AWS Key Management Service best practices

AWS Prescriptive Guidance

AWS Prescriptive Guidance: AWS Key Management Service best practices

Table of Contents

AWS Key Management Service best practices

Frank Phillis, Ken Beer, Zach Miller, Peter M. O'Donnell, Patrick Palmer, Jeremy Stieglitz, and Dave Walker, Amazon Web Services

AWS Key Management Service (AWS KMS) is a managed service that makes it easy for you to create and control the cryptographic keys that are used to protect your data. This guide describes how to effectively use AWS KMS and provides best practices. It helps you compare configuration options and choose the best set for your needs.

This guide includes recommendations for how your organization can use AWS KMS to protect sensitive information and implement signing for multiple use cases. It considers current recommendations that use the following dimensions:

- **Managing keys** – Delegation options for management and key storage choices
- **Data protection** – Encrypting data within your own applications vs in AWS services doing it on your behalf
- **Access management** – Using AWS KMS key policies and AWS Identity and Access Management (IAM) policies to implement role-based access control (RBAC) or attribute-based access control (ABAC).
- **Multi-account and multi-Region architecture** – Recommendations for large scale deployments.
- **Billing and cost management** – Understanding your cost and usage, and recommendations for ways to reduce costs.
- **Detective controls** – Monitoring the status of your KMS keys, encryption settings, and encrypted data.
- **Incident response** – Correcting misconfigurations that result in non-compliance with your data protection policies.

Targeted business outcomes

Your data is a critical and sensitive asset for your business. With AWS KMS, you manage the cryptographic keys that are used to protect and verify your data. You control how your data is used, who has access to it, and how it is encrypted. This guide is intended to help developers, system administrators, and security professionals implement encryption best practices that help you secure sensitive data that is stored or transmitted through AWS services. By understanding

and implementing the recommendations in this guide, you can promote data confidentiality and integrity across your AWS environment. You can meet your data protection requirements, whether those requirements are formulated internally, or you have requirements specific to a compliance or validation program. For more information about how AWS KMS can help you secure data in your AWS environment, see Using AWS KMS encryption with AWS services in the AWS KMS documentation.

About AWS KMS keys

AWS Key Management Service (AWS KMS) allows you to create cryptographic keys that can be used on data that you pass to the service. The primary resource type is the KMS key, of which there are three types:

- **Advanced Encryption Standard (AES) symmetric keys** – These are 256-bit keys that are used under the Galois Counter Mode (GCM) mode of AES. These keys provide authenticated encryption and decryption of data that is less than 4 KB in size. This is the most common type of key. It is used to protect other data keys, such as those used in your applications or by AWS services that encrypt data on your behalf.
- **RSA or elliptic curve asymmetric keys** -– These keys are available in various sizes and support many algorithms. Depending on the algorithm, they can be used for encryption and decryption and for sign and verify operations.
- **Symmetric keys for performing hash-based message authentication code (HMAC) operations** – These keys are 256-bit keys that are used for sign and verify operations.

KMS keys cannot be exported from the service in plaintext. They are generated by and can only be used within the hardware security modules (HSMs) used by the service. This is a foundational security property of AWS KMS to prevent key compromise. In China (Beijing) and China (Ningxia) Regions, these HSMs are certified by OSCCA. In all other Regions, the HSMs used in AWS KMS are validated under the FIPS 140 program within NIST at Security Level 3. For more information about design and controls in AWS KMS that help protect your keys, see AWS Key Management Service Cryptographic Details.

You can submit data to AWS KMS by using various cryptographic APIs in order to perform encrypt, decrypt, sign, or verify operations with KMS keys. You can also choose to have a KMS key act like a *key-encryption key*, which protects a key type called a *data key*. A data key can be exported from AWS KMS for use within your local application or an AWS service that is protecting data on your behalf. The use of data keys is common in all key management systems and is often referred to as envelope encryption. *Envelope encryption* allows a data key to be used on the remote system that is handling your sensitive data, instead of having to send your sensitive data to AWS KMS for encryption directly under a KMS key.

For more information, see AWS KMS keys and AWS KMS cryptography essentials in the AWS KMS documentation.

Key management best practices for AWS KMS

When using AWS Key Management Service (AWS KMS), there are some fundamental design decisions that you must make. These include whether to use a centralized or decentralized model for key management and access, the type of keys to use, and the type of key store to use. The following sections help you make decisions that are right for your organization and use cases. This section concludes with important considerations for disabling and deleting KMS keys, including actions that you should take to help protect your data and keys.

This section contains the following topics:

Choosing a centralized or decentralized model

AWS recommends that you use multiple AWS accounts and manage those accounts as a single organization in AWS Organizations. There are two broad approaches to managing AWS KMS keys in multi-account environments.

The first approach is a decentralized approach, where you create keys in each account that uses those keys. When you store the KMS keys in the same accounts as the resources they protect, it is easier to delegate permissions to local administrators who understand access requirements for their AWS principals and keys. You can authorize key usage by using just a key policy, or you can combine a key policy and identity-based policies in AWS Identity and Access Management (IAM).

The second approach is a centralized approach, where you maintain KMS keys in one or a few designated AWS accounts. You allow other accounts to only use the keys for cryptographic operations. You manage keys, their life cycle, and their permissions from the centralized account. You permit other AWS accounts to use the key but don't allow other permissions. The external accounts cannot manage anything about the key's life cycle or access permission. This centralized model can help minimize the risk of unintended deletion of keys or privilege escalation by delegated administrators or users.

The option you choose depends on several factors. Consider the following when choosing an approach:

1. Do you have an automated or manual process for provisioning key and resource access? This includes resources such as deployment pipelines and infrastructure as code (IaC) templates. These tools can help you deploy and manage resources (such as KMS keys, key policies, IAM roles, and IAM policies) across many AWS accounts. If you don't have these deployment tools, a centralized approach to key management might be more manageable for your business.
2. Do you have administrative control over all of the AWS accounts that contain resources that use KMS keys? If so, a centralized model can simplify management and eliminate the need to switch AWS accounts to manage keys. Note, however, that IAM roles and user permissions to use keys still must be managed per account.
3. Do you need to offer access to use your KMS keys to customers or partners who have their own AWS accounts and resources? For these keys, a centralized approach can reduce administrative burden on your customers and partners.
4. Do you have authorization requirements for access to AWS resources that are better solved by either a centralized or local access approach? For example, if different applications or business units are responsible for managing the security of their own data, a decentralized approach to key management is better.
5. Are you exceeding service resource quotas for AWS KMS? Because these quotas are set per AWS account, a decentralized model distributes the load across accounts, effectively multiplying service quotas.

> **Note**
>
> The management model for keys is irrelevant when considering request quotas because these quotas are applied to the account principal making a request against the key, not the account that owns or manages the key.

In general, we recommend that you start with a decentralized approach unless you can articulate a need for a centralized KMS key model.

Choosing customer managed keys, AWS managed keys, or AWS owned keys

The KMS keys that you create and manage for use in your own cryptographic applications are known as *customer managed keys*. AWS services can use customer managed keys to encrypt the data the service stores on your behalf. Customer managed keys are recommended if you want full control over the life cycle and usage of your keys. There is a monthly cost to have a customer managed key in your account. In addition, requests to use or manage the key incur a usage cost. For more information, see AWS KMS pricing.

If you want an AWS service to encrypt your data but don't want the overhead or costs of managing keys, you can use an *AWS managed key*. This type of key exists in your account, but it can only be used under certain circumstances. It can only be used in the context of the AWS service that you're operating in, and it can only be used by principals within the account that contains the key. You cannot manage anything about the life cycle or permissions of these keys. Some AWS services use AWS managed keys. The format of an AWS managed key alias is `aws/<service code>`. For example, an `aws/ebs` key can only be used to encrypt Amazon Elastic Block Store (Amazon EBS) volumes in the same account as the key and can only be used by IAM principals in that account. An AWS managed key can be used only by users in that account and for resources in that account. You cannot share resources encrypted under an AWS managed key with other accounts. If this is a limitation for your use case, we recommend using a customer managed key instead; you can share use of that key with any other account. You are not charged for the existence of an AWS managed key in your account, but you are charged for any use of this key type by the AWS service that is assigned to the key.

An AWS managed key is a legacy key type that is no longer being created for new AWS services as of 2021. Instead, new (and legacy) AWS services are using an *AWS owned key* to encrypt your data by default. AWS owned keys are a collection of KMS keys that an AWS service owns and manages for use in multiple AWS accounts. Although these keys are not in your AWS account, an AWS service can use one to protect the resources in your account.

We recommend that you use customer managed keys when granular control is most important and use AWS owned keys when convenience is most important.

The following table describes the key policy, logging, management, and pricing differences between each key type. For more information about key types, see AWS KMS concepts.

Consideration	Customer managed keys	AWS managed keys	AWS owned keys
Key policy	Exclusively controlled by the customer	Controlled by service; viewable by customer	Exclusively controlled and only viewable by the AWS service that encrypts your data
Logging	AWS CloudTrail customer trail or event data store	CloudTrail customer trail or event data store	Not viewable by the customer
Life cycle management	Customer manages rotation, deletion, and AWS Region	AWS service manages rotation (annual), deletion, and Region	AWS service manages rotation (annual), deletion, and Region
Pricing	Monthly fee for key existence (pro-rated hourly); the caller is charged for API usage	No charge for key existence; the caller is charged for API usage	No charges to customer

Choosing an AWS KMS key store

A *key store* is a secure location for storing and using cryptographic key material. The industry best practice for key stores is to use a device known as a hardware security module (HSM) that has been validated under the NIST Federal Information Processing Standards (FIPS) 140 Cryptographic Module Validation Program at Security Level 3. There are other programs to support key stores that are used to process payments. AWS Payment Cryptography is a service you can use to protect data related to your payment workloads.

AWS KMS supports multiple key store types to help protect your key material when using AWS KMS to create and manage your encryption keys. All of the key store options supplied by AWS KMS are continually validated under FIPS 140 at Security Level 3. They are designed to prevent anyone, including AWS operators, from accessing your plaintext keys or using them without your permission. For more information about the available types of key stores, see Key stores in the AWS KMS documentation.

The AWS KMS standard key store is the best choice for the majority of workloads. If you need to choose a different type of key store, carefully consider whether regulatory or other requirements (such as internal) mandate making this choice, and carefully weigh the costs and benefits.

Deleting and disabling KMS keys

The deletion of a KMS key can have significant impact. Before you delete a KMS key that you no longer intend to use, consider whether it's adequate to set the key state to **Disabled**. While a key is disabled, it cannot be used for cryptographic operations. It still exists in AWS, and you can re-enable it in the future if required. Disabled keys continue to incur storage charges. We recommend that you disable keys instead of deleting them until you are confident that the key does not protect any data or data keys.

> **Important**
>
> Deleting a key must be carefully planned. Data cannot be decrypted if the corresponding key has been deleted. AWS has no means to recover a deleted key after it's been deleted. As with other critical operations in AWS, you should apply a policy that limits who can schedule keys for deletion and require multi-factor authentication (MFA) for key deletion.

To help prevent accidental key deletion, AWS KMS enforces a default minimum waiting period of seven days after the execution of a `DeleteKey` call before it deletes the key. You can set the waiting period to a maximum value of 30 days. During the waiting period, the key is still stored in AWS KMS in a **Pending Deletion** state. It can't be used for encrypt or decrypt operations. Any attempt to use a key that is in the **Pending Deletion** state for encryption or decryption is logged to AWS CloudTrail. You can set an Amazon CloudWatch alarm for these events in your CloudTrail logs. If you receive alarms on these events, you can choose to cancel the deletion process if needed. Until the waiting period has expired, you can recover the key from the **Pending Deletion** state and restore it to either a **Disabled** or **Enabled** state.

Deletion of a multi-Region key requires that you delete replicas before the original copy. For more information, see Deleting multi-Region keys.

If you are using a key with imported key material, you can delete the imported key material immediately. This is different from deleting a KMS key in several ways. When you perform the `DeleteImportedKeyMaterial` action, AWS KMS deletes the key material, and the key state changes to **Pending import**. After you delete the key material, the key is immediately unusable.

There is no waiting period. To enable use of the key again, you need to import the same key material again. The waiting period for KMS key deletion also applies to KMS keys with imported key material.

If data keys are protected by a KMS key and are actively in use by AWS services, they are not immediately affected if their associated KMS key is disabled or if its imported key material is deleted. For example, say that a key with imported material was used to encrypt an object with SSE-KMS. You are uploading the object to an Amazon Simple Storage Service (Amazon S3) bucket. Before you upload the object to the bucket, you import the material into your key. After the object is uploaded, you delete the imported key material from that key. The object remains in the bucket in an encrypted state, but no one can access the object until the deleted key material is re-imported into the key. Though this flow requires precise automation for importing and deleting key material from a key, it can provide an additional level of control within an environment.

AWS offers prescriptive guidance to help you monitor and remediate (if necessary) the scheduled deletion of KMS keys. For more information, see Monitor and remediate scheduled deletion of AWS KMS keys.

Data protection best practices for AWS KMS

This section helps you to make choices about AWS Key Management Service (AWS KMS) key usage for data protection, such as which keys to use for each data type. It also provides specific examples of using AWS KMS with different AWS services. These recommendations and examples help you understand how many keys you might need and which principals require permissions to use those keys.

The section also discusses key rotation. *Key rotation* is the practice of either replacing an existing KMS key with a new key or replacing the cryptographic material associated with an existing KMS key with new material. This guide provides examples and instructions for how to rotate KMS keys for commonly used AWS services. The recommendations and examples are designed to help you make informed choices about your key rotation strategy.

Finally, this section makes recommendations for how to use the AWS Encryption SDK, a tool for implementing client-side encryption in your applications. This section includes design choices that you can make based on the feature set and capabilities of the AWS Encryption SDK.

This section discusses the following encryption topics:

- Encryption with AWS KMS
- Key rotation and scope of impact
- Recommendations for using the AWS Encryption SDK

Encryption

Encryption is a general best practice to protect the confidentiality and integrity of sensitive information. You should use your existing data classification levels and have at least one AWS Key Management Service (AWS KMS) key per level. For example, you could define a KMS key for data classified as **Confidential**, one for **Internal-Only**, and one for **Sensitive**. This helps you make sure that only authorized users have permissions to use the keys associated with each classification level.

> **Note**
>
> A single customer managed KMS key can be used across any combination of AWS services or your own applications that store data of a particular classification. The limiting factor

in using a key across multiple workloads and AWS services is how complex the usage permissions need to be to control access to the data across a set of users. The AWS KMS key policy JSON document must be less than 32 KB. If this size restriction becomes a limitation, consider using AWS KMS grants or creating multiple keys to minimize the size of the key policy document.

Instead of relying only on data classification to partition your KMS key, you could also choose to assign a KMS key to be used for a data classification within a single AWS service. For example, all data tagged `Sensitive` in Amazon Simple Storage Service (Amazon S3) should be encrypted under a KMS key that has a name like `S3-Sensitive`. You can further distribute your data across multiple KMS keys within your defined data classification and AWS service or application. For example, you might be able to delete some datasets in a specific time-period and delete other datasets in a different time period. You can use resource tags to help you identify and sort data that is encrypted with specific KMS keys.

If you choose a decentralized management model for KMS keys, you should apply guardrails to make sure that new resources with a given classification are created and use the expected KMS keys with the right permissions. For more information about how you can enforce, detect, and manage resource configuration by using automation, see the Detection and monitoring section of this guide.

This section discusses the following encryption topics:

- Log data encryption with AWS KMS
- Encryption by default
- Database encryption with AWS KMS
- PCI DSS data encryption with AWS KMS
- Using KMS keys with Amazon EC2 Auto Scaling

Log data encryption with AWS KMS

Many AWS services, such as Amazon GuardDuty and AWS CloudTrail, offer options to encrypt log data that is sent to Amazon S3. When exporting findings from GuardDuty to Amazon S3, you must use a KMS key. We recommend that you encrypt all log data and granting decrypt access only to authorized principals, such as security teams, incident responders, and auditors.

The AWS Security Reference Architecture recommends creating a central AWS account for logging. When you do this, you can also reduce your key management overhead. For example, with CloudTrail, you can create an organization trail or event data store to log events across your organization. When you configure your organizational trail or event data store, you can specify a single Amazon S3 bucket and KMS key in your designated logging account. This configuration applies to all member accounts in the organization. All accounts then send their CloudTrail logs to the Amazon S3 bucket in the logging account, and the log data is encrypted with the specified KMS key. You need to update the key policy for this KMS key to grant CloudTrail the necessary permissions to use the key. For more information, see Configure AWS KMS key policies for CloudTrail in the CloudTrail documentation.

To help protect the GuardDuty and CloudTrail logs, the Amazon S3 bucket and KMS key must be in the same AWS Region. The AWS Security Reference Architecture also provides guidance on logging and multi-account architectures. When aggregating logs across multiple Regions and accounts, review Creating a trail for an organization in the CloudTrail documentation to learn more about opt-in Regions and make sure that your centralized logging works as designed.

Encryption by default

AWS services that store or process data typically offer encryption at rest. This security feature helps protect your data by encrypting it when it's not in use. Authorized users can still access it when needed.

The implementation and encryption options vary between AWS services. Many provide encryption by default. It's important to understand how encryption works for each service that you use. The following are some examples:

- **Amazon Elastic Block Store (Amazon EBS)** – When you enable encryption by default, all new Amazon EBS volumes and snapshot copies are encrypted. AWS Identity and Access Management (IAM) roles or users can't launch instances with unencrypted volumes or volumes that don't support encryption. This feature helps with security, compliance, and auditing by making sure that all data stored on Amazon EBS volumes is encrypted. For more information about encryption in this service, see Amazon EBS encryption in the Amazon EBS documentation.
- **Amazon Simple Storage Service (Amazon S3)** – All new objects are encrypted by default. Amazon S3 automatically applies server-side encryption with Amazon S3 managed keys (SSE-S3) for each new object, unless you specify a different encryption option. IAM principals can still upload unencrypted objects to Amazon S3 by explicitly stating so in the API call. In Amazon S3, to enforce SSE-KMS encryption, you must use a bucket policy with conditions that require

encryption. For a sample policy, see Require SSE-KMS for all objects written to a bucket in the Amazon S3 documentation. Some Amazon S3 buckets receive and serve large numbers of objects. If those objects are encrypted with KMS keys, a large number of Amazon S3 operations results in a large number of `GenerateDataKey` and `Decrypt` calls to AWS KMS. This can increase the charges you incur for AWS KMS usage. You can configure Amazon S3 bucket keys, which may significantly reduce your AWS KMS costs. For more information about encryption in this service, see Protecting data with encryption in the Amazon S3 documentation.

- **Amazon DynamoDB** – DynamoDB is a fully managed NoSQL database service that enables server-side encryption at rest by default, and you can't disable it. We recommend that you use a customer managed key for encrypting your DynamoDB tables. This approach helps you implement least privilege with granular permissions and separation of duties by targeting specific IAM users and roles in your AWS KMS key policies. You can also choose AWS managed or AWS owned keys when configuring encryption settings for your DynamoDB tables. For data requiring a high degree of protection (where data should only be visible as cleartext to the client), consider using client-side encryption with the AWS Database Encryption SDK. For more information about encryption in this service, see Data protection in the DynamoDB documentation.

Database encryption with AWS KMS

The level at which you implement encryption affects database functionality. The following are the tradeoffs that you must consider:

- If you use only AWS KMS encryption, the storage that backs your tables is encrypted for DynamoDB and Amazon Relational Database Service (Amazon RDS). This means that the operating system that runs the database sees the contents of the storage as cleartext. All database functions, including index generation and other higher-order functions that require access to the cleartext data, continue to work as expected.
- Amazon RDS builds on Amazon Elastic Block Store (Amazon EBS) encryption to provide full disk encryption for database volumes. When you create an encrypted database instance with Amazon RDS, Amazon RDS creates an encrypted Amazon EBS volume on your behalf to store the database. Data stored at rest on the volume, database snapshots, automated backups, and read replicas are all encrypted under the KMS key that you specified when you created the database instance.
- Amazon Redshift integrates with AWS KMS and creates a four-tier hierarchy of keys that are used to encrypt the cluster level through the data level. When you launch your cluster, you can choose

to use AWS KMS encryption. Only the Amazon Redshift application and users with appropriate permissions can see cleartext when the tables are opened (and decrypted) in memory. This is broadly analogous to the *transparent* or *table-based data encryption (TDE)* features that are available in some commercial databases. This means that all database functions, including index generation and other higher-order functions that require access to the cleartext data, continue to work as expected.

- The client-side data-level encryption implemented through the AWS Database Encryption SDK (and similar tools) means that both the operating system and the database see only ciphertext. Users can view cleartext only if they access the database from a client that has the AWS Database Encryption SDK installed and they have access to the relevant key. Higher-order database functions that require access to cleartext to work as intended—such as index generation—won't work if directed to operate on encrypted fields. When choosing to use client-side encryption, make sure that you use a robust encryption mechanism that helps prevent common attacks against encrypted data. This includes using a strong encryption algorithm and appropriate techniques, such as a salt, to help mitigate ciphertext attacks.

We recommend using the AWS KMS integrated encryption capabilities for AWS database services. For workloads that process sensitive data, client-side encryption should be considered for the sensitive data fields. When using client-side encryption, you should consider the impact to database access, such as joins within SQL queries or index creation.

PCI DSS data encryption with AWS KMS

The security and quality controls in AWS KMS have been validated and certified to meet the requirements of the Payment Card Industry Data Security Standard (PCI DSS). This means that you can encrypt primary account number (PAN) data with a KMS key. The use of a KMS key to encrypt data removes some of the burden of managing encryption libraries. Additionally, KMS keys cannot be exported from AWS KMS, which reduces concern about the encryption keys being stored in an unsecure manner.

There are other ways you can use AWS KMS to meet PCI DSS requirements. For example, if you are using AWS KMS with Amazon S3, you can store PAN data in Amazon S3 because the access control mechanism for each service is distinct from the other.

As always, when reviewing your compliance requirements, make sure that you obtain advice from suitably experienced, qualified, and verified parties. Be aware of the AWS KMS request quotas when

you design applications that use the key directly to protect card transaction data that is in scope of the PCI DSS.

Because all AWS KMS requests are logged in AWS CloudTrail, you can audit key usage by reviewing the CloudTrail logs. However, if you use Amazon S3 bucket keys, there is no entry that corresponds to every Amazon S3 action. This is because the bucket key encrypts the data keys that you use to encrypt the objects in Amazon S3. While the use of a bucket key does not eliminate all API calls to AWS KMS, it reduces the number of them. As a result, there is no longer a one-to-one match between Amazon S3 object access attempts and API calls to AWS KMS.

Using KMS keys with Amazon EC2 Auto Scaling

Amazon EC2 Auto Scaling is a recommended service for automating the scaling of your Amazon EC2 instances. It helps you make sure that you have the correct number of instances available to handle the load for your application. Amazon EC2 Auto Scaling uses a service-linked role that provides appropriate permissions to the service and authorizes its activities within your account. To use KMS keys with Amazon EC2 Auto Scaling, your AWS KMS key policies must allow the service linked role to use your KMS key with some API operations, such as `Decrypt`, for the automation to be useful. If the AWS KMS key policy does not authorize the IAM principal who is performing the operation to conduct an action, that action will be denied. For more information about how to correctly apply permissions in the key policy to allow access, see Data protection in Amazon EC2 Auto Scaling in the Amazon EC2 Auto Scaling documentation.

Key rotation for AWS KMS and scope of impact

We do not recommend AWS Key Management Service (AWS KMS) key rotation unless you are required to rotate keys for regulatory compliance. For example, you might be required to rotate your KMS keys due to business policies, contract rules, or government regulations. The design of AWS KMS significantly reduces the types of risk that key rotation is typically used to mitigate. If you must rotate KMS keys, we recommend that you use automatic key rotation and use manual key rotation only if automatic key rotation is not supported.

This section discusses the following key rotation topics:

- AWS KMS symmetric key rotation
- Manual key rotation for Amazon EBS volumes
- Key rotation for Amazon RDS

- Key rotation for Amazon S3 and Same-Region Replication
- Rotating KMS keys with imported material

AWS KMS symmetric key rotation

AWS KMS supports automatic key rotation only for symmetric encryption KMS keys with key material that AWS KMS creates. Automatic rotation is optional for customer managed KMS keys. On an annual basis, AWS KMS rotates the key material for AWS managed KMS keys. AWS KMS saves all previous versions of the cryptographic material in perpetuity, so you can decrypt any data that is encrypted with that KMS key. AWS KMS does not delete any rotated key material until you delete the KMS key. Also, when you decrypt an object by using AWS KMS, the service determines the correct backing material to use for the decrypt operation; no additional input parameters need to be supplied.

Because AWS KMS retains previous versions of the cryptographic key material and because you can use that material to decrypt data, key rotation doesn't provide any additional security benefits. The key rotation mechanism exists to make it easier to rotate keys if you are operating a workload in a context where regulatory or other requirements demand it.

Key rotation for Amazon EBS volumes

You can rotate Amazon Elastic Block Store (Amazon EBS) data keys by using one of the following approaches. The approach depends on your workflows, deployment methods, and application architecture. You might want to do this when changing from an AWS managed key to a customer managed key.

To use operating system tools to copy the data from one volume to another

1. Create the new KMS key. For instructions, see Create a KMS key.
2. Create a new Amazon EBS volume that is the same size as or larger than the original. For encryption, specify the KMS key that you created. For instructions, see Create an Amazon EBS volume.
3. Mount the new volume on the same instance or container as the original volume. For instructions, see Attach an Amazon EBS volume to an Amazon EC2 instance.
4. Using your preferred operating system tool, copy data from the existing volume to the new volume.

5. When the sync is complete, during a pre-scheduled maintenance window, stop the traffic to the instance. For instructions, see Manually stop and start your instances.
6. Unmount the original volume. For instructions, see Detach an Amazon EBS volume from an Amazon EC2 instance.
7. Mount the new volume to the original mount point.
8. Verify that the new volume is operating correctly.
9. Delete the original volume. For instructions, see Delete an Amazon EBS volume.

To use an Amazon EBS snapshot to copy the data from one volume to another

1. Create the new KMS key. For instructions, see Create a KMS key.
2. Create an Amazon EBS snapshot of the original volume. For instructions, see Create Amazon EBS snapshots.
3. Create a new volume from the snapshot. For encryption, specify the new KMS key that you created. For instructions, see Create an Amazon EBS volume.

> **Note**
>
> Depending on your workload, you might want to use Amazon EBS fast snapshot restore to minimize initial latency on the volume.

4. Create a new Amazon EC2 instance. For instructions, see Launch an Amazon EC2 instance.
5. Attach the volume that you created to the Amazon EC2 instance. For instructions, see Attach an Amazon EBS volume to an Amazon EC2 instance.
6. Rotate the new instance into production.
7. Rotate the original instance out of production and delete it. For instructions, see Delete an Amazon EBS volume.

> **Note**
>
> It is possible to copy snapshots and modify the encryption key used for the target copy. After you copy the snapshot and encrypt it with your preferred KMS keys, you can also create an Amazon Machine Image (AMI) from snapshots. For more information, see Amazon EBS encryption in the Amazon EC2 documentation.

Key rotation for Amazon RDS

For some services, such as Amazon Relational Database Service (Amazon RDS), data encryption occurs within the service and is provided by AWS KMS. Use the following instructions to rotate a key for an Amazon RDS database instance.

To rotate a KMS key for an Amazon RDS database

1. Create a snapshot of the original encrypted database. For instructions, see Managing manual backups in the Amazon RDS documentation.
2. Copy the snapshot to a new snapshot. For encryption, specify the new KMS key. For instructions, see Copying a DB snapshot for Amazon RDS.
3. Use the new snapshot to create a new Amazon RDS cluster. For instructions, see Restoring to a DB instance in the Amazon RDS documentation. By default, the cluster uses the new KMS key.
4. Verify the operation of the new database and the data in it.
5. Rotate the new database into production.
6. Rotate the old database out of production and delete it. For instructions, see Deleting a DB instance.

Key rotation for Amazon S3 and Same-Region Replication

For Amazon Simple Storage Service (Amazon S3), to change the encryption key of an object, you need to read and rewrite the object. When you rewrite the object, you explicitly specify the new encryption key in the write operation. To do this for many objects, you can use Amazon S3 Batch Operations. Within the job settings, for the copy operation, specify the new encryption settings. For example, you might choose **SSE-KMS** and enter the **keyId**.

Alternatively, you could use Amazon S3 Same-Region Replication (SRR). SSR can re-encrypt the objects in transit.

Rotating KMS keys with imported material

AWS KMS does not recover or rotate your imported key material. To rotate a KMS key with imported key material, you must rotate the key manually.

Recommendations for using the AWS Encryption SDK

The AWS Encryption SDK is a powerful tool for implementing client-side encryption in your applications. Libraries are available for Java, JavaScript, C, Python, and other programming languages. It integrates with AWS Key Management Service (AWS KMS). You can also use it as a stand-alone SDK without referencing KMS keys.

Recommended practices for using this tool include carefully considering the requirements of your application. Balance those requirements against risks that can be introduced by certain configurations, such as introducing key caching into your application. For more information about data key caching, see Data key caching in the AWS Encryption SDK documentation.

Consider the following questions when determining whether to use the AWS Encryption SDK:

- Is there a requirement for client-side encryption that cannot be met by server-side encryption with services that integrate with AWS KMS?
- Can you adequately protect the keys that are used to encrypt data client-side, and how will you do that?
- Are there other, fit-for-purpose encryption libraries that might fit your use case more appropriately? Consider alternative AWS offerings, such as Amazon S3 client-side encryption and the AWS Database Encryption SDK.

Find more information about choosing the right service for your use case, see the AWS Crypto Tools Documentation.

Identity and access management best practices for AWS KMS

To use AWS Key Management Service (AWS KMS), you must have credentials that AWS can use to authenticate and authorize your requests. No AWS principal has any permissions to a KMS key unless that permission is provided explicitly and never denied. There are no implicit or automatic permissions to use or manage a KMS key. The topics in this section define security best practices to help you determine which AWS KMS access management controls you should use to secure your infrastructure.

This section discusses the following identity and access management topics:

- AWS KMS key policies and IAM policies
- Least-privilege permissions for AWS KMS
- Role-based access control for AWS KMS
- Attribute-based access control for AWS KMS
- Encryption context for AWS KMS
- Troubleshooting AWS KMS permissions

AWS KMS key policies and IAM policies

The primary way to manage access to your AWS KMS resources is with *policies*. Policies are documents that describe which principals can access which resources. Policies that are attached to an AWS Identity and Access Management (IAM) identity (users, groups of users, or roles) are called identity-based policies. IAM policies that attach to resources are called resource-based policies. AWS KMS resource policies for KMS keys are called key policies. In addition to IAM policies and AWS KMS key policies, AWS KMS supports grants. Grants provide a flexible and powerful way to delegate permissions. You can use grants to issue time-bound KMS key access to IAM principals in your AWS account or in other AWS accounts.

All KMS keys have a key policy. If you don't provide one, AWS KMS creates one for you. The default key policy that AWS KMS uses differs depending on whether you create the key using the AWS KMS console or you use the AWS KMS API. We recommend that you edit the default key policy to align with your organization's requirements for least-privilege permissions. This should also align with your strategy for using IAM policies in conjunction with key policies. For more

recommendations on using IAM policies with AWS KMS, see Best practices for IAM policies in the AWS KMS documentation.

You can use the key policy to delegate authorization for an IAM principal to the identity-based policy. You can also use the key policy to refine authorization in conjunction with the identity-based policy. In either case, both the key policy and identity-based policy determine access, along with any other applicable policies that scope access, such as service control policies (SCPs), resource control policies (RCPs), or permission boundaries. If the principal is in a different account than the KMS key, essentially, only cryptographic and grant actions are supported. For more information about this cross-account scenario, see Allowing users in other accounts to use a KMS key in the AWS KMS documentation.

You must use IAM identity-based policies in combination with key policies to control access to your KMS keys. Grants can also be used in combination with these policies to control access to a KMS key. To use an identity-based policy to control access to a KMS key, the key policy must allow the account to use identity-based policies. You can either specify a key policy statement that enables IAM policies, or you can explicitly specify allowed principals in the key policy.

When writing policies, make sure that you have strong controls that restrict who can perform the following actions:

- **Update, create, and delete IAM policies and KMS key policies**
- **Attach and detach identity-based policies from users, roles, and groups**
- **Attach and detach AWS KMS key polices from KMS keys**
- **Create grants for your KMS keys** – Whether you control access to your KMS keys exclusively with key policies, or you combine key policies with IAM policies, you should restrict the ability to modify the policies. Implement an approval process for changing any existing policies. An approval process can help prevent the following:
  - **Accidental loss of IAM principal permissions** – It's possible to make changes that would prevent IAM principals from being able to manage the key or use it in cryptographic operations. In extreme scenarios, it's possible to revoke key management permissions from all users. If this happens, you need to contact AWS Support to regain access to the key.
  - **Unapproved changes to KMS key policies** – If an unauthorized user gains access to the key policy, they could modify it to delegate permissions to an unintended AWS account or principal.
  - **Unapproved changes to IAM policies** – If an unauthorized user obtains a set of credentials with permissions to manage a group's membership, they could elevate their own permissions

and make changes to your IAM policies, key policies, KMS key configuration, or other AWS resource configurations.

Carefully review the IAM roles and users that are associated with the IAM principals who are designated as your KMS key administrators. This can help prevent unauthorized deletion or changes. If you need to change the principals who have access to your KMS keys, verify that the new administrator principals are added to all required key policies. Test their permissions before you delete the previous managing principal. We strongly recommend following all IAM security best practices and using temporary credentials instead of the long-term credentials.

We recommend issuing time-bound access through grants if you don't know the names of the principals at the time that the policies are created or if the principals that require access frequently change. The grantee principal can be in the same account as the KMS key or in a different account. If the principal and KMS key are in different accounts, then you must specify an identity-based policy in addition to the grant. Grants require additional management because you must call an API to create the grant and to retire or revoke the grant when it is no longer needed.

No AWS principal, including the account root user or key creator, has any permissions to a KMS key unless they are explicitly allowed and not explicitly denied in a key policy, IAM policy, or grant. By extension, you should consider what would happen if a user gains unintended access to use a KMS key and what the impact would be. To mitigate such a risk, consider the following:

- You can maintain different KMS keys for different categories of data. This helps you separate keys and maintain more concise key policies that contain policy statements that specifically target principal access to that data category. It also means that if relevant IAM credentials are accessed unintentionally, the identity tied to that access has access only to the keys specified in the IAM policy and only if the key policy allows access to that principal.
- You can assess if a user with unintended access to the key can access the data. For example, with Amazon Simple Storage Service (Amazon S3), the user must also have the appropriate permissions to access encrypted objects in Amazon S3. Alternately, if a user has unintended access (by using RDP or SSH) to an Amazon EC2 instance that has a volume encrypted with a KMS key, the user could access the data by using operating system tools.

Note

AWS services that use AWS KMS don't expose the ciphertext to users (most current approaches to cryptoanalysis require access to the ciphertext). Additionally, ciphertext

is not available for physical examination outside of an AWS data center because all storage media is physically destroyed when it is decommissioned, in accordance with NIST SP800-88 requirements.

Least-privilege permissions for AWS KMS

Because your KMS keys protect sensitive information, we recommend following the principle of least privileged access. Delegate the minimum permissions required to perform a task when you define your key policies. Allow all actions (`kms:*`) on a KMS key policy only if you plan to further restrict permissions with additional identity-based policies. If you plan to manage permissions with identity-based policies, limit who has the ability to create and attach IAM policies to IAM principals and monitor for policy changes.

If you allow all actions (`kms:*`) in both the key policy and the identity-based policy, the principal has both administrative and usage permissions to the KMS key. As a security best practice, we recommend delegating these permissions only to specific principals. Consider how you assign permissions to principals who will manage your keys and principals who will use your keys. You can do this by explicitly naming the principal in the key policy or by limiting which principals the identity-based policy is attached to. You can also use condition keys to restrict permissions. For example, you can use the aws:PrincipalTag to allow all actions if the principal making the API call has the tag specified in the condition rule.

For help understanding how policy statements are evaluated in AWS, see Policy evaluation logic in the IAM documentation. We recommend reviewing this topic before writing policies to help reduce the chance that your policy has unintended effects, such as providing access to principals that should not have access.

Tip

When testing an application in a non-production environment, use AWS Identity and Access Management Access Analyzer (IAM Access Analyzer) to help you apply least-privilege permissions in your IAM policies.

If you use IAM users instead of IAM roles, we strongly recommend using AWS multi-factor authentication (MFA) to mitigate the vulnerability of long-term credentials. You can use MFA to do the following:

- Require that users validate their credentials with MFA before performing privileged actions, such as scheduling key deletion.
- Split ownership of an administrator account password and MFA device between individuals to implement split authorization.

For sample policies that can help you configure least-privilege permissions, see IAM policy examples in the AWS KMS documentation.

Role-based access control for AWS KMS

Role-based access control (RBAC) is an authorization strategy that provides users with only the permissions required to perform their job duties, and nothing more. It is an approach that can help you implement the principle of least privilege.

AWS KMS supports RBAC. It allows you to control access to your keys by specifying granular permissions within key policies. Key policies specify a resource, action, effect, principal, and optional conditions to grant access to keys. To implement RBAC in AWS KMS, we recommend separating the permissions for key users and key administrators.

For key users, assign only the permissions that the user needs. Use the following questions to help you further refine permissions:

- Which IAM principals need access to the key?
- Which actions does each principal need to perform with the key? For example, does the principal need only `Encrypt` and `Sign` permissions?
- Which resources does the principal need to access?
- Is the entity a human or an AWS service? If it's a service, you can use the kms:ViaService condition key to restrict key usage to a specific service.

For key administrators, assign only the permissions that the administrator needs. For example, an administrator's permissions might vary depending on whether the key is used in test or production environments. If you use less restrictive permissions in certain non-production environments, implement a process to test the policies before they are released to production.

For sample policies that can help you configure role-based access control for key users and administrators, see RBAC for AWS KMS.

Attribute-based access control for AWS KMS

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes. Like RBAC, it is an approach that can help you implement the principle of least privilege.

AWS KMS supports ABAC by allowing you to define permissions based on tags associated with the target resource, such as a KMS key, and tags associated with the principal making the API call. In AWS KMS, you can use tags and aliases to control access to your customer managed keys. For example, you can define IAM policies that use tag condition keys to allow operations when the principal's tag matches the tag associated with the KMS key. For a tutorial, see Define permissions to access AWS resources based on tags in the AWS KMS documentation.

As a best practice, use ABAC strategies to simplify IAM policy management. With ABAC, administrators can use tags to allow access to new resources instead of updating existing policies. ABAC requires fewer policies because you don't have to create different policies for different job functions. For more information, see Comparison of ABAC to the traditional RBAC model in the IAM documentation.

Apply the best practice of least-privilege permissions to the ABAC model. Provide IAM principals with only the permissions they need to perform their jobs. Carefully control access to tagging APIs that would allow users to modify tags on roles and resources. If you use key alias condition keys to support ABAC in AWS KMS, make sure that you also have strong controls that restrict who can create keys and modify aliases.

You can also use tags to link a specific key to a business category and verify that the correct key is being used for a given action. For example, you can use AWS CloudTrail logs to verify that the key used to perform a specific AWS KMS action belongs to the same business category as the resource that it's being used on.

> **Warning**
>
> Do not include confidential or sensitive information in the tag key or tag value. Tags are not encrypted. They are accessible to many AWS services, including billing.

Before implementing an ABAC approach to your access control, consider whether the other services that you use support this approach. For help determining which services support ABAC, see AWS services that work with IAM in the IAM documentation.

For more information about implementing ABAC for AWS KMS and the conditions keys that can help you configure policies, see ABAC for AWS KMS.

Encryption context for AWS KMS

All AWS KMS cryptographic operations with symmetric encryption KMS keys accept an encryption context. *Encryption context* is an optional set of non-secret key–value pairs that can contain additional contextual information about the data. As a best practice, you can insert encryption context in `Encrypt` operations in AWS KMS to enhance the authorization and auditability of your decryption API calls to AWS KMS. AWS KMS uses the encryption context as additional authenticated data (AAD) to support authenticated encryption. The encryption context is cryptographically bound to the ciphertext so that the same encryption context is required to decrypt the data.

The encryption context is not secret and not encrypted. It appears in plaintext in AWS CloudTrail logs so that you can use it to identify and categorize your cryptographic operations. Because the encryption context is not secret, you should allow only authorized principals to access your CloudTrail log data.

You can also use the kms:EncryptionContext:context-key and kms:EncryptionContextKeys condition keys to control access to a symmetric encryption KMS key based on the encryption context. You can also use these condition keys to require that encryption contexts are used in cryptographic operations. For these condition keys, review the guidance about the use of `ForAnyValue` or `ForAllValues` set operators in order to make sure that your policies reflect your intended permissions.

Troubleshooting AWS KMS permissions

When you write access control policies for a KMS key, consider how the IAM policy and key policy work together. The effective permissions for a principal are the permissions that are granted (and not explicitly denied) by all of the effective policies. Within an account, the permissions to a KMS key can be affected by IAM identity-based policies, key policies, permissions boundaries, service control policies, or session policies. For example, if you use both identity-based and key policies to control access to the KMS key, all policies relating to both the principal and the resource are evaluated to determine a principal's authorization to perform a given action. For more information, see Policy evaluation logic in the IAM documentation.

For detailed information and a flowchart for troubleshooting key access, see Troubleshooting key access in the AWS KMS documentation.

To troubleshoot an access denied error message

1. Confirm that the IAM identity-based policies and KMS key policies allow access.
2. Confirm that a permissions boundary in IAM is not restricting access.
3. Confirm that a service control policy (SCP) or resource control policy (RCP) in AWS Organizations is not restricting access.
4. If you are using VPC endpoints, confirm that the endpoint policies are correct.
5. In the identity-based policies and key policies, remove any conditions or resource references that restrict access to the key. After removing these restrictions, confirm that the principal can successfully call the API that previously failed. If successful, reapply the conditions and resource references one at a time and, after each, verify that the principal still has access. This helps you identify the condition or resource reference that is causing the error.

For more information, see Troubleshooting access denied error messages in the IAM documentation.

Detection and monitoring best practices for AWS KMS

Detection and monitoring are an important part of understanding the availability, state, and usage of your AWS Key Management Service (AWS KMS) keys. Monitoring helps maintain the security, reliability, availability, and performance of your AWS solutions. AWS provides several tools for monitoring your KMS keys and AWS KMS operations. This section describes how to configure and use these tools to gain greater visibility into your environment and monitor the usage of your KMS keys.

This section discusses the following detection and monitoring topics:

- Monitoring AWS KMS operations with AWS CloudTrail
- Monitoring access to KMS keys with IAM Access Analyzer
- Monitoring the encryption settings of other AWS services with AWS Config
- Monitoring KMS keys with Amazon CloudWatch alarms
- Automating responses with Amazon EventBridge

Monitoring AWS KMS operations with AWS CloudTrail

AWS KMS is integrated with AWS CloudTrail, a service that can record all calls to AWS KMS by users, roles, and other AWS services. CloudTrail captures all API calls to AWS KMS as events, including calls from the AWS KMS console, AWS KMS APIs, AWS CloudFormation, the AWS Command Line Interface (AWS CLI), and AWS Tools for PowerShell.

CloudTrail logs all AWS KMS operations, including read-only operations, such as `ListAliases` and `GetKeyRotationStatus`. It also logs operations that manage KMS keys, such as `CreateKey` and `PutKeyPolicy, and cryptographic operations, such as GenerateDataKey` and `Decrypt`. It also logs internal operations that AWS KMS calls for you, such as `DeleteExpiredKeyMaterial`, `DeleteKey`, `SynchronizeMultiRegionKey`, and `RotateKey`.

CloudTrail is enabled on your AWS account when you create it. By default, the Event history provides a viewable, searchable, downloadable, and immutable record of the past 90 days of recorded management-event API activity in an AWS Region. To monitor or audit the usage of your KMS keys beyond the 90 days, we recommend creating a CloudTrail trail for your AWS account. If

you have created an organization in AWS Organizations, you can create an organization trail or an event data store that logs events for all AWS accounts in that organization.

After you establish a trail for your account or organization, you can use other AWS services to store, analyze, and automatically respond to events that are logged in the trail. For example, you can do the following:

- You can set up Amazon CloudWatch alarms that notify you of certain events in the trail. For more information, see Monitoring KMS keys with Amazon CloudWatch alarms in this guide.
- You can create Amazon EventBridge rules that automatically perform an action when an event occurs in the trail. For more information, see Automating responses with Amazon EventBridge in this guide.
- You can use Amazon Security Lake to collect and store logs from multiple AWS services, including CloudTrail. For more information, see Collecting data from AWS services in Security Lake in the Amazon Security Lake documentation.
- To enhance your analysis of operational activity, you can query CloudTrail logs with Amazon Athena. For more information, see Query AWS CloudTrail logs in the Amazon Athena documentation.

For more information about monitoring AWS KMS operations with CloudTrail, see the following:

- Logging AWS KMS API calls with AWS CloudTrail
- Examples of AWS KMS log entries
- Monitor KMS keys with Amazon EventBridge
- CloudTrail integration with Amazon EventBridge

Monitoring access to KMS keys with IAM Access Analyzer

AWS Identity and Access Management Access Analyzer (IAM Access Analyzer) helps you identify the resources in your organization and accounts (such as KMS keys) that are shared with an external entity. This service can help you identify unintended or overly broad access to your resources and data, which is a security risk. IAM Access Analyzer identifies resources that are shared with external principals by using logic-based reasoning to analyze the resource-based policies in your AWS environment.

You can use IAM Access Analyzer to identify which external entities have access to your KMS keys. When you enable IAM Access Analyzer, you create an analyzer for an entire organization or for a target account. The organization or account you choose is known as the *zone of trust* for the analyzer. The analyzer monitors the supported resources within the zone of trust. Any access to resources by principals within the zone of trust is considered trusted.

For KMS keys, IAM Access Analyzer analyzes the key policies and grants applied to a key. It generates a finding if a key policy or grant allows an external entity to access the key. Use IAM Access Analyzer to determine if external entities have access to your KMS keys, and then verify whether those entities should have access.

For more information about using IAM Access Analyzer to monitor KMS key access, see the following:

- Using AWS Identity and Access Management Access Analyzer
- IAM Access Analyzer resource types for external access
- IAM Access Analyzer resource types: AWS KMS keys
- Findings for external and unused access

Monitoring the encryption settings of other AWS services with AWS Config

AWS Config provides a detailed view of the configuration of AWS resources in your AWS account. You can use AWS Config to verify that the AWS services that use your KMS keys have their encryption settings configured appropriately. For example, you can use the encrypted-volumes AWS Config rule to validate that your Amazon Elastic Block Store (Amazon EBS) volumes are encrypted.

AWS Config includes *managed rules* that help you quickly choose rules against which to assess your resources. Check AWS Config in your AWS Regions to determine if the managed rules you need are supported in that Region. Available managed rules include checks for configuration of Amazon Relational Database Service (Amazon RDS) snapshots, CloudTrail trail encryption, default encryption for Amazon Simple Storage Service (Amazon S3) buckets, Amazon DynamoDB table encryption, and more.

You can also create *custom rules* and apply your business logic to determine whether your resources are compliant with your requirements. Open source code for many managed rules is

available in the AWS Config Rules Repository on GitHub. These can be a useful starting point for developing your own custom rules.

When a resource is noncompliant with a rule, you can initiate responsive actions. AWS Config includes remediation actions that AWS Systems Manager Automation carries out. For example, if you have applied the cloud-trail-encryption-enabled rule and the rule returns a `NON_COMPLIANT` result, AWS Config can initiate an Automation document that remediates the problem by encrypting the CloudTrail logs for you.

AWS Config lets you proactively check for compliance with AWS Config rules before you provision resources. Applying rules in proactive mode helps you evaluate the configurations of your cloud resources before they are created or updated. Applying rules in proactive mode as part of your deployment pipeline lets you test resource configurations before you deploy your resources.

You can also implement AWS Config rules as controls through AWS Security Hub CSPM. Security Hub CSPM offers security standards that you can apply to your AWS accounts. These standards help you assess your environment against recommend practices. The AWS Foundational Security Best Practices standard includes controls within the protect control category to verify that encryption at rest is configured and that KMS key policies follow recommended practices.

For more information about using AWS Config to monitor the encryption settings in AWS services, see the following:

- Getting started with AWS Config
- AWS Config managed rules
- AWS Config custom rules
- Remediating noncompliant resources with AWS Config

Monitoring KMS keys with Amazon CloudWatch alarms

Amazon CloudWatch monitors your AWS resources and the applications you run on AWS in real time. You can use CloudWatch to collect and track *metrics*, which are variables that you can measure.

The expiration of imported key material, or the deletion of a key, are potentially catastrophic events if they are unintended or not properly planned for. We recommend that you configure CloudWatch alarms to alert you to these events before they occur. We also recommend that you

configure AWS Identity and Access Management (IAM) policies or AWS Organizations service control policies (SCPs) to prevent the deletion of important keys.

CloudWatch alarms help you take corrective action, such as cancelling key deletion, or remediation actions, such as reimporting deleted or expired key material.

Automating responses with Amazon EventBridge

You can also use Amazon EventBridge to notify you of important events that affect your KMS keys. EventBridge is an AWS service that delivers a near real-time stream of system events that describe changes to AWS resources. EventBridge automatically receives events from CloudTrail and Security Hub CSPM. In EventBridge, you can create rules that respond to events recorded by CloudTrail.

AWS KMS events include the following:

- The key material in a KMS key was automatically rotated
- The imported key material in a KMS key expired
- A KMS key that had been scheduled for deletion was deleted

These events can initiate additional actions in your AWS account. These actions are different from the CloudWatch alarms described in the previous section because they can only be acted on after the event occurs. For example, you might want to delete resources that are connected to a specific key after that key has been deleted, or you might want to inform a compliance or auditing team that the key has been deleted.

You can also filter for any other API event that is logged in CloudTrail by using EventBridge. This means that if key policy-related API actions are of specific concern, you can filter for them. For example, you could filter in EventBridge for the `PutKeyPolicy` API action. More broadly, you can filter for any API action that starts with `Disable*` or `Delete*` to initiate automated responses.

Using EventBridge, you can monitor (which is a detective control) and investigate and respond (which are responsive controls) to unexpected or selected events. For example, you can alert security teams and take specific actions if an IAM user or role is created, when a KMS key is created, or when a key policy is changed. You can create an EventBridge event rule that filters the API actions you specify and then associate targets to the rule. Example targets include AWS Lambda functions, Amazon Simple Notification Service (Amazon SNS) notifications, Amazon Simple Queue Service (Amazon SQS) queues, and more. For more information about sending events to targets, see Event bus targets in Amazon EventBridge.

For more information about monitoring AWS KMS with EventBridge and automating responses, see Monitor KMS keys with Amazon EventBridge in the AWS KMS documentation.

Cost and billing management best practices for AWS KMS

Through breadth and depth, AWS services offer the flexibility to manage your costs while meeting business requirements. This section covers pricing for key storage in AWS Key Management Service (AWS KMS), and it provides recommendations to reduce costs, such as through key caching. You can also review KMS key usage to determine if there are additional opportunities to reduce costs.

This section discusses the following cost and billing management topics:

- AWS KMS pricing for key storage
- Amazon S3 bucket keys with default encryption
- Caching data keys by using the AWS Encryption SDK
- Alternatives to key caching and AWS KMS bucket keys
- Reviewing customer managed key usage in Amazon Athena

AWS KMS pricing for key storage

Each AWS KMS key that you create in AWS KMS incurs a charge. The monthly charge is the same for symmetric keys, asymmetric keys, HMAC keys, multi-Region keys (each primary and each replica multi-Region key), keys with imported key material, and KMS keys with a key origin of either AWS CloudHSM or an external key store.

For KMS keys that you rotate automatically or on demand, the first and second rotation of the key adds an additional monthly charge (prorated hourly) in cost. After the second rotation, any subsequent rotations in that month are not billed. Please see AWS KMS pricing for latest pricing information.

You can use AWS Budgets to configure a usage budget. AWS Budgets can alert you when the spend within your account exceeds certain thresholds. For costs related to AWS KMS, you can create a usage budget to alert based on KMS keys or requests. This can improve your visibility into your AWS KMS key storage and use costs.

Amazon S3 bucket keys with default encryption

In some use cases, workloads that access or generate large numbers of objects in Amazon Simple Storage Service (Amazon S3) can generate high volumes of requests to AWS KMS, which increases your costs. Configuring Amazon S3 bucket keys can help you reduce costs by up to 99%. This is a recommended alternative to disabling encryption to help reduce costs associated with AWS KMS.

Caching data keys by using the AWS Encryption SDK

When using the AWS Encryption SDK to perform client-side encryption, data key caching can help improve the performance of your application, reduce the risk that your application's requests to AWS KMS are throttled, and help you reduce costs. For more information about how to get started, see How to use data key caching.

Alternatives to key caching and Amazon S3 bucket keys

If key caching is not an option for you because of your data handling requirements, you can also request AWS KMS quota increases by using the AWS Management Console or the Service Quotas API. Consider the volume of API calls that you might make. The number of API calls that you make is a significant factor in AWS KMS pricing. If you increase the request-rate quota to scale your performance, the increasing number of requests to AWS KMS incurs additional costs.

Managing logging costs for KMS key usage

All AWS KMS API calls are logged to AWS CloudTrail. Applications and services can generate large volumes of AWS KMS API calls (such as for cryptographic operations, including encrypting and decrypting). It can be challenging to review CloudTrail logs without a tool that helps you organize that data, investigate trends, and search for anomalous API activity. Amazon Athena provides predefined data structures that can help you quickly set up tables for CloudTrail logs and start analyzing your log data. It is especially useful for ad-hoc analysis or further investigation during incident response. For more information, see Query AWS CloudTrail logs in the Athena documentation.

Because you pay on a per-query basis for Athena, you can set up your tables in advance at no cost. There are no charges for data definition language statements. When you are responding to an incident, this helps you make sure that many prerequisites are already met. To help you prepare, it is a best practice to write your queries after creating your table, test them, and make sure that they

are producing the results you want. You can save your queries in Athena for future use. For more information about how to get started with Athena, see Getting started with Amazon Athena.

Data events provide visibility into the operations that are performed on or within a resource. These are also known as *data plane operations*. Examples include Amazon S3 `PutObject` events or Lambda function operation API calls. Data events are often high-volume activities, and you incur charges for logging them. To help control the volume of data events that are logged to trails or event data stores in CloudTrail, you can optimize your logging to reduce costs for CloudTrail, AWS KMS, and Amazon S3 by configuring advanced event selectors to limit which data events to log in CloudTrail. For more information, see How to optimize AWS CloudTrail costs by using advanced event selectors (AWS blog post).

Resources

AWS KMS documentation

- AWS KMS Developer Guide
- AWS KMS API Reference
- AWS KMS in the AWS CLI Reference

Tools

- AWS Encryption SDK

AWS Prescriptive Guidance

Strategies

- Creating an encryption strategy for data at rest

Guides

- Encryption best practices and features for AWS services
- AWS Privacy Reference Architecture (AWS PRA)

Patterns

- Automatically encrypt Amazon EBS volumes
- Automatically remediate unencrypted Amazon RDS DB instances and clusters
- Monitor and remediate scheduled deletion of AWS KMS keys

Contributors

Authoring

- Frank Phillis, Senior GTM Specialist Solutions Architect, AWS
- Ken Beer, Director of AWS KMS and Crypto Libraries, AWS
- Michael Miller, Senior Solutions Architect, AWS
- Jeremy Stieglitz, Principal Product Manager, AWS
- Zach Miller, Principal Solutions Architect, AWS
- Peter M. O'Donnell, Principal Solutions Architect, AWS
- Patrick Palmer, Principal Solutions Architect, AWS
- Dave Walker, Principal Solutions Architect, AWS

Reviewing

- Manigandan Shri, Senior Delivery Consultant, AWS

Technical writing

- Lilly AbouHarb, Senior Technical Writer, AWS
- Kimberly Garmoe, Senior Technical Writer, AWS

Document history

The following table describes significant changes to this guide.

Change	Description	Date
Initial publication	—	March 24, 2025