you may find it easier to individually import certificates to each client instance.
3. The entire certificate chain should be verifiable. For the certificate chain to be verifiable, you may need to add the CA certificate and intermediate certificates to the AWS CloudHSM key store. See the code snippet in Sign a JAR file using AWS CloudHSM and Jarsigner for instruction on using Java code to verify the certificate chain. If you prefer, you can use keytool to import certificates. For instructions on using keytool, see Using Keytool to import intermediate and root certificates into AWS CloudHSM Key Store.

Sign a JAR file using AWS CloudHSM and Jarsigner

Use the following command to sign a JAR file using AWS CloudHSM and jarsigner:

```
jarsigner -keystore example_keystore.store \
        -signedjar signthisclass_signed.jar \
        -sigalg sha512withrsa \
        -storetype CloudHSM \
        -J-classpath '-J/opt/cloudhsm/java/*:/usr/lib/jvm/java-1.8.0/lib/tools.jar' \
        -J-Djava.library.path=/opt/cloudhsm/lib \
        signthisclass.jar <key pair label>
```

Use the following command to verify a signed JAR:

```
jarsigner -verify \
        -keystore example_keystore.store \
        -sigalg sha512withrsa \
        -storetype CloudHSM \
        -J-classpath '-J/opt/cloudhsm/java/*:/usr/lib/jvm/java-1.8.0/lib/tools.jar' \
        -J-Djava.library.path=/opt/cloudhsm/lib \
        signthisclass_signed.jar <key pair label>
```

Known issues for AWS CloudHSM integration Java Keytool and Jarsigner using Client SDK 3

The following list provides the current list of known issues for integrations with AWS CloudHSM and Java Keytool and Jarsigner using Client SDK 3.

- When generating keys using keytool, the first provider in provider configuration cannot be CaviumProvider.
- When generating keys using keytool, the first (supported) provider in the security configuration file is used to generate the key. This is generally a software provider. The generated key is then given an alias and imported into the AWS CloudHSM HSM as a persistent (token) key during the key addition process.
- When using keytool with AWS CloudHSM key store, do not specify `-providerName`, `-providerclass`, or `-providerpath` options on the command line. Specify these options in the security provider file as described in the Key store prerequisites.
- When using non-extractable EC keys through keytool and Jarsigner, the SunEC provider needs to be removed/disabled from the list of providers in the java.security file. If you use extractable EC keys through keytool and Jarsigner, the providers export key bits from the AWS CloudHSM HSM and use the key locally for signing operations. We do not recommend you use exportable keys with keytool or Jarsigner.

Register pre-existing keys with AWS CloudHSM key store

For maximum security and flexibility in attributes and labeling, we recommend you generate your AWS CloudHSM signing keys using key_mgmt_util. You can also use a Java application to generate the key in AWS CloudHSM.

The following section provides a code sample that demonstrates how to generate a new key pair on the HSM and register it using existing keys imported to the AWS CloudHSM key store. The imported keys are available for use with third-party tools such as keytool and Jarsigner.

To use a pre-existing key, modify the code sample to look up a key by label instead of generating a new key. Sample code for looking up a key by label is available in the KeyUtilitiesRunner.java sample on GitHub.

> **Important**
>
> Registering a key stored on AWS CloudHSM with a local key store does not export the key. When the key is registered, the key store registers the key's alias (or label) and correlates locally store certificate objects with a key pair on the AWS CloudHSM. As long as the key pair is created as non-exportable, the key bits won't leave the HSM.

```
                      //
// Copyright 2018 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of
this
// software and associated documentation files (the "Software"), to deal in the
Software
// without restriction, including without limitation the rights to use, copy, modify,
// merge, publish, distribute, sublicense, and/or sell copies of the Software, and to
// permit persons to whom the Software is furnished to do so.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
// INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
// PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
// HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
// OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
```

```
 // SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
 //

package com.amazonaws.cloudhsm.examples;

import com.cavium.key.CaviumKey;
import com.cavium.key.parameter.CaviumAESKeyGenParameterSpec;
import com.cavium.key.parameter.CaviumRSAKeyGenParameterSpec;
import com.cavium.asn1.Encoder;
import com.cavium.cfm2.Util;

import javax.crypto.KeyGenerator;

import java.io.ByteArrayInputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileNotFoundException;

import java.math.BigInteger;

import java.security.*;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.security.interfaces.RSAPrivateKey;
import java.security.interfaces.RSAPublicKey;
import java.security.KeyStore.PasswordProtection;
import java.security.KeyStore.PrivateKeyEntry;
import java.security.KeyStore.Entry;

import java.util.Calendar;
import java.util.Date;
import java.util.Enumeration;

//
// KeyStoreExampleRunner demonstrates how to load a keystore, and associate a
 certificate with a
// key in that keystore.
//
// This example relies on implicit credentials, so you must setup your environment
 correctly.
//
```

```
// https://docs.aws.amazon.com/cloudhsm/latest/userguide/java-library-
install.html#java-library-credentials
//

public class KeyStoreExampleRunner {

     private static byte[] COMMON_NAME_OID = new byte[] { (byte) 0x55, (byte) 0x04,
 (byte) 0x03 };
     private static byte[] COUNTRY_NAME_OID = new byte[] { (byte) 0x55, (byte) 0x04,
 (byte) 0x06 };
     private static byte[] LOCALITY_NAME_OID = new byte[] { (byte) 0x55, (byte) 0x04,
 (byte) 0x07 };
     private static byte[] STATE_OR_PROVINCE_NAME_OID = new byte[] { (byte) 0x55,
 (byte) 0x04, (byte) 0x08 };
     private static byte[] ORGANIZATION_NAME_OID = new byte[] { (byte) 0x55, (byte)
 0x04, (byte) 0x0A };
     private static byte[] ORGANIZATION_UNIT_OID = new byte[] { (byte) 0x55, (byte)
 0x04, (byte) 0x0B };

     private static String helpString = "KeyStoreExampleRunner%n" +
            "This sample demonstrates how to load and store keys using a keystore.%n%n"
 +
            "Options%n" +
            "\t--help\t\t\tDisplay this message.%n" +
            "\t--store <filename>\t\tPath of the keystore.%n" +
            "\t--password <password>\t\tPassword for the keystore (not your CU
 password).%n" +
            "\t--label <label>\t\t\tLabel to store the key and certificate under.%n" +
            "\t--list\t\t\tList all the keys in the keystore.%n%n";

    public static void main(String[] args) throws Exception {
        Security.addProvider(new com.cavium.provider.CaviumProvider());
        KeyStore keyStore = KeyStore.getInstance("CloudHSM");

        String keystoreFile = null;
        String password = null;
        String label = null;
        boolean list = false;
        for (int i = 0; i < args.length; i++) {
            String arg = args[i];
            switch (args[i]) {
                case "--store":
                    keystoreFile = args[++i];
                    break;
```

```
                case "--password":
                    password = args[++i];
                    break;
                case "--label":
                    label = args[++i];
                    break;
                case "--list":
                    list = true;
                    break;
                case "--help":
                    help();
                    return;
            }
        }

        if (null == keystoreFile || null == password) {
            help();
            return;
        }

        if (list) {
            listKeys(keystoreFile, password);
            return;
        }

        if (null == label) {
            label = "Keystore Example Keypair";
        }

        //
        // This call to keyStore.load() will open the pkcs12 keystore with the supplied
        // password and connect to the HSM. The CU credentials must be specified using
        // standard CloudHSM login methods.
        //
        try {
            FileInputStream instream = new FileInputStream(keystoreFile);
            keyStore.load(instream, password.toCharArray());
        } catch (FileNotFoundException ex) {
            System.err.println("Keystore not found, loading an empty store");
            keyStore.load(null, null);
        }

        PasswordProtection passwd = new PasswordProtection(password.toCharArray());
        System.out.println("Searching for example key and certificate...");
```

```
        PrivateKeyEntry keyEntry = (PrivateKeyEntry) keyStore.getEntry(label, passwd);
        if (null == keyEntry) {
            //
            // No entry was found, so we need to create a key pair and associate a
 certificate.
            // The private key will get the label passed on the command line. The
 keystore alias
            // needs to be the same as the private key label. The public key will have
 ":public"
            // appended to it. The alias used in the keystore will We associate the
 certificate
            // with the private key.
            //
            System.out.println("No entry found, creating...");
            KeyPair kp = generateRSAKeyPair(2048, label + ":public", label);
            System.out.printf("Created a key pair with the handles %d/%d%n",
 ((CaviumKey) kp.getPrivate()).getHandle(), ((CaviumKey) kp.getPublic()).getHandle());

            //
            // Generate a certificate and associate the chain with the private key.
            //
            Certificate self_signed_cert = generateCert(kp);
            Certificate[] chain = new Certificate[1];
            chain[0] = self_signed_cert;
            PrivateKeyEntry entry = new PrivateKeyEntry(kp.getPrivate(), chain);

            //
            // Set the entry using the label as the alias and save the store.
            // The alias must match the private key label.
            //
            keyStore.setEntry(label, entry, passwd);

            FileOutputStream outstream = new FileOutputStream(keystoreFile);
            keyStore.store(outstream, password.toCharArray());
            outstream.close();

            keyEntry = (PrivateKeyEntry) keyStore.getEntry(label, passwd);
        }

        long handle = ((CaviumKey) keyEntry.getPrivateKey()).getHandle();
        String name = keyEntry.getCertificate().toString();
        System.out.printf("Found private key %d with certificate %s%n", handle, name);
    }
```

```
    private static void help() {
        System.out.println(helpString);
    }

    //
    // Generate a non-extractable / non-persistent RSA keypair.
    // This method allows us to specify the public and private labels, which
    // will make KeyStore aliases easier to understand.
    //
    public static KeyPair generateRSAKeyPair(int keySizeInBits, String publicLabel,
 String privateLabel)
            throws InvalidAlgorithmParameterException, NoSuchAlgorithmException,
 NoSuchProviderException {

        boolean isExtractable = false;
        boolean isPersistent = false;
        KeyPairGenerator keyPairGen = KeyPairGenerator.getInstance("rsa", "Cavium");
        CaviumRSAKeyGenParameterSpec spec = new
 CaviumRSAKeyGenParameterSpec(keySizeInBits, new BigInteger("65537"), publicLabel,
 privateLabel, isExtractable, isPersistent);

        keyPairGen.initialize(spec);

        return keyPairGen.generateKeyPair();
    }

    //
    // Generate a certificate signed by a given keypair.
    //
    private static Certificate generateCert(KeyPair kp) throws CertificateException {
        CertificateFactory cf = CertificateFactory.getInstance("X509");
        PublicKey publicKey = kp.getPublic();
        PrivateKey privateKey = kp.getPrivate();
        byte[] version = Encoder.encodeConstructed((byte) 0,
 Encoder.encodePositiveBigInteger(new BigInteger("2"))); // version 1
        byte[] serialNo = Encoder.encodePositiveBigInteger(new BigInteger(1,
 Util.computeKCV(publicKey.getEncoded())));

        // Use the SHA512 OID and algorithm.
        byte[] signatureOid = new byte[] {
            (byte) 0x2A, (byte) 0x86, (byte) 0x48, (byte) 0x86, (byte) 0xF7, (byte)
 0x0D, (byte) 0x01, (byte) 0x01, (byte) 0x0D };
        String sigAlgoName = "SHA512WithRSA";
```

```
        byte[] signatureId = Encoder.encodeSequence(
                                        Encoder.encodeOid(signatureOid),
                                        Encoder.encodeNull());

        byte[] issuer = Encoder.encodeSequence(
                                    encodeName(COUNTRY_NAME_OID, "<Country>"),
                                    encodeName(STATE_OR_PROVINCE_NAME_OID, "<State>"),
                                    encodeName(LOCALITY_NAME_OID, "<City>"),
                                    encodeName(ORGANIZATION_NAME_OID,
 "<Organization>"),
                                    encodeName(ORGANIZATION_UNIT_OID, "<Unit>"),
                                    encodeName(COMMON_NAME_OID, "<CN>")
                                );

        Calendar c = Calendar.getInstance();
        c.add(Calendar.DAY_OF_YEAR, -1);
        Date notBefore = c.getTime();
        c.add(Calendar.YEAR, 1);
        Date notAfter = c.getTime();
        byte[] validity = Encoder.encodeSequence(
                                        Encoder.encodeUTCTime(notBefore),
                                        Encoder.encodeUTCTime(notAfter)
                                    );
        byte[] key = publicKey.getEncoded();

        byte[] certificate = Encoder.encodeSequence(
                                        version,
                                        serialNo,
                                        signatureId,
                                        issuer,
                                        validity,
                                        issuer,
                                        key);
        Signature sig;
        byte[] signature = null;
        try {
            sig = Signature.getInstance(sigAlgoName, "Cavium");
            sig.initSign(privateKey);
            sig.update(certificate);
            signature = Encoder.encodeBitstring(sig.sign());

        } catch (Exception e) {
            System.err.println(e.getMessage());
```

```
                return null;
            }

            byte [] x509 = Encoder.encodeSequence(
                            certificate,
                            signatureId,
                            signature
                            );
            return cf.generateCertificate(new ByteArrayInputStream(x509));
    }

     //
     // Simple OID encoder.
     // Encode a value with OID in ASN.1 format
     //
     private static byte[] encodeName(byte[] nameOid, String value) {
         byte[] name = null;
         name = Encoder.encodeSet(
                     Encoder.encodeSequence(
                             Encoder.encodeOid(nameOid),
                             Encoder.encodePrintableString(value)
                     )
                 );
         return name;
     }

    //
    // List all the keys in the keystore.
    //
    private static void listKeys(String keystoreFile, String password) throws Exception
 {
        KeyStore keyStore = KeyStore.getInstance("CloudHSM");

        try {
            FileInputStream instream = new FileInputStream(keystoreFile);
            keyStore.load(instream, password.toCharArray());
        } catch (FileNotFoundException ex) {
            System.err.println("Keystore not found, loading an empty store");
            keyStore.load(null, null);
        }

        for(Enumeration<String> entry = keyStore.aliases(); entry.hasMoreElements();) {
            System.out.println(entry.nextElement());
        }
```

```
    }

}
```

Use Microsoft Manifest Generation and Editing Tool (Mage.exe) with AWS CloudHSM to sign files

Note

AWS CloudHSM supports only the 64-bit Mage tool included in the Windows SDK for .NET Framework 4.8.1 and later.

The following topics provide an overview of how to use Mage.exe with AWS CloudHSM.

Topics

Step 1: Set up the prerequisites

To use Microsoft Mage.exe with AWS CloudHSM, you need the following:

- An Amazon EC2 instance running a Windows operating system
- A certificate authority (CA), either self-maintained or from a third-party provider
- An active AWS CloudHSM cluster in the same virtual private cloud (VPC) as your EC2 instance, with at least one HSM
- A crypto user (CU) to own and manage keys in the AWS CloudHSM cluster
- An unsigned file or executable
- The Microsoft Windows Software Development Kit (SDK)

To set up the prerequisites for using AWS CloudHSM with Mage.exe

1. Launch a Windows EC2 instance and an AWS CloudHSM cluster by following the instructions in the Getting Started section of this guide.
2. If you want to host your own Windows Server CA, complete steps 1 and 2 in Configuring Windows Server as a Certificate Authority with AWS CloudHSM. Otherwise, use your publicly trusted third-party CA.
3. Download and install Microsoft Windows SDK for .NET Framework 4.8.1 or later on your Windows EC2 instance:

   - Microsoft Windows SDK 10

   The `mage.exe` executable is part of the Windows SDK Tools. The default installation location is:

   ```
   C:\Program Files (x86)\Windows Kits\<SDK version>\bin\<version number>\x64\Mage.exe
   ```

After completing these steps, you can use the Microsoft Windows SDK, your AWS CloudHSM cluster, and your CA to create a signing certificate.

Step 2: Create a signing certificate

Now that you've installed the Windows SDK on your EC2 instance, you can use it to generate a certificate signing request (CSR). The CSR is an unsigned certificate that you submit to your CA for signing. In this example, we use the `certreq` executable included with the Windows SDK to generate the CSR.

To generate a CSR using the certreq executable

1. Connect to your Windows EC2 instance. For more information, see Connect to Your Instance in the *Amazon EC2 User Guide*.
2. Create a file named `request.inf` with the following content. Replace the `Subject` information with your organization's details:

   ```
   [Version]
   Signature= $Windows NT$
   [NewRequest]
   ```

```
Subject = "C=<Country>,CN=<www.website.com>,O=<Organization>,OU=<Organizational-
Unit>,L=<City>,S=<State>"
RequestType=PKCS10
HashAlgorithm = SHA256
KeyAlgorithm = RSA
KeyLength = 2048
ProviderName = "CloudHSM Key Storage Provider"
KeyUsage = "CERT_DIGITAL_SIGNATURE_KEY_USAGE"
MachineKeySet = True
Exportable = False
```

For an explanation of each parameter, see Microsoft's documentation.

3. Run `certreq.exe` to generate the CSR:

```
certreq.exe -new request.inf request.csr
```

This command generates a new key pair on your AWS CloudHSM cluster and uses the private key to create the CSR.

4. Submit the CSR to your CA. If you're using a Windows Server CA, follow these steps:

   a. Open the CA tool:

   ```
   certsrv.msc
   ```

   b. In the new window, right-click the CA server's name. Choose **All Tasks**, and then choose **Submit new request**.

   c. Navigate to the location of `request.csr` and choose **Open**.

   d. Expand the **Server CA** menu and navigate to the **Pending Requests** folder. Right-click the request you just created, choose **All Tasks**, and then choose **Issue**.

   e. Navigate to the **Issued Certificates** folder.

   f. Choose **Open** to view the certificate, and then choose the **Details** tab.

   g. Choose **Copy to File** to start the Certificate Export Wizard. Save the DER-encoded X.509 file to a secure location as `signedCertificate.cer`.

   h. Exit the CA tool and run the following command to move the certificate file to the Personal Certificate Store in Windows:

   ```
   certreq.exe -accept signedCertificate.cer
   ```

You can now use your imported certificate to sign a file.

Step 3: Sign a file

Now that you have Mage.exe and your imported certificate, you can sign a file. You need to know the certificate's SHA-1 hash, or *thumbprint*. The thumbprint ensures that Mage.exe only uses certificates verified by AWS CloudHSM. In this example, we use PowerShell to get the certificate's hash.

To obtain a certificate's thumbprint and use it to sign a file

1. Navigate to the directory containing `mage.exe`. The default location is:

```
C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8.1 Tools\x64
```

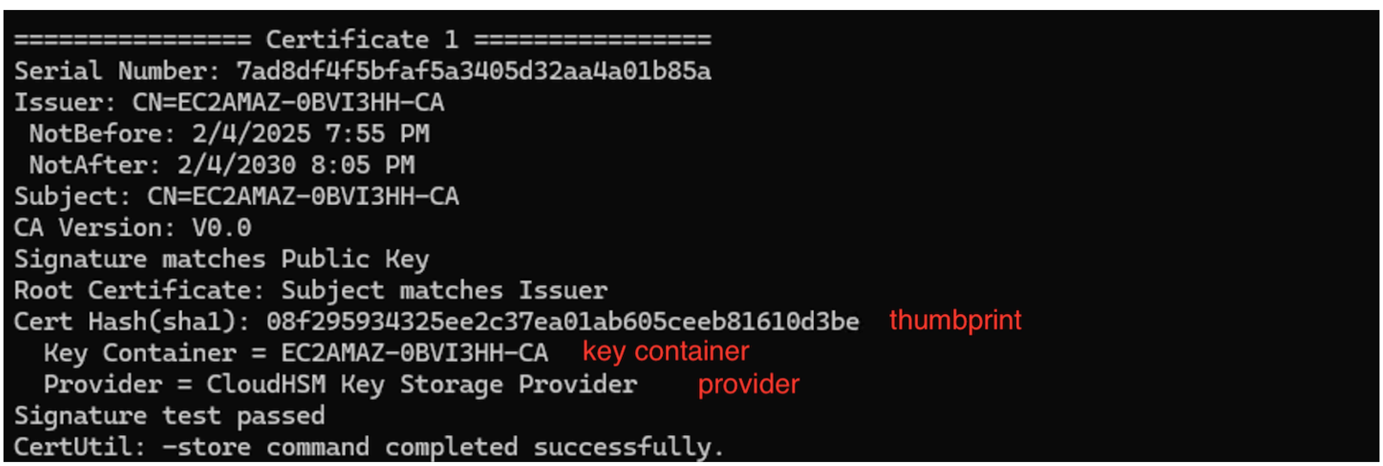
2. To create a sample application file using Mage.exe, run the following command:

```
mage.exe -New Application -ToFile C:\Users\Administrator\Desktop\sample.application
```

3. Open PowerShell as an administrator and run the following command:

```
Get-ChildItem -path cert:\LocalMachine\My
```

Copy the `Thumbprint`, `Key Container`, and `Provider` values from the output.



4. Sign your file by running the following command:

```
mage.exe -Sign -CertHash <thumbprint> -KeyContainer <keycontainer> -
CryptoProvider <CloudHSM Key Storage Provider/Cavium Key Storage Provider> C:\Users
\Administrator\Desktop\<sample.application>
```

If the command is successful, PowerShell returns a success message.

5. To verify the signature on the file, use the following command:

```
mage.exe -Verify -CryptoProvider <CloudHSM Key Storage Provider/Cavium Key Storage
 Provider> C:\Users\Administrator\Desktop\<sample.application>
```

Other third-party vendor integrations with AWS CloudHSM

Several third-party vendors support AWS CloudHSM as a root of trust. This means that you can utilize a software solution of your choice while creating and storing the underlying keys in your CloudHSM cluster. As a result, your workload in AWS can rely on the latency, availability, reliability, and elasticity benefits of CloudHSM. The following list includes third-party vendors that support CloudHSM.

> **Note**
>
> AWS does not endorse or vouch for any third-party vendor.

- **Hashicorp Vault** is a secrets management tool designed to enable collaboration and governance across organizations. It supports AWS Key Management Service and AWS CloudHSM as roots of trust for additional protection.
- **Thycotic Secrets Server** helps customers manage sensitive credentials across privileged accounts. It supports AWS CloudHSM as a root of trust.
- **P6R's KMIP adapter** allows you to utilize your AWS CloudHSM instances through a standard KMIP interface.
- **PrimeKey EJBCA** is a popular open source solution for PKI. It allows you to create and store key pairs securely with AWS CloudHSM.
- **Box KeySafe** provides encryption key management for cloud content to many organizations with strict security, privacy, and regulatory compliance requirements. Customers can further secure KeySafe keys directly in AWS Key Management Service or indirectly in AWS CloudHSM via AWS KMS Custom Key Store.
- **Insyde Software** supports AWS CloudHSM as a root of trust for firmware signing.
- **F5 BIG-IP LTM** supports AWS CloudHSM as a root of trust.

- **Cloudera Navigator Key HSM** allows you to use your CloudHSM cluster to create and store keys for Cloudera Navigator Key Trustee Server.
- **Venafi Trust Protection Platform** provides comprehensive machine identity management for TLS, SSH, and code signing with AWS CloudHSM key generation and protection.

Monitoring AWS CloudHSM

In addition to the logging features built into the Client SDK, you can also use AWS CloudTrail, Amazon CloudWatch Logs, and Amazon CloudWatch to monitor AWS CloudHSM.

Client SDK logs

Use Client SDK logging to monitor diagnostic and troubleshooting information from the applications you create.

CloudTrail

Use CloudTrail to monitor all API calls in your AWS account, including the calls you make to create and delete clusters, hardware security modules (HSM), and resource tags.

CloudWatch Logs

Use CloudWatch Logs to monitor the logs from your HSM instances, which include events for create and delete HSM users, change user passwords, create and delete keys, and more.

CloudWatch

Use CloudWatch to monitor the health of your cluster in real time.

Topics

Working with AWS CloudHSM client SDK logs

You can retrieve logs generated by the Client SDK. AWS CloudHSM offers an implementation of logging with Client SDK 3 and Client SDK 5.

Topics

Client SDK 5 logging

Client SDK 5 logs contain information for each component in a file named for the component. You can use the configure tool for Client SDK 5 to configure logging for each component.

If you do not specify a location for the file, the system writes logs to the default location:

PKCS #11 library

- Linux

  ```
  /opt/cloudhsm/run/cloudhsm-pkcs11.log
  ```

  Windows

  ```
  C:\Program Files\Amazon\CloudHSM\cloudhsm-pkcs11.log
  ```

OpenSSL Dynamic Engine

- Linux

  ```
  stderr
  ```

JCE provider

- Linux

  ```
  /opt/cloudhsm/run/cloudhsm-jce.log
  ```

  Windows

  ```
  C:\Program Files\Amazon\CloudHSM\cloudhsm-jce.log
  ```

For information on how to configure logging for Client SDK 5, see the Client SDK 5 Configure tool

Client SDK 3 logging

Client SDK 3 logs contain detailed information from the AWS CloudHSM client daemon. The location of the logs depends on the operating system of the Amazon EC2 client instance where you run the client daemon.

Amazon Linux

In Amazon Linux, the AWS CloudHSM client logs are written to the file named `/opt/cloudhsm/run/cloudhsm_client.log`. You can use *logrotate* or a similar tool to rotate and manage these logs.

Amazon Linux 2

In Amazon Linux 2, the AWS CloudHSM Client logs are collected and stored in the *journal*. You can use *journalctl* to view and manage these logs. For example, use the following command to view the AWS CloudHSM Client logs.

```
journalctl -f -u cloudhsm-client
```

CentOS 7

In CentOS 7, the AWS CloudHSM Client logs are collected and stored in the *journal*. You can use *journalctl* to view and manage these logs. For example, use the following command to view the AWS CloudHSM Client logs.

```
journalctl -f -u cloudhsm-client
```

CentOS 8

In CentOS 8, the AWS CloudHSM Client logs are collected and stored in the *journal*. You can use *journalctl* to view and manage these logs. For example, use the following command to view the AWS CloudHSM Client logs.

```
journalctl -f -u cloudhsm-client
```

RHEL 7

In Red Hat Enterprise Linux 7, the AWS CloudHSM Client logs are collected and stored in the *journal*. You can use *journalctl* to view and manage these logs. For example, use the following command to view the AWS CloudHSM Client logs.

```
journalctl -f -u cloudhsm-client
```

RHEL 8

In Red Hat Enterprise Linux 8, the AWS CloudHSM Client logs are collected and stored in the *journal*. You can use *journalctl* to view and manage these logs. For example, use the following command to view the AWS CloudHSM Client logs.

```
journalctl -f -u cloudhsm-client
```

Ubuntu 16.04

In Ubuntu 16.04, the AWS CloudHSM Client logs are collected and stored in the *journal*. You can use *journalctl* to view and manage these logs. For example, use the following command to view the AWS CloudHSM Client logs.

```
journalctl -f -u cloudhsm-client
```

Ubuntu 18.04

In Ubuntu 18.04, the AWS CloudHSM Client logs are collected and stored in the *journal*. You can use *journalctl* to view and manage these logs. For example, use the following command to view the AWS CloudHSM Client logs.

```
journalctl -f -u cloudhsm-client
```

Windows

- For Windows client 1.1.2+:

  AWS CloudHSM client logs are written to a `cloudhsm.log` file in the AWS CloudHSM program files folder (`C:\Program Files\Amazon\CloudHSM\`). Each log file name is suffixed with a timestamp indicating when the AWS CloudHSM client was started.

- For Windows client 1.1.1 and older:

  The client logs are not written to a file. The logs are displayed at the command prompt or in the PowerShell window where you started the AWS CloudHSM client.

Working with AWS CloudTrail and AWS CloudHSM

AWS CloudHSM is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS CloudHSM. CloudTrail captures all API calls for AWS CloudHSM as events. The calls captured include calls from the AWS CloudHSM console and code calls to the AWS CloudHSM API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS CloudHSM. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS CloudHSM, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the AWS CloudTrail User Guide. For a full list of AWS CloudHSM API operations, see Actions in the *AWS CloudHSM API Reference*.

AWS CloudHSM information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS CloudHSM, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing Events with CloudTrail Event History.

For an ongoing record of events in your AWS account, including events for AWS CloudHSM, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- Overview for Creating a Trail
- CloudTrail Supported Services and Integrations
- Configuring Amazon SNS Notifications for CloudTrail
- Receiving CloudTrail Log Files from Multiple Regions and Receiving CloudTrail Log Files from Multiple Accounts

CloudTrail logs all AWS CloudHSM operations, including read-only operations, such as `DescribeClusters` and `ListTags`, and management operations, such as `InitializeCluster`, `CreatHsm`, and `DeleteBackup`.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or AWS Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity Element.

Understanding AWS CloudHSM log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the AWS CloudHSM `CreateHsm` action.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAJZVM5NEGZSTCITAMM:ExampleSession",
        "arn": "arn:aws:sts::111122223333:assumed-role/AdminRole/ExampleSession",
        "accountId": "111122223333",
        "accessKeyId": "ASIAIY22AX6VRYNBGJSA",
        "sessionContext": {
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2017-07-11T03:48:44Z"
            },
            "sessionIssuer": {
                "type": "Role",
```

```
                "principalId": "AROAJZVM5NEGZSTCITAMM",
                "arn": "arn:aws:iam::111122223333:role/AdminRole",
                "accountId": "111122223333",
                "userName": "AdminRole"
            }
        }
    },
    "eventTime": "2017-07-11T03:50:45Z",
    "eventSource": "cloudhsm.amazonaws.com",
    "eventName": "CreateHsm",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "205.251.233.179",
    "userAgent": "aws-internal/3",
    "requestParameters": {
        "availabilityZone": "us-west-2b",
        "clusterId": "cluster-fw7mh6mayb5"
    },
    "responseElements": {
        "hsm": {
            "eniId": "eni-65338b5a",
            "clusterId": "cluster-fw7mh6mayb5",
            "state": "CREATE_IN_PROGRESS",
            "eniIp": "10.0.2.7",
            "hsmId": "hsm-6lz2hfmnzbx",
            "subnetId": "subnet-02c28c4b",
            "availabilityZone": "us-west-2b"
        }
    },
    "requestID": "1dae0370-65ec-11e7-a770-6578d63de907",
    "eventID": "b73a5617-8508-4c3d-900d-aa8ac9b31d08",
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```


Working with Amazon CloudWatch Logs and AWS CloudHSM Audit Logs

When an HSM in your account receives a command from the AWS CloudHSM command line tools or software libraries, it records its execution of the command in audit log form. The HSM audit logs include all client-initiated management commands, including those that create and delete the

HSM, log into and out of the HSM, and manage users and keys. These logs provide a reliable record of actions that have changed the state of the HSM.

AWS CloudHSM collects your HSM audit logs and sends them to Amazon CloudWatch Logs on your behalf. You can use the features of CloudWatch Logs to manage your AWS CloudHSM Audit Logs, including searching and filtering the logs and exporting log data to Amazon S3. You can work with your HSM audit logs in the Amazon CloudWatch console or use the CloudWatch Logs commands in the AWS CLI and CloudWatch Logs SDKs.

Topics

How HSM audit logging works

Audit logging is automatically enabled in all AWS CloudHSM clusters. It cannot be disabled or turned off, and no settings can prevent AWS CloudHSM from exporting the logs to CloudWatch Logs. Each log event has a time stamp and sequence number that indicate the order of events and help you detect any log tampering.

Each HSM instance generates its own log. The audit logs of various HSMs, even those in the same cluster, are likely to differ. For example, only the first HSM in each cluster records initialization of the HSM. Initialization events do not appear in the logs of HSMs that are cloned from backups. Similarly, when you create a key, the HSM that generates the key records a key generation event. The other HSMs in the cluster record an event when they receive the key via synchronization.

AWS CloudHSM collects the logs and posts them to CloudWatch Logs in your account. To communicate with the CloudWatch Logs service on your behalf, AWS CloudHSM uses a service-linked role. The IAM policy that is associated with the role allows AWS CloudHSM to perform only the tasks required to send the audit logs to CloudWatch Logs.

> **Important**
>
> If you created a cluster before January 20, 2018, and have not yet created an attached service-linked role, you must manually create one. This is necessary for CloudWatch to

receive audit logs from your AWS CloudHSM cluster. For more information about service-linked role creation, see Understanding Service-Linked Roles, as well as Creating a Service-Linked Role in the *IAM User Guide*.

Viewing AWS CloudHSM audit logs in CloudWatch Logs

Amazon CloudWatch Logs organizes the audit logs into *log groups* and, within a log group, into *log streams*. Each log entry is an *event*. AWS CloudHSM creates one *log group* for each cluster and one *log stream* for each HSM in the cluster. You do not have to create any CloudWatch Logs components or change any settings.

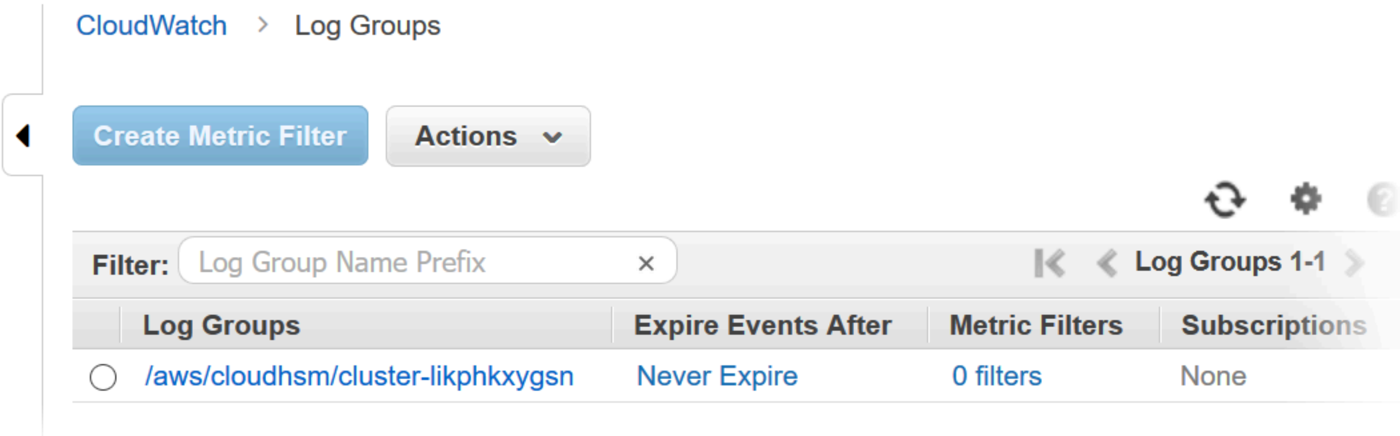
- The *log group* name is `/aws/cloudhsm/<cluster ID>`; for example `/aws/cloudhsm/cluster-likphkxygsn`. When you use the log group name in a AWS CLI or PowerShell command, be sure to enclose it in double quotation marks.
- The *log stream* name is the HSM ID; for example, `hsm-nwbbiqbj4jk`.

  In general, there is one log stream for each HSM. However, any action that changes the HSM ID, such as when an HSM fails and is replaced, creates a new log stream.

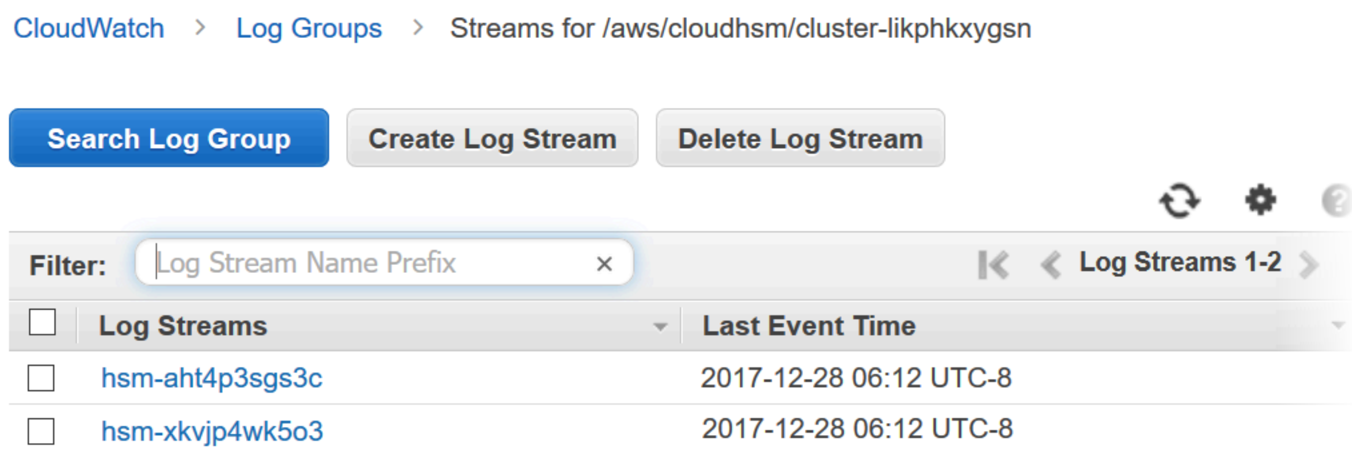
For more information about CloudWatch Logs concepts, see Concepts in the *Amazon CloudWatch Logs User Guide*.

You can view the audit logs for an HSM from the CloudWatch Logs page in the AWS Management Console, the CloudWatch Logs commands in the AWS CLI, the CloudWatch Logs PowerShell cmdlets, or the CloudWatch Logs SDKs. For instructions, see View Log Data in the *Amazon CloudWatch Logs User Guide*.

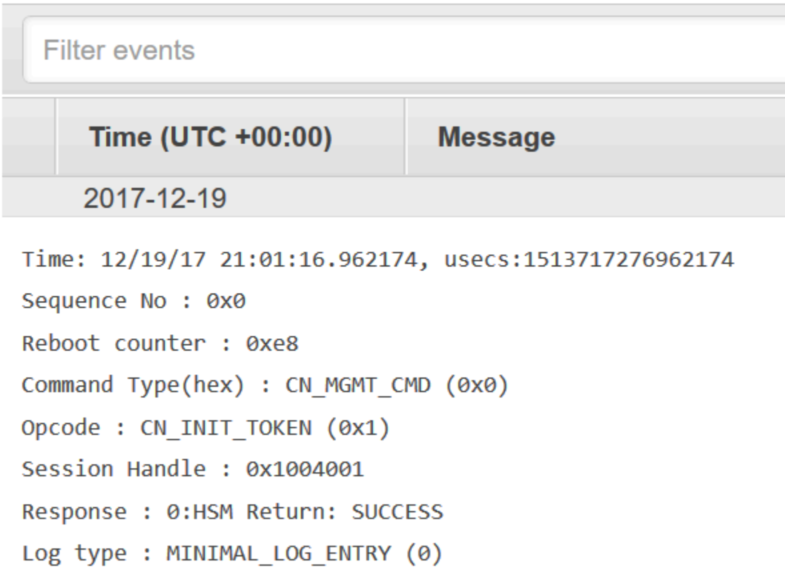
For example, the following image shows the log group for the `cluster-likphkxygsn` cluster in the AWS Management Console.



When you choose the cluster log group name, you can view the log stream for each of the HSMs in the cluster. The following image shows the log streams for the HSMs in the `cluster-likphkxygsn` cluster.

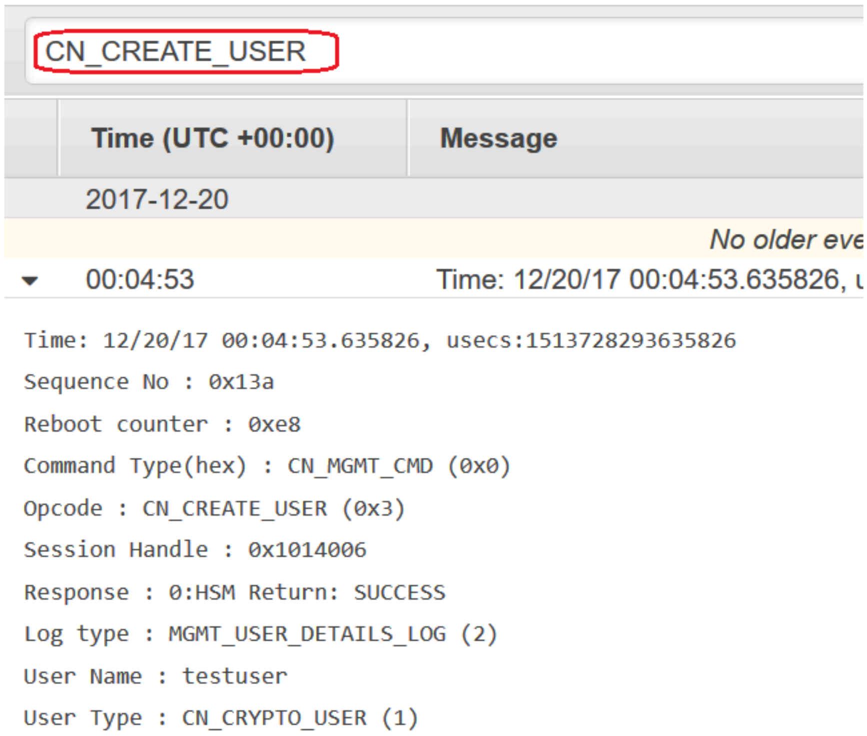


When you choose an HSM log stream name, you can view the events in the audit log. For example, this event, which has a sequence number of 0x0 and an `Opcode` of `CN_INIT_TOKEN`, is typically the first event for the first HSM in each cluster. It records the initialization of the HSM in the cluster.



You can use all the many features in CloudWatch Logs to manage your audit logs. For example, you can use the **Filter events** feature to find particular text in an event, such as the `CN_CREATE_USER` `Opcode`.

To find all events that do not include the specified text, add a minus sign (-) before the text. For example, to find events that do not include `CN_CREATE_USER`, enter **-CN_CREATE_USER**.



Interpreting AWS CloudHSM audit logs

The events in the HSM audit logs have standard fields. Some event types have additional fields that capture useful information about the event. For example, user login and user management events include the user name and user type of the user. Key management commands include the key handle.

Several of the fields provide particularly important information. The `Opcode` identifies the management command that is being recorded. The `Sequence No` identifies an event in the log stream and indicates the order in which it was recorded.

For example, the following example event is the second event (`Sequence No: 0x1`) in the log stream for an HSM. It shows the HSM generating a password encryption key, which is part of its startup routine.

```
Time: 12/19/17 21:01:17.140812, usecs:1513717277140812
Sequence No : 0x1
Reboot counter : 0xe8
```

```
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_GEN_PSWD_ENC_KEY (0x1d)
Session Handle : 0x1004001
Response : 0:HSM Return: SUCCESS
Log type : MINIMAL_LOG_ENTRY (0)
```

The following fields are common to every AWS CloudHSM event in the audit log.

Time

The time that the event occurred in the UTC time zone. The time is displayed as a human-readable time and Unix time in microseconds.

Reboot counter

A 32-bit persistent ordinal counter that is incremented when the HSM hardware is rebooted.

All events in a log stream have the same reboot counter value. However, the reboot counter might not be unique to a log stream, as it can differ across different HSM instances in the same cluster.

Sequence No

A 64-bit ordinal counter that is incremented for each log event. The first event in each log stream has a sequence number of 0x0. There should be no gaps in the `Sequence No` values. The sequence number is unique only within a log stream.

Command type

A hexadecimal value that represents the category of the command. Commands in the AWS CloudHSM log streams have a command type of `CN_MGMT_CMD` (0x0) or `CN_CERT_AUTH_CMD` (0x9).

Opcode

Identifies the management command that was executed. For a list of `Opcode` values in the AWS CloudHSM audit logs, see AWS CloudHSM audit log reference.

Session handle

Identifies the session in which the command was run and the event was logged.

Response

Records the response to the management command. You can search the `Response` field for `SUCCESS` and `ERROR` values.

Log type

Indicates the log type of the AWS CloudHSM log that recorded the command.

- `MINIMAL_LOG_ENTRY (0)`
- `MGMT_KEY_DETAILS_LOG (1)`
- `MGMT_USER_DETAILS_LOG (2)`
- `GENERIC_LOG`

Examples of audit log events

The events in a log stream record the history of the HSM from its creation to deletion. You can use the log to review the lifecycle of your HSMs and gain insight into its operation. When you interpret the events, note the `Opcode`, which indicates the management command or action, and the `Sequence No`, which indicates the order of events.

Topics

Example: Initialize the first HSM in a cluster

The audit log stream for the first HSM in each cluster differs significantly from the log streams of other HSMs in the cluster. The audit log for the first HSM in each cluster records its creation and initialization. The logs of additional HSMs in the cluster, which are generated from backups, begin with a login event.

> **Important**
>
> The following initialization entries will not appear in the CloudWatch logs of clusters initialized before the release of the CloudHSM audit logging feature (August 30, 2018). For more information, see Document History.

The following example events appear in the log stream for the first HSM in a cluster. The first event in the log — the one with `Sequence No 0x0` — represents the command to initialize the HSM (`CN_INIT_TOKEN`). The response indicates that the command was successful (`Response : 0: HSM Return: SUCCESS`).

```
Time: 12/19/17 21:01:16.962174, usecs:1513717276962174
Sequence No : 0x0
Reboot counter : 0xe8
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_INIT_TOKEN (0x1)
Session Handle : 0x1004001
Response : 0:HSM Return: SUCCESS
Log type : MINIMAL_LOG_ENTRY (0)
```

The second event in this example log stream (`Sequence No 0x1`) records the command to create the password encryption key that the HSM uses (`CN_GEN_PSWD_ENC_KEY`).

This is a typical startup sequence for the first HSM in each cluster. Because subsequent HSMs in the same cluster are clones of the first one, they use the same password encryption key.

```
Time: 12/19/17 21:01:17.140812, usecs:1513717277140812
Sequence No : 0x1
Reboot counter : 0xe8
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_GEN_PSWD_ENC_KEY (0x1d)
Session Handle : 0x1004001
Response : 0:HSM Return: SUCCESS
Log type : MINIMAL_LOG_ENTRY (0)
```

The third event in this example log stream (`Sequence No 0x2`) is the creation of the appliance user (AU), which is the AWS CloudHSM service. Events that involve HSM users include extra fields for the user name and user type.

```
Time: 12/19/17 21:01:17.174902, usecs:1513717277174902
Sequence No : 0x2
Reboot counter : 0xe8
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_CREATE_APPLIANCE_USER (0xfc)
Session Handle : 0x1004001
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : app_user
User Type : CN_APPLIANCE_USER (5)
```

The fourth event in this example log stream (`Sequence No 0x3`) records the `CN_INIT_DONE` event, which completes the initialization of the HSM.

```
Time: 12/19/17 21:01:17.298914, usecs:1513717277298914
Sequence No : 0x3
Reboot counter : 0xe8
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_INIT_DONE (0x95)
Session Handle : 0x1004001
Response : 0:HSM Return: SUCCESS
Log type : MINIMAL_LOG_ENTRY (0)
```

You can follow the remaining events in the startup sequence. These events might include several login and logout events, and the generation of the key encryption key (KEK). The following event records the command that changes the password of the [precrypto officer (PRECO)](). This command activates the cluster.

```
Time: 12/13/17 23:04:33.846554, usecs:1513206273846554
Sequence No: 0x1d
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_CHANGE_PSWD (0x9)
Session Handle: 0x2010003
Response: 0:HSM Return: SUCCESS
Log type: MGMT_USER_DETAILS_LOG (2)
User Name: admin
User Type: CN_CRYPTO_PRE_OFFICER (6)
```

Login and logout events

When interpreting your audit log, note events that record users logging and in and out of the HSM. These events help you to determine which user is responsible for management commands that appear in sequence between the login and logout commands.

For example, this log entry records a login by a crypto officer named `admin`. The sequence number, `0x0`, indicates that this is the first event in this log stream.

When a user logs into an HSM, the other HSMs in the cluster also record a login event for the user. You can find the corresponding login events in the log streams of other HSMs in the cluster shortly after the initial login event.

```
Time: 01/16/18 01:48:49.824999, usecs:1516067329824999
Sequence No : 0x0
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0x7014006
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : admin
User Type : CN_CRYPTO_OFFICER (2)
```

The following example event records the `admin` crypto officer logging out. The sequence number, `0x2`, indicates that this is the third event in the log stream.

If the logged in user closes the session without logging out, the log stream includes an `CN_APP_FINALIZE` or close session event (`CN_SESSION_CLOSE`), instead of a `CN_LOGOUT` event. Unlike the login event, this logout event typically is recorded only by the HSM that executes the command.

```
Time: 01/16/18 01:49:55.993404, usecs:1516067395993404
Sequence No : 0x2
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGOUT (0xe)
Session Handle : 0x7014000
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : admin
```

```
User Type : CN_CRYPTO_OFFICER (2)
```

If a login attempt fails because the user name is invalid, the HSM records a CN_LOGIN event with the user name and type provided in the login command. The response displays error message 157, which explains that the user name does not exist.

```
Time: 01/24/18 17:41:39.037255, usecs:1516815699037255
Sequence No : 0x4
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0xc008002
Response : 157:HSM Error: user isn't initialized or user with this name doesn't exist
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : ExampleUser
User Type : CN_CRYPTO_USER (1)
```

If a login attempt fails because the password is invalid, the HSM records a CN_LOGIN event with the user name and type provided in the login command. The response displays the error message with the RET_USER_LOGIN_FAILURE error code.

```
Time: 01/24/18 17:44:25.013218, usecs:1516815865013218
Sequence No : 0x5
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0xc008002
Response : 163:HSM Error: RET_USER_LOGIN_FAILURE
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : testuser
User Type : CN_CRYPTO_USER (1)
```

Example: Create and delete users

This example shows the log events that are recorded when a crypto officer (CO) creates and deletes users.

The first event records a CO, admin, logging into the HSM. The sequence number of 0x0 indicates that this is the first event in the log stream. The name and type of the user who logged in are included in the event.

```
Time: 01/16/18 01:48:49.824999, usecs:1516067329824999
Sequence No : 0x0
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0x7014006
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : admin
User Type : CN_CRYPTO_OFFICER (2)
```

The next event in the log stream (sequence `0x1`) records the CO creating a new crypto user (CU). The name and type of the new user are included in the event.

```
Time: 01/16/18 01:49:39.437708, usecs:1516067379437708
Sequence No : 0x1
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_CREATE_USER (0x3)
Session Handle : 0x7014006
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : bob
User Type : CN_CRYPTO_USER (1)
```

Then, the CO creates another crypto officer, `alice`. The sequence number indicates that this action followed the previous one with no intervening actions.

```
Time: 01/16/18 01:49:55.993404, usecs:1516067395993404
Sequence No : 0x2
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_CREATE_CO (0x4)
Session Handle : 0x7014007
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : alice
User Type : CN_CRYPTO_OFFICER (2)
```

Later, the CO named `admin` logs in and deletes the crypto officer named `alice`. The HSM records a `CN_DELETE_USER` event. The name and type of the deleted user are included in the event.

```
Time: 01/23/18 19:58:23.451420, usecs:1516737503451420
Sequence No : 0xb
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_DELETE_USER (0xa1)
Session Handle : 0x7014007
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : alice
User Type : CN_CRYPTO_OFFICER (2)
```

Example: Create and delete a key pair

This example shows the events that are recorded in an HSM audit log when you create and delete a key pair.

The following event records the crypto user (CU) named `crypto_user` logging in to the HSM.

```
Time: 12/13/17 23:09:04.648952, usecs:1513206544648952
Sequence No: 0x28
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_LOGIN (0xd)
Session Handle: 0x2014005
Response: 0:HSM Return: SUCCESS
Log type: MGMT_USER_DETAILS_LOG (2)
User Name: crypto_user
User Type: CN_CRYPTO_USER (1)
```

Next, the CU generates a key pair (`CN_GENERATE_KEY_PAIR`). The private key has key handle `131079`. The public key has key handle `131078`.

```
Time: 12/13/17 23:09:04.761594, usecs:1513206544761594
Sequence No: 0x29
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_GENERATE_KEY_PAIR (0x19)
Session Handle: 0x2014004
Response: 0:HSM Return: SUCCESS
Log type: MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle: 131079
```

```
Public Key Handle: 131078
```

The CU immediately deletes the key pair. A CN_DESTROY_OBJECT event records the deletion of the public key (131078).

```
Time: 12/13/17 23:09:04.813977, usecs:1513206544813977
Sequence No: 0x2a
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_DESTROY_OBJECT (0x11)
Session Handle: 0x2014004
Response: 0:HSM Return: SUCCESS
Log type: MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle: 131078
Public Key Handle: 0
```

Then, a second `CN_DESTROY_OBJECT` event records the deletion of the private key (`131079`).

```
Time: 12/13/17 23:09:04.815530, usecs:1513206544815530
Sequence No: 0x2b
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_DESTROY_OBJECT (0x11)
Session Handle: 0x2014004
Response: 0:HSM Return: SUCCESS
Log type: MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle: 131079
Public Key Handle: 0
```

Finally, the CU logs out.

```
Time: 12/13/17 23:09:04.817222, usecs:1513206544817222
Sequence No: 0x2c
Reboot counter: 0xe8
Command Type(hex): CN_MGMT_CMD (0x0)
Opcode: CN_LOGOUT (0xe)
Session Handle: 0x2014004
Response: 0:HSM Return: SUCCESS
Log type: MGMT_USER_DETAILS_LOG (2)
User Name: crypto_user
User Type: CN_CRYPTO_USER (1)
```

Example: Generate and synchronize a key

This example shows the effect of creating a key in a cluster with multiple HSMs. The key is generated on one HSM, extracted from the HSM as a masked object, and inserted in the other HSMs as a masked object.

> **Note**
>
> The client tools might fail to synchronize the key. Or the command might include the **min_srv** parameter, which synchronizes the key only to the specified number of HSMs. In either case, the AWS CloudHSM service synchronizes the key to the other HSMs in the cluster. Because the HSMs record only client-side management commands in their logs, the server-side synchronization is not recorded in the HSM log.

First consider the log stream of the HSM that receives and executes the commands. The log stream is named for HSM ID, `hsm-abcde123456`, but the HSM ID does not appear in the log events.

First, the `testuser` crypto user (CU) logs in to the `hsm-abcde123456` HSM.

```
Time: 01/24/18 00:39:23.172777, usecs:1516754363172777
Sequence No : 0x0
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0xc008002
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : testuser
User Type : CN_CRYPTO_USER (1)
```

The CU runs an exSymKey command to generate a symmetric key. The `hsm-abcde123456` HSM generates a symmetric key with a key handle of 262152. The HSM records a CN_GENERATE_KEY event in its log.

```
Time: 01/24/18 00:39:30.328334, usecs:1516754370328334
Sequence No : 0x1
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_GENERATE_KEY (0x17)
Session Handle : 0xc008004
```

```
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 262152
Public Key Handle : 0
```

The next event in the log stream for `hsm-abcde123456` records the first step in the key synchronization process. The new key (key handle `262152`) is extracted from the HSM as a masked object.

```
Time: 01/24/18 00:39:30.330956, usecs:1516754370330956
Sequence No : 0x2
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_EXTRACT_MASKED_OBJECT_USER (0xf0)
Session Handle : 0xc008004
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 262152
Public Key Handle : 0
```

Now consider the log stream for HSM `hsm-zyxwv987654`, another HSM in the same cluster. This log stream also includes a login event for the `testuser` CU. The time value shows that occurs shortly after the user logs in to the `hsm-abcde123456` HSM.

```
Time: 01/24/18 00:39:23.199740, usecs:1516754363199740
Sequence No : 0xd
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0x7004004
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : testuser
User Type : CN_CRYPTO_USER (1)
```

This log stream for this HSM does not have a `CN_GENERATE_KEY` event. But it does have an event that records synchronization of the key to this HSM. The `CN_INSERT_MASKED_OBJECT_USER` event records the receipt of key `262152` as a masked object. Now key `262152` exists on both HSMs in the cluster.

```
Time: 01/24/18 00:39:30.408950, usecs:1516754370408950
```

```
Sequence No : 0xe
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_INSERT_MASKED_OBJECT_USER (0xf1)
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 262152
Public Key Handle : 0
```

When the CU user logs out, this `CN_LOGOUT` event appears only in the log stream of the HSM that received the commands.

Example: Export a key

This example shows the audit log events that are recorded when a crypto user (CU) exports keys from a cluster with multiple HSMs.

The following event records the CU (`testuser`) logging into key_mgmt_util.

```
Time: 01/24/18 19:42:22.695884, usecs:1516822942695884
Sequence No : 0x26
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_LOGIN (0xd)
Session Handle : 0x7004004
Response : 0:HSM Return: SUCCESS
Log type : MGMT_USER_DETAILS_LOG (2)
User Name : testuser
User Type : CN_CRYPTO_USER (1)
```

The CU runs an exSymKey command to export key 7, a 256-bit AES key. The command uses key 6, a 256-bit AES key on the HSMs, as the wrapping key.

The HSM that receives the command records a `CN_WRAP_KEY` event for key 7, the key that is being exported.

```
Time: 01/24/18 19:51:12.860123, usecs:1516823472860123
Sequence No : 0x27
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_WRAP_KEY (0x1a)
```

```
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 7
Public Key Handle : 0
```

Then, the HSM records a `CN_NIST_AES_WRAP` event for the wrapping key, key 6. The key is wrapped and then immediately unwrapped, but the HSM records only one event.

```
Time: 01/24/18 19:51:12.905257, usecs:1516823472905257
Sequence No : 0x28
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_NIST_AES_WRAP (0x1e)
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 6
Public Key Handle : 0
```

The **exSymKey** command writes the exported key to a file but does not change the key on the HSM. Consequently, there are no corresponding events in the logs of other HSMs in the cluster.

Example: Import a key

This example shows the audit log events that are recorded when you import keys into the HSMs in a cluster. In this example, the crypto user (CU) uses the imSymKey command to import an AES key into the HSMs. The command uses key 6 as the wrapping key.

The HSM that receives the commands first records a `CN_NIST_AES_WRAP` event for key 6, the wrapping key.

```
Time: 01/24/18 19:58:23.170518, usecs:1516823903170518
Sequence No : 0x29
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_NIST_AES_WRAP (0x1e)
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 6
Public Key Handle : 0
```

Then, the HSM records a CN_UNWRAP_KEY event that represents the import operation. The imported key is assigned a key handle of 11.

```
Time: 01/24/18 19:58:23.200711, usecs:1516823903200711
Sequence No : 0x2a
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_UNWRAP_KEY (0x1b)
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 11
Public Key Handle : 0
```

When a new key is generated or imported, the client tools automatically attempt to synchronize the new key to other HSMs in the cluster. In this case, the HSM records a CN_EXTRACT_MASKED_OBJECT_USER event when key 11 is extracted from the HSM as a masked object.

```
Time: 01/24/18 19:58:23.203350, usecs:1516823903203350
Sequence No : 0x2b
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_EXTRACT_MASKED_OBJECT_USER (0xf0)
Session Handle : 0x7004003
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 11
Public Key Handle : 0
```

The log streams of other HSMs in the cluster reflect the arrival of the newly imported key.

For example, this event was recorded in the log stream of a different HSM in the same cluster. This CN_INSERT_MASKED_OBJECT_USER event records the arrival of a masked object that represents key 11.

```
Time: 01/24/18 19:58:23.286793, usecs:1516823903286793
Sequence No : 0xb
Reboot counter : 0x107
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_INSERT_MASKED_OBJECT_USER (0xf1)
Session Handle : 0xc008004
```

```
Response : 0:HSM Return: SUCCESS
Log type : MGMT_KEY_DETAILS_LOG (1)
Priv/Secret Key Handle : 11
Public Key Handle : 0
```

Example: Share and unshare a key

This example shows the audit log event that is recorded when a crypto user (CU) shares or unshares ECC private key with other crypto users. The CU uses the shareKey command and provides the key handle, the user ID, and the value 1 to share or value 0 to unshare the key.

In the following example, the HSM that receives the command, records a CM_SHARE_OBJECT event that represents the share operation.

```
Time: 02/08/19 19:35:39.480168, usecs:1549654539480168
Sequence No : 0x3f
Reboot counter : 0x38
Command Type(hex) : CN_MGMT_CMD (0x0)
Opcode : CN_SHARE_OBJECT (0x12)
Session Handle : 0x3014007
Response : 0:HSM Return: SUCCESS
Log type : UNKNOWN_LOG_TYPE (5)
```

AWS CloudHSM audit log reference

AWS CloudHSM records HSM management commands in audit log events. Each event has an operation code (Opcode) value that identifies the action that occurred and its response. You can use the Opcode values to search, sort, and filter the logs.

The following table defines the Opcode values in an AWS CloudHSM audit log.

Operation Code (Opcode)	Description
User Login: These events include the user name and user type	
CN_LOGIN (0xd)	User login
CN_LOGOUT (0xe)	User logout

Operation Code (Opcode)	Description
CN_APP_FINALIZE	The connection with the HSM was closed. Any session keys or quorum tokens from this connection were deleted.
CN_CLOSE_SESSION	The session with the HSM was closed. Any session keys or quorum tokens from this session were deleted.
User Management: These events include the user name and user type	
CN_CREATE_USER (0x3)	Create a crypto user (CU)
CN_CREATE_CO	Create a crypto officer (CO)
CN_DELETE_USER	Delete a user
CN_CHANGE_PSWD	Change a user password
CN_SET_M_VALUE	Set quorum authentication (M of N) for a user action
CN_APPROVE_TOKEN	Approve a quorum authentication token for a user action
CN_DELETE_TOKEN	Delete one or more quorum tokens
CN_GET_TOKEN	Request a signing token to initiate a quorum operation
Key Management: These events include the key handle	
CN_GENERATE_KEY	Generate a symmetric key
CN_GENERATE_KEY_PAIR (0x19)	Generate an asymmetric key pair
CN_CREATE_OBJECT	Import a public key (without wrapping)
CN_MODIFY_OBJECT	Set a key attribute

Operation Code (Opcode)	Description
CN_DESTROY_OBJECT (0x11)	Deletion of a session key
CN_TOMBSTONE_OBJECT	Deletion of a token key
CN_SHARE_OBJECT	Share or unshare a key
CN_WRAP_KEY	Export an encrypted copy of a key (wrapKey)
CN_UNWRAP_KEY	Import an encrypted copy of a key (unwrapKey)
CN_DERIVE_KEY	Derive a symmetric key from an existing key
CN_NIST_AES_WRAP	Encrypt or decrypt a key with an AES key
CN_INSERT_MASKED_OBJECT_USER	Insert an encrypted key with attributes from another HSM in the cluster.
CN_EXTRACT_MASKED_OBJECT_USER	Wraps/encrypts a key with attributes from the HSM to be sent to another HSM in the cluster.
Back up HSMs	
CN_BACKUP_BEGIN	Begin the backup process
CN_BACKUP_END	Completed the backup process
CN_RESTORE_BEGIN	Begin restoring from a backup
CN_RESTORE_END	Completed the restoration process from a backup
Certificate-Based Authentication	
CN_CERT_AUTH_STORE_CERT	Stores the cluster certificate
HSM Instance Commands	
CN_INIT_TOKEN (0x1)	Start the HSM initialization process

Operation Code (Opcode)	Description
`CN_INIT_DONE`	The HSM initialization process has finished
`CN_GEN_KEY_ENC_KEY`	Generate a key encryption key (KEK)
`CN_GEN_PSWD_ENC_KEY (0x1d)`	Generate a password encryption key (PEK)
HSM crypto commands	
`CN_FIPS_RAND`	Generate a FIPS-compliant random number

Getting CloudWatch metrics for AWS CloudHSM

Use CloudWatch to monitor your AWS CloudHSM cluster in real time. The metrics can be grouped by region, cluster ID, or cluster ID and HSM ID.

The `AWS/CloudHSM` namespace includes the following metrics:

Metric	Description
HsmUnhealthy	The HSM instance is not performing properly. AWS CloudHSM automatically replaces unhealthy instances for you. You may choose to proactively expand cluster size to reduce performance impact while we are replacing the HSM.
HsmTemperature [1]	The junction temperature of the hardware processor. The system shuts down if temperature reaches 110 degrees Centigrade.
HsmKeysSessionOccupied	The number of session keys being used by the HSM instance.
HsmKeysTokenOccupied	The number of token keys being used by the HSM instance and the cluster.
HsmSslCtxsOccupied [1]	The number of end-to-end encrypted channels currently established for the HSM instance. Up to 2,048 channels are allowed.
HsmSessionCount	The number of open connections to the HSM instance. Up to 2,048 are allowed. By default, the client daemon is configured to open two sessions

Metric	Description
	with each HSM instance under one end-to-end encrypted channel. AWS CloudHSM may also have up to 2 connections open with the HSM to monitor the health of the HSMs.
HsmUsersAvailable	The number of additional users that can be created. This equals the maximum number of users (listed in **HsmUsersMax**) minus the users created to date.
HsmUsersMax [1]	The maximum number of users that can be created on the HSM instance.
InterfaceEth2OctetsInput [1]	The cumulative sum of incoming traffic to the HSM to date.
InterfaceEth2OctetsOutput [1]	The cumulative sum of outgoing traffic to the HSM to date.

- [1] This metric is not available for hsm2m.medium.

AWS CloudHSM performance information

For production AWS CloudHSM clusters, you should have at least two hardware security module (HSM) instances spread across different availability zones in a region. We recommend load testing your cluster to determine the peak load you should anticipate, and then add one more HSM to it to ensure high availability. For applications requiring durability of newly generated keys, we recommend at least three HSM instances spread across different availability zones in a region.

Performance data

The performance of AWS CloudHSM clusters vary based on specific workload. To increase performance, you can add additional HSM instances to your clusters. Performance can vary based on configuration, data size, and additional application load on your EC2 instances. We encourage load testing your application to determine scaling needs.

The following table shows approximate performance for common cryptographic algorithms running on an EC2 instance with hsm1.medium instances.

Performance data for hsm1.medium

Operation	Two-HSM cluster[1]	Three-HSM cluster[2]	Six-HSM cluster[3]
RSA 2048-bit sign	2,000 ops/sec	3,000 ops/sec	5,000 ops/sec
EC P256 sign	500 ops/sec	750 ops/sec	1,500 ops/sec

The following table shows approximate performance for common cryptographic algorithms running on an EC2 instance with hsm2m.medium.

Performance data for hsm2m.medium

Operation	Two-HSM cluster[1]	Three-HSM cluster[2]	Six-HSM cluster[3]
RSA 2048-bit sign	2000 ops/sec	3000 ops/sec	5000 ops/sec
EC P256 sign	3000 ops/sec	4500 ops/sec	7000 ops/sec

- [1] A two-HSM cluster with the Java multi-threaded application running on one c4.large EC2 instance with one HSM in the same AZ as the EC2 instance.
- [2] A three-HSM cluster with the Java multi-threaded application running on one c4.large EC2 instance with one HSM in the same AZ as the EC2 instance.
- [3] A six-HSM cluster with the Java multi-threaded application running on one c4.large EC2 instance with two HSMs in the same AZ as the EC2 instance.

HSM throttling

When your workload exceeds your cluster's HSM capacity, you will receive error messages stating HSMs are busy or throttled. For details on what to do when this happens, see HSM throttling

Security in AWS CloudHSM

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS CloudHSM, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS CloudHSM. The following topics show you how to configure AWS CloudHSM to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS CloudHSM resources.

Contents

Control API access with IAM policies

Upgrade IAM policies to IPv6

AWS CloudHSM customers use IAM policies to control access to AWS CloudHSM APIs and prevent any IP addresses outside the configured range from being able to access AWS CloudHSM APIs.

The *cloudhsmv2.`<region>`.api.aws* dual-stack endpoint where AWS CloudHSM APIs are hosted supports IPv6 in addition to IPv4.

Customers who need to support both IPv4 and IPv6 must update their IP address filtering policies to handle IPv6 addresses, otherwise it will impact their ability to connect to AWS CloudHSM over IPv6.

Who should upgrade?

Customers who use dual addressing with policies containing *aws:sourceIp* are impacted by this upgrade. *Dual addressing* means that the network supports both IPv4 and IPv6.

If you are using dual addressing, you must update your IAM policies that are currently configured with IPv4 format addresses to include IPv6 format addresses.

For help with access issues, contact Support.

> **Note**
>
> The following customers are *not* impacted by this upgrade:
>
> - Customers who are on *only* IPv4 networks.

What is IPv6?

IPv6 is the next generation IP standard intended to eventually replace IPv4. The previous version, IPv4, uses a 32-bit addressing scheme to support 4.3 billion devices. IPv6 instead uses 128-bit addressing to support approximately 340 trillion trillion trillion (or 2 to the 128th power) devices.

For more details, refer VPC IPv6 webpage.

```
2001:cdba:0000:0000:0000:0000:3257:9652
```

```
2001:cdba:0:0:0:0:3257:9652
2001:cdba::3257:965
```

Updating an IAM policy for IPv6

IAM policies are currently used to set an allowed range of IP addresses using the `aws:SourceIp` filter.

Dual addressing supports both IPv4 and IPv6 traffic. If your network uses dual addressing, you must update any IAM polices used for IP address filtering to include IPv6 address ranges.

For example, below policy identifies allowed IPv4 address ranges `192.0.2.0.*` and `203.0.113.0.*` in the `Condition` element.

```
# https://docs.aws.amazon.com/IAM/latest/UserGuide/
reference_policies_examples_aws_deny-ip.html
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Deny",
        "Action": "*",
        "Resource": "*",
        "Condition": {
            "NotIpAddress": {
                "*aws:SourceIp*": [
                    "*192.0.2.0/24*",
                    "*203.0.113.0/24*"
                ]
            },
            "Bool": {
                "aws:ViaAWSService": "false"
            }
        }
    }
}
```

To update this policy, change the `Condition` element to include the IPv6 address ranges `2001:DB8:1234:5678::/64` and `2001:cdba:3257:8593::/64`.

Note

DO NOT REMOVE the existing IPv4 addresses because they are needed for backward compatibility.

```
"Condition": {
                "NotIpAddress": {
                    "*aws:SourceIp*": [
                        "*192.0.2.0/24*", <<DO NOT REMOVE existing IPv4 address>>
                        "*203.0.113.0/24*", <<DO NOT REMOVE existing IPv4 address>>
                        "*2001:DB8:1234:5678::/64*", <<New IPv6 IP address>>
                        "*2001:cdba:3257:8593::/64*" <<New IPv6 IP address>>
                    ]
                },
                "Bool": {
                    "aws:ViaAWSService": "false"
                }
            }
```

Verify your client supports IPv6

Customers using the *cloudhsmv2.{region}.api.aws* endpoint are advised to verify if they are able to connect to it. The following steps describe how to perform the verification.

This examples uses Linux and curl version 8.6.0 and uses the AWS CloudHSM service endpoints which has IPv6 enabled endpoints located at the *api.aws endpoint*.

Note

Switch the AWS Region to the same Region where the client is located. In this example, we use the US East (N. Virginia) – `us-east-1` endpoint.

1. Determine if the endpoint resolves with an IPv6 address using the following `dig` command.

   ```
   dig +short AAAA cloudhsmv2.us-east-1.api.aws
   2600:1f18:e2f:4e05:1a8a:948e:7c08:c1c3
   ```

2. Determine if the client network can make an IPv6 connection using the following `curl` command. A 404 response code means the connection succeeded, while a 0 response code means the connection failed.

```
curl --ipv6 -o /dev/null --silent -w "\nremote ip: %{remote_ip}\nresponse code:
 %{response_code}\n" https://cloudhsmv2.us-east-1.api.aws

remote ip: 2600:1f18:e2f:4e05:1a8a:948e:7c08:c1c3
response code: 404
```

If a remote IP was identified **and** the response code is not `0`, a network connection was successfully made to the endpoint using IPv6. The remote IP should be an IPv6 address because the operating system should select the protocol that is valid for the client. If the remote IP is not an IPv6 address, use the following command to force `curl` to use IPv4.

```
curl --ipv4 -o /dev/null --silent -w "\nremote ip: %{remote_ip}\nresponse code:
 %{response_code}\n" https://cloudhsmv2.us-east-1.api.aws

remote ip: 3.123.154.250
response code: 404
```

If the remote IP is blank or the response code is `0`, the client network or the network path to the endpoint is IPv4-only. You can verify this configuration with the following `curl` command.

```
curl -o /dev/null --silent -w "\nremote ip: %{remote_ip}\nresponse code:
 %{response_code}\n" https://cloudhsmv2.us-east-1.api.aws

remote ip: 3.123.154.250
response code: 404
```

Data protection in AWS CloudHSM

The AWS shared responsibility model applies to data protection in AWS CloudHSM. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ.

For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS CloudHSM or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Encryption at rest

When AWS CloudHSM makes a backup from an HSM, the HSM encrypts its data before sending it to AWS CloudHSM. The data is encrypted using a unique, ephemeral encryption key. For more information, see AWS CloudHSM cluster backups.

Encryption in transit

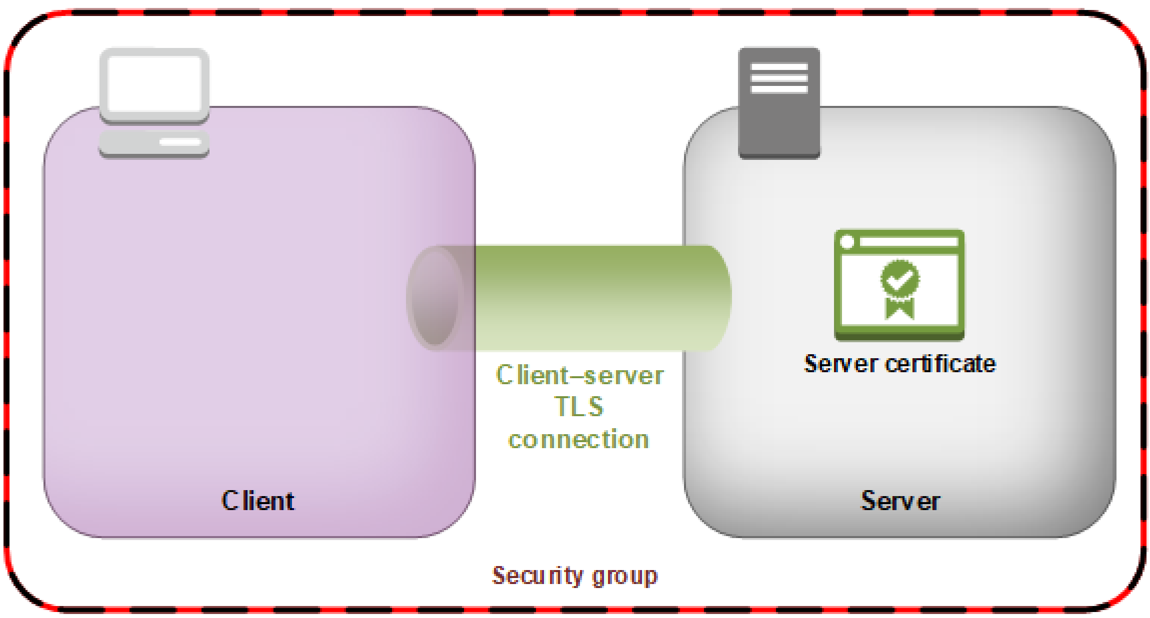
Communication between the AWS CloudHSM client and the HSM in your cluster is encrypted from end to end. This communication can be decrypted only by your client and your HSM. For more information, see End-to-end encryption.

AWS CloudHSM client end-to-end encryption

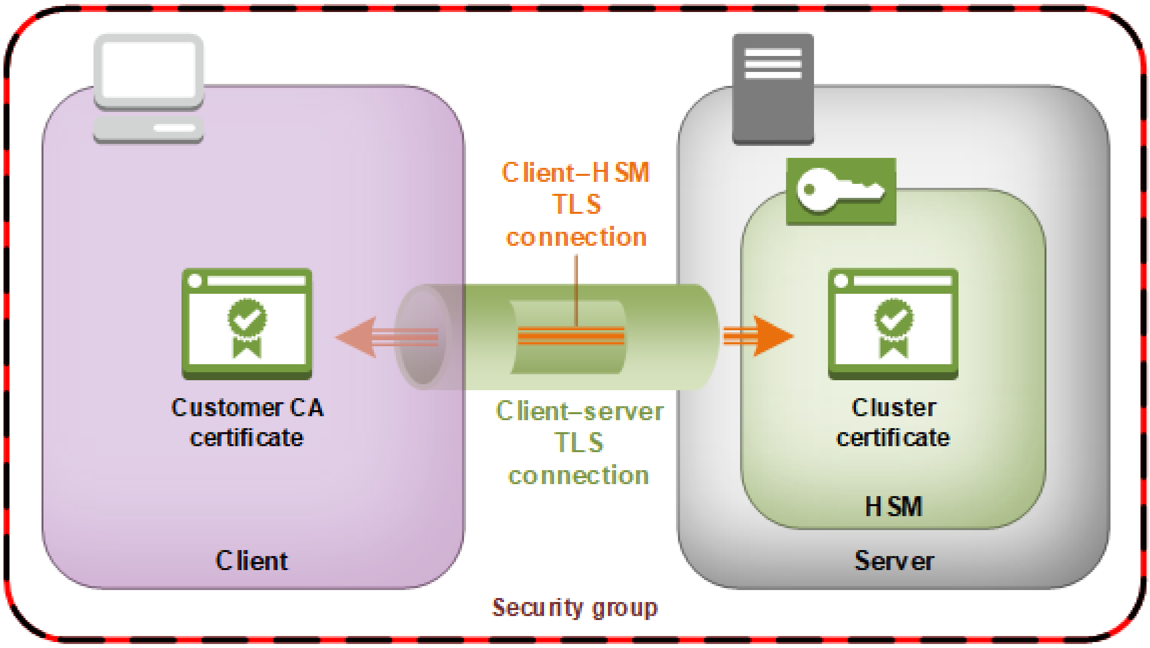
Communication between the client instance and the HSMs in your cluster is encrypted from end to end. Only your client and your HSMs can decrypt the communication.

The following process explains how the client establishes end-to-end encrypted communication with an HSM.

1. Your client establishes a Transport Layer Security (TLS) connection with the server that hosts your HSM hardware. Your cluster's security group allows inbound traffic to the server only from client instances in the security group. The client also checks the server's certificate to ensure that it's a trusted server.



2. Next, the client establishes an encrypted connection with the HSM hardware. The HSM has the cluster certificate that you signed with your own certificate authority (CA), and the client has the CA's root certificate. Before the client–HSM encrypted connection is established, the client verifies the HSM's cluster certificate against its root certificate. The connection is established only when the client successfully verifies that the HSM is trusted.



Security of cluster backups

When AWS CloudHSM makes a backup from the HSM, the HSM encrypts all of its data before sending it to AWS CloudHSM. The data never leaves the HSM in plaintext form. Additionally, backups cannot be decrypted by AWS because AWS doesn't have access to key used to decrypt the backups.

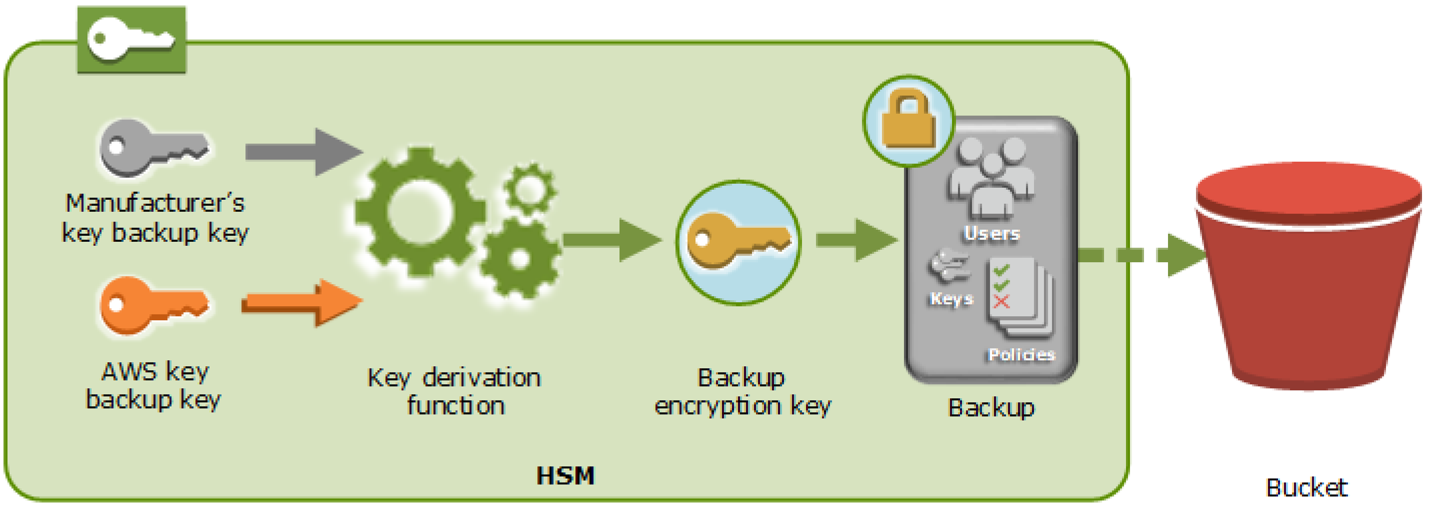
To encrypt its data, the HSM uses a unique, ephemeral encryption key known as the ephemeral backup key (EBK). The EBK is an AES 256-bit encryption key generated inside the HSM when AWS CloudHSM makes a backup. The HSM generates the EBK, then uses it to encrypt the HSM's data with a FIPS-approved AES key wrapping method that complies with [NIST special publication 800-38F](). Then the HSM gives the encrypted data to AWS CloudHSM. The encrypted data includes an encrypted copy of the EBK.

To encrypt the EBK, the HSM uses another encryption key known as the persistent backup key (PBK). The PBK is also an AES 256-bit encryption key. To generate the PBK, the HSM uses a FIPS-approved key derivation function (KDF) in counter mode that complies with [NIST special publication 800-108](). The inputs to this KDF include the following:

- A manufacturer key backup key (MKBK), permanently embedded in the HSM hardware by the hardware manufacturer.

- An AWS key backup key (AKBK), securely installed in the HSM when it's initially configured by AWS CloudHSM.

The encryption processes are summarized in the following figure. The backup encryption key represents the persistent backup key (PBK) and the ephemeral backup key (EBK).



AWS CloudHSM can restore backups onto only AWS-owned HSMs made by the same manufacturer. Because each backup contains all users, keys, and configuration from the original HSM, the restored HSM contains the same protections and access controls as the original. The restored data overwrites all other data that might have been on the HSM prior to restoration.

A backup consists of only encrypted data. Before the service stores a backup in Amazon S3, the service encrypts the backup again using AWS Key Management Service (AWS KMS).

Identity and access management for AWS CloudHSM

AWS uses security credentials to identify you and to grant you access to your AWS resources. You can use features of AWS Identity and Access Management (IAM) to allow other users, services, and applications to use your AWS resources fully or in a limited way. You can do this without sharing your security credentials.

By default, IAM users don't have permission to create, view, or modify AWS resources. To allow an IAM user to access resources such as a load balancer, and to perform tasks, you:

1. Create an IAM policy that grants the IAM user permission to use the specific resources and API actions they need.
2. Attach the policy to the IAM user or the group that the IAM user belongs to.

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources.

For example, you can use IAM to create users and groups under your AWS account. An IAM user can be a person, a system, or an application. Then you grant permissions to the users and groups to perform specific actions on the specified resources using an IAM policy.

Grant permissions using IAM policies

When you attach a policy to a user or group of users, it allows or denies the users permission to perform the specified tasks on the specified resources.

An IAM policy is a JSON document that consists of one or more statements. Each statement is structured as shown in the following example.

```
{
  "Version": "2012-10-17",
  "Statement":[{
    "Effect": "<effect>",
    "Action": "<action>",
    "Resource": "<resource-arn>",
    "Condition": {
      "<condition>": {
        "<key>":"<value>"
      }
    }
  }]
}
```

- **Effect**— The *effect* can be `Allow` or `Deny`. By default, IAM users don't have permission to use resources and API actions, so all requests are denied. An explicit allow overrides the default. An explicit deny overrides any allows.
- **Action**— The *action* is the specific API action for which you are granting or denying permission. For more information about specifying *action*, see API actions for AWS CloudHSM.
- **Resource**— The resource that's affected by the action. AWS CloudHSM does not support resource-level permissions. You must use the * wildcard to specify all AWS CloudHSM resources.
- **Condition**— You can optionally use conditions to control when your policy is in effect. For more information, see Condition keys for AWS CloudHSM.

For more information, see the IAM User Guide.

API actions for AWS CloudHSM

In the **Action** element of your IAM policy statement, you can specify any API action that AWS CloudHSM offers. You must prefix the action name with the lowercase string `cloudhsm:`, as shown in the following example.

```
"Action": "cloudhsm:DescribeClusters"
```

To specify multiple actions in a single statement, enclose them in square brackets and separate them with a comma, as shown in the following example.

```
"Action": [
    "cloudhsm:DescribeClusters",
    "cloudhsm:DescribeHsm"
]
```

You can also specify multiple actions using the * wildcard. The following example specifies all API action names for AWS CloudHSM that start with `List`.

```
"Action": "cloudhsm:List*"
```

To specify all API actions for AWS CloudHSM, use the * wildcard, as shown in the following example.

```
"Action": "cloudhsm:*"
```

For the list of API actions for AWS CloudHSM, see AWS CloudHSM Actions.

Condition keys for AWS CloudHSM

When you create a policy, you can specify the conditions that control when the policy is in effect. Each condition contains one or more key-value pairs. There are global condition keys and service-specific condition keys.

AWS CloudHSM has no service-specific context keys.

For more information about global condition keys, see AWS Global Condition Context Keys in the *IAM User Guide*.

Predefined AWS managed policies for AWS CloudHSM

The managed policies created by AWS grant the required permissions for common use cases. You can attach these policies to your IAM users, based on the access to AWS CloudHSM that they require:

- **AWSCloudHSMFullAccess** — Grants full access required to use AWS CloudHSM features.
- **AWSCloudHSMReadOnlyAccess** — Grants read-only access to AWS CloudHSM features.

Customer managed policies for AWS CloudHSM

We recommend that you create an IAM administrators group for AWS CloudHSM that contains only the permissions required to run AWS CloudHSM. Attach the policy with the appropriate permissions to this group. Add IAM users to the group as needed. Each user that you add inherits the policy from the administrators group.

Also, we recommend that you create additional user groups based on the permissions that your users need. This ensures that only trusted users have access to critical API actions. For example, you could create a user group that you use to grant read-only access to clusters and HSMs. Because this group does not allow a user to delete clusters or HSMs, an untrusted user cannot affect the availability of a production workload.

As new AWS CloudHSM management features are added over time, you can ensure that only trusted users are given immediate access. By assigning limited permissions to policies at creation, you can manually assign new feature permissions to them later.

The following are example policies for AWS CloudHSM. For information about how to create a policy and attach it to an IAM user group, see Creating Policies on the JSON Tab in the *IAM User Guide*.

Examples

- Read Only Permissions
- Power User Permissions
- Admin Permissions

Example Example: Read-only permissions

This policy allows access to the `DescribeClusters` and `DescribeBackups` API actions. It also includes additional permissions for specific Amazon EC2 API actions. It does not allow the user to delete clusters or HSMs.

```
{
   "Version": "2012-10-17",
   "Statement": {
      "Effect": "Allow",
      "Action": [
         "cloudhsm:DescribeClusters",
         "cloudhsm:DescribeBackups",
         "cloudhsm:ListTags"
      ],
      "Resource": "*"
   }
}
```

Example Example: Power user permissions

This policy allows access to a subset of the AWS CloudHSM API actions. It also includes additional permissions for specific Amazon EC2 actions. It does not allow the user to delete clusters or HSMs. You must include the `iam:CreateServiceLinkedRole` action to allow AWS CloudHSM to automatically create the **AWSServiceRoleForCloudHSM** service-linked role in your account. This role allows AWS CloudHSM to log events. For more information, see Service-linked roles for AWS CloudHSM.

> **Note**
>
> To see the specific permissions for each API, refer to Actions, resources, and condition keys for AWS CloudHSM in the *Service Authorization Reference*.

```
{
   "Version": "2012-10-17",
   "Statement": {
      "Effect": "Allow",
      "Action": [
         "cloudhsm:DescribeClusters",
```

```
            "cloudhsm:DescribeBackups",
            "cloudhsm:CreateCluster",
            "cloudhsm:CreateHsm",
            "cloudhsm:RestoreBackup",
            "cloudhsm:CopyBackupToRegion",
            "cloudhsm:InitializeCluster",
            "cloudhsm:ListTags",
            "cloudhsm:TagResource",
            "cloudhsm:UntagResource",
            "ec2:CreateNetworkInterface",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeNetworkInterfaceAttribute",
            "ec2:DetachNetworkInterface",
            "ec2:DeleteNetworkInterface",
            "ec2:CreateSecurityGroup",
            "ec2:AuthorizeSecurityGroupIngress",
            "ec2:AuthorizeSecurityGroupEgress",
            "ec2:RevokeSecurityGroupEgress",
            "ec2:DescribeSecurityGroups",
            "ec2:DeleteSecurityGroup",
            "ec2:CreateTags",
            "ec2:DescribeVpcs",
            "ec2:DescribeSubnets",
            "iam:CreateServiceLinkedRole"
         ],
         "Resource": "*"
      }
}
```

Example Example: Admin permissions

This policy allows access to all AWS CloudHSM API actions, including the actions to delete HSMs and clusters. It also includes additional permissions for specific Amazon EC2 actions. You must include the `iam:CreateServiceLinkedRole` action to allow AWS CloudHSM to automatically create the **AWSServiceRoleForCloudHSM** service-linked role in your account. This role allows AWS CloudHSM to log events. For more information, see Service-linked roles for AWS CloudHSM.

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":[
         "cloudhsm:*",
```

```
            "ec2:CreateNetworkInterface",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeNetworkInterfaceAttribute",
            "ec2:DetachNetworkInterface",
            "ec2:DeleteNetworkInterface",
            "ec2:CreateSecurityGroup",
            "ec2:AuthorizeSecurityGroupIngress",
            "ec2:AuthorizeSecurityGroupEgress",
            "ec2:RevokeSecurityGroupEgress",
            "ec2:DescribeSecurityGroups",
            "ec2:DeleteSecurityGroup",
            "ec2:CreateTags",
            "ec2:DescribeVpcs",
            "ec2:DescribeSubnets",
            "iam:CreateServiceLinkedRole"
         ],
         "Resource":"*"
      }
}
```

Service-linked roles for AWS CloudHSM

The IAM policy that you created previously to Customer managed policies for AWS CloudHSM includes the `iam:CreateServiceLinkedRole` action. AWS CloudHSM defines a service-linked role named **AWSServiceRoleForCloudHSM**. The role is predefined by AWS CloudHSM and includes permissions that AWS CloudHSM requires to call other AWS services on your behalf. The role makes setting up your service easier because you don't need to manually add the role policy and trust policy permissions.

The role policy allows AWS CloudHSM to create Amazon CloudWatch Logs log groups and log streams and write log events on your behalf. You can view it below and in the IAM console.

```
{
    "Version": "2018-06-12",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
```

```
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:*:*:*"
            ]
        }
    ]
}
```

The trust policy for the **AWSServiceRoleForCloudHSM** role allows AWS CloudHSM to assume the role.

```
{
  "Version": "2018-06-12",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudhsm.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Creating a service-linked role (automatic)

AWS CloudHSM creates the **AWSServiceRoleForCloudHSM** role when you create a cluster if you include the `iam:CreateServiceLinkedRole` action in the permissions that you defined when you created the AWS CloudHSM administrators group. See Customer managed policies for AWS CloudHSM.

If you already have one or more clusters and just want to add the **AWSServiceRoleForCloudHSM** role, you can use the console, the create-cluster command, or the CreateCluster API operation to create a cluster. Then use the console, the delete-cluster command, or the DeleteCluster API operation to delete it. Creating the new cluster creates the service-linked role and applies it to all clusters in your account. Alternatively, you can create the role manually. See the following section for more information.

> **Note**
>
> You do not need to perform all of the steps outlined in Getting started with AWS CloudHSM to create a cluster if you are only creating it to add the **AWSServiceRoleForCloudHSM** role.

Creating a service-linked role (manual)

You can use the IAM console, AWS CLI, or API to create the **AWSServiceRoleForCloudHSM** role. For more information, see Creating a Service-Linked Role in the *IAM User Guide*.

Editing the service-linked role

AWS CloudHSM does not allow you to edit the **AWSServiceRoleForCloudHSM** role. After the role is created, for example, you cannot change its name because various entities might reference the role by name. Also, you cannot change the role policy. You can, however, use IAM to edit the role description. For more information, see Editing a Service–Linked Role in the *IAM User Guide*.

Deleting the service-linked role

You cannot delete a service-linked role as long as a cluster to which it has been applied still exists. To delete the role, you must first delete each HSM in your cluster and then delete the cluster. Every cluster in your account must be deleted. You can then use the IAM console, AWS CLI, or API to delete the role. For more information about deleting a cluster, see Deleting an AWS CloudHSM cluster. For more information, see Deleting a Service-Linked Role in the *IAM User Guide*.

Compliance

For clusters in FIPS mode, AWS CloudHSM provides FIPS-approved HSMs that meet PCI-PIN, PCI-3DS, and SOC2 compliance requirements. AWS CloudHSM also gives customers the option of choosing clusters that are non-FIPS mode. For details on what certification and compliance requirements apply to each, see AWS CloudHSM cluster modes.

Relying on a FIPS-validated HSM can help you meet corporate, contractual, and regulatory compliance requirements for data security in the AWS Cloud.

FIPS 140-2 Compliance

The Federal Information Processing Standard (FIPS) Publication 140-2 is a US government security standard that specifies security requirements for cryptographic modules that protect sensitive information. The type hsm1.medium HSMs provided by AWS CloudHSM are FIPS 140-2 level 3 certified (Certificate #4218). For more information, refer to FIPS validation for hardware.

FIPS 140-3 Compliance

The Federal Information Processing Standard (FIPS) Publication 140-3 is a US government security standard that specifies security requirements for cryptographic modules that protect sensitive information. The type hsm2m.medium HSMs provided by AWS CloudHSM are FIPS 140-3 level 3 certified (Certificate #4703). For more information, refer to FIPS validation for hardware.

PCI DSS Compliance

The Payment Card Industry Data Security Standard (PCI DSS) is a proprietary information security standard administered by the PCI Security Standards Council. The HSMs provided by AWS CloudHSM comply with PCI DSS.

PCI PIN Compliance

PCI PIN provides security requirement and assessment standards for transmitting, processing, and managing personal identification number (PIN) data, information that is used for transactions at ATMs and point-of-sale (POS) terminals. The hsm1.medium and hsm2m.medium HSMs that are provided by AWS CloudHSM are both PCI PIN compliant. For more information, refer to the article AWS CloudHSM is now PCI PIN certified.

PCI-3DS Compliance

PCI 3DS (or Three Domain Secure, 3-D Secure) provides security of data for EMV 3D secure e-commerce payments. PCI 3DS provides another layer of security for online shopping. The type hsm1.medium HSMs provided by AWS CloudHSM are PCI-3DS compliant.

SOC2

SOC2 is a framework to help service organizations demonstrate their cloud and data center security controls. AWS CloudHSM has implemented SOC2 controls in critical areas to adhere to the trusted service principles. For further information, refer to The AWS SOC FAQs page.

AWS CloudHSM PCI-PIN compliance FAQs

PCI PIN provides security requirement and assessment standards for transmitting, processing, and managing personal identification number (PIN) data, information that is used for transactions at ATMs and point-of-sale (POS) terminals.

The PCI-PIN Attestation of Compliance (AOC) and Responsibility Summary is available to customers through AWS Artifact, a self-service portal for on-demand access to AWS compliance reports. For more information, sign in to AWS Artifact in the AWS Management Console, or learn more at Getting Started with AWS Artifact.

FAQs

Q: What is the Attestation of Compliance and Responsibility Summary?

Attestation Of Compliance (AOC) is produced by a Qualified PIN Assessor (QPA) attesting AWS CloudHSM meets the applicable controls in the PCI-PIN standard. The responsibility summary matrix describes the controls which are the respective responsibilities of AWS CloudHSM and its customers.

Q: How do I obtain the AWS CloudHSM Attestation of Compliance?

The PCI-PIN Attestation of Compliance (AOC) is available to customers through AWS Artifact, a self-service portal for on-demand access to AWS compliance reports. For more information, sign in to AWS Artifact in the AWS Management Console, or learn more at Getting Started with AWS Artifact.

Q: How can I learn which PCI PIN controls I am responsible for?

For detailed information please see "AWS CloudHSM PCI PIN Responsibility Summary" from the AWS PCI PIN Compliance Package, available to customers through AWS Artifact, a self-service portal for on-demand access to AWS compliance reports. For more information, sign in to AWS Artifact in the AWS Management Console, or learn more at Getting Started with AWS Artifact.

Q: As an AWS CloudHSM customer, can I rely on PCI-PIN Attestation of Compliance (AOC)?

Customers must manage their own PCI-PIN compliance. You are required to go through a formal PCI-PIN attestation process through a Qualified PIN Assessor (QPA) to verify that your payment workload satisfies all PCI-PIN controls/requirements. However, for the controls which AWS is responsible for, your QPA can rely on AWS CloudHSM Attestation of Compliance (AOC) without further testing.

Q: Is AWS CloudHSM responsible for PCI-PIN requirements related to Key Management Life cycle?

AWS CloudHSM is responsible for the physical device lifecycle of the HSMs. Customers are responsible for the key management life cycle requirements in the PCI-PIN standard.

Q: Which AWS CloudHSM controls are PCI-PIN compliant?

The AOC summarizes the AWS CloudHSM controls which are assessed by QPA. The PCI-PIN Responsibility Summary is available to customers through AWS Artifact, a self-service portal for on-demand access to AWS compliance reports.

Q: Does AWS CloudHSM support payment functions such as PIN translation and DUKPT?

No, AWS CloudHSM provides general purpose HSMs. Over time we may provide payment functions. Although the service does not perform payment functions directly, the AWS CloudHSM PCI PIN attestation of compliance enables customers to attain their own PCI compliance for their services running on AWS CloudHSM. If you are interested in using AWS Payment Cryptography services for your workload, please refer to the blog "Move Payment Processing to the Cloud with AWS Payment Cryptography."

Deprecation Notifications

From time to time, AWS CloudHSM may deprecate functionality in order to remain compliant with the requirements of FIPS 140, PCI-DSS, PCI-PIN, PCI-3DS, SOC2, or because of end-of-support hardware. This page lists the changes that currently apply.

HSM1 Deprecation

The AWS CloudHSM hsm1.medium instance type will reach its end of support on December 1, 2025. To ensure continued service, we're introducing the following changes:

- Starting April 2025, you won't be able to create new hsm1.medium clusters.
- Starting April 2025, we will begin automatically migrating existing hsm1.medium clusters to the new hsm2m.medium instance type.

The hsm2m.medium instance type is compatible with your current AWS CloudHSM instance type and offers improved performance. To avoid disruption to your applications, you must upgrade to latest version of client SDK. For upgrade instructions, see ???.

You have two options for migration:

1. Opt in to a CloudHSM-managed migration when you're ready. For more information, ???.
2. Create a new hsm2m.medium cluster from a backup of your hsm1 cluster and redirect your application to the new cluster. We recommend using a blue/green deployment strategy for this approach. For more information, see ???.

FIPS 140 Compliance: 2024 Mechanism Deprecation

The National Institute of Standards and Technology (NIST)[1] advises that support for Triple DES (DESede, 3DES, DES3) encryption and RSA key wrap and unwrap with PKCS#1 v1.5 padding is disallowed after December 31, 2023. Therefore, support for these end on January 1, 2024 in our Federal Information Processing Standard (FIPS) mode clusters. Support for these remain for clusters in non-FIPs mode.

This guidance applies to the following cryptographic operations:

- Triple DES key generation
  - `CKM_DES3_KEY_GEN` for the PKCS#11 Library
  - `DESede` Keygen for the JCE Provider
  - `genSymKey` with `-t=21` for the KMU
- Encryption with Triple DES keys (note: decrypt operations are allowed)
  - For the PKCS #11 Library: `CKM_DES3_CBC` encrypt, `CKM_DES3_CBC_PAD` encrypt, and `CKM_DES3_ECB` encrypt
  - For the JCE Provider: `DESede/CBC/PKCS5Padding` encrypt, `DESede/CBC/NoPadding` encrypt, `DESede/ECB/Padding` encrypt, and `DESede/ECB/NoPadding` encrypt
- RSA key wrap, unwrap, encrypt, and decrypt with PKCS#1 v1.5 padding
  - `CKM_RSA_PKCS` wrap, unwrap, encrypt, and decrypt for the PKCS#11 SDK
  - `RSA/ECB/PKCS1Padding` wrap, unwrap, encrypt, and decrypt for the JCE SDK
  - `wrapKey` and `unWrapKey` with `-m 12` for the KMU (note `12` is the value for mechanism `RSA_PKCS`)

[1] For details on this change, refer to Table 1 and Table 5 in Transitioning the Use of Cryptographic Algorithms and Key Lengths.

Resilience in AWS CloudHSM

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure. For more information about AWS CloudHSM features to support resiliency, see AWS CloudHSM cluster high availability and load balancing.

Infrastructure security in AWS CloudHSM

As a managed service, AWS CloudHSM is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access AWS CloudHSM through the network. Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Network isolation

A virtual private cloud (VPC) is a virtual network in your own logically isolated area in the AWS cloud. You can create a cluster in a private subnet in your VPC. You can create private subnets when you create a VPC. For more information, see Create a virtual private cloud (VPC) for AWS CloudHSM.

When you create an HSM, AWS CloudHSM put an elastic network interface (ENI) in your subnet so that you can interact with your HSMs. For more information, see AWS CloudHSM cluster architecture.

AWS CloudHSM creates a security group that allows inbound and outbound communication between HSMs in your cluster. You can use this security group to enable your EC2 instances to communicate with the HSMs in your cluster. For more information, see Configure the Client Amazon EC2 instance security groups for AWS CloudHSM.

Authorization of users

With AWS CloudHSM, operations performed on the HSM require the credentials of an authenticated HSM user. For more information, see the section called "User types".

AWS CloudHSM and VPC endpoints

You can establish a private connection between your VPC and AWS CloudHSM by creating an *interface VPC endpoint*. Interface endpoints are powered by AWS PrivateLink, a technology that enables you to privately access AWS CloudHSM APIs without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. Instances in your VPC don't need public IP addresses to communicate with AWS CloudHSM APIs. Traffic between your VPC and AWS CloudHSM does not leave the Amazon network.

Each interface endpoint is represented by one or more Elastic Network Interfaces in your subnets.

For more information, see Interface VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Considerations for AWS CloudHSM VPC endpoints

Before you set up an interface VPC endpoint for AWS CloudHSM, ensure that you review Interface endpoint properties and limitations in the *Amazon VPC User Guide*.

- AWS CloudHSM supports making calls to all of its API actions from your VPC.

Creating an interface VPC endpoint for AWS CloudHSM

You can create a VPC endpoint for the AWS CloudHSM service using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

To create a VPC endpoint for AWS CloudHSM, use the following service name:

```
com.amazonaws.<region>.cloudhsmv2
```

For example, in the US West (Oregon) Region (`us-west-2`), the service name would be:

```
com.amazonaws.us-west-2.cloudhsmv2
```

To make it easier to use the VPC endpoint, you can enable a private DNS hostname for your VPC endpoint. If you select the **Enable Private DNS Name** option, the standard AWS CloudHSM DNS hostnames (`https://cloudhsmv2.<region>.amazonaws.com` and `https://cloudhsmv2.<region>.api.aws`) resolves to your VPC endpoint.

This option makes it easier to use the VPC endpoint. The AWS SDKs and AWS CLI use the standard AWS CloudHSM DNS hostname by default, so you do not need to specify the VPC endpoint URL in applications and commands.

For more information, see Accessing a service through an interface endpoint in the *Amazon VPC User Guide*.

Creating a VPC endpoint policy for AWS CloudHSM

You can attach an endpoint policy to your VPC endpoint that controls access to AWS CloudHSM. The policy specifies the following information:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see Controlling access to services with VPC endpoints in the *Amazon VPC User Guide*.

Example: VPC endpoint policy for AWS CloudHSM actions

The following is an example of an endpoint policy for AWS CloudHSM. When attached to an endpoint, this policy grants access to the listed AWS CloudHSM actions for all principals on all resources. See Identity and access management for AWS CloudHSM for other AWS CloudHSM actions and their corresponding IAM permissions.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "<cloudhsm>:<DescribeBackups>",
            "<cloudhsm>:<DescribeClusters>",
            "<cloudhsm>:<ListTags>",
```

```
        ],
        "Resource":"*"
      }
   ]
}
```

Update management in AWS CloudHSM

AWS manages the firmware. Firmware is maintained by a third party, and must be evaluated by NIST for FIPS 140-2 Level 3 or FIPS 140-3 Level 3 compliance depending on the hsm type. Only firmware that has been cryptographically signed by the FIPS key, which AWS does not have access to, can be installed.

Troubleshooting AWS CloudHSM

If you encounter problems with AWS CloudHSM, the following topics can help you resolve them.

Topics

AWS CloudHSM known issues

AWS CloudHSM has the following known issues. Choose a topic to learn more.

Topics

Known issues for all HSM instances

The following issues impact all AWS CloudHSM users regardless of whether they use the key_mgmt_util command line tool, the PKCS #11 SDK, the JCE SDK, or the OpenSSL SDK.

Topics

Issue: AES key wrapping uses PKCS #5 padding instead of providing a standards-compliant implementation of key wrap with zero padding

Additionally, key wrap with no padding and zero padding is not supported.

- **Impact:** There is no impact if you wrap and unwrap using this algorithm within AWS CloudHSM. However, keys wrapped with AWS CloudHSM cannot be unwrapped within other HSMs or software that expects compliance to the no-padding specification. This is because eight bytes of padding data might be added to the end of your key data during a standards-compliant unwrap. Externally wrapped keys cannot be properly unwrapped into an AWS CloudHSM instance.
- **Workaround:** To externally unwrap a key that was wrapped with AES Key Wrap with PKCS #5 Padding on an AWS CloudHSM instance, strip the extra padding before you attempt to use the key. You can do this by trimming the extra bytes in a file editor or copying only the key bytes into a new buffer in your code.
- **Resolution status:** With the 3.1.0 client and software release, AWS CloudHSM provides standards-compliant options for AES key wrapping. For more information, see [AES Key Wrapping](#).

Issue: The client daemon requires at least one valid IP address in its configuration file to successfully connect to the cluster

- **Impact:** If you delete every HSM in your cluster and then add another HSM, which gets a new IP address, the client daemon continues to search for your HSMs at their original IP addresses.
- **Workaround:** If you run an intermittent workload, we recommend that you use the `IpAddress` argument in the [CreateHsm](#) function to set the elastic network interface (ENI) to its original value. Note than an ENI is specific to an Availability Zone (AZ). The alternative is to delete the `/opt/cloudhsm/daemon/1/cluster.info` file and then reset the client configuration to the IP address of your new HSM. You can use the `client -a <IP address>` command. For more information, see [Install and Configure the AWS CloudHSM Client (Linux)](#) or [Install and Configure the AWS CloudHSM Client (Windows)](#).

Issue: There was an upper limit of 16 KB on data that can be hashed and signed by AWS CloudHSM using Client SDK 3

- **Resolution status:** Data less than 16KB in size continues to be sent to the HSM for hashing. We have added capability to hash locally, in software, data between 16KB and 64KB in size. Client

SDK 5 will explicitly fail if the data buffer is larger than 64KB. You must update your client and SDK(s) to a version greater than 5.0.0 or higher to benefit from the fix.

Issue: Imported keys could not be specified as non-exportable

- **Resolution Status:** This issue is fixed. No action is required on your part to benefit from the fix.

Issue: The default mechanism for the wrapKey and unWrapKey commands in the key_mgmt_util has been removed

- **Resolution:** When using the wrapKey or unWrapKey commands, you must use the `-m` option to specify the mechanism. See the examples in the wrapKey or unWrapKey articles for more information.

Issue: If you have a single HSM in your cluster, HSM failover does not work correctly

- **Impact:** If the single HSM instance in your cluster loses connectivity, the client will not reconnect with it even if the HSM instance is later restored.
- **Workaround:** We recommend at least two HSM instances in any production cluster. If you use this configuration, you will not be impacted by this issue. For single-HSM clusters, bounce the client daemon to restore connectivity.
- **Resolution status:** This issue has been resolved in the AWS CloudHSM client 1.1.2 release. You must upgrade to this client to benefit from the fix.

Issue: If you exceed the key capacity of the HSMs in your cluster within a short period of time, the client enters an unhandled error state

- **Impact:** When the client encounters the unhandled error state, it freezes and must be restarted.
- **Workaround:** Test your throughput to ensure you are not creating session keys at a rate that the client is unable to handle. You can lower your rate by adding an HSM to the cluster or slowing down the session key creation.
- **Resolution status:** This issue has been resolved in the AWS CloudHSM client 1.1.2 release. You must upgrade to this client to benefit from the fix.

Issue: Digest operations with HMAC keys of size greater than 800 bytes are not supported

- **Impact:** HMAC keys larger than 800 bytes can be generated on or imported into the HSM. However, if you use this larger key in a digest operation via the JCE or key_mgmt_util, the operation will fail. Note that if you are using PKCS11, HMAC keys are limited to a size of 64 bytes.
- **Workaround:** If you will be using HMAC keys for digest operations on the HSM, ensure the size is smaller than 800 bytes.
- **Resolution status:** None at this time.

Issue: The client_info tool, distributed with Client SDK 3, deletes the contents of the path specified by the optional output argument

- **Impact:** All existing files and sub-directories under the specified output path may be permanently lost.
- **Workaround:** Do not use the optional argument `-output path` when using the `client_info` tool.
- **Resolution status:** This issue has been resolved in the Client SDK 3.3.2 release. You must upgrade to this client to benefit from the fix.

Issue: You receive an error when running the SDK 5 configure tool using the `--cluster-id` argument in containerized environments

You receive the following error when using the --cluster-id argument with the Configure Tool:

```
No credentials in the property bag
```

This error is caused by an update to Instance Metadata Service Version 2 (IMDSv2). For more information, see the IMDSv2 documentation.

- **Impact:** This issue will impact users running the configure tool on SDK versions 5.5.0 and later in containerized environments and utilizing EC2 instance metadata to provide credentials.
- **Workaround:** Set the PUT response hop limit to at least two. For guidance on how to do this, see Configure the instance metadata options.

Issue: You receive the error "Failed to create cert/key from provided pfx file. Error: NotPkcs8"

- **Workaround:** You can convert the custom SSL private key to PKCS8 format with openssl command: `openssl pkcs8 -topk8 -inform PEM -outform PEM -in ssl_private_key -out ssl_private_key_pkcs8`
- **Resolution status:** This issue has been resolved in the client SDK 5.12.0 release. You must upgrade to this client version or later to benefit from the fix.

Issue: ECDSA signing fails with "invalid mechanism" error starting with SDK 5.16

- **Impact:** ECDSA signing operations fail when using hash functions that are weaker than the key strength. This failure occurs because FIPS 186-5 requires the hash function to be at least as strong as the key strength.

  You might see an error similar to this in your client logs:

  ```
  [cloudhsm_provider::hsm1::session::ecdsa::sign::common][][] Digest security strength
   (80) is weaker than the key security strength (128)
  ```
- **Workaround:** If you can't update your hash functions, you can migrate to non-FIPS clusters, which don't enforce the hash strength requirement. However, we recommend updating your hash functions to maintain FIPS compliance.

  As an additional workaround, we have added a configuration option to bypass this requirement. Please note, this option is **not recommended**, as using ECDSA with weaker hash functions does not follow security best practices. To use this option, run the following command (replacing `configure-cli` with the configure tool for the SDK being used: the section called “Syntax”):

  ```
  sudo /opt/cloudhsm/bin/configure-cli --enable-ecdsa-with-weak-hash-function
  ```
- **Resolution:** Use a hash function that is at least as strong as your ECDSA key. For information about hash function and ECDSA key strengths, see Tables 2 and 3 in NIST SP 800-57 Part 1 Rev 5.

Known issues for AWS CloudHSM hsm1.medium instances

The following issues impact all AWS CloudHSM hsm1.medium instances.

Topics

Issue: The HSM cannot create more than 250 users

- **Workaround:** This issue is resolved on AWS CloudHSM hsm2m.medium instance types.
- **Resolution status:** None at this time.

Known issues for AWS CloudHSM hsm2m.medium instances

The following issues impact all AWS CloudHSM hsm2m.medium instances.

Topics

Issue: Increased login latency on hsm2m.medium

- **Impact:** The hsm2m.medium adheres to the latest FIPS 140-3 Level 3 requirements. Logging into hsm2m.medium follows enhanced security and compliance requirements, which results in increased latency.
- **Workaround:** If possible, serialize login requests in the same application to avoid extended latency during login. Multiple login requests in parallel will cause increased latency.

Issue: A CO using trying to set the trusted attribute of a key will fail with Client SDK 5.12.0 and earlier

- **Impact:** Any CO user attempting to set the trusted attribute of a key will receive an error indicating that `User type should be CO or CU`.
- **Resolution:** Future versions of the Client SDK will resolve this issue. Updates will be announced in our user guide's Document history.

Issue: ECDSA verify will fail with Client SDK 5.12.0 and earlier for clusters in FIPS mode

- **Impact:** ECDSA verify operation performed for HSMs in FIPS mode will fail.
- **Resolution status:** This issue has been resolved in the client SDK 5.13.0 release. You must upgrade to this client version or later to benefit from the fix.

Issue: Only the PEM-formatted certificates can be registered as mtls trust anchors with CloudHSM CLI

- **Impact:** Certificates in DER format cannot be registered as mTLS trust anchors with CloudHSM CLI.
- **Workaround:** You can convert a certificate in DER format to PEM format with openssl command: `openssl x509 -inform DER -outform PEM -in *certificate.der* -out *certificate.pem*`

Issue: Customer applications will stop processing all requests when using mTLS with a passphrase protected client private key.

- **Impact:** All operations performed by the application will be halted and the user will be prompted for the passphrase on standard input multiple times throughout the lifetime of application. Operations will timeout and fail if passphrase is not provided before the operation's timeout duration.

- **Workaround:** Passphrase encrypted private keys are not supported for mTLS. Remove passphrase encryption from client private key

Issue: User replicate fails when using the CloudHSM CLI

- **Impact:** User replication fails on hsm2m.medium instances when using the CloudHSM CLI. The `user replicate` command works as expected on hsm1.medium instances.
- **Resolution:** We're working to resolve this issue. For updates, see the Document history in the user guide.

Issue: Operations can fail during backup creation

- **Impact:** Operations like generating random numbers can fail on hsm2m.medium instances while AWS CloudHSM creates a backup.
- **Resolution:** To minimize service interruptions, implement these best practices:
  - Create a multi-HSM cluster
  - Configure your applications to retry cluster operations

  For more information about best practices, see Best practices for AWS CloudHSM.

Issue: Client SDK 5.8 and above do not perform automatic retries for HSM throttled operations in some scenarios on hsm2m.medium

- **Impact:** Client SDK 5.8 and above will not retry some HSM throttled operations
- **Workaround:** Follow best practices to architect your cluster to handle load and implement application level retries. We are currently working on a fix. Updates will be announced in our user guide's Document history.

Known issues for AWS CloudHSM backups

The following issues impact all AWS CloudHSM backups.

Issue: Cross-account backup sharing unavailable in Europe (Spain)

- **Impact:** Cross-account backup sharing is not supported in this region. The GetResourcePolicy, DeleteResourcePolicy, and PutResourcePolicy APIs are temporarily unavailable in Europe (Spain).
- **Workaround:** To transfer HSM data between accounts, manually export the data from the source account's HSM and import it into the destination account's HSM.
- **Resolution Status:** We are currently working on enabling this feature in Europe (Spain).

Known issues for the PKCS #11 library for AWS CloudHSM

The following issues impact the PKCS #11 library for AWS CloudHSM.

Topics

Issue: AES key wrap in version 3.0.0 of the PKCS #11 library does not validate IVs before use

If you specify an IV shorter than 8 bytes in length, it is padded with unpredictable bytes before use.

> **Note**
>
> This impacts `C_WrapKey` with `CKM_AES_KEY_WRAP` mechanism only.

- **Impact:** If you provide an IV that is shorter than 8 bytes in version 3.0.0 of PKCS #11 library, you may be unable to unwrap the key.
- **Workarounds:**
  - We strongly recommend you upgrade to version 3.0.1 or higher of the PKCS #11 library, which properly enforces IV length during AES key wrap. Amend your wrapping code to pass a NULL IV, or specify the default IV of `0xA6A6A6A6A6A6A6A6`. For more information, see Custom IVs with Non-Compliant Length for AES Key Wrap.
  - If you wrapped any keys with version 3.0.0 of the PKCS #11 library using an IV shorter than 8 bytes, reach out to us for support.
- **Resolution status:** This issue has been resolved in version 3.0.1 of the PKCS #11 library. To wrap keys using AES key wrap, specify an IV that is NULL or 8 bytes long.

Issue: PKCS#11 SDK 2.0.4 and earlier versions always used the default IV of `0xA6A6A6A6A6A6A6A6` for AES key wrap and unwrap

User-provided IVs were silently ignored.

> **Note**
>
> This impacts `C_WrapKey` with `CKM_AES_KEY_WRAP` mechanism only.

- **Impact:**
  - If you used PKCS#11 SDK 2.0.4 or an earlier version and a user-provided IV, your keys are wrapped with the default IV of `0xA6A6A6A6A6A6A6A6`.
  - If you used PKCS#11 SDK 3.0.0 or later and a user-provided IV, your keys are wrapped with the user-provided IV.
- **Workarounds:**
  - To unwrap keys wrapped with PKCS#11 SDK 2.0.4 or earlier use the default IV of `0xA6A6A6A6A6A6A6A6`.

  - To unwrap keys wrapped with PKCS#11 SDK 3.0.0 or later, use the user-provided IV.
- **Resolution status:** We strongly recommend that you amend your wrapping and unwrapping code to pass a NULL IV, or specify the default IV of `0xA6A6A6A6A6A6A6A6`.

Issue: The `CKA_DERIVE` attribute was not supported and was not handled

- **Resolution status:** We have implemented fixes to accept `CKA_DERIVE` if it is set to `FALSE`. `CKA_DERIVE` set to `TRUE` will not be supported until we begin to add key derivation function support to AWS CloudHSM. You must update your client and SDK(s) to version 1.1.1 or higher to benefit from the fix.

Issue: The `CKA_SENSITIVE` attribute was not supported and was not handled

- **Resolution status:** We have implemented fixes to accept and properly honor the `CKA_SENSITIVE` attribute. You must update your client and SDK(s) to version 1.1.1 or higher to benefit from the fix.

Issue: Multipart hashing and signing are not supported

- **Impact:** `C_DigestUpdate` and `C_DigestFinal` are not implemented. `C_SignFinal` is also not implemented and will fail with `CKR_ARGUMENTS_BAD` for a non-`NULL` buffer.
- **Workaround:** Hash your data within your application and use AWS CloudHSM only for signing the hash.
- **Resolution status:** We are fixing the client and the SDKs to correctly implement multipart hashing. Updates will be announced in the AWS CloudHSM forum and on the version history page.

Issue: `C_GenerateKeyPair` does not handle `CKA_MODULUS_BITS` or `CKA_PUBLIC_EXPONENT` in the private template in a manner that is compliant with standards

- **Impact:** `C_GenerateKeyPair` should return `CKA_TEMPLATE_INCONSISTENT` when the private template contains `CKA_MODULUS_BITS` or `CKA_PUBLIC_EXPONENT`. It instead generates a private key for which all usage fields are set to `FALSE`. The key cannot be used.

- **Workaround:** We recommend that your application check the usage field values in addition to the error code.
- **Resolution status:** We are implementing fixes to return the proper error message when an incorrect private key template is used. The updated PKCS #11 library will be announced on the version history page.

Issue: Buffers for the `C_Encrypt` and `C_Decrypt` API operations cannot exceed 16 KB when using the `CKM_AES_GCM` mechanism

AWS CloudHSM does not support multipart AES-GCM encryption.

- **Impact:** You cannot use the `CKM_AES_GCM` mechanism to encrypt data larger than 16 KB.
- **Workaround:** You can use an alternative mechanism such as `CKM_AES_CBC`, `CKM_AES_CBC_PAD`, or you can divide your data into pieces and encrypt each piece using `AES_GCM` individually. If you're using `AES_GCM`, you must manage the division of your data and subsequent encryption. AWS CloudHSM does not perform multipart AES-GCM encryption for you. Note that FIPS requires that the initialization vector (IV) for `AES-GCM` be generated on the HSM. Therefore, the IV for each piece of your AES-GCM encrypted data will be different.
- **Resolution status:** We are fixing the SDK to fail explicitly if the data buffer is too large. We return `CKR_MECHANISM_INVALID` for the `C_EncryptUpdate` and `C_DecryptUpdate` API operations. We are evaluating alternatives to support larger buffers without relying on multipart encryption. Updates will be announced in the AWS CloudHSM forum and on the version history page.

Issue: Elliptic-curve Diffie-Hellman (ECDH) key derivation is executed partially within the HSM

Your EC private key remains within the HSM at all times, but the key derivation process is performed in multiple steps. As a result, intermediate results from each step are available on the client.

- **Impact:** In Client SDK 3, the key derived using the `CKM_ECDH1_DERIVE` mechanism is first available on the client and is then imported into the HSM. A key handle is then returned to your application.
- **Workaround:** If you are implementing SSL/TLS Offload in AWS CloudHSM, this limitation may not be an issue. If your application requires your key to remain within an FIPS boundary at all times, consider using an alternative protocol that does not rely on ECDH key derivation.

- **Resolution status:** SDK 5.16 now supports ECDH with Key Derivation which is performed entirely within the HSM.

Issue: Verification of secp256k1 signatures fails on EL6 platforms such as CentOS6 and RHEL 6

This happens because the CloudHSM PKCS#11 library avoids a network call during initialization of the verification operation by using OpenSSL to verify EC curve data. Since Secp256k1 is not supported by the default OpenSSL package on EL6 platforms, the initialization fails.

- **Impact:** Secp256k1 signature verification will fail on EL6 platforms. The verify call will fail with a `CKR_HOST_MEMORY` error.
- **Workaround:** We recommend using either Amazon Linux 1 or any EL7 platform if your PKCS#11 application needs to verify secp256k1 signatures. Alternatively, upgrade to a version of the OpenSSL package that supports the secp256k1 curve.
- **Resolution status:** We are implementing fixes to fall back to the HSM if local curve validation is not available. The updated PKCS#11 library will be announced on the version history page.

Issue: Incorrect sequence of function calls gives undefined results instead of failing

- **Impact**: If you call an incorrect sequence of functions, the final result is incorrect even though the individual function calls return success. For instance, decrypted data may not match the original plaintext or signatures may fail to verify. This issue affects both single part and multi-part operations.

  Examples of incorrect function sequences:
  - `C_EncryptInit/C_EncryptUpdate` followed by `C_Encrypt`
  - `C_DecryptInit/C_DecryptUpdate` followed by `C_Decrypt`
  - `C_SignInit/C_SignUpdate` followed by `C_Sign`
  - `C_VerifyInit/C_VerifyUpdate` followed by `C_Verify`
  - `C_FindObjectsInit` followed by `C_FindObjectsInit`
- **Workaround**: Your application should, in compliance with the PKCS #11 specification, use the right sequence of function calls for both single and multi-part operations. Your application should not rely on the CloudHSM PKCS #11 library to return an error under this circumstance.

Issue: Read Only Session is not supported in SDK 5

- **Issue:** SDK 5 does not support opening Read-Only sessions with `C_OpenSession`.
- **Impact:** If you attempt to call `C_OpenSession` without providing `CKF_RW_SESSION`, the call will fail with the error `CKR_FUNCTION_FAILED`.
- **Workaround:** When opening a session, you must pass the `CKF_SERIAL_SESSION | CKF_RW_SESSION` flags to the `C_OpenSession` function call.

Issue: `cryptoki.h` header file is Windows-only

- **Issue:** With AWS CloudHSM Client SDK 5 versions 5.0.0 through 5.4.0 on Linux, the header file `/opt/cloudhsm/include/pkcs11/cryptoki.h` is only compatible with Windows operating systems.
- **Impact:** You may encounter issues when trying to include this header file in your application on Linux-based operating systems.
- **Resolution status:** Upgrade to AWS CloudHSM Client SDK 5 version 5.4.1 or above, which includes a Linux-compatible version of this header file.

Known issues for the JCE SDK for AWS CloudHSM

The following issues impact the JCE SDK for AWS CloudHSM.

Topics

Issue: When working with asymmetric key pairs, you see occupied key capacity even when you are not explicitly creating or importing keys

- **Impact:** This issue can cause your HSMs to unexpectedly run out of key space and occurs when your application uses a standard JCE key object for crypto operations instead of a `CaviumKey` object. When you use a standard JCE key object, `CaviumProvider` implicitly imports that key into the HSM as a session key and does not delete this key until the application exits. As a result, keys build up while the application is running and can cause your HSMs to run out of free key space, thus freezing your application.
- **Workaround:** When using the `CaviumSignature` class, `CaviumCipher` class, `CaviumMac` class, or the `CaviumKeyAgreement` class, you should supply the key as a `CaviumKey` instead of a standard JCE key object.

  You can manually convert a normal key to a `CaviumKey` using the `ImportKey` class, and can then manually delete the key after the operation is complete.
- **Resolution status:** We are updating the `CaviumProvider` to properly manage implicit imports. The fix will be announced on the version history page once available.

Issue: The JCE KeyStore is read only

- **Impact:** You cannot store an object type that is not supported by the HSM in the JCE keystore today. Specifically, you cannot store certificates in the keystore. This precludes interoperability with tools like jarsigner, which expect to find the certificate in the keystore.
- **Workaround:** You can rework your code to load certificates from local files or from an S3 bucket location instead of from the keystore.
- **Resolution status:** You can use AWS CloudHSM keystore to store certificates.

Issue: Buffers for AES-GCM encryption cannot exceed 16,000 bytes

Multi-part AES-GCM encryption is not supported.

- **Impact:** You cannot use AES-GCM to encrypt data larger than 16,000 bytes.
- **Workaround:** You can use an alternative mechanism, such as AES-CBC, or you can divide your data into pieces and encrypt each piece individually. If you divide the data, you must manage the divided ciphertext and its decryption. Because FIPS requires that the initialization vector (IV) for

AES-GCM be generated on the HSM, the IV for each AES-GCM-encrypted piece of data will be different.

- **Resolution status:** We are fixing the SDK to fail explicitly if the data buffer is too large. We are evaluating alternatives that support larger buffers without relying on multi-part encryption. Updates will be announced in the AWS CloudHSM forum and on the version history page.

Issue: Elliptic-curve Diffie-Hellman (ECDH) key derivation is executed partially within the HSM

Your EC private key remains within the HSM at all times, but the key derivation process is performed in multiple steps. As a result, intermediate results from each step are available on the client. An ECDH key derivation sample is available in the Java code samples.

- **Impact:** Client SDK 3 adds ECDH functionality to the JCE. When you use the `KeyAgreement` class to derive a SecretKey, it is first available on the client and is then imported into the HSM. A key handle is then returned to your application.
- **Workaround:** If you are implementing SSL/TLS Offload in AWS CloudHSM, this limitation may not be an issue. If your application requires your key to remain within an FIPS boundary at all times, consider using an alternative protocol that does not rely on ECDH key derivation.
- **Resolution status:** SDK 5.16 now supports ECDH with Key Derivation which is performed entirely within the HSM.

Issue: KeyGenerator and KeyAttribute incorrectly interprets key size parameter as number of bytes instead of bits

When generating a key using the `init` function of the KeyGenerator class or the `SIZE` attribute of the AWS CloudHSM KeyAttribute enum, the API incorrectly expects the argument to be the number of key bytes, when it should instead be the number of key bits.

- **Impact:** Client SDK versions 5.4.0 through 5.4.2 incorrectly expects the key size to be provided to the specified APIs as bytes.
- **Workaround:** Convert the key size from bits to bytes before using the KeyGenerator class or KeyAttribute enum to generate keys using the AWS CloudHSM JCE provider if using Client SDK versions 5.4.0 through 5.4.2.

- **Resolution status:** Upgrade your client SDK version to 5.5.0 or later, which includes a fix to correctly expect key sizes in bits when using the KeyGenerator class or KeyAttribute enum to generate keys.

Issue: Client SDK 5 throws the warning “An illegal reflective access operation has occurred”

When using Client SDK 5 with Java 11, CloudHSM throws the following Java warning:

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by
 com.amazonaws.cloudhsm.jce.provider.CloudHsmKeyStore (file:/opt/cloudhsm/java/
cloudhsm-jce-5.6.0.jar) to field java.security .KeyStore.keyStoreSpi
WARNING: Please consider reporting this to the maintainers of
 com.amazonaws.cloudhsm.jce.provider.CloudHsmKeyStore
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective
 access operations
WARNING: All illegal access operations will be denied in a future release
```

This issue is fixed in Client SDK version 5.8 and later.

Issue: JCE session pool is exhausted

Impact: You may not be able to perform operations in JCE after seeing the following message:

```
com.amazonaws.cloudhsm.jce.jni.exception.InternalException: There are too many
 operations
happening at the same time: Reached max number of sessions in session pool: 1000
```

Workarounds:

- Restart your JCE application if you’re experiencing impact.
- When performing an operation, you may need to complete the JCE operation before losing reference to the operation.

> **Note**
>
> Depending on the operation, a completion method may be needed.

Operation	Completion method(s)
Cipher	`doFinal()` in encrypt or decrypt mode `wrap()` in wrap mode `unwrap()` in unwrap mode
KeyAgreement	`generateSecret()` or `generateSecret(String)`
KeyPairGenerator	`generateKeyPair()` , `genKeyPair()` , or `reset()`
KeyStore	No method needed
MAC	`doFinal()` or `reset()`
MessageDigest	`digest()` or `reset()`
SecretKeyFactory	No method needed
SecureRandom	No method needed
Signature	`sign()` in sign mode `verify()` in verify mode

Resolution status: We have resolved this issue in Client SDK 5.9.0 and later. To fix this issue, upgrade your Client SDK to one of these versions.

Issue: Client SDK 5 memory leak with getKey operations

- **Impact:** The API `getKey` operation has a memory leak in JCE in Client SDK versions 5.10.0 and earlier. If you're using the `getKey` API multiple times in your application, it will lead to increased memory growth and consequently increase the memory footprint in your application. Over time this may cause throttling errors or require the application to be restarted.
- **Workaround:** We recommend upgrading to Client SDK 5.11.0. If this can't be done, we recommend not calling the `getKey` API multiple times in your application. Rather, reuse the previously returned key from the prior `getKey` operation as much as possible.
- **Resolution status:** Upgrade your client SDK version to 5.11.0 or later, which includes a fix for this issue.

Known issues for the OpenSSL Dynamic Engine for AWS CloudHSM

These are the known issues for OpenSSL Dynamic Engine for AWS CloudHSM.

Topics

Issue: You cannot install AWS CloudHSM OpenSSL Dynamic Engine on RHEL 6 and CentOS6

- **Impact:** The OpenSSL Dynamic Engine only supports OpenSSL 1.0.2[f+]. By default, RHEL 6 and CentOS 6 ship with OpenSSL 1.0.1.
- **Workaround:** Upgrade the OpenSSL library on RHEL 6 and CentOS 6 to version 1.0.2[f+].

Issue: Only RSA offload to the HSM is supported by default

- **Impact:** To maximize performance, the SDK is not configured to offload additional functions such as random number generation or EC-DH operations.
- **Workaround:** Please contact us through a support case if you need to offload additional operations.
- **Resolution status:** We are adding support to the SDK to configure offload options through a configuration file. The update will be announced on the version history page once available.

Issue: RSA encryption and decryption with OAEP padding using a key on the HSM is not supported

- **Impact:** Any call to RSA encryption and decryption with OAEP padding fails with a divide-by-zero error. This occurs because the OpenSSL dynamic engine calls the operation locally using the fake PEM file instead of offloading the operation to the HSM.
- **Workaround:** You can perform this procedure by using either the PKCS #11 library for AWS CloudHSM Client SDK 5 or the JCE provider for AWS CloudHSM Client SDK 5.
- **Resolution status:** We are adding support to the SDK to correctly offload this operation. The update will be announced on the version history page once available.

Issue: Only private key generation of RSA and ECC keys is offloaded to the HSM

For any other key type, the OpenSSL AWS CloudHSM engine is not used for call processing. The local OpenSSL engine is used instead. This generates a key locally in software.

- **Impact:** Because the failover is silent, there is no indication that you have not received a key that was securely generated on the HSM. You will see an output trace that contains the string `"...........++++++"` if the key is locally generated by OpenSSL in software. This trace is absent when the operation is offloaded to the HSM. Because the key is not generated or stored on the HSM, it will be unavailable for future use.

- **Workaround:** Only use the OpenSSL engine for key types it supports. For all other key types, use PKCS #11 or JCE in applications, or use `key_mgmt_util` in the CLI.

Issue: You cannot install OpenSSL Dynamic Engine for Client SDK 3 on RHEL 8, CentOS 8, or Ubuntu 18.04 LTS

- **Impact:** By default, RHEL 8, CentOS 8, and Ubuntu 18.04 LTS ship a version of OpenSSL that is not compatible with OpenSSL Dynamic Engine for Client SDK 3.
- **Workaround:** Use a Linux platform that provides support for OpenSSL Dynamic Engine. For more information about supported platforms, see Supported Platforms.
- **Resolution status:** AWS CloudHSM supports these platforms with OpenSSL Dynamic Engine for Client SDK 5. For more information, see Supported Platforms and OpenSSL Dynamic Engine.

Issue: SHA-1 Sign and Verify deprecation on RHEL 9 (9.2+)

- **Impact:** The usage of the SHA-1 message digest for cryptographic purposes has been deprecated in RHEL 9 (9.2+). As a result, sign and verify operations with SHA-1 using the OpenSSL Dynamic Engine will fail.
- **Workaround:** If your scenario requires the use of SHA-1 for signing/verifying existing or third-party cryptographic signatures, see Enhancing RHEL Security: Understanding SHA-1 deprecation on RHEL 9 (9.2+) and RHEL 9 (9.2+) Release Notes for further details.

Issue: AWS CloudHSM OpenSSL Dynamic Engine is incompatible with the FIPS provider for OpenSSL v3.x

- **Impact:** You will receive an error if you attempt to utilize the AWS CloudHSM OpenSSL Dynamic Engine when the FIPS provider is enabled for OpenSSL versions 3.x.
- **Workaround:** To use the AWS CloudHSM OpenSSL Dynamic Engine with OpenSSL versions 3.x, ensure that the "default" provider is configured. Read more about the default provider on the OpenSSL Website.

Issue: SSL/TLS offload fails with ECDSA cipher suites in TLS 1.0 and TLS 1.1 starting with SDK 5.16

- **Impact:** Connection attempts using TLS 1.0 or TLS 1.1 fail because these versions use SHA-1 for signing, which doesn't meet FIPS 186-5 requirements.
- **Workaround:** If you can't upgrade TLS versions immediately, you can migrate to non-FIPS clusters, which don't enforce the hash strength requirement. However, we recommend upgrading to TLS 1.2 or TLS 1.3 to maintain FIPS compliance and security best practices.
- **Resolution:** Upgrade your implementation to use TLS 1.2 or TLS 1.3. The Internet Engineering Task Force (IETF) has deprecated TLS 1.0 and TLS 1.1 due to security concerns.

Known issues for the Key Storage Provider (KSP) for AWS CloudHSM

These are the known issues for Key Storage Provider (KSP) for AWS CloudHSM.

Topics

- Issue: Verification of a certificate store fails
- Issue: Container name inconsistency in the certificate store while using SDK3 compatibility mode for Client SDK 5

Issue: Verification of a certificate store fails

When using Client SDK versions 5.14 and 5.15, calling `certutil -store my CERTIFICATE_SERIAL_NUMBER` throws the following error:

```
ERROR: Could not verify certificate public key against private key
```

- **Impact:** You cannot use `certutil` to validate a certificate store created with Client SDK 5.
- **Workaround:** Validate the key pair associated with the certificate by signing a file using the private key and verifying the signature using the public key. This can be done using Microsoft SignTool by following the steps provided here.
- **Resolution Status:** We're working to add support for verifying certificates using `certutil`. The fix will be announced on the version history page once available.

Issue: Container name inconsistency in the certificate store while using SDK3 compatibility mode for Client SDK 5

When using the `certutil -store my CERTIFICATE_SERIAL_NUMBER` command to view certificates whose key-reference files were generated using generate-file command in AWS CLI 5.16.0, the following error occurs:

```
ERROR: Container name inconsistent: CONTAINER_NAME
```

This error occurs because there is a mismatch between the container name stored in the certificate and the key reference file name generated by the CloudHSM CLI.

- **Impact:** Despite this error, the certificates and their associated keys remain fully functional. All applications using these certificates will continue to work normally.
- **Workaround:** To resolve this error, rename the key reference filename to Simple or Unique container name. Refer to the following sample output of the command `certutil -store my`

  ```
  Subject: CN=www.website.com, OU=Organizational-Unit, O=Organization, L=City, S=State,
   C=US
  Non-root Certificate
  Cert Hash(sha1): 1add52
  Key Container = 7e3c-b2f5
  Simple container name: tq-3daacd89
  Unique container name: tq-3daacd89
  ERROR: Container name inconsistent: 7e3c-b2f5
  ```

  By default, the key reference files will be stored in `C:\Users\Default\AppData\Roaming\Microsoft\Crypto\CaviumKSP\GlobalPartition`

  1. Rename the key reference file to the simple container name.
  2. Repair the certificate store with the new key container name. Refer to steps 12 to 14 in KSP Migration for more details.
- **Resolution status:** This issue has been fixed in Client SDK version 5.16.1. To resolve this problem, upgrade your Client SDK to version 5.16.1 or later.

Known issues for Amazon EC2 instances running Amazon Linux 2 with AWS CloudHSM

The following issues impact AWS CloudHSM and Amazon EC2 instances that are running on Amazon Linux 2.

Issue: Amazon Linux 2 version 2018.07 uses an updated `ncurses` package (version 6) that is currently incompatible with the AWS CloudHSM SDKs

You see the following error returned upon running the AWS CloudHSM cloudhsm_mgmt_util or key_mgmt_util:

```
/opt/cloudhsm/bin/cloudhsm_mgmt_util: error while loading shared libraries:
 libncurses.so.5: cannot open shared object file: No such file or directory
```

- **Impact:** Instances running on Amazon Linux 2 version 2018.07 will be unable to use *all* AWS CloudHSM utilities.
- **Workaround:** Issue the following command on your Amazon Linux 2 EC2 instances to install the supported `ncurses` package (version 5):

  ```
  sudo yum update && yum install ncurses-compat-libs
  ```

- **Resolution status:** This issue has been resolved in the AWS CloudHSM client 1.1.2 release. You must upgrade to this client to benefit from the fix.

Known issues for integrating third-party applications with AWS CloudHSM

The following issues impact AWS CloudHSM when integrating with third-party applications.

Issue: Client SDK 3 does not support Oracle setting PKCS #11 attribute `CKA_MODIFIABLE` during master key generation

This limit is defined in the PKCS #11 library. For more information, see annotation 1 on Supported PKCS #11 Attributes.

- **Impact:** Oracle master key creation fails.

- **Workaround:** Set the special environment variable `CLOUDHSM_IGNORE_CKA_MODIFIABLE_FALSE` to TRUE when creating a new master key. This environment variable is only needed for master key generation and you do not need to use this environment variable for anything else. For example, you would use this variable for the first master key you create and then you would only use this environment variable again if you wanted to rotate your master key edition. For more information, see Generate the Oracle TDE Master Encryption Key.
- **Resolution status:** We are improving the HSM firmware to fully support the CKA_MODIFIABLE attribute. Updates will be announced in the AWS CloudHSM forum and on the version history page

Known issues for AWS CloudHSM cluster modification

The following issues impact customers attempting to use the modify-cluster API to change the HSM type of a cluster.

Topics

Issue: Login latency increases due to increased PBKDF2 iterations

- **Impact:** Clusters with a large amount of users will experience an extended migration period. This is due to changes in the backup restoration process performing PBKDF2 operations per user when restoring an hsm1.medium backup to hsm2m.medium for the first time.
- **Workaround:** Design your applications to be resilient to an extended migration period.
- **Resolution status:** No resolution status.

Issue: Unable to modify HSM type due to token key creation

- **Impact:** Customers performing token key based workloads will be unable to start their migration. This is done because the HSM will be placed into a limited-write mode to prevent dataloss scenarios during the HSM type modification.
- **Workaround:** Stop creating and deleting token keys and then wait 7 days. Alternatively, please reach out to support if you

  - Cannot handle blocking token key migrations and cannot do a blue/green deployment.
  - Can handle blocking token key operations for the duration of the migration, but can’t wait the full 7 day period.
- **Resolution status:** This issue has been resolved. Customers performing token key based workloads can now begin the migration. Token key creations and deletions will be blocked for the duration of the migration.

Known issues of operation failure using AWS CloudHSM client version 5.12.0 on hsm2.medium

The following issues impact AWS CloudHSM when using AWS CloudHSM client version 5.12.0

Issue: Error during get-attribute operation

If you're migrating from hsm1.medium to hsm2m.medium and using CloudHSM Client SDK 5.12.0, you may observe errors related to attribute handling.

You might see the following error message in the client logs: `Error in deserialization of data: Invalid integer conversion`

Impact: Below operations will fail using client version 5.12.0

- In PKCS#11 SDK, calls to C_GetAttributeValue fail
- In CloudHSM CLI, the key list command shows no attributes in the output
- In CloudHSM CLI, key generate-file may fail for keys generated using hsm1.medium

Resolution: We recommend upgrading to the latest version of the SDK which resolves this issue.

AWS CloudHSM Client SDK 3 key synchronization failures

In Client SDK 3, if client-side synchronization fails, AWS CloudHSM makes a best-effort response to clean up any unwanted keys that may have been created (and are now unwanted). This process involves removing unwanted key material immediately or marking unwanted material for later removal. In both these cases, the resolution does not require any action from you. In the rare case that AWS CloudHSM cannot remove *and* cannot mark unwanted key material, you must delete the key material.

Problem: You attempt a token key generation, import, or unwrap operation and see errors that specify a failure to *tombstone*.

```
2018-12-24T18:28:54Z liquidSecurity ERR: print_node_ts_status:
[create_object_min_nodes]Key: 264617 failed to tombstone on node:1
```

Cause: AWS CloudHSM was unsuccessful removing *and* marking unwanted key material.

Resolution: An HSM in your cluster contains unwanted key material that is not marked as unwanted. You must manually remove the key material. To manually delete unwanted key material, use key_mgmt_util (KMU) or an API from the PKCS #11 library or the JCE provider. For more information, see deleteKey or Client SDKs.

To make token keys more durable, AWS CloudHSM fails key creation operations that don't succeed on the minimum number of HSMs specified in client-side synchronization settings. For more information, see Key Synchronization in AWS CloudHSM.

AWS CloudHSM Client SDK 3 verify HSM performance with the pkpspeed tool

This topic describes how to verify AWS CloudHSM hardware security module (HSM) performance with Client SDK 3.

To verify the performance of the HSMs in your AWS CloudHSM cluster, you can use the pkpspeed (Linux) or pkpspeed_blocking (Windows) tool that is included with Client SDK 3. The pkpspeed tool executes under ideal conditions and directly calls the HSM to execute operations without going through an SDK such as PKCS11. We recommend load testing your application independently to determine your scaling needs. We do not recommend running the following tests: Random (I), ModExp (R), and EC point mul (Y).

For more information about installing the client on a Linux EC2 instance, see Install and configure the AWS CloudHSM client for CMU (Linux). For more information about installing the client on a Windows instance, see Install and configure the AWS CloudHSM client for CMU (Windows).

After you install and configure the AWS CloudHSM client, run the following command to start it.

Amazon Linux

```
$ sudo start cloudhsm-client
```

Amazon Linux 2

```
$ sudo service cloudhsm-client start
```

CentOS 7

```
$ sudo service cloudhsm-client start
```

CentOS 8

```
$ sudo service cloudhsm-client start
```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- For Windows client 1.1.2+:

```
C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
```

- For Windows clients 1.1.1 and older:

```
C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe C:
\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

If you have already installed the client software, you might need to download and install the latest version to get pkpspeed. You can find the pkpspeed tool at `/opt/cloudhsm/bin/pkpspeed` in Linux or `C:\Program Files\Amazon\CloudHSM\` in Windows.

To use pkpspeed, run the **pkpspeed** command or **pkpspeed_blocking.exe**, specifying the user name and password of a crypto user (CU) on the HSM. Then set the options to use while considering the following recommendations.

Test recommendations

- To test the performance of RSA sign and verify operations, choose the `RSA_CRT` cipher in Linux or option B in Windows. Don't choose `RSA` (option A in Windows). The ciphers are equivalent, but `RSA_CRT` is optimized for performance.
- Start with a small number of threads. For testing AES performance, one thread is typically enough to show maximum performance. For testing RSA performance(`RSA_CRT`), three or four threads is typically enough.

Configurable options for the pkpspeed tool

- **FIPS Mode**: AWS CloudHSM is always in FIPS mode (See AWS CloudHSM FAQs for details). This can be verified by using the CLI tools as documented in the AWS CloudHSM User Guide and running the **Get hardware information for each HSM in an AWS CloudHSM cluster with CMU** command which will indicate the FIPS mode status.
- **Test type (blocking versus non-blocking)**: This specifies how operations are performed in a threaded manner. You will most likely get better numbers using non-blocking. This is because they utilize threads and concurrency.
- **Number of threads**: Number of threads to run the test with.
- **Time in seconds to run the test (max = 600)**: pkpspeed produces results measured in "OPERATIONS/second" and reports this value for each second that the test is run. For example, if the test is run for 5 seconds the output may look like the following sample values:
  - `OPERATIONS/second 821/1`
  - `OPERATIONS/second 833/1`
  - `OPERATIONS/second 845/1`
  - `OPERATIONS/second 835/1`
  - `OPERATIONS/second 837/1`

Tests that can be ran with the pkpspeed tool

- **AES GCM**: Tests AES GCM mode encryption.
- **Basic 3DES CBC**: Tests 3DES CBC mode encryption. See note 1 below for an upcoming change.
- **Basic AES**: Tests AES CBC/ECB encryption.
- **Digest**: Tests hash digest.
- **ECDSA Sign**: Tests ECDSA sign.
- **ECDSA Verify**: Tests ECDSA verify.
- **FIPS Random**: Tests generation of a FIPS-compliant random number (Note: this can only be used in blocking mode).
- **HMAC**: Tests HMAC.
- **Random**: This test is not relevant because we are using FIPS 140-2 HSM's.
- **RSA non-CRT versus RSA_CRT**: Tests RSA sign and verify operations.
- **RSA OAEP Enc**: Tests RSA OAEP encryption.
- **RSA OAEP Dec**: Tests RSA OAEP decryption.
- **RSA private dec non-CRT**: Tests RSA Private key encryption (non-optimized).
- **RSA private key dec CRT**: Tests RSA Private key encryption (optimized).
- **RSA PSS Sign**: Tests RSA PSS sign.
- **RSA PSS Verify**: Tests RSA PSS verify.
- **RSA public key enc**: Tests RSA Public key encryption.

RSA public key encryption, RSA private decryption non-CRT, and RSA private key decryption CRT will also prompt the user to answer the following:

```
Do you want to use static key [y/n]
```

If y is entered, a pre-computed key is imported into the HSM.

If n is entered, a new key is generated.

[1] In accordance with NIST guidance, this is disallowed for clusters in FIPS mode after 2023. For clusters in non-FIPS mode, it is still allowed after 2023. See FIPS 140 Compliance: 2024 Mechanism Deprecation for details.

Examples

The following examples show the options that you can choose with pkpspeed (Linux) or pkpspeed_blocking (Windows) to test the HSM's performance for RSA and AES operations.

Example – Using pkpspeed to test RSA performance

You can run this example on Windows, Linux, and compatible operating systems.

Linux

Use these instructions for Linux and compatible operating systems.

```
/opt/cloudhsm/bin/pkpspeed -s CU user name -p password

SDK Version: 2.03

        Available Ciphers:
                AES_128
                AES_256
                3DES
                RSA  (non-CRT. modulus size can be 2048/3072)
                RSA_CRT (same as RSA)
For RSA, Exponent will be 65537

Current FIPS mode is: 00002
Enter the number of thread [1-10]: 3
Enter the cipher: RSA_CRT
Enter modulus length: 2048
Enter time duration in Secs: 60
Starting non-blocking speed test using data length of 245 bytes...
[Test duration is 60 seconds]

Do you want to use static key[y/n] (Make sure that KEK is available)?n
```

Windows

```
c:\Program Files\Amazon\CloudHSM>pkpspeed_blocking.exe -s CU user name -p password

Please select the test you want to run

RSA non-CRT------------------->A
RSA CRT----------------------->B
```

```
Basic 3DES CBC----------------->C
Basic AES---------------------->D
FIPS Random-------------------->H
Random------------------------->I
AES GCM ----------------------->K

eXit------------------------>X
B

Running 4 threads for 25 sec

Enter mod size(2048/3072):2048
Do you want to use Token key[y/n]n
Do you want to use static key[y/n] (Make sure that KEK is available)?  n
OPERATIONS/second                821/1
OPERATIONS/second                833/1
OPERATIONS/second                845/1
OPERATIONS/second                835/1
OPERATIONS/second                837/1
OPERATIONS/second                836/1
OPERATIONS/second                837/1
OPERATIONS/second                849/1
OPERATIONS/second                841/1
OPERATIONS/second                856/1
OPERATIONS/second                841/1
OPERATIONS/second                847/1
OPERATIONS/second                838/1
OPERATIONS/second                843/1
OPERATIONS/second                852/1
OPERATIONS/second                837/
```

Example – Using pkpspeed to test AES performance

Linux

Use these instructions for Linux and compatible operating systems.

```
/opt/cloudhsm/bin/pkpspeed -s <CU user name> -p <password>

SDK Version: 2.03

        Available Ciphers:
                AES_128
```

```
                AES_256
                3DES
                RSA  (non-CRT. modulus size can be 2048/3072)
                RSA_CRT (same as RSA)
For RSA, Exponent will be 65537

Current FIPS mode is: 00000002
Enter the number of thread [1-10]: 1
Enter the cipher: AES_256
Enter the data size [1-16200]: 8192
Enter time duration in Secs: 60
Starting non-blocking speed test using data length of 8192 bytes...
```

Windows

```
c:\Program Files\Amazon\CloudHSM>pkpspeed_blocking.exe -s CU user name -p password
login as USER
Initializing Cfm2 library
        SDK Version: 2.03

 Current FIPS mode is: 00000002
Please enter the number of threads [MAX=400] : 1
Please enter the time in seconds to run the test [MAX=600]: 20


Please select the test you want to run

RSA non-CRT------------------->A
RSA CRT----------------------->B
Basic 3DES CBC---------------->C
Basic AES--------------------->D
FIPS Random------------------->H
Random------------------------>I
AES GCM ---------------------->K

eXit------------------------>X
D

Running 1 threads for 20 sec

Enter the key size(128/192/256):256
Enter the size of the packet in bytes[1-16200]:8192
OPERATIONS/second                9/1
```

```
OPERATIONS/second                10/1
OPERATIONS/second                11/1
OPERATIONS/second                10/1
OPERATIONS/second                10/1
OPERATIONS/second                10/...
```

AWS CloudHSM Client SDK 5 user contains inconsistent values

The `user list` command in AWS CloudHSM Client SDK 5 returns a list of all users, and user properties, in your cluster. If any of a user's properties have the value "**inconsistent**", this user is not synchronized across your cluster. This means that the user exists with different properties on different HSMs in the cluster. Based on which property is inconsistent, different repair steps can be taken.

The following table includes steps to resolve inconsistencies for a single user. If a single user has multiple inconsistencies, resolve them by following these steps from top to bottom. If there are multiple users with inconsistencies, work through this list for each user, fully resolving the inconsistencies for that user before moving on the next.

> **Note**
>
> To perform these steps you should ideally be logged in as an admin. If your admin account is not consistent, go through these steps logging in with the admin and repeating the steps until all properties are consistent. After your admin account is consistent, you can proceed to use that admin to synchronize other users in the cluster.

Inconsistent property	Example output of user list	Implication	Recovery method
User "role" is "inconsis tent"	`{ "username": "test_user", "role": "inconsistent ", "locked": "false",`	This user is a CryptoUser on some HSMs, and an Admin on other HSMs. This can happen if two SDKs attempt to create the same	1. Login as an admin. 2. Delete the user on all HSMs: **user delete --username**

Inconsistent property	Example output of user list	Implication	Recovery method

	`"mfa": [],`
	`"cluster-coverage":`
	` "full"`
	`}`
		user, at the same time, with different roles.You must remove this user, and re-create it with the desired role.	**`<user's name>` --role admin**
			
			user delete --username `<user's name>` --role crypto-user
			
			3. Create the user with the desired role:
			
			user create --username `<user's name>` --role `<desired role>`

Inconsistent property	Example output of user list	Implication	Recovery method
User "cluster-coverage" is "inconsistent"	`{ "username": "test_user", "role": "crypto-user", "locked": "false", "mfa": [], "cluster-coverage": "`**`inconsistent`**` " }`	This user exists on a subset of HSMs in the cluster.This can happen if a **user create** partially succeeded, or if a **user delete** partially succeeded. You must finish your previous operation, either creating or removing this user from your cluster.	If the user should not exist, follow these steps: 1. Login as an admin. 2. Run this command: **user delete --username <user's name> --role admin** 3. Now, run the following command: **user delete --username <user's name> --role crypto-user** If the user should exist, follow these steps: 1. Login as an admin. 2. Run the following command: **user create --username <user's name> --role <desired role>**

Inconsistent property	Example output of user list	Implication	Recovery method
User "locked" parameter is "inconsistent" or "true"	`{ "username": "test_user", "role": "crypto-user", "locked" : `**`inconsistent`**` , "mfa": [], "cluster-coverage": "full" }`	This user is locked out on a subset of HSMs. This can happen if a user uses the wrong password and only connects to a subset of HSMs in the cluster. You must change the user's credentials to be consistent across the cluster.	If the user has MFA activated, follow these steps: 1. Login as an admin. 2. Run the following command to temporarily deactivate MFA: **user change-mfa token-sign --username `<user's name>` --role `<desired role>` --disable** 3. Change the user's password so they can log into all HSMs: **user change-password --username `<user's name>` --role `<desired role>`** If MFA should be active for the user, follow these steps: 1. Have the user log in and re-enable MFA (this

Inconsistent property	Example output of user list	Implication	Recovery method
			will require them to sign tokens and provide their public key in a PEM file): **user change-mfa token-sign --username `<user's name>` --role `<desired role>` —token <File>**

Inconsistent property	Example output of user list	Implication	Recovery method
MFA status is "inconsistent"	`{ "username": "test_user", "role": "crypto-user", "locked": "false", "mfa": [ { "strategy": "token-sign", "status": "`**`inconsistent`**` " } ], "cluster-coverage": "full" }`	This user has different MFA flags on different HSMs in the cluster. This can happen if an MFA operation only completed on a subset of HSMs. You must reset the user's password, and allow them to re-enable MFA.	If the user has MFA activated, follow these steps: 1. Login as an admin. 2. Run the following command to temporarily deactivate MFA: **user change-mfa token-sign --username `<user's name>` --role `<desired role>` --disable** 3. You will also need to then change the user's password so they can log into all HSMs: **user change-password --username `<user's name>` --role `<desired role>`** If MFA should be active for the user, follow these steps: 1. Have the user log in and re-

Inconsistent property	Example output of user list	Implication	Recovery method
			enable MFA (this will require them to sign tokens and provide their public key in a PEM file): **user change-mfa token-sign --username `<user's name>` --role `<desired role>` —token <File>**

AWS CloudHSM Client SDK 5 user replicate failures

The `user replicate` command in the CloudHSM CLI replicates a user between cloned AWS CloudHSM clusters. This guide addresses failures due to user inconsistencies within the source cluster or between the source and destination clusters. User replicate verifies that users are consistent by checking the following attributes:

- User Role
- Account Lock Status
- Quorum Status
- Multi-Factor Authentication (MFA) Status

Problem: The selected user is not synchronized throughout the cluster

The user replication process checks for user synchronization throughout the source cluster. If a user's attribute has the value "inconsistent", this means the user isn't synchronized across the cluster. User replication fails with the following error message:

```
{
  "error_code": 1,
  "data": "Specified user is inconsistent across the cluster"
}
```

To check for user desynchronization in the source cluster:

- Run the `user list` command in the CloudHSM CLI.

```
aws-cloudhsm > user list
{
  "error_code": 0,
  "data": {
    "users": [
      {
        "username": "admin",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      },
      {
        "username": "example-inconsistent-user",
        "role": "admin",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "inconsistent"
      },
      {
        "username": "app_user",
        "role": "internal(APPLIANCE_USER)",
        "locked": "false",
        "mfa": [],
        "quorum": [],
        "cluster-coverage": "full"
      }
    ]
  }
}
```

Resolution: Synchronize user attributes throughout the source cluster

- To synchronize user information throughout the source cluster, refer to the following: AWS CloudHSM Client SDK 5 user contains inconsistent values.

Problem: User exists on the destination cluster with different attributes

If a user already exists with the same reference exists in one or more HSMs in the destination cluster but has different user attributes, the following error may occur:

```
{
  "error_code": 1,
  "data": "User replicate failed on 1 of 3 connections"
}
```

Resolution

1. Determine which version of the user should be kept.
2. Delete the unwanted user in the appropriate cluster by running the `user delete` command. See Delete an AWS CloudHSM user with CloudHSM CLI for more information.
3. Replicate the user by running the `user replicate` command.

AWS CloudHSM Client SDK 5 key replicate failures

The `key replicate` command in the CloudHSM CLI replicates a key from a source AWS CloudHSM cluster to a destination AWS CloudHSM cluster. This guide addresses failures caused by inconsistencies within the source cluster or between the source and destination clusters.

Problem: The selected key is not synchronized throughout the cluster

The key replication process checks for key synchronization throughout the source cluster. If any key information or attributes have the value "inconsistent", this means the key isn't synchronized across the cluster. Key replication fails with the following error message:

```
{
```

```
  "error_code": 1,
  "data": "The selected key is not synchronized throughout the cluster"
}
```

To check for key desynchronization in the source cluster:

1. Run the `key list` command in the CloudHSM CLI.
2. Use the `--filter` flag to specify the key.
3. Add the `--verbose` flag to see the full output with key coverage information.

```
aws-cloudhsm > key list --filter attr.label=example-desynchronized-key-label --verbose
{
  "error_code": 0,
  "data": {
    "matched_keys": [
      {
        "key-reference": "0x000000000048000f",
        "key-info": {
          "key-owners": [
            {
              "username": "cu1",
              "key-coverage": "full"
            }
          ],
          "shared-users": [],
        "key-quorum-values": {
          "manage-key-quorum-value": 0,
          "use-key-quorum-value": 0
        },
          "cluster-coverage": "full"
        },
        "attributes": {
          "key-type": "aes",
          "label": "example-desynchronized-key-label",
          "id": "0x",
          "check-value": "0xbe79db",
          "class": "secret-key",
          "encrypt": false,
          "decrypt": false,
          "token": true,
          "always-sensitive": true,
```

```
            "derive": false,
            "destroyable": true,
            "extractable": true,
            "local": true,
            "modifiable": true,
            "never-extractable": false,
            "private": true,
            "sensitive": true,
            "sign": "inconsistent",
            "trusted": false,
            "unwrap": false,
            "verify": true,
            "wrap": false,
            "wrap-with-trusted": false,
            "key-length-bytes": 16
          }
        }
      ],
      "total_key_count": 1,
      "returned_key_count": 1
    }
}
```

Resolution: Synchronize key information and attributes throughout the source cluster

To synchronize key information and attributes throughout the source cluster:

1. For inconsistent key attributes: Use the `key set-attribute` command to set the desired attribute for the specific key.
2. For inconsistent shared user coverage: Use the `key share` or `key unshare` commands to adjust key sharing with the desired users.

Problem: Key with same reference exists in destination cluster with different information or attributes

If a key with the same reference exists in the destination cluster but has different information or attributes, the following error may occur:

```
{
  "error_code": 1,
```

```
  "data": "Key replicate failed on 1 of 3 connections"
}
```

Resolution

1. Determine which version of the key should be kept.
2. Delete the unwanted key version using the `key delete` command in the appropriate cluster.
3. Replicate the key from the cluster that has the correct version.

AWS CloudHSM error seen during key availability check

Problem: An AWS CloudHSM hardware security module (HSM) is returning the following error:

```
Key <KEY HANDLE> does not meet the availability requirements - The key must be
 available on at least 2 HSMs before being used.
```

Cause: Key availability checks look for keys that, under rare but possible conditions, could be lost. This error usually occurs in clusters with only one HSM or in clusters with two HSMs during a period in which one of them is being replaced. In these situations, the following customer operations likely prompted the above error:

- A new key was generated using a command like **The generate-symmetric category in CloudHSM CLI** or **The generate-asymmetric-pair category in CloudHSM CLI**.
- A **List keys for a user with CloudHSM CLI** operation was started.
- A new instance of the SDK was started.

> **Note**
>
> OpenSSL frequently forks new instances of the SDK.

Resolution/recommendation: Choose from the following actions to prevent this error from occurring:

- Use the **--disable-key-availability-check** parameter to set key availability to false in the configure file of your configure tool. For more information, see the AWS CloudHSM Client SDK 5 configuration parameters section of the Configure tool.

- If using a cluster with two HSMs, avoid using the operations that prompted the error, except during initialization code.
- Increase the amount of HSMs in your cluster to at least three.

AWS CloudHSM extracting keys using JCE

Use the following sections to troubleshoot issues extracting AWS CloudHSM keys using JCE.

getEncoded, getPrivateExponent, or getS returns null

`getEncoded`, `getPrivateExponent`, and `getS` will return null because they are by default disabled. To enable them, refer to Key extraction using JCE for AWS CloudHSM.

If `getEncoded`, `getPrivateExponent`, and `getS` return null after being enabled, your key does not meet the right prerequisites. For more information, refer to Key extraction using JCE for AWS CloudHSM.

getEncoded, getPrivateExponent, or getS return key bytes outside of the HSM

You or someone with access to your system has enabled clear key extraction. See the following pages for more information, including how to reset this configuration to the default disabled state.

- Key extraction using JCE for AWS CloudHSM
- Protecting and extracting keys from an HSM

HSM throttling

When your workload exceeds your AWS CloudHSM cluster's hardware security module (HSM) capacity, you will receive error messages stating HSMs are busy or throttled. When this happens, you may see reduced throughput or an increased rate of rejection requests from HSMs. Additionally, HSMs may send the following busy errors.

For Client SDK 5

- In PKCS11, busy errors map to `CKR_FUNCTION_FAILED`. This error can happen for multiple reasons, but if HSM throttling causes this error the following log lines will appear in your log:

  - `[cloudhsm_provider::hsm1::hsm_connection::e2e_encryption::error] Failed to prepare E2E response. Error: Received error response code from Server. Response Code: 187`
  - `[cloudhsm_pkcs11::decryption::aes_gcm] Received error from the server. Error: This operation is already in progress. Internal error code: 0x000000BB`
- In JCE, busy errors map to `com.amazonaws.cloudhsm.jce.jni.exception.InternalException: Unexpected error with the Provider: The HSM could not queue the request for processing.`
- Other SDKs' busy errors print out the following message: `Received error response code from Server. Response Code: 187.`

For Client SDK 3

- In PKCS11, busy errors map to `CKR_OPERATION_ACTIVE` errors.
- In JCE, busy errors map to `CFM2Exception` with status of `0xBB (187)`. Applications can use `getStatus()` function on `CFM2Exception` to check what status is returned by the HSM.
- Other SDKs busy errors will print out the following message: `HSM Error: HSM is already busy generating the keys(or random bytes) for another request.`

Resolution

You can resolve these issues by completing one or more of the following actions:

- Add retry commands for rejected HSM operations in your application layer. Before enabling retry commands, ensure your cluster is adequately sized to meet peak loads.

> **Note**
>
> For Client SDK 5.8.0 and above, retry commands are turned on by default. For details on each SDK's retry command configuration, refer to Advanced configurations for the Client SDK 5 configure tool.

- Add more HSMs to your cluster by following the instructions in Scaling HSMs in an AWS CloudHSM cluster.

> **Important**
>
> We recommend load testing your cluster to determine the peak load you should anticipate, and then add one more HSM to it to ensure high availability.

Keep HSM users in sync across HSMs in the AWS CloudHSM cluster

To manage your HSM's users, you use a AWS CloudHSM command line tool known as cloudhsm_mgmt_util. It communicates only with the HSMs that are in the tool's configuration file. It's not aware of other HSMs in the cluster that are not in the configuration file.

AWS CloudHSM synchronizes the keys on your HSMs across all other HSMs in the cluster, but it doesn't synchronize the HSM's users or policies. When you use cloudhsm_mgmt_util to manage HSM users, these user changes might affect only some of the cluster's HSMs—the ones that are in the cloudhsm_mgmt_util configuration file. This can cause problems when AWS CloudHSM syncs keys across HSMs in the cluster, because the users that own the keys might not exist on all HSMs in the cluster.

To avoid these problems, edit the cloudhsm_mgmt_util configuration file *before* managing users. For more information, see ???.

Lost connection to the AWS CloudHSM cluster

When you configured the AWS CloudHSM client, you provided the IP address of the first HSM in your cluster. This IP address is saved in the configuration file for the AWS CloudHSM client. When the client starts, it tries to connect to this IP address. If it can't—for example, because the HSM failed or you deleted it—you might see errors like the following:

```
LIQUIDSECURITY: Daemon socket connection error
```

```
LIQUIDSECURITY: Invalid Operation
```

To resolve these errors, update the configuration file with the IP address of an active, reachable HSM in the cluster.

To update the configuration file for the AWS CloudHSM client

1. Use one of the following ways to find the IP address of an active HSM in your cluster.

   - View the **HSMs** tab on the cluster details page in the AWS CloudHSM console.
   - Use the AWS Command Line Interface (AWS CLI) to issue the **describe-clusters** command.

   You need this IP address in a subsequent step.

2. Use the following command to stop the client.

   Amazon Linux

   ```
   $ sudo stop cloudhsm-client
   ```

   Amazon Linux 2

   ```
   $ sudo service cloudhsm-client stop
   ```

   CentOS 7

   ```
   $ sudo service cloudhsm-client stop
   ```

   CentOS 8

   ```
   $ sudo service cloudhsm-client stop
   ```

   RHEL 7

   ```
   $ sudo service cloudhsm-client stop
   ```

   RHEL 8

   ```
   $ sudo service cloudhsm-client stop
   ```

   Ubuntu 16.04 LTS

   ```
   $ sudo service cloudhsm-client stop
   ```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client stop
```

Windows

- For Windows client 1.1.2+:

  ```
  C:\Program Files\Amazon\CloudHSM>net.exe stop AWSCloudHSMClient
  ```

- For Windows clients 1.1.1 and older:

  Use **Ctrl**+**C** in the command window where you started the AWS CloudHSM client.

3. Use the following command to update the client's configuration file, providing the IP address that you found in a previous step.

   ```
   $ sudo /opt/cloudhsm/bin/configure -a <IP address>
   ```

4. Use the following command to start the client.

   Amazon Linux

   ```
   $ sudo start cloudhsm-client
   ```

   Amazon Linux 2

   ```
   $ sudo service cloudhsm-client start
   ```

   CentOS 7

   ```
   $ sudo service cloudhsm-client start
   ```

   CentOS 8

   ```
   $ sudo service cloudhsm-client start
   ```

RHEL 7

```
$ sudo service cloudhsm-client start
```

RHEL 8

```
$ sudo service cloudhsm-client start
```

Ubuntu 16.04 LTS

```
$ sudo service cloudhsm-client start
```

Ubuntu 18.04 LTS

```
$ sudo service cloudhsm-client start
```

Windows

- For Windows client 1.1.2+:

```
C:\Program Files\Amazon\CloudHSM>net.exe start AWSCloudHSMClient
```

- For Windows clients 1.1.1 and older:

```
C:\Program Files\Amazon\CloudHSM>start "cloudhsm_client" cloudhsm_client.exe
 C:\ProgramData\Amazon\CloudHSM\data\cloudhsm_client.cfg
```

Missing AWS CloudHSM audit logs in CloudWatch

If you created an AWS CloudHSM cluster before January 20th, 2018, you will need to manually configure a service-linked role in order to enable the delivery of that cluster's audit logs. For instructions on how to enable a service-linked role on an HSM cluster, see Understanding Service-Linked Roles, as well as Creating a Service-Linked Role in the IAM User Guide.

Custom IVs with non-compliant length for AES key wrap in AWS CloudHSM

This troubleshooting topic helps you determine if your application generates irrecoverable wrapped keys. If you are impacted by this issue, use this topic to address the problem.

Topics

- Determine whether your code generates irrecoverable wrapped keys
- Actions you must take if your code generates irrecoverable wrapped keys

Determine whether your code generates irrecoverable wrapped keys

You are impacted only if you meet *all* the conditions below:

Condition	How do I know?
Your application uses PKCS #11 library	The PKCS #11 library is installed as the `libpkcs11.so` file in your `/opt/cloudhsm/lib` folder. Applications written in the C language generally use the PKCS #11 library directly, while application written in Java may be using the library indirectly via a Java abstraction layer. If you're using Windows, you are NOT affected, as PKCS #11 library is not presently available for Windows.
Your application specifically uses version 3.0.0 of the PKCS #11 library	If you received an email from the AWS CloudHSM team, you are likely using version 3.0.0 of the PKCS #11 library. To check the software version on your application instances, use this command: `rpm -qa | grep ^cloudhsm`

Condition	How do I know?
You wrap keys using AES key wrapping	AES key wrapping means you use an AES key to wrap out some other key. The correspon ding mechanism name is `CKM_AES_K EY_WRAP` . It is used with the function `C_WrapKey` . Other AES based wrapping mechanisms that use initialization vectors (IVs), such as `CKM_AES_GCM` and `CKM_CLOUD HSM_AES_GCM` , are not affected by this issue. [Learn more about functions and mechanisms](#).
You specify a custom IV when calling AES key wrapping, and the length of this IV is shorter than 8	AES key wrap is generally initialized using a `CK_MECHANISM` structure as follows: `CK_MECHANISM mech = {CKM_AES_ KEY_WRAP, IV_POINTER, IV_LENGTH };` This issue applies to you only if: • IV_POINTER is not NULL • IV_LENGTH is less than 8 bytes

If you do not meet all the conditions above, you may stop reading now. Your wrapped keys can be unwrapped properly, and this issue does not impact you. Otherwise, see the section called “Actions you must take if your code generates irrecoverable wrapped keys”.

Actions you must take if your code generates irrecoverable wrapped keys

You should take the following three steps:

1. **Immediately upgrade your PKCS #11 library to a newer version**
    - Latest PKCS #11 library for Amazon Linux, CentOS 6 and RHEL 6
    - Latest PKCS #11 library for Amazon Linux 2, CentOS 7 and RHEL 7

- Latest PKCS #11 library for Ubuntu 16.04 LTS

2. **Update your software to use a standards-compliant IV**

   We strongly recommend you follow our sample code and simply specify a NULL IV, which causes the HSM to utilize the standards-compliant default IV. Alternatively, you may explicitly specify the IV as `0xA6A6A6A6A6A6A6A6` with a corresponding IV length of `8`. We do not recommend using any other IV for AES key wrapping, and will explicitly disable custom IVs for AES key wrapping in a future version of the PKCS #11 library.

   Sample code for properly specifying the IV appears in aes_wrapping.c on GitHub.

3. **Identify and recover existing wrapped keys**

   You should identify any keys you wrapped using version 3.0.0 of the PKCS #11 library, and then contact support for assistance (https://aws.amazon.com/support) in recovering these keys.

> **Important**
>
> This issue only impacts keys wrapped with version 3.0.0 of the PKCS #11 library. You can wrap keys using earlier versions (2.0.4 and lower-numbered packages) or later versions (3.0.1 and higher-numbered packages) of the PKCS #11 library.

Resolving AWS CloudHSM cluster creation failures

When you create a cluster, AWS CloudHSM creates the AWSServiceRoleForCloudHSM service-linked role, if the role does not already exist. If AWS CloudHSM cannot create the service-linked role, your attempt to create a cluster might fail.

This topic explains how to resolve the most common problems so you can create a cluster successfully. You need to create this role only one time. Once the service-linked role is created in your account, you can use any of the supported methods to create additional clusters and to manage them.

The following sections offer suggestions to troubleshoot cluster creation failures that are related to the service-linked role. If you try them but are still unable to create a cluster, contact Support. For more information about the AWSServiceRoleForCloudHSM service-linked role, see Service-linked roles for AWS CloudHSM.

Topics

- [Add the missing permission]
- [Create the service-linked role manually]
- [Use a non-federated user]

Add the missing permission

To create a service-linked role, the user must have the `iam:CreateServiceLinkedRole` permission. If the IAM user who is creating the cluster does not have this permission, the cluster creation process fails when it tries to create the service-linked role in your AWS account.

When a missing permission causes the failure, the error message includes the following text.

```
This operation requires that the caller have permission to call
 iam:CreateServiceLinkedRole to create the CloudHSM Service Linked Role.
```

To resolve this error, give the IAM user who is creating the cluster the `AdministratorAccess` permission or add the `iam:CreateServiceLinkedRole` permission to the user's IAM policy. For instructions, see Adding Permissions to a New or Existing User.

Then try to create the cluster again.

Create the service-linked role manually

You can use the IAM console, CLI, or API to create the AWSServiceRoleForCloudHSM service-linked role. For more information, see Creating a Service-Linked Role in the *IAM User Guide*.

Use a non-federated user

Federated users, whose credentials originate outside of AWS, can perform many of the tasks of a non-federated user. However, AWS does not allow users to make the API calls to create a service-linked role from a federated endpoint.

To resolve this problem, create a non-federated user with the `iam:CreateServiceLinkedRole` permission, or give an existing non-federated user the `iam:CreateServiceLinkedRole` permission. Then have that user create a cluster from the AWS CLI. This creates the service-linked role in your account.

Once the service-linked role is created, if you prefer, you can delete the cluster that the non-federated user created. Deleting the cluster does not affect the role. Thereafter, any user with the required permissions, included federated users, can create AWS CloudHSM clusters in your account.

To verify that the role was created, open the IAM console at https://console.aws.amazon.com/iam/ and choose **Roles**. Or use the IAM get-role command in the AWS CLI.

```
$  aws iam get-role --role-name AWSServiceRoleForCloudHSM
{
    "Role": {
        "Description": "Role for CloudHSM service operations",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Action": "sts:AssumeRole",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "cloudhsm.amazonaws.com"
                    }
                }
            ]
        },
        "RoleId": "AROAJ4I6WN5QVGG5G7CBY",
        "CreateDate": "2017-12-19T20:53:12Z",
        "RoleName": "AWSServiceRoleForCloudHSM",
        "Path": "/aws-service-role/cloudhsm.amazonaws.com/",
        "Arn": "arn:aws:iam::111122223333:role/aws-service-role/cloudhsm.amazonaws.com/
AWSServiceRoleForCloudHSM"
    }
}
```

Retrieving AWS CloudHSM client configuration logs

AWS CloudHSM offers tools for Client SDK 3 and Client SDK 5 to gather information about your environment for AWS Support to troubleshoot problems.

Topics

- AWS CloudHSM Client SDK 5 support tool
- AWS CloudHSM Client SDK 3 support tool

AWS CloudHSM Client SDK 5 support tool

The script for AWS CloudHSM Client SDK 5 extracts the following information:

- The configuration file for the Client SDK 5 component
- Available log files
- Current version of the operating system
- Package information

Running the info tool for Client SDK 5

Client SDK 5 includes a client support tool for each component, but all tools function the same. Run the tool to create an output file with all the gathered information.

The tools use a syntax like this:

```
[ pkcs11 | dyn | jce ]_info
```

For example, to gather information for support from a Linux host running PKCS #11 library and have the system write to the default directory, you would run this command:

```
/opt/cloudhsm/bin/pkcs11_info
```

The tool creates the output file inside the `/tmp` directory.

PKCS #11 library

To gather support data for PKCS #11 library on Linux

- Use the support tool to gather data.

  ```
  /opt/cloudhsm/bin/pkcs11_info
  ```

To gather support data for PKCS #11 library on Windows

- Use the support tool to gather data.

  ```
  PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\pkcs11_info.exe"
  ```

OpenSSL Dynamic Engine

To gather support data for OpenSSL Dynamic Engine on Linux

- Use the support tool to gather data.

```
/opt/cloudhsm/bin/dyn_info
```

JCE provider

To gather support data for JCE provider on Linux

- Use the support tool to gather data.

```
/opt/cloudhsm/bin/jce_info
```

To gather support data for JCE provider on Windows

- Use the support tool to gather data.

```
PS C:\> & "C:\Program Files\Amazon\CloudHSM\bin\jce_info.exe"
```

Retrieving logs from a serverless environment

To configure for serverless environments, like Fargate or Lambda, we recommend you configure your AWS CloudHSM log type to `term`. Once configured to `term`, the serverless environment will be able to output to CloudWatch.

To get the client logs from CloudWatch, see [Working with log groups and log streams] in the Amazon CloudWatch Logs User Guide.

AWS CloudHSM Client SDK 3 support tool

The script for the AWS CloudHSM Client SDK 3 extracts the following information:

- Operating system and its current version
- Client configuration information from `cloudhsm_client.cfg`, `cloudhsm_mgmt_util.cfg`, and `application.cfg` files

- Client logs from the location specific to the platform
- Cluster and HSM information by using cloudhsm_mgmt_util
- OpenSSL information
- Current client and build version
- Installer version

Running the info tool for Client SDK 3

The script creates an output file with all the gathered information. The script creates the output file inside the `/tmp` directory.

Linux: `/opt/cloudhsm/bin/client_info`

Windows: `C:\Program Files\Amazon\CloudHSM\client_info`

> **Warning**
>
> This script has a known issue for Client SDK 3 versions 3.1.0 through 3.3.1. We strongly recommend you upgrade to version 3.3.2 which includes a fix for this issue. Please refer to the Known Issues page for more information before using this tool.

AWS CloudHSM quotas

Quotas, formerly known as limits, are the assigned values for AWS resources. The following quotas apply to your AWS CloudHSM resources per AWS Region and AWS account. The default quota is the initial value applied by AWS, and these values are listed in the table below. An adjustable quota can be increased above the default quota.

Service quotas

Resource	Default Quota	Adjustable?
Clusters	4	Yes
HSMs	6	Yes
HSMs per cluster	28	No

The recommended way of requesting a quota increase is to open the Service Quotas console. In the console, choose your service and quota, and submit your request. For more information, see the Service Quotas documentation.

The quotas in the following System Quotas table are not adjustable.

System quotas

Resource	Quota for hsm1.medium	Quota for hsm2m.medium
Maximum keys per cluster	3,300	16,666 total keys, with asymmetric keys having a maximum of 3,333
Maximum users per cluster	250	1,024
Maximum length of a user name	31 characters	31 characters
Required password length	8 to 32 characters	8 to 32 characters

Resource	Quota for hsm1.medium	Quota for hsm2m.medium
Maximum number of concurrent client connections per cluster[1]	900	900
Maximum number of PKCS#11 sessions per application	1,024	1,024

[1] A client connection for Client SDK 3 is a client daemon. For Client SDK 5, a client connection is an application.

Downloads for AWS CloudHSM Client SDK

The following topics provide downloads for AWS CloudHSM client SDKs.

> **Note**
>
> For information on what platforms are supported by each Client SDK, refer to AWS CloudHSM Client SDK 5 supported platforms and AWS CloudHSM Client SDK 3 supported platforms.

Topics

AWS CloudHSM latest Client SDK release

In March 2021, AWS CloudHSM released Client SDK version 5.0.0, which introduces an all-new Client SDK with different requirements, capabilities, and platform support.

Client SDK 5 is fully supported for production environments, and offers the same components and level of support as Client SDK 3. For more information, see Compare AWS CloudHSM Client SDK component support.

This section includes the latest version of the Client SDK.

Client SDK 5 release: Version 5.16.1

Amazon Linux 2023

Download version 5.16.1 software for Amazon Linux 2023 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 342e81846436708cfc1fb459a7ef1c7b065d8a68b50a5e63653c24918d0338a0)
- OpenSSL Dynamic Engine (SHA256 checksum 4e83bebcd20201c04629a03b70136df8d225d3056346789054c4e2ce8c9b3cba)

- JCE provider (SHA256 checksum 014e3c804a56087f8855ed30e480f9c193def0e1132a4769eae3ecd235d76c4f)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum 854c5c21f9fbef4f53a710340f213c272365c2bb1233106266eca5a5aa547e11)

Download version 5.16.1 software for Amazon Linux 2023 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum dbe9498d91e1e2e8f80fc870a4a6576b0dc801dc20cb7a82bfded79b3a362ac0)
- OpenSSL Dynamic Engine (SHA256 checksum 76cff941f36275163db146e05bf5fe64440248643ceb5dfcb3b64103f4f016a7)
- JCE provider (SHA256 checksum 8c38082797172b5630c74a8fc0c9e2652f0898bd2232b72911fbbaaa0260b5a6)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum 2a0640e32405a7db138afb2444a542eab11c0a7d9610d5406478c7ac4602a14b)

Amazon Linux 2

Download version 5.16.1 software for Amazon Linux 2 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum cfa59458ec239553c86ac55773bebc69a5dc7cdc08a3927917fd5918e10abe93)
- OpenSSL Dynamic Engine (SHA256 checksum 49e50bece7d73f3b7dc95eed1df856a6dee40e27f24f40f015c6a4a2e8dee839)
- JCE provider (SHA256 checksum 85c8853860aa36a6c54e75c94d607a334d64dd34683ba70430ff5be61eedcc56)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum 0efc3783e8429331aa446e45776fef46c7f9726a4768763ba4881dc2b05e090e)

Download version 5.16.1 software for Amazon Linux 2 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum dad60a0380ef0ea9c469cae4de10dc47124bbecedafb6956d009734cd49abceb)
- OpenSSL Dynamic Engine (SHA256 checksum 7ee303421d94544cbe9df03022c48af327833e631d8f1ec59a466673b6e9395d)
- JCE provider (SHA256 checksum 5ebe8157abb042ee92a7edfbfcb98bf79fc3a9907684565176fd28c387c08e88)

- [Javadocs for AWS CloudHSM] (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum 5e76b4a9021a3c92e59fc608c8263af731013835738977f376fb8ad9189add56)

RHEL 9 (9.2+)

Download version 5.16.1 software for RHEL 9 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 696e58f74d89bd8e39f85bddc547f8c94fa2fbca498318a7bb94f22e8be7a668)
- OpenSSL Dynamic Engine (SHA256 checksum 86197cd8bdd70db91331bb8380ea094352b4087c95a04768d2cefc3bba18dffa)
- JCE provider (SHA256 checksum 37ccc2df3e8aaddda74dc060f7f5bbe63e4769311f2ccbf313a8ab5d8831f206)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum 45a6db22db9ae64c239b49ea9281d29d2b4364dd63ee2408f463de17c4da1877)

Download version 5.16.1 software for RHEL 9 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum fd79fbccf504f53a6e70497135fcea4e20982fca75b628124c7510dcbaddf56c)
- OpenSSL Dynamic Engine (SHA256 checksum 23fe7407be9c5f8da0ae64e560a9741887bc31936f0b88c8d1490e3c6893a8a8)
- JCE provider (SHA256 checksum bd1c43e3e28c2f71e78a41342c064ff873fd8e052f89f90227fe00eaa28f089e)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum 3e877a232303052b8a6b2b869dcf228edbb0da913d3d41393f622831f4455a27)

RHEL 8 (8.3+)

Download version 5.16.1 software for RHEL 8 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum d396c53c229b1eaa7cb30d4fcd17addc9170c7942795d1c82b23a157ec379686)
- OpenSSL Dynamic Engine (SHA256 checksum c01f2cb66a6c5be839906b25c6a7f7990507b1b8ec3da34c128c1a90838e48df)
- JCE provider (SHA256 checksum 1745ab78de0712111659e91301f80e771e1b7ebc2ff8c4cc5e37bce591058520)

  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum b3ca40ba66062856ef63848c71f6e9dfa0a46a2b18d44c2d96a798fc4a4fb9cf)

Ubuntu 24.04 LTS

Download version 5.16.1 software for Ubuntu 24.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum ade113ee72547cb6a8fe91a4f3ac403f462d0acd0a306f01d16ac90699f9b59c)
- OpenSSL Dynamic Engine (SHA256 checksum 9e00cb32b16fbb286a853f65c5c5154c3e805ad67820d829a0c5343f12cc9e00)
- JCE provider (SHA256 checksum 4fab322a138e14372aba394de29f444af640479dd338539be84e0cf659c8993b)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum f8c1e369885eb77ba4517388dab47c067a617e860fb67a028a1ebde0f96acaef)

Download version 5.16.1 software for Ubuntu 24.04 LTS on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 5e15a69f8fb1429ed5c10429c91323fe82a63c0fe7bcc2f968820733d1737449)
- OpenSSL Dynamic Engine (SHA256 checksum f5136ee61bd34b74d59a01d37964b70a0f80ec5981f732b265af1c5466309e6f)
- JCE provider (SHA256 checksum acf5bebe7009461cdc1e31b95caadcbf80da09ec78d8f69142072da9baf61b79)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum 511376d2faf9f991f2193d08f80fc1edd53bf06daaa6fd8bd7a3fba0d6563ad1)

Ubuntu 22.04 LTS

Download version 5.16.1 software for Ubuntu 22.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum e1d787f10ee51d94732732811e4d2110f1b8e448e67fd47df8b53a2f7e56e3c8)
- OpenSSL Dynamic Engine (SHA256 checksum 29a9c41379754ce098a025feadc026f2f75a8638981f5b95ed07007d5b3d8510)
- JCE provider (SHA256 checksum 390d21e499ba181dc2774c3140433f6e60b49d9de3a892a89038c27e1aff157b)

  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum cadadafaa9464f944e5c2446df92fb1777a9e1e5116cab1b359a8aa101cf7ae8)

Download version 5.16.1 software for Ubuntu 22.04 LTS on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 3b6a3d50b4a55c150f12308ee3914451897bc4cc4c8420accf1d2d706316fcce)
- OpenSSL Dynamic Engine (SHA256 checksum c366a3c5faf4de32a6dedc4613234ba3d331b0abac3241bed7d25e0109a44f64)
- JCE provider (SHA256 checksum 705416b47bc62781b5a6dfc78dd02dadb4f18f842b569c3e210b9535813b9e70)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum 811b501615cf34665b70f103905094ca84ed7c126f72d91b040aacda99dbf22b)

Ubuntu 20.04 LTS

Download version 5.16.1 software for Ubuntu 20.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 6a2c33c78ada33fb435bbf8939b7cde3efe968e9f03250083dc6024b7ebd45b8)
- OpenSSL Dynamic Engine (SHA256 checksum 6ee178454e78d88e0ab92cd7c5b056d0c04cd6de192aea731d7ebcbd4c5ed761)
- JCE provider (SHA256 checksum 1e61ccd8d2b37c64467051d26e8d8bd592465ecef04282d6f4a9491707ba059d)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum 645693f2ede4fa0d7879eedaa41ae23120d902f24c9a637ba088f277e703cb96)

Windows Server 2025

Download version 5.16.1 software for Windows Server 2025 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 25beadaf28323412c60ed6695d89b6261d34da9cccf08869923b979854aa0329)
- JCE provider (SHA256 checksum 65f487f22c0786b80d81b387f5f4c8a6c9395c8d31a020c91fc0105829b8ec2c)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum d4dc9f5efd2b5e2eb07a59fb237f7f8dfd5476c859cdb9f91841354ce88783c4)

- Key Storage Provider (KSP) (SHA256 checksum 80f70eb3ba22d34e49b5b5da3fa183c86c751bff46229ae16fb83a5fc69a4d0c)

Windows Server 2022

Download version 5.16.1 software for Windows Server 2022 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 25beadaf28323412c60ed6695d89b6261d34da9cccf08869923b979854aa0329)
- JCE provider (SHA256 checksum 65f487f22c0786b80d81b387f5f4c8a6c9395c8d31a020c91fc0105829b8ec2c)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum d4dc9f5efd2b5e2eb07a59fb237f7f8dfd5476c859cdb9f91841354ce88783c4)
- Key Storage Provider (KSP) (SHA256 checksum 80f70eb3ba22d34e49b5b5da3fa183c86c751bff46229ae16fb83a5fc69a4d0c)

Windows Server 2019

Download version 5.16.1 software for Windows Server 2019 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 25beadaf28323412c60ed6695d89b6261d34da9cccf08869923b979854aa0329)
- JCE provider (SHA256 checksum 65f487f22c0786b80d81b387f5f4c8a6c9395c8d31a020c91fc0105829b8ec2c)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)
- CloudHSM CLI (SHA256 checksum d4dc9f5efd2b5e2eb07a59fb237f7f8dfd5476c859cdb9f91841354ce88783c4)
- Key Storage Provider (KSP) (SHA256 checksum 80f70eb3ba22d34e49b5b5da3fa183c86c751bff46229ae16fb83a5fc69a4d0c)

Windows Server 2016

Download version 5.16.1 software for Windows Server 2016 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 25beadaf28323412c60ed6695d89b6261d34da9cccf08869923b979854aa0329)
- JCE provider (SHA256 checksum 65f487f22c0786b80d81b387f5f4c8a6c9395c8d31a020c91fc0105829b8ec2c)
  - Javadocs for AWS CloudHSM (SHA256 checksum 70320f9aee48a250f9384f626fc16db00ea35c32af4c29cd63e9c82065a0fe1b)

- CloudHSM CLI (SHA256 checksum d4dc9f5efd2b5e2eb07a59fb237f7f8dfd5476c859cdb9f91841354ce88783c4)
- Key Storage Provider (KSP) (SHA256 checksum 80f70eb3ba22d34e49b5b5da3fa183c86c751bff46229ae16fb83a5fc69a4d0c)

Client SDK 5.16.1 adds support for signing and verifying prehashed data in the CloudHSM CLI.

Platform support

- SDK 5.16.1 is the last release to provide Ubuntu 20.04 LTS platform support. For more information, see the Ubuntu website.

CloudHSM CLI

- Added support for signing and verifying prehashed data in the CloudHSM CLI. For more information, see The crypto sign category in CloudHSM CLI and The crypto verify category in CloudHSM CLI.

JCE provider

- Updated AES/CBC/Pkcs5Padding Encryption mode to automatically generate a random Initialization Vector (IV) when null IV is provided. Previously, null IV would result in an operation failure. Explicit IVs remain mandatory for decrypt operations.

Bug fixes/Improvements

- Reduced latency for repeated digest update operations in JCE.
- Updated the `generate key-reference` command to correctly name KSP key reference files based on their Attribute ID values when running on Windows Server. For more information, see Generating KSP key references (Windows).

AWS CloudHSM previous Client SDK releases

This section lists previous Client SDK releases.

Version 5.16.0

Amazon Linux 2023

Download version 5.16.0 software for Amazon Linux 2023 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum d63271304f32f49838390a58d94a2140ae8a744ac64efcf0e6a65983c858d862)
- OpenSSL Dynamic Engine (SHA256 checksum 25d78c1df82355601ed6887bb47d64a06380d001da15a070cfa89dce65417fb6)
- JCE provider (SHA256 checksum fcd876813e5d114e2b042765ff98b256410e988a4e373ea1aae6f260aebdd554)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum 7f1441eb08daa7ceab91161900d89974ee86e0a3c54d878a29e43db775bd04f5)

Download version 5.16.0 software for Amazon Linux 2023 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 0b42e4e89cddda10c5be5f295a52d7a541bba057a3a07d3d9b192cf9eb49c776)
- OpenSSL Dynamic Engine (SHA256 checksum e8ee949d44b9f9ba64ed36eb7944acacbd1d76d7cf09d5492deae5928a9953e9)
- JCE provider (SHA256 checksum 67a6fd8f7b5dfc60ec6d6d380c31035675373b542205ec7e5a687175191cd275)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum 1e6c3053b19aded54045a3baa9fe2545f81a35332366cf2d77dca92ee888e654)

Amazon Linux 2

Download version 5.16.0 software for Amazon Linux 2 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 5a89ff9801f89f51e27e70869f2713cfe9bbd87d1198246cffaafe74aff9e809)
- OpenSSL Dynamic Engine (SHA256 checksum 991b6288289d07972915aa0be6dfd6b3c33c1d6312bc304225715e30832e688f)
- JCE provider (SHA256 checksum c5e673f79c2efd83195a288a5c8860c4bf74dba5bc8e422741dd5cc1be230fb3)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum b9021b0b348c08f433e7cce247ab1710e2a79f33a82ac36e1d6a4672171028db)

Download version 5.16.0 software for Amazon Linux 2 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 97677a7f9759637b548205d31c9a08a3e7b27dfa5a38e9a4e8e2398ae9869a5a)
- OpenSSL Dynamic Engine (SHA256 checksum dde369dc4efd9f59e5c9a5459a532ad5d273f87839fba6dd5dcde2c3a2b61517)
- JCE provider (SHA256 checksum b7d96e7453e26ff07ccd9437b102f4c21857c79daa6315054728fc7ac66150bc)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum c7386ac016b14ea8cb96a6ed655a67f46826cdb925bd87902277b5ce9b593d51)

RHEL 9 (9.2+)

Download version 5.16.0 software for RHEL 9 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 178d3e1ff0ec8cdc7f8c8be9aab772b4195602bb1c48dea692b628689be5ea3d)
- OpenSSL Dynamic Engine (SHA256 checksum 0488ad66cd825fafee70d1a1ccb1c8045b089af7b7d6ccadbec75f535376048c)
- JCE provider (SHA256 checksum cc9a60fa981a46c674de1e715d92ec1b8cd5c801394bfb3f0d101880589d4202)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum cb7e3ce6704f015d5b9462f12e6ad5489366fb3e80f554a2bdf7f04e688be1a9)

Download version 5.16.0 software for RHEL 9 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 9567fb016b641821869919d4d6e9754ae08fd549381446640ff135a1e225b809)
- OpenSSL Dynamic Engine (SHA256 checksum 8c46a9dd908e77a0d8412e7050621d0c570e091845480815c230e26d1917227b)
- JCE provider (SHA256 checksum 78a7c714aedd068e5e2f12f5044a6ce1618e6dfacbcdd91b4a9b505842dcc6c9)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum d2340ef4092ca6c1fa1ac8b484550508a5d9d8670e663bef6e595a07f8151be3)

RHEL 8 (8.3+)

Download version 5.16.0 software for RHEL 8 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 47bf628a4a2663b8e8c7207b43e5af1d05ed2247e85e1371773596c132c9a4ec)
- OpenSSL Dynamic Engine (SHA256 checksum 49dd44243863121f7781e0f4ac556828c746d85db723864ccf16ea5fb042846a)
- JCE provider (SHA256 checksum c5e8a6664e6a8e5338a98b3845bed029f5936b517e6043aeb65ad58c3364eed8)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum 71f2cd6bd8892ccaea845bfe575cdf20016dd7176e5216b76290e499c822a2f2)

Ubuntu 24.04 LTS

Download version 5.16.0 software for Ubuntu 24.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum fd7e78050aba017b7dc859f0b1e6f354f56b22091cb262b328aad204064a2960)
- OpenSSL Dynamic Engine (SHA256 checksum 9d2a9e8049caa12c8027ad2963fa30f22e1032d4025e39a4e2d5985073c283f5)
- JCE provider (SHA256 checksum ae19c13f55e01538e989e8693aaf3a5d998c60f1ed71e721a34404a228947c20)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum 7ebedb3e8c4b8e01cc1e5dfe32ae18c9542d9c2940c77bd71ee102093aaf1d1c)

Download version 5.16.0 software for Ubuntu 24.04 LTS on ARM64 architecture:

- PKCS #11 library (SHA256 checksum cfd9526cf495e81a2477b464ca3c124a89d07b1a366a14b39eeb27d6b5e2404b)
- OpenSSL Dynamic Engine (SHA256 checksum 02a5cc2cd9442b7bf982bb3ae44c9b134758b46e68e46fe4729b4adf1320d75c)
- JCE provider (SHA256 checksum 124d04f667a7222d624f528404a289405669f4e4a1983b8a7926758bf67e35ec)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum b874556b34ff5f0d082ca19f62822cf97d6e7074ce3861d7577090f99cba0cbd)

Ubuntu 22.04 LTS

Download version 5.16.0 software for Ubuntu 22.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum e36d9bf900e195e417db261034d820cb935bec5135a0aa332d46019c70257cac)

- OpenSSL Dynamic Engine (SHA256 checksum 2c39129a5acce96b8693ce8c4b4aa174d9c13bc8ea3c2652fa505b4e0a933842)
- JCE provider (SHA256 checksum 03b70d834985a2500b73eecd831bff4f30389eb67460bf1d799466e59a9f9cfd)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum 3fc4fc662188ebbd430d9b5e01a57774567b926f3478d0a455fee4968c47460d)

Download version 5.16.0 software for Ubuntu 22.04 LTS on ARM64 architecture:

- PKCS #11 library (SHA256 checksum c835d9bb604b5c47a702fd45267ffb9dfa37d68dd0734391f060835ea680a169)
- OpenSSL Dynamic Engine (SHA256 checksum 943213a0f14b50fbdf3a90286e7f7ab219ab77750698f5062013873419484a04)
- JCE provider (SHA256 checksum 0ecc3e65a3b94c2dd3913f44aba58764e5558a8bedd3b742a42e4769533f3734)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum 8558328b224b850cf9e1b3c4f8773565a90e38ab43e2c9e90bc5b841f98898d2)

Ubuntu 20.04 LTS

Download version 5.16.0 software for Ubuntu 20.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 64a01e84d898aca3cd03f22022cf8dbbc806a1735a84df5f820ba5bfa3339b0e)
- OpenSSL Dynamic Engine (SHA256 checksum 53a73cfc60c3fddd3ad9173bf6e602faf289f20935b3c469a137847148442954)
- JCE provider (SHA256 checksum 09544cb8129af4d06a241e1fbc0b6e6c207e1a518e5cb202f7b6c9802507d1b8)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum e1fdf1e014541f57d3ba4688bad88e7557e32ba974e020b4da76f85f1fc6aa29)

Windows Server 2025

Download version 5.16.0 software for Windows Server 2025 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 6d3142d178127372de9ab40e7af5dc81fa7c627a18286dd521ad786d0947f17d)
- JCE provider (SHA256 checksum 9f5c28583b5127b90e753dd7ce6d081bbe4810c0c30c424192704efa6ad1be34)

  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum 22cfed533f528e0975f452af51d334baa1b5c143e722182dca107622e703b1ca)
- Key Storage Provider (KSP) (SHA256 checksum 9045c4f3e81093c4b49b93f3ea3f5caf2a4e6980628e0db2d971c437dc203bfc)

Windows Server 2022

Download version 5.16.0 software for Windows Server 2022 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 6d3142d178127372de9ab40e7af5dc81fa7c627a18286dd521ad786d0947f17d)
- JCE provider (SHA256 checksum 9f5c28583b5127b90e753dd7ce6d081bbe4810c0c30c424192704efa6ad1be34)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum 22cfed533f528e0975f452af51d334baa1b5c143e722182dca107622e703b1ca)
- Key Storage Provider (KSP) (SHA256 checksum 9045c4f3e81093c4b49b93f3ea3f5caf2a4e6980628e0db2d971c437dc203bfc)

Windows Server 2019

Download version 5.16.0 software for Windows Server 2019 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 6d3142d178127372de9ab40e7af5dc81fa7c627a18286dd521ad786d0947f17d)
- JCE provider (SHA256 checksum 9f5c28583b5127b90e753dd7ce6d081bbe4810c0c30c424192704efa6ad1be34)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum 22cfed533f528e0975f452af51d334baa1b5c143e722182dca107622e703b1ca)
- Key Storage Provider (KSP) (SHA256 checksum 9045c4f3e81093c4b49b93f3ea3f5caf2a4e6980628e0db2d971c437dc203bfc)

Windows Server 2016

Download version 5.16.0 software for Windows Server 2016 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 6d3142d178127372de9ab40e7af5dc81fa7c627a18286dd521ad786d0947f17d)

- JCE provider (SHA256 checksum 9f5c28583b5127b90e753dd7ce6d081bbe4810c0c30c424192704efa6ad1be34)
  - Javadocs for AWS CloudHSM (SHA256 checksum bbbbe99e427b7b4d51d7018a97c4d44372db020e44fdbb76fd3954fb104010e3)
- CloudHSM CLI (SHA256 checksum 22cfed533f528e0975f452af51d334baa1b5c143e722182dca107622e703b1ca)
- Key Storage Provider (KSP) (SHA256 checksum 9045c4f3e81093c4b49b93f3ea3f5caf2a4e6980628e0db2d971c437dc203bfc)

Client SDK 5.16 adds ECDH with X963 KDF support on hsm2m.medium cluster type for JCE provider and PKCS #11 library. Client SDK 5.16 also adds support for generating KSP key reference files on Windows Server with CloudHSM CLI.

CloudHSM CLI

- Added support for generating KSP key reference files on Windows Server, for more information, see Generating KSP key references (Windows).

JCE provider

- Added ECDH with X963 KDF support on hsm2m.medium cluster type, see Supported mechanisms for JCE provider for AWS CloudHSM Client SDK 5.

PKCS #11 library

- Added ECDH with X963 KDF support on hsm2m.medium cluster type, see Supported mechanisms for the PKCS #11 library for AWS CloudHSM Client SDK 5.

Bug fixes/Improvements

- Fixed a bug where session keys were not properly cleaned during disconnection.
- Fixed an issue where successful mTLS logout operations on hsm2m.medium incorrectly returned an error response.
- Fixed an issue with connection failure during the windows startup.
- Fixed an issue where SDK info tools displayed incorrect output on Ubuntu systems.
- Fixed an issue that previously allowed ECDSA with weaker hashes in FIPS mode.

Version 5.15.0

Amazon Linux 2023

Download version 5.15.0 software for Amazon Linux 2023 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 41ef3178811df1dbb03b2cbac83fe0f4768bdc9b17005c409f1c0229f93ef11c)
- OpenSSL Dynamic Engine (SHA256 checksum afa1f9f8bd99f54866dea1b8928c00b951a6e492f5f36d0d6c7c38fff341d609)
- JCE provider (SHA256 checksum 6ea775e05570ef3497a4df5c35a6ec1c682aea73c48e7fecec3e541af995759e)
  - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum 78c10bb213dd14fcfc5836e358de6aedac61db05125fd61137b0082214fdecbe)

Download version 5.15.0 software for Amazon Linux 2023 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum cd0617d29b6d64c00c8e14fd9a92f604e14acda4746f4f0b86b9de42367192fb)
- OpenSSL Dynamic Engine (SHA256 checksum bc9acfdd04eb1246eb3d5b0a8f3736ec017c0d1699d5395f85868d4a1722cd83)
- JCE provider (SHA256 checksum 96d05301486206577b7be05aac561649167b57272c9c06ff839fd8ff2b5d96d5)
  - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum b870eadb27736a2cde98022d57e9704c67ae15878cf0b910738859cdabaa35a2)

Amazon Linux 2

Download version 5.15.0 software for Amazon Linux 2 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum c70ae4f0181a8187c9380481c51c1d03e12236dd86863ec818ed3f210b294c8e)
- OpenSSL Dynamic Engine (SHA256 checksum 08e9fd1dd80efa637f9a1727bb0de205ba124a3776b2e8bc21008ee458063a42)
- JCE provider (SHA256 checksum 7932ed060e72c53b2556f30694b0ffe5342b244b6628c7a9dc03966aa49c8fe6)
  - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum 0a939e4d5d0a2ff308c7a1d9e73ebc865e426214d556bde1bd29dbe807fbb583)

Download version 5.15.0 software for Amazon Linux 2 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum bcf907a8fb4722b09c54b8e5785fe2b8cffcedd6ee3fdda2d21879a012138077)
- OpenSSL Dynamic Engine (SHA256 checksum 8f70edc3a6a4a1bf0264c6567b1dda1ac69055f206753f88eeadbb8bf3bf9f38)
- JCE provider (SHA256 checksum 75dd67736bb08fe7e46e113af10803a255ba8edee3016ca963c1ee94fe59d43b)
  - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum 1c2ecf90c955281d99bcd8d1956d63debb15bbc8419744c83f88821ef8b78aee)

RHEL 9 (9.2+)

Download version 5.15.0 software for RHEL 9 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 65bd0b815eebc806674a7bf7c54e9f884595881547f5fffd08ff6a38aabdccbe)
- OpenSSL Dynamic Engine (SHA256 checksum f2af9f5882ab2e5a11defecc660f8af5c4d9d6e2e2b89873e6833fc2976f44ac)
- JCE provider (SHA256 checksum 5124bace5d1544775c891f13a0e309b30dd73699116c46ecc9a77bba5f9cf633)
  - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum 1eec31fb2c0ab3c2839d9bdb37874cf8dbc74a383279068fce9e8613966d06e0)

Download version 5.15.0 software for RHEL 9 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum f498697519afa04b6b0362a4388a6a38e2cb6813b781d6ce3d97a7de89c5cbfe)
- OpenSSL Dynamic Engine (SHA256 checksum 5410da63108a1b209e567e9bccf8bd7e4035af88b0d58b9d78b10917be1b40c1)
- JCE provider (SHA256 checksum b46f233e6994d2c0ed505dc5c717ee3009daaeb2063d260aa78e06273770bffa)
  - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum e9f93eaa58db2f7ea1174164ef96ab219700933d353243c3c6ab1aebac5ccffe)

RHEL 8 (8.3+)

Download version 5.15.0 software for RHEL 8 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 87131e179d0e60ade302ec07b22803cfb39294bf060b786c41f154d95791ac94)
- OpenSSL Dynamic Engine (SHA256 checksum f412a2f5cd761db5940288bb252ce060d44735c6b436bb6d4fa7d3687a44a026)
- JCE provider (SHA256 checksum d0844f55c08f9ff9c393138a9041efe1b59dc3dce20c0b2c23406efe6acc43db)
  - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum 9fd8033e478ce6d7d640c063c4f007359cb04c19d519826a745ad0885f96a0f8)

Ubuntu 24.04 LTS

Download version 5.15.0 software for Ubuntu 24.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum ca5f2f80ae921cfebdc5c8bc35c39d2b19cfabfd5981932409eaf2e7c00a9097)
- OpenSSL Dynamic Engine (SHA256 checksum e44cd7b678a421957c84e4fc0f70280360fd4e1e66f4cabdd1b20b955ee5fcca)
- JCE provider (SHA256 checksum bc6382f12769c3d87c34522d0d616b7a8c108574c003814e1300938c386655ac)
  - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum f786be86c204680e670f7817376ff376733ebd96b51c31e9c98213078596637f)

Download version 5.15.0 software for Ubuntu 24.04 LTS on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 1ecf057d67137c0e9e1bb0dda57d581690c647b0360c3a617a8dc668919d08de)
- OpenSSL Dynamic Engine (SHA256 checksum 4f3d23e6b798f88be587108d8e6a225d796a3d080aef61ea384eb74a1270612a)
- JCE provider (SHA256 checksum b9fd16bdcc1fcf59fde0d3e0debee500b0b7edfdff69209e84d14393097fe9d2)
  - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum a95922d9b44e64a64723db0c21ac89566515a5a6c87de990af4a9e1f40c7424f)

Ubuntu 22.04 LTS

Download version 5.15.0 software for Ubuntu 22.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 966be12eb32de813ca07e766abf7b5616c0d2e105e9296d920aadaca10e5afdf)

- OpenSSL Dynamic Engine (SHA256 checksum f2840151d87b7f9cbff68993c25397afd48a16f054abf0f2fd4624662d3087d6)
- JCE provider (SHA256 checksum 01aebdda05640e50a82cae04c5cb6d33ab909dadd917bd834957d4f75ad8c577)
  - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum 2e5eba9409abc429828505779c512cd2424719766c0e488f50aee288966cf61d)

Download version 5.15.0 software for Ubuntu 22.04 LTS on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 614f74f101c9c64fff515128c5a59fa23de5047494fe248e8aabdab441092b3e)
- OpenSSL Dynamic Engine (SHA256 checksum c15d6db77b76bce690749b73b947567eb5f2d76669887843116d0ce56c1f8ea7)
- JCE provider (SHA256 checksum bca8511d5c0a173b0fef326016ce5091b6e6829fa2b3cb45ed5621290ae3e42a)
  - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum 2f5b673148b682d7e34619c51b5e8799abe7dc7fd4f046158a0d05320ba24dc1)

Ubuntu 20.04 LTS

Download version 5.15.0 software for Ubuntu 20.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 86c8394b5ddff91a71194fb87c327efde36baa2380e559c04f9d543a6e74d61b)
- OpenSSL Dynamic Engine (SHA256 checksum 0a4227389fea61e6e7ac7cfa715eb341f7a4eeae9ed10e4c96da2c0dd4a18f9e)
- JCE provider (SHA256 checksum 65b2d926ff9dfbe6c7864bc3a41b3da2383bc731dd199bcef8805a0543fbe612)
  - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum 09c2e55bcf72f9e530717950d8c5fdfd48574ae6ccb09a049526ca5b2a3b8aa9)

Windows Server 2025

Download version 5.15.0 software for Windows Server 2025 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum a903b63fe286f15bf669c0555b1fa4d86b33592ed05af0809acac28c0d3ace16)
- JCE provider (SHA256 checksum fdef6251f06d77d51fddbc2184d3eec87ddec4fe35b3ac620343eb66c95ddf64)

    - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum 6cc9205fe1fc514ddd9774f824e512940d6b1bc4cc0e251265bc46ab99746c28)
- Key Storage Provider (KSP) (SHA256 checksum 52ed9b08cd0ce100b8dcd3d8e8f411b6201f9f1b27872b19d1136c0bf36a29b8)

Windows Server 2022

Download version 5.15.0 software for Windows Server 2022 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum a903b63fe286f15bf669c0555b1fa4d86b33592ed05af0809acac28c0d3ace16)
- JCE provider (SHA256 checksum fdef6251f06d77d51fddbc2184d3eec87ddec4fe35b3ac620343eb66c95ddf64)
    - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum 6cc9205fe1fc514ddd9774f824e512940d6b1bc4cc0e251265bc46ab99746c28)
- Key Storage Provider (KSP) (SHA256 checksum 52ed9b08cd0ce100b8dcd3d8e8f411b6201f9f1b27872b19d1136c0bf36a29b8)

Windows Server 2019

Download version 5.15.0 software for Windows Server 2019 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum a903b63fe286f15bf669c0555b1fa4d86b33592ed05af0809acac28c0d3ace16)
- JCE provider (SHA256 checksum fdef6251f06d77d51fddbc2184d3eec87ddec4fe35b3ac620343eb66c95ddf64)
    - Javadocs for AWS CloudHSM (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- CloudHSM CLI (SHA256 checksum 6cc9205fe1fc514ddd9774f824e512940d6b1bc4cc0e251265bc46ab99746c28)
- Key Storage Provider (KSP) (SHA256 checksum 52ed9b08cd0ce100b8dcd3d8e8f411b6201f9f1b27872b19d1136c0bf36a29b8)

Windows Server 2016

Download version 5.15.0 software for Windows Server 2016 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum a903b63fe286f15bf669c0555b1fa4d86b33592ed05af0809acac28c0d3ace16)

- [JCE provider] (SHA256 checksum fdef6251f06d77d51fddbc2184d3eec87ddec4fe35b3ac620343eb66c95ddf64)
  - [Javadocs for AWS CloudHSM] (SHA256 checksum ce5d92731f2a9c7f46c92bba92f3d0adff4ceb06b9d653597994100dfc352fbe)
- [CloudHSM CLI] (SHA256 checksum 6cc9205fe1fc514ddd9774f824e512940d6b1bc4cc0e251265bc46ab99746c28)
- [Key Storage Provider (KSP)] (SHA256 checksum 52ed9b08cd0ce100b8dcd3d8e8f411b6201f9f1b27872b19d1136c0bf36a29b8)

Client SDK 5.15 adds support for replicating a user across cloned clusters with CloudHSM CLI. Client SDK 5.15 also adds installation packages for PKCS #11 library, JCE provider, CloudHSM CLI, and Key Storage Provider (KSP) for Windows Server 2025.

Platform support

- Added support for Windows Server 2025 for PKCS #11 library, JCE provider, CloudHSM CLI, and Key Storage Provider (KSP).

CloudHSM CLI

- Added the following command:
  - Replicate a user with CloudHSM CLI

Version 5.14.0

Amazon Linux 2023

Download version 5.14.0 software for Amazon Linux 2023 on x86_64 architecture:

- [PKCS #11 library] (SHA256 checksum 05e7a3882166c694a7a09bc735f08f91c8145a4215176665eacacdf3e509abe8)
- [OpenSSL Dynamic Engine] (SHA256 checksum f4dd9966988418e100c276dc0d521f91afdfc0e6c008dbf8eda446ebaca83c14)
- [JCE provider] (SHA256 checksum c4dee5c1173f6a1c7683aedb58e61d329c36933435c416f288d94bc9a68a6b31)
  - [Javadocs for AWS CloudHSM] (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- [CloudHSM CLI] (SHA256 checksum 1fad069fde305450254287e43750d597db9af0cb8fd168300cd5eaed9e2af33a)

Download version 5.14.0 software for Amazon Linux 2023 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 9f34163d02bce26c1280e589310cda891d27995c50cec0a7fe083100ecff2b69)
- OpenSSL Dynamic Engine (SHA256 checksum 7444a4daad6e4715c82d6c39a7b03a07ee0201a13fe0f98da96acbf9d24abf6c)
- JCE provider (SHA256 checksum 3393fe3a0f5c3a9c92106d74b7de074c818e095f97d2cfd600dbd47779b90b37)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum d9a2edce48c5f6646d5a351cb431712f2d2fc62d21f8318e7bb1ce579819d7f4)

Amazon Linux 2

Download version 5.14.0 software for Amazon Linux 2 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 9c47b90bfa0ad51627cdb0dd8f148a56090fbdeb2490f1ab4009170c7b9c1120)
- OpenSSL Dynamic Engine (SHA256 checksum 215f9768331565085a317585b3dbe0514b251fdc428c96ed32491c4abb9fea56)
- JCE provider (SHA256 checksum a802f941e95fcbf0ef37775fc096c0d6ae4c916fa08330a9d56defc5f99ff2b7)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum ac10f23dd81264b1d10a0760f62b5006d2b2b66bf1c6378248ca9326afb65a83)

Download version 5.14.0 software for Amazon Linux 2 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum f2d1550f043c655ef5d51e2ca14a0416886d99902e0702bd7f30f93b2c563d4d)
- OpenSSL Dynamic Engine (SHA256 checksum 2c8e2c81af53ba3646d1f947894d84b9b54780bbca79b58f354125e4ac9427c0)
- JCE provider (SHA256 checksum 3f9c881056e6905d46358585db72143b59971958a708fa4ac75cb53994487213)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum 6d5985e44c9852409dd3d342239fbbf7d0f9ad43d445e0980153c9b9eafe2b6f)

RHEL 9 (9.2+)

Download version 5.14.0 software for RHEL 9 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum c4812210421ff2fa5dac8477a2e6b10552aabd88b1f03d717e044e7293823158)

- OpenSSL Dynamic Engine (SHA256 checksum 3f6aeaa6ae1faae7d8bba1596f358cad1eec9e562cf08aaab9ded92cabe94719)
- JCE provider (SHA256 checksum bd8f120b08f738ad4d1534b0e32aad903758d5e75d3aba7cb9ff6a77dec533db)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum 84f13a9babd767edf90dabb0b14034ec5b2208898123a13966dd6b8519961c27)

Download version 5.14.0 software for RHEL 9 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 3faa1de563aa9773939cab7b39907dcb7b981ed7225019f9070a9dd52db7ae70)
- OpenSSL Dynamic Engine (SHA256 checksum 13b41ac47ef7ee7bf78f585b8347ca4da9ebc296e4fc1e6a0c2ff5b333354ca6)
- JCE provider (SHA256 checksum 06803655ebe54d59c180bb17ac6fe56337bdd58ad0f6fe87c50ff8df32f70258)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum e6b4e9688d0db9d72bbee3450fe19736d640c9931adbd4f4ef73cb7ac2a08cf4)

RHEL 8 (8.3+)

Download version 5.14.0 software for RHEL 8 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum e400aeea6dbf7721f97e71c643a0db4f5f1094fb197fc46dc0ab293de9d16f2d)
- OpenSSL Dynamic Engine (SHA256 checksum bfcc27d251e62f9eba0fd508e7d08dc62126642d4cdd0b5566183957768b8c54)
- JCE provider (SHA256 checksum b5500031b572c918a8df4a0347e01c8ea00a7366b865b310bd92427fa1ed53e3)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum 000712d0a691efc64a6c5d54bfeb1ab48b315ebb5dd6926b0502e326bf700291)

Ubuntu 24.04 LTS

Download version 5.14.0 software for Ubuntu 24.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum bbec70a198a4b173620b4018accc297ce6a6a80d372706e0101997d6bca35bca)

- OpenSSL Dynamic Engine (SHA256 checksum fcb77f75bd465b22401a09a20c4109858333402951012 63b7171cdcc4ac9f980)
- JCE provider (SHA256 checksum e1fa16aae2f6095c89a8bf392ef2cb9ca4db8853c858904ff90abf4bb491d74b)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum af32e0ac1c5f8c7f8cb40d255abe7c63ce9d981293187a3fb452562fa05756f4)

Download version 5.14.0 software for Ubuntu 24.04 LTS on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 1232318b084347889f92136536537ca519373683ce39bcec20f6ac3fa2f42d7c)
- OpenSSL Dynamic Engine (SHA256 checksum 1745ee3a33d8e6ea72644e903c55c6a206204cf0a8bea200bc4a7b15736ed801)
- JCE provider (SHA256 checksum 7a44acabbc90c996594ed53661937f9242850823347f7c386f02fc041a97471a)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum e6778bd12c55fd152b50033833531fe569472f4f2bd9927a345eb126e8305739)

Ubuntu 22.04 LTS

Download version 5.14.0 software for Ubuntu 22.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 6b4b1620e9a85267950633b171dd188b7ac7094e371e188fabc1bef7a911a16f)
- OpenSSL Dynamic Engine (SHA256 checksum fee5f0a65fab0f46ad58689af5dc510721581f31364d3be5cbbf79f5d9a60db8)
- JCE provider (SHA256 checksum 5883a3d15e160d65f8e26f185e1ee30f68becad5f6fcd16abc1c4586689800b5)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum 7b1a8cd2962f4be7325154c080dabfb820beac5629e272c3a3ebc0e6cab11d27)

Download version 5.14.0 software for Ubuntu 22.04 LTS on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 8680f63fd74272ea0e4d4c32dfb96b3eb9b60e03483c202d4e9b65ee101a178f)
- OpenSSL Dynamic Engine (SHA256 checksum c757304e8fc5f38be3bab7ac6d37a35dcb56d31e62f7194da62b3a176593d1d8)

- JCE provider (SHA256 checksum a78ee59341da56af315d126ec7ed9f0dafe6e99649f659db2436be3863cb035b)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum 18931bff869a0bd54846b3296d870fa19beba5652f979352be8fba6307e6d1aa)

Ubuntu 20.04 LTS

Download version 5.14.0 software for Ubuntu 20.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum f1461c16b135ebcc17deec46aab88bd113ea122b8942fc188d4f05cd03e919a8)
- OpenSSL Dynamic Engine (SHA256 checksum 89211a7a7ed50eda2dc385c31ea76f1fbabd389ca691204873531d983c3eb0f7)
- JCE provider (SHA256 checksum f098c32d61a53b073459a75d88b68e377a9b16335874fae060cb10df0da00df0)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum ff91fb930717c917344af2ba344dc6e02bd5abc004dcb6147e9412b67e2aa7ab)

Windows Server 2022

Download version 5.14.0 software for Windows Server 2022 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum ee5a8e8e85fff7acd0bbafa23740e3981f7dc52e708972b600c2b26603786838)
- JCE provider (SHA256 checksum 2ae0274f09f66981c03fd1e3c264e896ba7cd211168ea31369335db1b3ea2e77)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum 3938654f88ce010042e48909e23991e178e411e2bd9e3c9ec25fcb8157c0cd55)
- Key Storage Provider (KSP) (SHA256 checksum b026e4d8c11e9ff6f22a7b9e10b8bb29e7572665f0d7978a3cef7d2354b7693f)

Windows Server 2019

Download version 5.14.0 software for Windows Server 2019 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum ee5a8e8e85fff7acd0bbafa23740e3981f7dc52e708972b600c2b26603786838)
- JCE provider (SHA256 checksum 2ae0274f09f66981c03fd1e3c264e896ba7cd211168ea31369335db1b3ea2e77)

  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum 3938654f88ce010042e48909e23991e178e411e2bd9e3c9ec25fcb8157c0cd55)
- Key Storage Provider (KSP) (SHA256 checksum b026e4d8c11e9ff6f22a7b9e10b8bb29e7572665f0d7978a3cef7d2354b7693f)

Windows Server 2016

Download version 5.14.0 software for Windows Server 2016 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum ee5a8e8e85fff7acd0bbafa23740e3981f7dc52e708972b600c2b26603786838)
- JCE provider (SHA256 checksum 2ae0274f09f66981c03fd1e3c264e896ba7cd211168ea31369335db1b3ea2e77)
  - Javadocs for AWS CloudHSM (SHA256 checksum 173087a8f286dc4f88dc915ecc00aa515bec0ae2faf219654df9f3422d8e83bb)
- CloudHSM CLI (SHA256 checksum 3938654f88ce010042e48909e23991e178e411e2bd9e3c9ec25fcb8157c0cd55)
- Key Storage Provider (KSP) (SHA256 checksum b026e4d8c11e9ff6f22a7b9e10b8bb29e7572665f0d7978a3cef7d2354b7693f)

Client SDK 5.14 adds support for quorum controlled key usage and key management operations using CloudHSM CLI. Client SDK 5.14 also adds support for Key storage provider (KSP) for AWS CloudHSM Client SDK 5 for windows platforms. Additionally, Client SDK 5.14 adds installation packages for PKCS #11 library, JCE provider, CloudHSM CLI, and Key Storage Provider (KSP) for Windows Server 2022.

Platform support

- Added support for Windows Server 2022 for PKCS #11 library, JCE provider, CloudHSM CLI, and Key Storage Provider (KSP).

CloudHSM CLI

- Added support for quorum controlled key usage and key management operations.

Key Storage Provider (KSP)

- Added support for Key Storage Provider (KSP), a cryptographic API specific to the Microsoft Windows operating system. For more information, see Key storage provider (KSP) for AWS CloudHSM Client SDK 5

Version 5.13.0

Amazon Linux 2

Download version 5.13.0 software for Amazon Linux 2 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum cefbcfe15f0bed09a2bc9b0c15824067dfede8ceb1ad6373659c7e583a604c95)
- OpenSSL Dynamic Engine (SHA256 checksum 7b384253f0a124b55092e6ab18e23d9c95067d55fa8167ef7817bd2ae1becd29)
- JCE provider (SHA256 checksum cfac14b593b027bdb8010d6019328e7129143be06ffe223d2d50c4b7e1ac747a)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum 6b762f0884368d2e234c5f6d45b4aeefb52d686105ec2c1affdbcdbb8dda7500)

Download version 5.13.0 software for Amazon Linux 2 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum ed0352cb33b4cb9fd3d2a00a8654f53e7290535474641a1714151b4190c1de07)
- OpenSSL Dynamic Engine (SHA256 checksum 5e55e24175167f38a7358178ba252cb7629def0de4c99eee8a25d44649ebe5ec)
- JCE provider (SHA256 checksum 4e19807e792f10ffd9819d381f02ad1485aaf45fee7f660054211b8f52224ed2)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum bf90dec12f39eb685df34d82fdf2dac1c87a86fbf8a03aabde2107113081a083)

Amazon Linux 2023

Download version 5.13.0 software for Amazon Linux 2023 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 7f95ca9dcdf19627333257d28b81d06cd5f10c70df1e2aa10a57af34213328eb)

- OpenSSL Dynamic Engine (SHA256 checksum 52de525d691b404b87c6381d4c71c9b5a51a80ada1c078d6433032bb4840ebe7)
- JCE provider (SHA256 checksum 98c69a66e353568e416a1daba161cf49e95e3196c82ae66628519aec82479787)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum 5b08e80ff26fef91c2693a8def394ec02d69ea36604c189885f9e1205aa83da0)

Download version 5.13.0 software for Amazon Linux 2023 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 3797803ceaea2ea2f495b7e08c9e344fad755b1919b93f341b5dc7246c484988)
- OpenSSL Dynamic Engine (SHA256 checksum 71bbd800adc024df13dd503268217530a6e85fae2ab0c07c75cd3f5905fd526a)
- JCE provider (SHA256 checksum 3d7213810899ebace2e6664fbd722edbf2a771f70d68a35885ed75007f3de2cb)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum 381420610beed60b7a402fb0f7c518b1b8df74690b6539f16c115342ef75cbee)

RHEL 8 (8.3+)

Download version 5.13.0 software for RHEL 8 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 1a4d42b88f79f64ebc9fa55d091556cf04a16796014fa0488ade43cda62a0731)
- OpenSSL Dynamic Engine (SHA256 checksum d7658ea876c1a6209637fc4a4ef47e0421ea47e54d1d7d10eacc7eefabb86021)
- JCE provider (SHA256 checksum 54aae2a6e8b2a43e806c1320fff638345f88ade7e510a6b63c55573327ba160c)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum 587592eb395af73b33beadd67af5765354904d1cc92f83c0548a308af842a6c7)

RHEL 9 (9.2+)

Download version 5.13.0 software for RHEL 9 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 4e68cd8055300c40e8b4cb9a4303e84870c2b517a74c16f2bd6a10fcbab5f426)

- OpenSSL Dynamic Engine (SHA256 checksum a8ae26dc0eda9f143c4a44a3a7e399772039e238d8b5b0f36256cdd8ae6dc30b)
- JCE provider (SHA256 checksum 2948e6cec865f0934ac501a2d4724b1b8c4dc2d15b61155c41d60a0257e74110)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum 0f4b84b572722de119edbb35d50a00e3c390b019a88a9c3f279a9b76225b4520)

Download version 5.13.0 software for RHEL 9 on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 0bea8d7e46bc7e9bd5fa36f64d43416ea400332602720f0ae162eb7b12eda312)
- OpenSSL Dynamic Engine (SHA256 checksum d96eddd33c5034357e8cc3c157ff1a03dafbaeb3f09b31ed324a2cbe9e424c01)
- JCE provider (SHA256 checksum 34bcabf11d0b7d34e6fc48c07ba9a383a4df26491e7c4c00cd7fcbf50cd30298)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum be7a22e5f64db4211f86eb361f79c5db93c237bd28ac0db5c274bba210cd431a)

Ubuntu 20.04 LTS

Download version 5.13.0 software for Ubuntu 20.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 97b3686b007c3d3a0d97f6774ad182d5702f6a233060522f4674fd233b3eafe9)
- OpenSSL Dynamic Engine (SHA256 checksum 3d453a428a920c2fccd40bb18fe11b7dba3194da6fb3e457ade77d1d2cfe2b35)
- JCE provider (SHA256 checksum 6e6e68d1ee6f14df9370bf6d37055328a49bf28e57de23152ddc9c51e8014508)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum 2515705e66e118deaee9694a47fdb74ad64e66067a690545039dc2802e4e198f)

Ubuntu 22.04 LTS

Download version 5.13.0 software for Ubuntu 22.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 53ffd1d6353e6facb022631e4f258200a6efabeaf00ee9f4bf4418ec27633a39)

- OpenSSL Dynamic Engine (SHA256 checksum 4f49e0946ba376b3c2cef05c5ee63cd78202a08907ea0ac8027095e16e47eed1)
- JCE provider (SHA256 checksum 2840a8938c22de6a9e6130b250bc7dd7fc512d274d7a702e944db3d1396c0222)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum 9e8592967c330f50552249017695d116adbbc54321a35be33a48ca18d739beae)

Download version 5.13.0 software for Ubuntu 22.04 LTS on ARM64 architecture:

- PKCS #11 library (SHA256 checksum 4ab4961eb97ec0cf8bd818176c99da763a416903e24855e0dd6b31f776a01f26)
- OpenSSL Dynamic Engine (SHA256 checksum f1a396bf9ac2d1e970e027e2ab7d388fc0f0634d3e9c16b91d6dd889698514ad)
- JCE provider (SHA256 checksum 4035bc68fe7bf978b83f4fd0eb99e49efe874c2e128f62e800b9ec95c8142ec0)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum 099137b934ec81d6bb87137f919d99a810f3149858ccdd69df51418f7e5485d9)

Ubuntu 24.04 LTS

Download version 5.13.0 software for Ubuntu 24.04 LTS on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 7057ce0d74c50635eeede91edcf3ef2e7915bed6b1f73b7bca45ba3a44392b7e)
- OpenSSL Dynamic Engine (SHA256 checksum d76b59bf0ba1325adbb1ad3cea8050a38db1517e48c9d9bd1001a232df285904)
- JCE provider (SHA256 checksum e4296cef92f99e49d6ca6c0d07a82de5e1551a6ec550252c52329561533f4f6d)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum 47516fc88c8089edcd82e942f17351df7bfba8c7d640c9f909ad48ba4d980022)

Download version 5.13.0 software for Ubuntu 24.04 LTS on ARM64 architecture:

- PKCS #11 library (SHA256 checksum f2aefc6517ad4c6ef63124f380ae1f26fc2eb423d0a02e5b7ceda6769784a74f)
- OpenSSL Dynamic Engine (SHA256 checksum 6159f4eb648159d37f304982725e5ed0dc34e7fd0658a8dc8ccacf2b75a1f4d2)

- JCE provider (SHA256 checksum eeaf7e0345dcf78ae14595e4ab7967dd95fb6da06a42913423f76234f47ca3fc)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum 8656590f2caa5cd8c930b116ed12504caf99254f00c7563b90d799e6f69b2e77)

Windows Server 2016

Download version 5.13.0 software for Windows Server 2016 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 77e32ad8d28b1073286e95f8b350f99dd26c62ff32897fb86e9d79aef9c190fb)
- JCE provider (SHA256 checksum 191135271e912cf858d24ad4b07c7ff57c9c4a1b3635513cc6ab8dd5dc1a0e42)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum bb7960dd7bff73a1430cf2edc1bf36b0161309e5c354f0db44eaf086568507d5)

Windows Server 2019

Download version 5.13.0 software for Windows Server 2019 on x86_64 architecture:

- PKCS #11 library (SHA256 checksum 77e32ad8d28b1073286e95f8b350f99dd26c62ff32897fb86e9d79aef9c190fb)
- JCE provider (SHA256 checksum 191135271e912cf858d24ad4b07c7ff57c9c4a1b3635513cc6ab8dd5dc1a0e42)
  - Javadocs for AWS CloudHSM (SHA256 checksum 11b5de4a400861cc796b4b2ecaec706603e61ae7640bd0c4e2b090c7034d6318)
- CloudHSM CLI (SHA256 checksum bb7960dd7bff73a1430cf2edc1bf36b0161309e5c354f0db44eaf086568507d5)

Client SDK 5.13 adds support for configuring mutual TLS on hsm2m.medium cluster types. For information on using mutual TLS with CloudHSM, see Set up mutual TLS between client and AWS CloudHSM (recommended). Client SDK 5.13 also adds installation packages for Ubuntu 24.04 LTS.

Platform support

- Added support for Ubuntu 24.04 LTS on x86_64 and ARM64 architectures for all SDKs.

CloudHSM CLI

- Added support for admin users to run the **Replicate a key with CloudHSM CLI** command. Client SDK 5.12 introduced the key replicate command for use by crypto users.
- Added the following command:
  - **The cluster mtls category in CloudHSM CLI**

Bug fixes/Improvements

- Fixed an issue to decrease the time required for the client to detect unhealthy HSM connections, which helps prevent connection drop errors during lambda warm starts.

Version 5.12.0

Client SDK 5.12.0 adds ARM support to several platforms and performance improvements for all SDKs. New features have been added to the CloudHSM CLI and JCE provider.

Platform support

- Added support for Amazon Linux 2023 on ARM64 architecture for all SDKs.
- Added support for Red Hat Enterprise Linux 9 (9.2+) on ARM64 architecture for all SDKs.
- Added support for Ubuntu 22.04 LTS on ARM64 architecture for all SDKs.

CloudHSM CLI

- Added the following command:
  - Replicate a key with CloudHSM CLI
- Added support for connecting to multiple clusters. For more information, see Connecting to multiple clusters with CloudHSM CLI.

JCE provider

- Added `KeyReferenceSpec` for retrieving keys using `KeyStoreWithAttributes`.
- Added `getKeys` for retrieving multiple keys at once using `KeyStoreWithAttributes`.

Performance improvements

- Performance improvements for the AES CBC NoPadding operation for all SDKs.

Version 5.11.0

Client SDK 5.11.0 adds new features, improves stability, and includes bug fixes for all SDKs.

Platform support

- Added support for Amazon Linux 2023 and RHEL 9 (9.2+) for all SDKs.
- Removed support for Ubuntu 18.04 LTS due to its recent end of life.
- Removed support for Amazon Linux due to its recent end of life.

CloudHSM CLI

- Added the following commands:
  - The crypto sign category in CloudHSM CLI
  - The crypto verify category in CloudHSM CLI
  - Import a PEM format key with CloudHSM CLI
  - The key unwrap command in CloudHSM CLI
  - The key wrap command in CloudHSM CLI
- Export an asymmetric key with CloudHSM CLI now supports export of public keys.

OpenSSL Dynamic Engine

- The AWS CloudHSM OpenSSL Dynamic Engine is now supported on platforms that come installed with an OpenSSL library version of 3.x. This includes Amazon Linux 2023, RHEL 9 (9.2+), and Ubuntu 22.04.

JCE

- Added support for JDK 17 and JDK 21.
- Added support for AES keys to be used for HMAC operations.
- Added the new key attribute `ID`.

- Introduced a new `DataExceptionCause` variant for key exhaustion: `DataExceptionCause.KEY_EXHAUSTED`.

Bug fixes/Improvements

- Increased the max length for the `label` attribute from 126 to 127 characters.
- Fixed a bug which prevented unwrapping of EC keys with the RsaOaep mechanism.
- Resolved a known issue for the getKey operation in the JCE provider. Refer to Issue: Client SDK 5 memory leak with getKey operations for details.
- Improved logging in all SDKs for Triple DES keys that have hit their maximum encryption block limit, per FIPS 140-2.
- Added known issues for the OpenSSL Dynamic Engine. See Known issues for the OpenSSL Dynamic Engine for AWS CloudHSM for details.

Version 5.10.0

Client SDK 5.10.0 improves stability and includes bug fixes for all SDKs.

CloudHSM CLI

- Added new commands that allow customers to manage keys using CloudHSM CLI, including:
  - Create symmetric keys and asymmetric key pairs
  - Share and unshare keys
  - List and filter keys using key attributes
  - Set key attributes
  - Generate key reference files
  - Delete keys
- Improved error logging.
- Added support for multi-line unicode commands in interactive mode.

Bug fixes/Improvements

- Improved performance for importing, unwrapping, deriving, and creating sessions keys for all SDKs.

- Fixed a bug in the JCE Provider that prevented temp files to be removed on exit.
- Fixed a bug that caused a connection error under certain conditions after HSMs in the cluster are replaced.
- Modified JCE `getVersion` output format to handle large minor version numbers and include patch number.

Platform support

- Added support for Ubuntu 22.04 with JCE, PKCS #11, and CloudHSM CLI (support for OpenSSL Dynamic Engine is not yet available).

Version 5.9.0

Client SDK 5.9.0 improves stability and includes bug fixes for all SDKs. An optimization has been made for all SDKs to inform applications of operation failure immediately when an HSM is determined unavailable. This release includes performance enhancements for JCE.

JCE provider

- Enhanced performance
- Fixed a known issue for session pool exhaustion

AWS CloudHSM deprecated Client SDK releases

Versions 5.8.0 and earlier are deprecated. We do not recommend using deprecated releases in production workloads. We do not provide backwards compatible updates for deprecated releases, nor do we host deprecated releases for download. If you experience production impact while using deprecated releases, you must upgrade to obtain software fixes.

Deprecated Client SDK 5 releases

This section lists deprecated Client SDK 5 releases.

Version 5.8.0

Version 5.8.0 introduces quorum authentication for CloudHSM CLI, SSL/TLS offload with JSSE, multi-slot support for PKCS #11, multi-cluster/multi-user support for JCE, key extraction with JCE, supported keyFactory for JCE, new retry configurations for non-terminal return codes, and includes improved stability and bug fixes for all SDKs.

PKCS #11 library

- Added support for multi-slot configuration.

JCE provider

- Added configuration based key extraction.
- Added support for multi-cluster and multi-user configurations.
- Added support for SSL and TLS offload with JSSE.
- Added unwrap support for AES/CBC/NoPadding.
- Added new types of key factories: SecretKeyFactory and KeyFactory.

CloudHSM CLI

- Added support for quorum authentication

Version 5.7.0

Version 5.7.0 introduces CloudHSM CLI and includes a new cipher-based message authentication code (CMAC) algorithm. This release adds ARM architecture on Amazon Linux 2. JCE provider Javadocs are now available for AWS CloudHSM.

PKCS #11 library

- Improved stability and bug fixes.
- Now supported on ARM architecture with Amazon Linux 2.
- Algorithms
  - CKM_AES_CMAC (sign and verify)

OpenSSL Dynamic Engine

- Improved stability and bug fixes.
- Now supported on ARM architecture with Amazon Linux 2.

JCE provider

- Improved stability and bug fixes.
- Algorithms
  - AESCMAC

Version 5.6.0

Version 5.6.0 includes new mechanism support for PKCS #11 library and JCE provider. Additionally, version 5.6 supports Ubuntu 20.04.

PKCS #11 library

- Improved stability and bug fixes.
- Mechanisms
  - CKM_RSA_X_509, for encrypt, decrypt, sign, and verify modes

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Improved stability and bug fixes.
- Ciphers
  - RSA/ECB/NoPadding, for encrypt and decrypt modes

Supported keys

- EC with curves secp224r1 and secp521r1

Platform support

- Added support for Ubuntu 20.04.

Version 5.5.0

Version 5.5.0 adds support for OpenJDK 11, Keytool and Jarsigner integration, and additional mechanisms to the JCE provider. Resolves a known issue regarding a KeyGenerator class incorrectly interpreting key size parameter as number of bytes instead of bits.

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Support for the Keytool and Jarsigner utilities
- Support for OpenJDK 11 on all platforms
- Ciphers
  - AES/CBC/NoPadding Encrypt and Decrypt mode
  - AES/ECB/PKCS5Padding Encrypt and Decrypt mode
  - AES/CTR/NoPadding Encrypt and Decrypt mode
  - AES/GCM/NoPadding Wrap and Unwrap mode
  - DESede/ECB/PKCS5Padding Encrypt and Decrypt mode
  - DESede/CBC/NoPadding Encrypt and Decrypt mode
  - AESWrap/ECB/NoPadding Wrap and Unwrap mode
  - AESWrap/ECB/PKCS5Padding Wrap and Unwrap mode
  - AESWrap/ECB/ZeroPadding Wrap and Unwrap mode
  - RSA/ECB/PKCS1Padding Wrap and Unwrap mode
  - RSA/ECB/OAEPPadding Wrap and Unwrap mode
  - RSA/ECB/OAEPWithSHA-1ANDMGF1Padding Wrap and Unwrap mode
  - RSA/ECB/OAEPWithSHA-224ANDMGF1Padding Wrap and Unwrap mode
  - RSA/ECB/OAEPWithSHA-256ANDMGF1Padding Wrap and Unwrap mode
  - RSA/ECB/OAEPWithSHA-384ANDMGF1Padding Wrap and Unwrap mode

  - RSA/ECB/OAEPWithSHA-512ANDMGF1Padding Wrap and Unwrap mode
  - RSAAESWrap/ECB/OAEPPadding Wrap and Unwrap mode
  - RSAAESWrap/ECB/OAEPWithSHA-1ANDMGF1Padding Wrap and Unwrap mode
  - RSAAESWrap/ECB/OAEPWithSHA-224ANDMGF1Padding Wrap and Unwrap mode
  - RSAAESWrap/ECB/OAEPWithSHA-256ANDMGF1Padding Wrap and Unwrap mode
  - RSAAESWrap/ECB/OAEPWithSHA-384ANDMGF1Padding Wrap and Unwrap mode
  - RSAAESWrap/ECB/OAEPWithSHA-512ANDMGF1Padding Wrap and Unwrap mode
- KeyFactory and SecretKeyFactory
  - RSA – 2048-bit to 4096-bit RSA keys, in increments of 256 bits
  - AES – 128, 192, and 256-bit AES keys
  - EC key pairs for NIST curves secp256r1 (P-256), secp384r1 (P-384), and secp256k1
  - DESede (3DES)
  - GenericSecret
  - HMAC – with SHA1, SHA224, SHA256, SHA384, SHA512 hash support
- Sign/Verify
  - RSASSA-PSS
  - SHA1withRSA/PSS
  - SHA224withRSA/PSS
  - SHA256withRSA/PSS
  - SHA384withRSA/PSS
  - SHA512withRSA/PSS
  - SHA1withRSAandMGF1
  - SHA224withRSAandMGF1
  - SHA256withRSAandMGF1
  - SHA384withRSAandMGF1
  - SHA512withRSAandMGF1

Version 5.4.2

Version 5.4.2 includes improved stability and bug fixes for all SDKs. This is also the last release for the CentOS 8 platform. For more information, see the CentOS website .

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Improved stability and bug fixes.

Version 5.4.1

Version 5.4.1 resolves a known issue with the PKCS #11 library. This is also the last release for the CentOS 8 platform. For more information, see the CentOS website.

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Improved stability and bug fixes.

Version 5.4.0

Version 5.4.0 adds initial support for the JCE provider for all platforms. The JCE provider is compatible with OpenJDK 8.

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- **Key types**
  - RSA – 2048-bit to 4096-bit RSA keys, in increments of 256 bits.
  - AES – 128, 192, and 256-bit AES keys.
  - ECC key pairs for NIST curves secp256r1 (P-256), secp384r1 (P-384), and secp256k1.
  - DESede (3DES)
  - HMAC – with SHA1, SHA224, SHA256, SHA384, SHA512 hash support.
- **Ciphers (encrypt and decrypt only)**
  - AES/GCM/NoPadding
  - AES/ECB/NoPadding
  - AES/CBC/PKCS5Padding
  - DESede/ECB/NoPadding
  - DESede/CBC/PKCS5Padding
  - AES/CTR/NoPadding
  - RSA/ECB/PKCS1Padding
  - RSA/ECB/OAEPPadding
  - RSA/ECB/OAEPWithSHA-1ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-224ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-256ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-384ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-512ANDMGF1Padding
- **Digests**
  - SHA-1
  - SHA-224
  - SHA-256
  - SHA-384

  - SHA-512
- **Sign/Verify**
  - NONEwithRSA
  - SHA1withRSA
  - SHA224withRSA
  - SHA256withRSA
  - SHA384withRSA
  - SHA512withRSA
  - NONEwithECDSA
  - SHA1withECDSA
  - SHA224withECDSA
  - SHA256withECDSA
  - SHA384withECDSA
  - SHA512withECDSA
- Integration with the Java KeyStore

Version 5.3.0

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Add support for ECDSA sign/verify with curves P-256, P-384, and secp256k1.
- Add support for the platforms: Amazon Linux, Amazon Linux 2, CentOS 7.8+, RHEL 7 (7.8+).
- Add support for OpenSSL version 1.0.2.
- Improved stability and bug fixes.

JCE provider

- **Key types**
  - RSA – 2048-bit to 4096-bit RSA keys, in increments of 256 bits.

  - AES – 128, 192, and 256-bit AES keys.
  - EC key pairs for NIST curves secp256r1 (P-256), secp384r1 (P-384), and secp256k1.
  - DESede (3DES)
  - HMAC – with SHA1, SHA224, SHA256, SHA384, SHA512 hash support.
- **Ciphers (encrypt and decrypt only)**
  - AES/GCM/NoPadding
  - AES/ECB/NoPadding
  - AES/CBC/PKCS5Padding
  - DESede/ECB/NoPadding
  - DESede/CBC/PKCS5Padding
  - AES/CTR/NoPadding
  - RSA/ECB/PKCS1Padding
  - RSA/ECB/OAEPPadding
  - RSA/ECB/OAEPWithSHA-1ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-224ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-256ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-384ANDMGF1Padding
  - RSA/ECB/OAEPWithSHA-512ANDMGF1Padding
- **Digests**
  - SHA-1
  - SHA-224
  - SHA-256
  - SHA-384
  - SHA-512
- **Sign/Verify**
  - NONEwithRSA
  - SHA1withRSA
  - SHA224withRSA
  - SHA256withRSA
  - SHA384withRSA

  - SHA512withRSA
  - NONEwithECDSA
  - SHA1withECDSA
  - SHA224withECDSA
  - SHA256withECDSA
  - SHA384withECDSA
  - SHA512withECDSA
- Integration with the Java KeyStore

Version 5.2.1

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

Version 5.2.0

Version 5.2.0 adds support additional key types and mechanisms to the PKCS #11 library.

PKCS #11 library

Key Types

- ECDSA– P-224, P-256, P-384, P-521 and secp256k1 curves
- Triple DES (3DES)

Mechanisms

- CKM_EC_KEY_PAIR_GEN
- CKM_DES3_KEY_GEN
- CKM_DES3_CBC
- CKM_DES3_CBC_PAD

- CKM_DES3_ECB
- CKM_ECDSA
- CKM_ECDSA_SHA1
- CKM_ECDSA_SHA224
- CKM_ECDSA_SHA256
- CKM_ECDSA_SHA384
- CKM_ECDSA_SHA512
- CKM_RSA_PKCS for Encrypt/Decrypt

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

Version 5.1.0

Version 5.1.0 adds support for additional mechanisms to the PKCS #11 library.

PKCS #11 library

Mechanisms

- CKM_RSA_PKCS for Wrap/Unwrap
- CKM_RSA_PKCS_PSS
- CKM_SHA1_RSA_PKCS_PSS
- CKM_SHA224_RSA_PKCS_PSS
- CKM_SHA256_RSA_PKCS_PSS
- CKM_SHA384_RSA_PKCS_PSS
- CKM_SHA512_RSA_PKCS_PSS
- CKM_AES_ECB
- CKM_AES_CTR
- CKM_AES_CBC
- CKM_AES_CBC_PAD
- CKM_SP800_108_COUNTER_KDF

- CKM_GENERIC_SECRET_KEY_GEN
- CKM_SHA_1_HMAC
- CKM_SHA224_HMAC
- CKM_SHA256_HMAC
- CKM_SHA384_HMAC
- CKM_SHA512_HMAC
- CKM_RSA_PKCS_OAEP Wrap/Unwrap only
- CKM_RSA_AES_KEY_WRAP
- CKM_CLOUDHSM_AES_KEY_WRAP_NO_PAD
- CKM_CLOUDHSM_AES_KEY_WRAP_PKCS5_PAD
- CKM_CLOUDHSM_AES_KEY_WRAP_ZERO_PAD

API Operations

- C_CreateObject
- C_DeriveKey
- C_WrapKey
- C_UnWrapKey

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

Version 5.0.1

Version 5.0.1 adds initial support for OpenSSL Dynamic Engine.

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Initial release of OpenSSL Dynamic Engine.

- This release offers introductory support for key types and OpenSSL APIs:
  - RSA key generation for 2048, 3072, and 4096-bit keys
  - OpenSSL APIs:
    - RSA Sign using RSA PKCS with SHA1/224/256/384/512 & RSA PSS
    - RSA Key Generation

  For more information, see OpenSSL Dynamic Engine .
- Platforms supported: CentOS 8.3+, Red Hat Enterprise Linux (RHEL) 8.3+, and Ubuntu 18.04 LTS
  - Requires: OpenSSL 1.1.1

  For more information, see Supported Platforms.
- Support for SSL/TLS Offload on CentOS 8.3+, Red Hat Enterprise Linux (RHEL) 8.3, and Ubuntu 18.04 LTS, including NGINX 1.19 (for select cipher suites).

  For more information, see SSL/TLS Offload on Linux using Tomcat or SSL/TLS Offload on Linux using NGINX or Apache.

Version 5.0.0

Version 5.0.0 is the first release.

PKCS #11 library

- This is the initial release.

Introductory PKCS #11 library support in client SDK version 5.0.0

This section details support for key types, mechanisms, API operations and attributes Client SDK version 5.0.0.

Key Types:

- **AES–** 128, 192, and 256-bit AES keys
- **RSA–** 2048-bit to 4096-bit RSA keys, in increments of 256 bits

Mechanisms:

- CKM_AES_GCM

- CKM_AES_KEY_GEN
- CKM_CLOUDHSM_AES_GCM
- CKM_RSA_PKCS
- CKM_RSA_X9_31_KEY_PAIR_GEN
- CKM_SHA1
- CKM_SHA1_RSA_PKCS
- CKM_SHA224
- CKM_SHA224_RSA_PKCS
- CKM_SHA256
- CKM_SHA256_RSA_PKCS
- CKM_SHA384
- CKM_SHA384_RSA_PKCS
- CKM_SHA512
- CKM_SHA512_RSA_PKCS

API Operations:

- C_CloseAllSessions
- C_CloseSession
- C_Decrypt
- C_DecryptFinal
- C_DecryptInit
- C_DecryptUpdate
- C_DestroyObject
- C_Digest
- C_DigestFinal
- C_DigestInit
- C_DigestUpdate
- C_Encrypt
- C_EncryptFinal

- C_EncryptInit
- C_EncryptUpdate
- C_Finalize
- C_FindObjects
- C_FindObjectsFinal
- C_FindObjectsInit
- C_GenerateKey
- C_GenerateKeyPair
- C_GenerateRandom
- C_GetAttributeValue
- C_GetFunctionList
- C_GetInfo
- C_GetMechanismInfo
- C_GetMechanismList
- C_GetSessionInfo
- C_GetSlotInfo
- C_GetSlotList
- C_GetTokenInfo
- C_Initialize
- C_Login
- C_Logout
- C_OpenSession
- C_Sign
- C_SignFinal
- C_SignInit
- C_SignUpdate
- C_Verify
- C_VerifyFinal
- C_VerifyInit
- C_VerifyUpdate

Attributes:

- GenerateKeyPair
  - All RSA Key attributes
- GenerateKey
  - All AES Key attributes
- GetAttributeValue
  - All RSA Key attributes
  - All AES Key attributes

Samples:

- Generate keys (AES, RSA, EC)
- List key attributes
- Encrypt and decrypt data with AES GCM
- Sign and verify data with RSA

Deprecated Client SDK 3 releases

This section lists deprecated Client SDK 3 releases.

Version 3.4.4

Version 3.4.4 adds updates to JCE provider.

AWS CloudHSM Client Software

- Updated the version for consistency.

PKCS #11 library

- Updated the version for consistency.

OpenSSL Dynamic Engine

- Updated the version for consistency.

JCE provider

- Update log4j to version 2.17.1.

Windows (CNG and KSP providers)

- Updated the version for consistency.

Version 3.4.3

Version 3.4.3 adds updates to JCE provider.

AWS CloudHSM Client Software

- Updated the version for consistency.

PKCS #11 library

- Updated the version for consistency.

OpenSSL Dynamic Engine

- Updated the version for consistency.

JCE provider

- Update log4j to version 2.17.0.

Windows (CNG and KSP providers)

- Updated the version for consistency.

Version 3.4.2

Version 3.4.2 adds updates to JCE provider.

AWS CloudHSM Client Software

- Updated the version for consistency.

PKCS #11 library

- Updated the version for consistency.

OpenSSL Dynamic Engine

- Updated the version for consistency.

JCE provider

- Update log4j to version 2.16.0.

Windows (CNG and KSP providers)

- Updated the version for consistency.

Version 3.4.1

Version 3.4.1 adds updates to JCE provider.

AWS CloudHSM Client Software

- Updated the version for consistency.

PKCS #11 library

- Updated the version for consistency.

OpenSSL Dynamic Engine

- Updated the version for consistency.

JCE provider

- Update log4j to version 2.15.0.

Windows (CNG and KSP providers)

- Updated the version for consistency.

Version 3.4.0

Version 3.4.0 adds updates to all components.

AWS CloudHSM Client Software

- Improved stability and bug fixes.

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Improved stability and bug fixes.

Windows (CNG and KSP providers)

- Improved stability and bug fixes.

Version 3.3.2

Version 3.3.2 resolves an issue with the client_info script.

AWS CloudHSM Client Software

- Updated the version for consistency.

PKCS #11 library

- Updated the version for consistency.

OpenSSL Dynamic Engine

- Updated the version for consistency.

JCE provider

- Updated the version for consistency.

Windows (CNG and KSP providers)

- Updated the version for consistency.

Version 3.3.1

Version 3.3.1 adds updates to all components.

AWS CloudHSM Client Software

- Improved stability and bug fixes.

PKCS #11 library

- Improved stability and bug fixes.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Improved stability and bug fixes.

Windows (CNG and KSP providers)

- Improved stability and bug fixes.

Version 3.3.0

Version 3.3.0 adds two-factor authentication (2FA) and other improvements.

AWS CloudHSM Client Software

- Added 2FA authentication for crypto officers (CO). For more information, see Managing Two-Factor Authentication for Crypto Officers.
- Removed platform support for RedHat Enterprise Linux 6 and CentOS 6. For more information, see Linux Support.
- Added a standalone version of CMU for use with Client SDK 5 or Client SDK 3. This is the same version of CMU included with the client daemon of version 3.3.0, and now you can download CMU without downloading the client daemon.

PKCS #11 library

- Improved stability and bug fixes.
- Removed platform support for RedHat Enterprise Linux 6 and CentOS 6. For more information, see Linux Support.

OpenSSL Dynamic Engine

- Updated the version for consistency
- Removed platform support for RedHat Enterprise Linux 6 and CentOS 6. For more information, see Linux Support.

JCE provider

- Improved stability and bug fixes.
- Removed platform support for RedHat Enterprise Linux 6 and CentOS 6. For more information, see Linux Support.

Windows (CNG and KSP providers)

- Updated the version for consistency

Version 3.2.1

Version 3.2.1 adds a compliance analysis between the AWS CloudHSM implementation of the PKCS #11 library and the PKCS #11 standard, new platforms, and other improvements.

AWS CloudHSM Client Software

- Add platform support for CentOS 8, RHEL 8, and Ubuntu 18.04 LTS. For more information, see ??? .

PKCS #11 library

- PKCS #11 library compliance report for client SDK 3.2.1
- Add platform support for CentOS 8, RHEL 8, and Ubuntu 18.04 LTS. For more information, see ??? .

OpenSSL Dynamic Engine

- No support for CentOS 8, RHEL 8, and Ubuntu 18.04 LTS. For more information, see ???.

JCE provider

- Add platform support for CentOS 8, RHEL 8, and Ubuntu 18.04 LTS. For more information, see ??? .

Windows (CNG and KSP providers)

- Improved stability and bug fixes.

Version 3.2.0

Version 3.2.0 adds support for masking passwords and other improvements.

AWS CloudHSM Client Software

- Adds support for hiding your password when using command-line tools. For more information, see loginHSM and logoutHSM (cloudhsm_mgmt_util) and loginHSM and logoutHSM (key_mgmt_util).

PKCS #11 library

- Adds support for hashing large data in software for some PKCS #11 mechanisms that were previously unsupported. For more information, see Supported Mechanisms.

OpenSSL Dynamic Engine

- Improved stability and bug fixes.

JCE provider

- Updated the version for consistency.

Windows (CNG and KSP providers)

- Improved stability and bug fixes.

Version 3.1.2

Version 3.1.2 adds updates to JCE provider.

AWS CloudHSM Client Software

- Updated the version for consistency

PKCS #11 library

- Updated the version for consistency

OpenSSL Dynamic Engine

- Updated the version for consistency

JCE provider

- Update log4j to version 2.13.3

Windows (CNG and KSP providers)

- Updated the version for consistency

Version 3.1.1

AWS CloudHSM Client Software

- Updated the version for consistency.

PKCS #11 Library

- Updated the version for consistency.

OpenSSL Dynamic Engine

- Updated the version for consistency.

JCE provider

- Bug fixes and performance improvements.

Windows (CNG, KSP)

- Updated the version for consistency.

Version 3.1.0

Version 3.1.0 adds standards-compliant AES key wrapping.

AWS CloudHSM Client Software

- A new requirement for upgrade: the version of your client must match the version of any software libraries you are using. To upgrade, you must use a batch command that upgrades the client and all the libraries at the same time. For more information, see Client SDK 3 Upgrade.
- Key_mgmt_util (KMU) includes the following updates:
  - Added two new AES key wrap methods – standards-compliant AES key wrap with zero padding and AES key wrap with no padding. For more information, see wrapKey and unwrapKey.

  - Disabled ability to specify custom IV when wrapping a key using AES_KEY_WRAP_PAD_PKCS5. For more information, see AES Key Wrapping.

PKCS #11 Library

- Added two new AES key wrap methods – standards-compliant AES key wrap with zero padding and AES key wrap with no padding. For more information, see AES Key Wrapping.
- You can configure salt length for RSA-PSS signatures. To learn how to use this feature, see Configurable salt length for RSA-PSS signatures on GitHub.

OpenSSL Dynamic Engine

- **BREAKING CHANGE**: TLS 1.0 and 1.2 cipher suites with SHA1 are not available in OpenSSL Engine 3.1.0. This issue will be resolved shortly.
- If you intend to install the OpenSSL Dynamic Engine library on RHEL 6 or CentOS 6, see a known issue about the default OpenSSL version installed on those operating systems.
- Improved stability and bug fixes

JCE provider

- **BREAKING CHANGE**: To address an issue with Java Cryptography Extension (JCE) compliance, AES wrap and unwrap now properly use the AESWrap algorithm instead of the AES algorithm. This means `Cipher.WRAP_MODE` and `Cipher.UNWRAP_MODE` no longer succeed for AES/ECB and AES/CBC mechanisms.

  To upgrade to client version 3.1.0, you must update your code. If you have existing wrapped keys, you must pay particular attention to the mechanism you use to unwrap and how IV defaults have changed. If you wrapped keys with client version 3.0.0 or earlier, then in 3.1.1 you must use AESWrap/ECB/PKCS5Padding to unwrap your existing keys. For more information, see AES Key Wrapping.
- You can list multiple keys with the same label from the JCE provider. To learn how to iterate through all available keys, see Find all keys on GitHub.
- You can set more restrictive values for attributes during key creation, including specifying different labels for public and private keys. For more information, see Supported Java Attributes.

Windows (CNG, KSP)

- Improved stability and bug fixes.

AWS CloudHSM end-of-life Client SDK releases

The following AWS CloudHSM client versions have reached the end of support. These AWS CloudHSM client versions are no longer compatible with the service and will not receive updates. To preserve the security of your application, AWS CloudHSM might refuse connections from releases that have reached the end of support.

- SDK versions 3.4.4 and earlier have reached the end of support.
- SDK versions 5.8.0 and earlier have reached the end of support.

Document history

This topic describes significant updates to the *AWS CloudHSM User Guide*.

Topics

Recent updates

The following table describes significant changes to this documentation since April 2018. In addition to major changes listed here, we also update the documentation frequently to improve the descriptions and examples, and to address the feedback you send us. To be notified about significant changes, use the link in the upper right corner to subscribe to the RSS feed.

For details on new releases, see Downloads for AWS CloudHSM Client SDK

Change	Description	Date
Added new release	Released AWS CloudHSM client version 5.16.1.	June 25, 2025
Added new release	Released AWS CloudHSM client version 5.16.0.	May 1, 2025
Added new release	Released AWS CloudHSM client version 5.15.0.	February 3, 2025
Added new release	Released AWS CloudHSM client version 5.14.0.	November 26, 2024
New HSM type and cluster mode	Added support for creating (*hsm2m.medium*) in FIPS mode clusters.	August 20, 2024
Added new release	Released AWS CloudHSM client version 5.13.0.	August 13, 2024

New HSM type and cluster mode	Launched a new HSM type (*hsm2m.medium*) and a new cluster mode (*non-FIPS*).	June 10, 2024
Added new release	Released AWS CloudHSM client version 5.12.0.	March 20, 2024
Added new release	Released AWS CloudHSM client version 5.11.0.	January 17, 2024
Added new release	Released AWS CloudHSM client version 5.10.0.	July 28, 2023
Added new release	Released AWS CloudHSM client version 5.9.0.	May 23, 2023
Added new release	Released AWS CloudHSM client version 5.8.0.	March 16, 2023
Added new release	Released AWS CloudHSM client version 5.7.0.	November 16, 2022
Added new release	Released AWS CloudHSM client version 5.6.0.	September 1, 2022
Added new release	Released AWS CloudHSM client version 5.5.0.	May 13, 2022
Added new release	Released AWS CloudHSM client version 5.4.2.	March 18, 2022
Added new release	Released AWS CloudHSM client version 5.4.1.	February 10, 2022
Added new release	Released the AWS CloudHSM JCE provider version 5.4.0 for Windows platforms.	February 1, 2022

Added new release	Released the AWS CloudHSM client version 5.4.0, which adds initial support for the JCE provider for all Linux platforms.	January 28, 2022
Added new release	Released AWS CloudHSM client version 5.3.0.	January 3, 2022
Added new release	Released AWS CloudHSM client version 3.4.4.	January 3, 2022
Added new release	Released AWS CloudHSM client version 3.4.3.	December 20, 2021
Added new release	Released AWS CloudHSM client version 3.4.2.	December 15, 2021
Added new release	Released AWS CloudHSM client version 3.4.1.	December 10, 2021
Added new release	Released AWS CloudHSM client version 5.2.1.	October 4, 2021
Added new release	Released AWS CloudHSM client version 3.4.0.	August 25, 2021
Added new release	Released AWS CloudHSM client version 5.2.0.	August 3, 2021
Added new release	Released AWS CloudHSM client version 3.3.2.	July 2, 2021
Added new release	Released AWS CloudHSM client version 5.1.0.	June 1, 2021
Added new release	Released AWS CloudHSM client version 3.3.1.	April 26, 2021

Added new release	Released AWS CloudHSM client version 5.0.1.	April 8, 2021
Added new release	Released AWS CloudHSM client version 5.0.0.	March 12, 2021
Added new content	Added interface VPC endpoint, an AWS feature that allows you to create a private connection between your VPC and AWS CloudHSM without requiring access over the internet or through a NAT device, a VPN connection, or an AWS Direct Connect connection.	February 10, 2021
Added new release	Released AWS CloudHSM client version 3.3.0.	February 3, 2021
Add new content	Added managed backup retention, a feature that automatically deletes old backups.	November 18, 2020
Add new content	Added a compliance report that analyzes the AWS CloudHSM Client SDK 3.2.1 implementation of the PKCS #11 library with the PKCS #11 standard.	October 29, 2020
Added new release	Released AWS CloudHSM client version 3.2.1.	October 8, 2020
Added new content	Added documentation that describes key synchronization settings in AWS CloudHSM.	September 1, 2020

Added new release	Released AWS CloudHSM client version 3.2.0.	August 31, 2020
Added new release	Released AWS CloudHSM client version 3.1.2.	July 30, 2020
Added new release	Released AWS CloudHSM client version 3.1.1.	June 3, 2020
Added new release	Released AWS CloudHSM client version 3.1.0.	May 21, 2020
Added new release	Released AWS CloudHSM client version 3.0.1.	April 20, 2020
Added new release	Released AWS CloudHSM client version 3.0.0 for Windows Server platform.	October 30, 2019
Added new release	Released AWS CloudHSM client version 3.0.0 for all platforms, except Windows.	October 22, 2019
Added new release	Released AWS CloudHSM client version 2.0.4.	August 26, 2019
Added new release	Released AWS CloudHSM client version 2.0.3.	May 13, 2019
Added new release	Released AWS CloudHSM client version 2.0.1.	March 21, 2019
Added new release	Released AWS CloudHSM client version 2.0.0.	February 6, 2019
Added region support	Added AWS CloudHSM support for the EU (Stockholm) and AWS GovCloud (US-East) regions.	December 19, 2018

Added new release	Released AWS CloudHSM client version 1.1.2 for Windows.	November 20, 2018
Updated known issues	New content was added to the Troubleshooting guide.	November 8, 2018
Added new release	Released AWS CloudHSM client version 1.1.2 for Linux platforms.	November 8, 2018
Added region support	Added AWS CloudHSM support for the EU (Paris) and Asia Pacific (Seoul) regions.	October 24, 2018
Added new content	Added the ability to delete and restore AWS CloudHSM backups.	September 10, 2018
Added new content	Added automatic audit log delivery to Amazon CloudWatch Logs.	August 13, 2018
Added new content	Added the ability to copy an AWS CloudHSM cluster backup across regions.	July 30, 2018
Added region support	Added AWS CloudHSM support for the EU (London) region.	June 13, 2018
Added new content	Added AWS CloudHSM client and library support for Amazon Linux 2, Red Hat Enterprise Linux (RHEL) 6, Red Hat Enterprise Linux (RHEL) 7, CentOS 6, CentOS 7, and Ubuntu 16.04 LTS.	May 10, 2018

Added new release	Added a Windows AWS CloudHSM client.	April 30, 2018

Earlier updates

The following table describes the important changes to the AWS CloudHSM prior to 2018.

Change	Description	Date
New content	Added quorum authentication (M of N access control) for crypto officers (COs). For more information, see Using CloudHSM Management Utility (CMU) to manage quorum authentication (M of N access control).	November 9, 2017
Update	Added documentation about using the key_mgmt_util command line tool. For more information, see Reference for AWS CloudHSM Key Management Utility commands.	November 9, 2017
New content	Added Oracle Transparent Data Encryption. For more information, see Oracle database encryption.	October 25, 2017
New content	Added SSL Offload. For more information, see SSL/TLS offload.	October 12, 2017

Change	Description	Date
New guide	This release introduces AWS CloudHSM	August 14, 2017