User Guide

AWS Private Certificate Authority

Version latest

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Private Certificate Authority: User Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is AWS Private CA? ... 1
Regional availability ... 1
Integrated services ... 2
Supported algorithms ... 2
RFC 5280 compliance ... 3
Pricing ... 5
Terms and concepts for AWS Private CA ... 5
Trust ... 6
TLS server certificates ... 6
Certificate signature ... 6
Certificate authority ... 6
Root CA ... 7
CA certificate ... 7
Root CA certificate ... 8
End-entity certificate ... 8
Self-signed certificates ... 8
Private certificate ... 9
Certificate path ... 10
Path length constraint ... 10
What is the best certificate service for my needs? ... 11
Best practices ... 12
Documenting CA structure and policies ... 12
Minimize use of the root CA if possible ... 12
Give the root CA its own AWS account ... 13
Separate administrator and issuer roles ... 13
Implement managed revocation of certificates ... 13
Turn on AWS CloudTrail ... 14
Rotate the CA private key ... 14
Delete unused CAs ... 14
Block public access to your CRLs ... 15
Amazon EKS application best practices ... 15
Use AWS Private CA with the AWS SDK for Java ... 16
API examples ... 16
Create and activate a root CA programmatically ... 17

Create and activate a subordinate CA programmatically ... 26
CreateCertificateAuthority ... 35
Using CreateCertificateAuthority to support Active Directory ... 39
CreateCertificateAuthorityAuditReport ... 48
CreatePermission ... 50
DeleteCertificateAuthority ... 53
DeletePermission ... 55
DeletePolicy ... 57
DescribeCertificateAuthority ... 59
DescribeCertificateAuthorityAuditReport ... 61
GetCertificate ... 64
GetCertificateAuthorityCertificate ... 66
GetCertificateAuthorityCsr ... 69
GetPolicy ... 71
ImportCertificateAuthorityCertificate ... 73
IssueCertificate ... 76
ListCertificateAuthorities ... 79
ListPermissions ... 83
ListTags ... 86
PutPolicy ... 88
RestoreCertificateAuthority ... 90
RevokeCertificate ... 92
TagCertificateAuthorities ... 94
UntagCertificateAuthority ... 96
UpdateCertificateAuthority ... 98
Create CAs and certificates with custom subject names ... 100
Create certificates with custom extensions ... 109
Matter examples ... 124
Activate a Product Attestation Authority (PAA) ... 124
Activate an Product Attestation Intermediate (PAI) ... 135
Create a Device Attestation Certificate (DAC) ... 146
Activate a Root CA for Node Operational Certificates (NOC). ... 150
Activate a Subordinate CA for Node Operational Certificates (NOC) ... 160
Create a Node Operational Certificate (NOC) ... 170
mDL examples ... 174
Activate an issuing authority certificate authority (IACA) certificate ... 175

Create a document signer certificate ... 184
Architect your solution for AWS Private CA ... 189
Design a CA hierarchy ... 189
Validate end-entity certificates ... 191
Plan the structure of a CA hierarchy ... 193
Set length constraints on the certification path ... 196
Manage the CA lifecycle ... 198
Choose validity periods ... 198
Manage CA succession ... 200
Revoke a CA ... 201
Plan certificate revocation ... 201
Requirements ... 203
Set up CRL ... 204
Customize OCSP URL ... 211
CA mode ... 213
General-purpose (default) ... 214
Short-lived certificate ... 214
Plan for resilience ... 215
Redundancy and disaster recovery ... 215
Certificate authorities ... 216
Set up ... 217
Sign up for an AWS account ... 217
Create a user with administrative access ... 218
Install the AWS Command Line Interface ... 219
Create a private CA ... 219
CLI examples ... 227
Install CA certificate ... 239
Compatible signing algorithms ... 239
Install a root CA certificate ... 242
Install a subordinate CA certificate hosted by AWS Private CA ... 249
Install a subordinate CA certificate signed by an external parent CA ... 250
Control access ... 251
Create single-account permissions for an IAM user ... 251
Attach a policy for cross-account access ... 254
List private CAs ... 256
View a private CA ... 258

Add tags ... 261
CA status ... 264
Relation between CA status and CA lifecycle ... 266
Update a CA ... 267
Update a CA (console) ... 267
Updating a CA (CLI) ... 271
Delete a CA ... 279
Restore a CA ... 281
Restoring a private CA (console) ... 281
Restore a private CA (AWS CLI) ... 281
Externally signed CA certificates ... 283
Issue and manage certificates ... 287
Issue private end-entity certificates ... 287
Issue a standard certificate (AWS CLI) ... 289
Issue a certificate with a custom subject name using an APIPassthrough template ... 291
Issue a certificate with custom extensions using an APIPassthrough template ... 293
Retrieve a private certificate ... 295
List private certificates ... 296
Export a certificate ... 301
Revoke a private certificate ... 301
Revoked certificates and OCSP ... 303
Revoked certificates in a CRL ... 303
Revoked certificates in an audit report ... 304
Automate export ... 305
Certificate templates ... 305
Template varieties ... 306
Template order of operations ... 317
Template definitions ... 318
Security ... 361
IAM ... 362
API permissions ... 363
AWS managed policies ... 368
Customer managed policies ... 370
Inline policies ... 371
Cross-account access ... 377
Resource-based policies ... 377

Data protection ... 381
Storage and security compliance of AWS Private CA private keys ... 382
Data encryption in AWS Private CA Connector for Active Directory ... 382
Compliance validation ... 383
Create an audit report ... 384
Infrastructure security ... 390
VPC Endpoints (AWS PrivateLink) ... 391
Dual-stack endpoint support ... 395
Using IPv6 addresses in IAM and AWS Private CA ... 395
CP/CPS ... 397
CP/CPS Requirements and Responsibilities ... 398
Monitor resources ... 409
AWS Private CA CloudWatch metrics ... 409
Monitor AWS Private CA with CloudWatch Events ... 410
Success or failure when creating a private CA ... 411
Success or failure when issuing a certificate ... 412
Success when revoking a certificate ... 413
Success or failure when generating a CRL ... 414
Success or failure when creating a CA audit report ... 416
CloudTrail logs ... 417
AWS Private CA information in CloudTrail ... 417
AWS Private CA management events ... 418
Example AWS Private CA events ... 419
Troubleshoot ... 423
Certificate revocation issues ... 423
OCSP response latency ... 423
Revocation of self-signed certificates ... 423
Exception messages ... 423
Matter-compliant certificate errors ... 426
Secure Kubernetes with AWS Private CA ... 430
Concepts ... 430
Considerations ... 432
Cross-account use of cert-manager ... 433
Get started ... 433
Install cert-manager ... 436
Configure IAM permissions ... 437

Install and configure the AWS Private CA cluster issuer ... 439
Manage the AWS Private CA client certificate with cert-manager ... 443
Issue your first TLS certificate ... 444
Examples ... 446
Monitor ... 446
Troubleshoot ... 446
Connector for Active Directory ... 382
Are You a First-Time Connector for AD User? ... 449
Access Connector for AD ... 449
Pricing ... 450
Set up ... 450
Step 1: Create a private CA using AWS Private CA ... 450
Step 2: Set up an Active Directory ... 450
(Active Directory Connector only) Step 3: Delegate permissions to service account ... 451
Step 4: Create IAM Policy ... 452
Step 5: Share your private CA with Connector for AD ... 454
Step 6: Create directory registration ... 455
Step 7: Configure security groups ... 455
Step 8: Configure network access for directory objects ... 455
Get started ... 456
Before you begin ... 457
Step 1: Create a connector ... 457
Step 2: Configure Microsoft Active Directory policies ... 457
Step 3: Create a template ... 459
Step 4: Configure Microsoft group permissions ... 459
Connectors for Active Directory ... 459
Create connector ... 459
Create template ... 462
Update template ... 466
List connectors ... 467
List templates ... 468
View connector ... 469
View template ... 470
Directory registrations ... 473
Template access control entries ... 475
Service principal name ... 476

Tags ... 477
Integrating with EventBridge ... 478
How EventBridge routes Connector for AD events ... 479
Connector for AD events ... 479
Creating event patterns ... 480
Receiving events ... 480
Troubleshoot Connector for Active Directory ... 481
Connector for AD error codes ... 481
Connector creation failure ... 486
SPN creation failure ... 491
Template update issues ... 492
Connector for SCEP ... 493
Features ... 493
How to get started with Connector for SCEP ... 494
Related services ... 494
Access Connector for SCEP ... 494
Pricing ... 495
Concepts ... 495
Considerations and limitations ... 496
Considerations ... 496
Limitations ... 497
Set up ... 498
Step 1: Create an AWS Identity and Access Management policy ... 498
Step 2: Create a private CA ... 500
Step 3: Create a resource share ... 500
Get started ... 501
Before you begin ... 501
Step 1: Create a connector ... 502
Step 2: Copy connector details into your MDM system ... 503
Configure your MDM system ... 504
General-purpose connector ... 504
AWS Private CA Connector for SCEP for Microsoft Intune ... 505
Configure Jamf Pro ... 506
Configure Microsoft Intune ... 513
Configure Omnissa Workspace ONE ... 516
Monitor ... 521

Automate using EventBridge 522
CloudTrail logs 527
Troubleshoot 536
HTTP errors 537
Client errors 555
Service quotas 557
Document History 558
Earlier Updates 567

What is AWS Private CA?

AWS Private CA enables creation of private certificate authority (CA) hierarchies, including root and subordinate CAs, without the investment and maintenance costs of operating an on-premises CA. Your private CAs can issue end-entity X.509 certificates useful in scenarios including:

- Creating encrypted TLS communication channels
- Authenticating users, computers, API endpoints, and IoT devices
- Cryptographically signing code
- Implementing Online Certificate Status Protocol (OCSP) for obtaining certificate revocation status

AWS Private CA operations can be accessed from the AWS Management Console, using the AWS Private CA API, or using the AWS CLI.

Topics

- Regional availability for AWS Private Certificate Authority
- Services integrated with AWS Private Certificate Authority
- Supported cryptographic algorithms in AWS Private Certificate Authority
- RFC 5280 compliance in AWS Private Certificate Authority
- Pricing for AWS Private Certificate Authority
- Terms and concepts for AWS Private CA

Regional availability for AWS Private Certificate Authority

Like most AWS resources, private certificate authorities (CAs) are Regional resources. To use private CAs in more than one Region, you must create your CAs in those Regions. You cannot copy private CAs between Regions. Visit AWS Regions and Endpoints in the *AWS General Reference* or the AWS Region Table to see the Regional availability for AWS Private CA.

> **Note**
>
> ACM is currently available in some regions that AWS Private CA is not.

Services integrated with AWS Private Certificate Authority

If you use AWS Certificate Manager to request a private certificate, you can associate that certificate with any service that is integrated with ACM. This applies both to certificates chained to a AWS Private CA root and to certificates chained to an external root. For more information, see Integrated Services in the AWS Certificate Manager User Guide.

You can also integrate private CAs into Amazon Elastic Kubernetes Service to provide certificate issuance inside a Kubernetes cluster. For more information, see Secure Kubernetes with AWS Private Certificate Authority.

> **Note**
>
> Amazon Elastic Kubernetes Service is not an ACM integrated service.

If you use the AWS Private CA API or AWS CLI to issue a certificate or to export a private certificate from ACM, you can install the certificate anywhere you want.

Supported cryptographic algorithms in AWS Private Certificate Authority

AWS Private CA supports the following cryptographic algorithms for private key generation and certificate signing.

Supported algorithm

Private key algorithms	Signing algorithms
ML_DSA_44	ML_DSA_44
ML_DSA_65	ML_DSA_65
ML_DSA_87	ML_DSA_87
RSA_2048	SHA256WITHRSA
RSA_3072	SHA384WITHRSA
RSA_4096	SHA512WITHRSA

Private key algorithms	Signing algorithms
EC_prime256v1	SHA256WITHECDSA
EC_secp384r1	SHA384WITHECDSA
EC_secp521r1	SHA512WITHECDSA
SM2 (China Regions only)	SM3WITHSM2

This list applies only to certificates issued directly by AWS Private CA through its console, API, or command line. When AWS Certificate Manager issues certificates using a CA from AWS Private CA, it supports some but not all of these algorithms. For more information, see Request a Private Certificate in the AWS Certificate Manager User Guide.

> **Note**
>
> For RSA or ECDSA, the specified signing algorithm family must match the key algorithm family of the CA's private key.
> For ML-DSA, the hash function is defined as part of the algorithm itself. There is no option to select a different hash function with ML-DSA. To maintain backward compatibility with the APIs, the same value is used for key algorithm and signing algorithm.

RFC 5280 compliance in AWS Private Certificate Authority

AWS Private CA does not enforce certain constraints defined in RFC 5280. The reverse situation is also true: Certain additional constraints appropriate to a private CA are enforced.

Enforced

- Not After date. In conformity with RFC 5280, AWS Private CA prevents the issuance of certificates bearing a `Not After` date later than the `Not After` date of the issuing CA's certificate.
- Basic constraints. AWS Private CA enforces basic constraints and path length in imported CA certificates.

  Basic constraints indicate whether or not the resource identified by the certificate is a CA and can issue certificates. CA certificates imported to AWS Private CA must include the basic constraints

extension, and the extension must be marked `critical`. In addition to the `critical` flag, `CA=true` must be set. AWS Private CA enforces basic constraints by failing with a validation exception for the following reasons:

- The extension is not included in the CA certificate.
- The extension is not marked `critical`.

Path length (pathLenConstraint) determines how many subordinate CAs may exist downstream from the imported CA certificate. AWS Private CA enforces path length by failing with a validation exception for the following reasons:

- Importing a CA certificate would violate the path length constraint in the CA certificate or in any CA certificate in the chain.
- Issuing a certificate would violate a path length constraint.

- Name constraints indicate a name space within which all subject names in subsequent certificates in a certification path must be located. Restrictions apply to the subject distinguished name and subject alternative names.

Not enforced

- Certificate policies. Certificate policies regulate the conditions under which a CA issue certificates.
- Inhibit anyPolicy. Used in certificates issued to CAs.
- Issuer Alternative Name. Allows additional identities to be associated with the issuer of the CA certificate.
- Policy Constraints. These constraints limit a CA's capacity to issue subordinate CA certificates.
- Policy Mappings. Used in CA certificates. Lists one or more pairs of OIDs; each pair includes an issuerDomainPolicy and a subjectDomainPolicy.
- Subject Directory Attributes. Used to convey identification attributes of the subject.
- Subject Information Access. How to access information and services for the subject of the certificate in which the extension appears.
- Subject Key Identifier (SKI) and Authority Key Identifier (AKI). The RFC requires a CA certificate to contain the SKI extension. Certificates issued by the CA must contain an AKI extension matching the CA certificate's SKI. AWS does not enforce these requirements. If your CA Certificate does not contain an SKI, the issued end-entity or subordinate CA certificate AKI will be the SHA-1 hash of the issuer public key instead.

- SubjectPublicKeyInfo and Subject Alternative Name (SAN). When issuing a certificate, AWS Private CA copies the SubjectPublicKeyInfo and SAN extensions from the provided CSR without performing validation.

Pricing for AWS Private Certificate Authority

Your account is charged a monthly price for each private CA starting from the time that you create it. You are also charged for each certificate that you issue. This charge includes certificates that you export from ACM and certificates that you create from the AWS Private CA API or AWS Private CA CLI. You are not charged for a private CA after it has been deleted. However, if you restore a private CA, you are charged for the time between deletion and restoration. Private certificates whose private key you cannot access are free. These include certificates that are used with Integrated Services such as Elastic Load Balancing, CloudFront, and API Gateway.

For the latest AWS Private CA pricing information, see AWS Private Certificate Authority Pricing. You can also use the AWS pricing calculator to estimate costs.

Terms and concepts for AWS Private CA

The following terms and concepts can help you as you work with AWS Private Certificate Authority.

Topics

- Trust
- TLS server certificates
- Certificate signature
- Certificate authority
- Root CA
- CA certificate
- Root CA certificate
- End-entity certificate
- Self-signed certificates
- Private certificate
- Certificate path
- Path length constraint

Trust

In order for a web browser to trust the identity of a website, the browser must be able to verify the website's certificate. Browsers, however, trust only a small number of certificates known as CA root certificates. A trusted third party, known as a certificate authority (CA), validates the identity of the website and issues a signed digital certificate to the website's operator. The browser can then check the digital signature to validate the identity of the website. If validation is successful, the browser displays a lock icon in the address bar.

TLS server certificates

HTTPS transactions require server certificates to authenticate a server. A server certificate is an X.509 v3 data structure that binds the public key in the certificate to the subject of the certificate. A TLS certificate is signed by a certificate authority (CA). It contains the name of the server, the validity period, the public key, the signature algorithm, and more.

Certificate signature

A digital signature is an encrypted hash over a certificate. A signature is used to affirm the integrity of the certificate data. Your private CA creates a signature by using a cryptographic hash function such as SHA256 over the variable-sized certificate content. This hash function produces an effectively unforgeable fixed-size data string. This string is called a hash. The CA then encrypts the hash value with its private key and concatenates the encrypted hash with the certificate.

Certificate authority

A certificate authority (CA) issues and if necessary revokes digital certificates. The most common type of certificate is based on the ISO X.509 standard. An X.509 certificate affirms the identity of the certificate subject and binds that identity to a public key. The subject can be a user, an application, a computer, or other device. The CA signs a certificate by hashing the contents and then encrypting the hash with the private key related to the public key in the certificate. A client application such as a web browser that needs to affirm the identity of a subject uses the public key to decrypt the certificate signature. It then hashes the certificate contents and compares the hashed value to the decrypted signature to determine whether they match. For information about certificate signing, see Certificate signature.

You can use AWS Private CA to create a private CA and use the private CA to issue certificates. Your private CA issues only private SSL/TLS certificates for use within your organization. For more

information, see Private certificate. Your private CA also requires a certificate before you can use it. For more information, see CA certificate.

Root CA

A cryptographic building block and root of trust upon which certificates can be issued. It comprises a private key for signing (issuing) certificates and a root certificate that identifies the root CA and binds the private key to the name of the CA. The root certificate is distributed to the trust stores of each entity in an environment. Administrators construct trust stores to include only the CAs that they trust. Administrators update or build the trust stores into the operating systems, instances, and host machine images of entities in their environment. When resources attempt to connect with one another, they check the certificates that each entity presents. A client checks the certificates for validity and whether a chain exists from the certificate to a root certificate installed in the trust store. If those conditions are met, a “handshake" is accomplished between the resources. This handshake cryptographically proves the identity of each entity to the other and creates an encrypted communication channel (TLS/SSL) between them.

CA certificate

A certificate authority (CA) certificate affirms the identity of the CA and binds it to the public key that is contained in the certificate.

You can use AWS Private CA to create a private root CA or a private subordinate CA, each backed by a CA certificate. Subordinate CA certificates are signed by another CA certificate higher in a chain of trust. But in the case of a root CA, the certificate is self-signed. You can also establish an external root authority (hosted on premises, for example). You can then use your root authority to sign a subordinate root CA certificate hosted by AWS Private CA.

The following example shows the typical fields contained in an AWS Private CA X.509 CA certificate. Note that for a CA certificate, the `CA:` value in the `Basic Constraints` field is set to TRUE.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4121 (0x1019)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=Washington, L=Seattle, O=Example Company Root CA, OU=Corp,
 CN=www.example.com/emailAddress=corp@www.example.com
        Validity
```

```
            Not Before: Feb 26 20:27:56 2018 GMT
            Not After : Feb 24 20:27:56 2028 GMT
        Subject: C=US, ST=WA, L=Seattle, O=Examples Company Subordinate CA,
OU=Corporate Office, CN=www.example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c0: ... a3:4a:51
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                F8:84:EE:37:21:F2:5E:0B:6C:40:C2:9D:C6:FE:7E:49:53:67:34:D9
            X509v3 Authority Key Identifier:
                keyid:0D:CE:76:F2:E3:3B:93:2D:36:05:41:41:16:36:C8:82:BC:CB:F8:A0

            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Digital Signature, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         6:bb:94: ... 80:d8
```

Root CA certificate

A certificate authority (CA) typically exists within a hierarchical structure that contains multiple other CAs with clearly defined parent–child relationships between them. Child or subordinate CAs are certified by their parent CAs, creating a certificate chain. The CA at the top of the hierarchy is referred to as the root CA, and its certificate is called the root certificate. This certificate is typically self-signed.

End-entity certificate

An end-entity certificate identifies a resource, such as a server, instance, container or device. Unlike CA certificates, end-entity certificates cannot be used to issue certificates. Other common terms for end-entity certificate are "client" or "leaf" certificate.

Self-signed certificates

A certificate signed by the issuer instead of a higher CA. Unlike certificates issued from a secure root maintained by a CA, self-signed certificates act as their own root, and as a result they have

significant limitations: They can be used to provide on the wire encryption but not to verify identity, and they cannot be revoked. They are unacceptable from a security perspective. But organizations use them nonetheless because they are easy to generate, require no expertise or infrastructure, and many applications accept them. There are no controls in place for issuing self-signed certificates. Organizations that use them incur greater risk of outages caused by certificate expirations because they have no way to track expiration dates.

Private certificate

AWS Private CA certificates are private SSL/TLS certificates that you can use within your organization, but are untrusted on the public internet. Use them to identify resources such as clients, servers, applications, services, devices, and users. When establishing a secure encrypted communications channel, each resource uses a certificate like the following as well as cryptographic techniques to prove its identity to another resource. Internal API endpoints, web servers, VPN users, IoT devices, and many other applications use private certificates to establish encrypted communication channels that are necessary for their secure operation. By default, private certificates are not publicly trusted. An internal administrator must explicitly configure applications to trust private certificates and distribute the certificates.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            e8:cb:d2:be:db:12:23:29:f9:77:06:bc:fe:c9:90:f8
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=WA, L=Seattle, O=Example Company CA, OU=Corporate,
 CN=www.example.com
        Validity
            Not Before: Feb 26 18:39:57 2018 GMT
            Not After : Feb 26 19:39:57 2019 GMT
        Subject: C=US, ST=Washington, L=Seattle, O=Example Company, OU=Sales,
 CN=www.example.com/emailAddress=sales@example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00...c7
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
```

```
            X509v3 Authority Key Identifier:
                keyid:AA:6E:C1:8A:EC:2F:8F:21:BC:BE:80:3D:C5:65:93:79:99:E7:71:65

            X509v3 Subject Key Identifier:
                C6:6B:3C:6F:0A:49:9E:CC:4B:80:B2:8A:AB:81:22:AB:89:A8:DA:19
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 CRL Distribution Points:

                Full Name:
                  URI:http://NA/crl/12345678-1234-1234-1234-123456789012.crl

    Signature Algorithm: sha256WithRSAEncryption
         58:32:...:53
```

Certificate path

A client that relies on a certificate validates that a path exists from the end-entity certificate, possibly through a chain of intermediate certificates, to a trusted root. The client checks that each certificate along the path is valid (not revoked). It also checks that the end-entity certificate has not expired, has integrity (has not been tampered with or modified), and that constraints in the certificate are enforced.

Path length constraint

The basic constraints *pathLenConstraint* for a CA certificate sets the number of subordinate CA certificates that may exist in the chain below it. For example, a CA certificate with a path length constraint of zero cannot have any subordinate CAs. A CA with a path length constraint of one may have up to one level of subordinate CAs underneath it. RFC 5280 defines this as, “the maximum number of non-self-issued intermediate certificates that may follow this certificate in a valid certification path." The path length value excludes the end-entity certificate, though informal language about the "length" or "depth" of a validation chain may include it...leading to confusion.

What is the best certificate service for my needs?

There are two AWS services for issuing and deploying X.509 certificates. Choose the one that best fits your needs. Considerations include whether you need public- or private-facing certificates, customized certificates, certificates you want to deploy into other AWS services, or automated certificate management and renewal.

1. **AWS Private CA**—This service is for enterprise customers building a public key infrastructure (PKI) inside the AWS cloud and intended for private use within an organization. With AWS Private CA, you can create your own CA hierarchy and issue certificates with it for authenticating internal users, computers, applications, services, servers, and other devices, and for signing computer code. Certificates issued by a private CA are trusted only within your organization, not on the internet.

   After creating a private CA, you have the ability to issue certificates directly (that is, without obtaining validation from a third-party CA) and to customize them to meet your organization's internal needs. For example, you may want to:

   - Create certificates with any subject name.
   - Create certificates with any expiration date.
   - Use any supported private key algorithm and key length.
   - Use any supported signing algorithm.
   - Control certificate issuance using templates.

   You are in the right place for this service. To get started, sign into the https://console.aws.amazon.com/acm-pca/ console.
2. **AWS Certificate Manager (ACM)**—This service manages certificates for enterprise customers who need a publicly trusted secure web presence using TLS. You can deploy ACM certificates into AWS Elastic Load Balancing, Amazon CloudFront, Amazon API Gateway, and other integrated services. The most common application of this kind is a secure public website with significant traffic requirements.

   With this service, you can use public certificates provided by ACM (ACM certificates) or certificates that you import into ACM. If you use AWS Private CA to create a CA, ACM can manage certificate issuance from that private CA and automate certificate renewals.

   For more information, see the AWS Certificate Manager User Guide.

AWS Private CA best practices

Best practices are recommendations that can help you use AWS Private CA effectively. The following best practices are based on real-world experience from current AWS Certificate Manager and AWS Private CA customers.

Documenting CA structure and policies

AWS recommends documenting all of your policies and practices for operating your CA. This might include:

- Reasoning for your decisions about CA structure
- A diagram showing your CAs and their relationships
- Policies on CA validity periods
- Planning for CA succession
- Policies on path length
- Catalog of permissions
- Description of administrative control structures
- Security

You can capture this information in two documents, known as Certification Policy (CP) and Certification Practices Statement (CPS). Refer to RFC 3647 for a framework for capturing important information about your CA operations.

Minimize use of the root CA if possible

A root CA should in general only be used to issue certificates for intermediate CAs. This allows the root CA to be stored out of harm's way while the intermediate CAs perform the daily task of issuing end-entity certificates.

However, if your organization's current practice is to issue end-entity certificates directly from a root CA, AWS Private CA can support this workflow while improving security and operational controls. Issuing end-entity certificates in this scenario requires an IAM permissions policy that permits your root CA to use an end-entity certificate template. For information about IAM policies, see Identity and Access Management (IAM) for AWS Private Certificate Authority.

Note

This configuration imposes limitations that might result in operational challenges. For example, if your root CA is compromised or lost, you must create a new root CA and distribute it to all of the clients in your environment. Until this recovery process is complete, you will not be able to issue new certificates. Issuing certificates directly from a root CA also prevents you from restricting access and limiting the number of certificates issued from your root, which are both considered best practices for managing a root CA.

Give the root CA its own AWS account

Creating a root CA and subordinate CA in two different AWS accounts is a recommended best practice. Doing so can provide you with additional protection and access controls for your root CA. You can do so by exporting the CSR from the subordinate CA in one account, and signing it with a root CA in a different account. The benefit of this approach is that you can separate control of your CAs by account. The disadvantage is that you cannot use the AWS Management Console wizard to simplify the process of signing the CA certificate of a subordinate CA from your root CA.

Important

We strongly recommend the use of multi-factor authentication (MFA) any time you access AWS Private CA.

Separate administrator and issuer roles

The CA administrator role should be separate from users who need only to issue end-entity certificates. If your CA administrator and certificate issuer reside in the same AWS account, you can limit issuer permissions by creating an IAM user specifically for that purpose.

Implement managed revocation of certificates

Managed revocation automatically provides notice to certificate clients when a certificate has been revoked. You might need to revoke a certificate if its cryptographic information has been compromised or if it was issued in error. Clients typically refuse to accept revoked certificates. AWS Private CA offers two standard options for managed revocation: Online Certificate Status Protocol

(OCSP), and certificate revocation lists (CRLs). For more information, see Plan your AWS Private CA certificate revocation method.

Turn on AWS CloudTrail

Turn on CloudTrail logging before you create and start operating a private CA. With CloudTrail, you can retrieve a history of AWS API calls for your account to monitor your AWS deployments. This history includes API calls made from the AWS Management Console, the AWS SDKs, the AWS Command Line Interface, and higher-level AWS services. You can also identify which users and accounts called the PCA API operations, the source IP address the calls were made from, and when the calls occurred. You can integrate CloudTrail into applications using the API, automate trail creation for your organization, check the status of your trails, and control how administrators turn CloudTrail logging on and off. For more information, see Creating a Trail. Go to Logging AWS Private Certificate Authority API calls using AWS CloudTrail to see example trails for AWS Private CA operations.

Rotate the CA private key

It is a best practice to periodically update the private key for your private CA. You can update a key by importing a new CA certificate, or you can replace the private CA with a new CA.

> **Note**
>
> If you replace the CA itself, be aware that the ARN of the CA changes. This would cause automation relying on a hard-coded ARN to fail.

Delete unused CAs

You can permanently delete a private CA. You might want to do so if you no longer need the CA or if you want to replace it with a CA that has a newer private key. To safely delete a CA, we recommend that you follow the process outlined in Delete your private CA.

> **Note**
>
> AWS bills you for a CA until it has been deleted.

Block public access to your CRLs

AWS Private CA recommends using the Amazon S3 Block Public Access (BPA) feature on buckets that contain CRLs. This avoids unnecessarily exposing details of your private PKI to potential adversaries. BPA is an S3 best practice and is enabled by default on new buckets. Additional setup is needed in some cases. For more information, see Enable S3 Block Public Access (BPA) with CloudFront.

Amazon EKS application best practices

When using AWS Private CA to provision Amazon EKS with X.509 certificates, follow the recommendations for securing multi-tenant environments in the Amazon EKS Best Practices Guides. For general information about integrating AWS Private CA with Kubernetes, see Secure Kubernetes with AWS Private Certificate Authority.

Use AWS Private CA with the AWS SDK for Java

You can use the AWS Private Certificate Authority API to programmatically interact with the service by sending HTTP requests. The service returns HTTP responses. For more information see AWS Private Certificate Authority API Reference.

In addition to the HTTP API, you can use the AWS SDKs and command line tools to interact with AWS Private CA. This is recommended over the HTTP API. For more information, see Tools for Amazon Web Services. The following topics show you how to use the AWS SDK for Java to program the AWS Private CA API.

The GetCertificateAuthorityCsr, GetCertificate, and DescribeCertificateAuthorityAuditReport operations support waiters. You can use waiters to control the progression of your code based on the presence or state of certain resources. For more information, see the following topics, as well as Waiters in the AWS SDK for Java in the AWS Developer Blog.

AWS Private CA API examples

The following code examples show how to use select AWS Private CA API actions and data types with the AWS SDK for Java.

Topics

- Create and activate a root CA programmatically
- Create and activate a subordinate CA programmatically
- CreateCertificateAuthority
- Using CreateCertificateAuthority to support Active Directory
- CreateCertificateAuthorityAuditReport
- CreatePermission
- DeleteCertificateAuthority
- DeletePermission
- DeletePolicy
- DescribeCertificateAuthority
- DescribeCertificateAuthorityAuditReport
- GetCertificate
- GetCertificateAuthorityCertificate

- GetCertificateAuthorityCsr
- GetPolicy
- ImportCertificateAuthorityCertificate
- IssueCertificate
- ListCertificateAuthorities
- ListPermissions
- ListTags
- PutPolicy
- RestoreCertificateAuthority
- RevokeCertificate
- TagCertificateAuthorities
- UntagCertificateAuthority
- UpdateCertificateAuthority
- Create CAs and certificates with custom subject names
- Create certificates with custom extensions

Create and activate a root CA programmatically

This Java sample shows how to activate a root CA using the following AWS Private CA API actions:

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.samples.GetCertificateAuthorityCertificate;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
```

```
import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
```

```
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

public class RootCAActivation {
    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = "region"; // Substitute your region here, e.g. "us-
west-2"

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setOrganization("Example Organization");
        subject.setOrganizationalUnit("Example");
        subject.setCountry("US");
        subject.setState("Virginia");
        subject.setLocality("Arlington");
        subject.setCommonName("www.example.com");

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
        configCA.withSubject(subject);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.ROOT;

        // ** Execute core code samples for Root CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCAArn = CreateCertificateAuthority(configCA, crlConfigure, CAtype,
 client);
```

```
        String csr = GetCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String rootCertificate = GetCertificate(rootCertificateArn, rootCAArn, client);
        ImportCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType CAtype, AWSACMPCA
client) {
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Create the request object.
```

```
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withRevocationConfiguration(revokeConfig);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String rootCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Root CA Arn: " + rootCAArn);

        return rootCAArn;
    }

    private static String GetCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
```

```
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/RootCACertificate/
V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
```

```
        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(3650L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("Root Certificate Arn: " + rootCertificateArn);

        return rootCertificateArn;
    }

    private static String GetCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(rootCertificateArn);

        // Set the certificate authority ARN.
```

```
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
```

```
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificate(certByteBuffer);

        importRequest.setCertificateChain(null);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(rootCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Root CA certificate successfully imported.");
        System.out.println("Root CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

Create and activate a subordinate CA programmatically

This Java sample shows how to activate a subordinate CA using the following AWS Private CA API actions:

- GetCertificateAuthorityCertificate
- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
```

```
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

public class SubordinateCAActivation {

    public static void main(String[] args) throws Exception {
        // Place your own Root CA ARN here.
        String rootCAArn = "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566";

        // Define the endpoint region for your sample.
        String endpointRegion = "region"; // Substitute your region here, e.g. "us-
west-2"
```

```
        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setOrganization("Example Organization");
        subject.setOrganizationalUnit("Example");
        subject.setCountry("US");
        subject.setState("Virginia");
        subject.setLocality("Arlington");
        subject.setCommonName("www.example.com");

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
        configCA.withSubject(subject);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.SUBORDINATE;

        // ** Execute core code samples for Subordinate CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCertificate = GetCertificateAuthorityCertificate(rootCAArn, client);
        String subordinateCAArn = CreateCertificateAuthority(configCA, crlConfigure,
CAtype, client);
        String csr = GetCertificateAuthorityCsr(subordinateCAArn, client);
        String subordinateCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String subordinateCertificate = GetCertificate(subordinateCertificateArn,
rootCAArn, client);
        ImportCertificateAuthorityCertificate(subordinateCertificate, rootCertificate,
subordinateCAArn, client);

    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
```

```
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String GetCertificateAuthorityCertificate(String rootCAArn,
AWSACMPCA client) {
        // ** GetCertificateAuthorityCertificate **

        // Create a request object and set the certificate authority ARN,
        GetCertificateAuthorityCertificateRequest getCACertificateRequest =
        new GetCertificateAuthorityCertificateRequest();
        getCACertificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create a result object.
        GetCertificateAuthorityCertificateResult getCACertificateResult = null;
        try {
            getCACertificateResult =
client.getCertificateAuthorityCertificate(getCACertificateRequest);
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
```

```
        } catch (InvalidArnException ex) {
            throw ex;
        }

        // Retrieve and display the certificate information.
        String rootCertificate = getCACertificateResult.getCertificate();
        System.out.println("Root CA Certificate / Certificate Chain:");
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType CAtype, AWSACMPCA
client) {
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withRevocationConfiguration(revokeConfig);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String subordinateCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Subordinate CA Arn: " + subordinateCAArn);

        return subordinateCAArn;
    }
```

```
   private static String GetCertificateAuthorityCsr(String subordinateCAArn, AWSACMPCA
client) {

       // Create the CSR request object and set the CA ARN.
       GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
       csrRequest.withCertificateAuthorityArn(subordinateCAArn);

       // Create waiter to wait on successful creation of the CSR file.
       Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
       try {
           getCSRWaiter.run(new WaiterParameters<>(csrRequest));
       } catch (WaiterUnrecoverableException e) {
           //Explicit short circuit when the recourse transitions into
           //an undesired state.
       } catch (WaiterTimedOutException e) {
           //Failed to transition into desired state even after polling.
       } catch(AWSACMPCAException e) {
           //Unexpected service exception.
       }

       // Retrieve the CSR.
       GetCertificateAuthorityCsrResult csrResult = null;
       try {
           csrResult = client.getCertificateAuthorityCsr(csrRequest);
       } catch (RequestInProgressException ex) {
           throw ex;
       } catch (ResourceNotFoundException ex) {
           throw ex;
       } catch (InvalidArnException ex) {
           throw ex;
       } catch (RequestFailedException ex) {
           throw ex;
       }

       // Retrieve and display the CSR;
       String csr = csrResult.getCsr();
       System.out.println("Subordinate CSR:");
       System.out.println(csr);

       return csr;
   }
```

```
    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the issuing CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
SubordinateCACertificate_PathLen0/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(730L); // Approximately two years
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
```

```
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String subordinateCertificateArn = issueResult.getCertificateArn();
        System.out.println("Subordinate Certificate Arn: " +
subordinateCertificateArn);

        return subordinateCertificateArn;
    }

    private static String GetCertificate(String subordinateCertificateArn, String
rootCAArn, AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(subordinateCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
```

```
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String subordinateCertificate = certificateResult.getCertificate();
        System.out.println("Subordinate CA Certificate:");
        System.out.println(subordinateCertificate);

        return subordinateCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String
subordinateCertificate, String rootCertificate, String subordinateCAArn, AWSACMPCA
client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(subordinateCertificate);
        importRequest.setCertificate(certByteBuffer);

        ByteBuffer rootCACertByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificateChain(rootCACertByteBuffer);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
```

```
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
        System.out.println("Subordinate CA certificate successfully imported.");
        System.out.println("Subordinate CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

CreateCertificateAuthority

The following Java sample shows how to use the CreateCerticateAuthority operation.

The operation creates a private subordinate certificate authority (CA). You must specify the CA configuration, the revocation configuration, the CA type, and an optional idempotency token.

The CA configuration specifies the following:

- The name of the algorithm and key size to be used to create the CA private key
- The type of signing algorithm that the CA uses to sign its own Certificate Signing Request, CRLs, and OCSP responses
- X.500 subject information

The CRL configuration specifies the following:

- The CRL expiration period in days (the validity period of the CRL)
- The Amazon S3 bucket that will contain the CRL
- A CNAME alias for the S3 bucket that is included in certificates issued by the CA

If successful, this function returns the Amazon Resource Name (ARN) of the CA.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.util.ArrayList;
import java.util.Objects;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;


public class CreateCertificateAuthority {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
```

```
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
 format.",
                    e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setOrganization("Example Organization");
        subject.setOrganizationalUnit("Example");
        subject.setCountry("US");
        subject.setState("Virginia");
        subject.setLocality("Arlington");
        subject.setCommonName("www.example.com");

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
        configCA.withSubject(subject);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
```

```
        crlConfigure.withS3BucketName("your-bucket-name");

        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Define a certificate authority type: ROOT or SUBORDINATE
        CertificateAuthorityType CAtype = CertificateAuthorityType.<<SUBORDINATE>>;

        // Create a tag - method 1
        Tag tag1 = new Tag();
        tag1.withKey("PrivateCA");
        tag1.withValue("Sample");

        // Create a tag - method 2
        Tag tag2 = new Tag()
            .withKey("Purpose")
            .withValue("WebServices");

        // Add the tags to a collection.
        ArrayList<Tag> tags = new ArrayList<Tag>();
        tags.add(tag1);
        tags.add(tag2);

        // Create the request object.
        CreateCertificateAuthorityRequest req = new
CreateCertificateAuthorityRequest();
        req.withCertificateAuthorityConfiguration(configCA);
        req.withRevocationConfiguration(revokeConfig);
        req.withIdempotencyToken("123987");
        req.withCertificateAuthorityType(CAtype);
        req.withTags(tags);


        // Create the private CA.
        CreateCertificateAuthorityResult result = null;
        try {
            result = client.createCertificateAuthority(req);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }
```

```
        // Retrieve the ARN of the private CA.
        String arn = result.getCertificateAuthorityArn();
        System.out.println(arn);
    }
}
```

Your output should be similar to the following:

```
arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

Using CreateCertificateAuthority to support Active Directory

The following Java sample shows how to use the CreateCerticateAuthority operation to create a CA that can be installed in the Enterprise NTAuth store of Microsoft Active Directory (AD).

The operation creates a private root certificate authority (CA) using custom object identifiers (OIDs). For more information and an AWS CLI example of an equivalent operation, see Create a CA for Active Directory login.

If successful, this function returns the Amazon Resource Name (ARN) of the CA.

```
package com.amazonaws.samples.appstream;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.samples.GetCertificateAuthorityCertificate;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
```

```
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
```

```
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.cert.jcajce.JcaX509ExtensionUtils;
import org.bouncycastle.openssl.PEMParser;
import org.bouncycastle.pkcs.PKCS10CertificationRequest;
import org.bouncycastle.util.io.pem.PemReader;

import lombok.SneakyThrows;

public class RootCAActivation {
    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.3") // OID for Common Name
                .withValue("root CA"),
            new CustomAttribute()
                .withObjectIdentifier("0.9.2342.19200300.100.1.25") // OID for Domain
 Component
                .withValue("example"),
            new CustomAttribute()
                .withObjectIdentifier("0.9.2342.19200300.100.1.25") // OID for Domain
 Component
                .withValue("com")

        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);
```

```
        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.ROOT;

        // ** Execute core code samples for Root CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCAArn = CreateCertificateAuthority(configCA, CAtype, client);
        String csr = GetCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String rootCertificate = GetCertificate(rootCertificateArn, rootCAArn, client);
        ImportCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CertificateAuthorityType CAtype, AWSACMPCA client) {
```

```
        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String rootCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Root CA Arn: " + rootCAArn);

        return rootCAArn;
    }

    private static String GetCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
```

```
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/RootCACertificate/
V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
```

```
        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(3650L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("Root Certificate Arn: " + rootCertificateArn);

        return rootCertificateArn;
    }

    private static String GetCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(rootCertificateArn);

        // Set the certificate authority ARN.
```

```
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
```

```
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificate(certByteBuffer);

        importRequest.setCertificateChain(null);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(rootCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Root CA certificate successfully imported.");
        System.out.println("Root CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

Your output should be similar to the following:

```
arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

CreateCertificateAuthorityAuditReport

The following Java sample shows how to use the CreateCertificateAuthorityAuditReport operation.

The operation creates an audit report that lists every time a certificate is issued or revoked. The report is saved in the Amazon S3 bucket that you specify on input. You can generate a new report once every 30 minutes.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import
 com.amazonaws.services.acmpca.model.CreateCertificateAuthorityAuditReportRequest;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityAuditReportResult;

import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;

public class CreateCertificateAuthorityAuditReport {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
```

```
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object and set the certificate authority ARN.
      CreateCertificateAuthorityAuditReportRequest req =
          new CreateCertificateAuthorityAuditReportRequest();

      // Set the certificate authority ARN.
      req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Specify the S3 bucket name for your report.
      req.setS3BucketName("your-bucket-name");

      // Specify the audit response format.
      req.setAuditReportResponseFormat("JSON");

      // Create a result object.
      CreateCertificateAuthorityAuditReportResult result = null;
      try {
         result = client.createCertificateAuthorityAuditReport(req);
      } catch (RequestInProgressException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
```

```
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidArgsException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      }

      String ID = result.getAuditReportId();
      String S3Key = result.getS3Key();

      System.out.println(ID);
      System.out.println(S3Key);

   }
}
```

Your output should be similar to the following:

```
58904752-7de3-4bdf-ba89-6953e48c3cc7
audit-report/16075838-061c-4f7a-b54b-49bbc111bcff/58904752-7de3-4bdf-
ba89-6953e48c3cc7.json
```

CreatePermission

The following Java sample shows how to use the CreatePermission operation.

The operation assigns access permissions from a private CA to a designated AWS service principal. Services can be given permission to create and retrieve certificates from a private CA, as well as list the active permissions that the private CA has granted. In order to automatically renew certificates through ACM, you must assign all possible permissions (`IssueCertificate`, `GetCertificate`, and `ListPermissions`) from the CA to the ACM service principal (`acm.amazonaws.com`). You can find a CA's ARN by calling the ListCertificateAuthorities function.

Once a permission is created, you can inspect it with the ListPermissions function or delete it with the DeletePermission function.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
```

```
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.CreatePermissionRequest;
import com.amazonaws.services.acmpca.model.CreatePermissionResult;

import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.PermissionAlreadyExistsException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

import java.util.ArrayList;

public class CreatePermission {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
```

```
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create a request object.
        CreatePermissionRequest req =
            new CreatePermissionRequest();

        //  Set the certificate authority ARN.
        req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

        // Set the permissions to give the user.
        ArrayList<String> permissions = new ArrayList<>();
        permissions.add("IssueCertificate");
        permissions.add("GetCertificate");
        permissions.add("ListPermissions");

        req.setActions(permissions);

        // Set the Principal.
        req.setPrincipal("acm.amazonaws.com");

        // Create a result object.
        CreatePermissionResult result = null;
        try {
            result = client.createPermission(req);
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (PermissionAlreadyExistsException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        }
    }
}
```

DeleteCertificateAuthority

The following Java sample shows how to use the DeleteCertificateAuthority operation.

This operation deletes the private certificate authority (CA) that you created using the CreateCertificateAuthority operation. The `DeleteCertificateAuthority` operation requires that you provide an ARN for the CA to be deleted. You can find the ARN by calling the ListCertificateAuthorities operation. You can delete the private CA immediately if its status is `CREATING` or `PENDING_CERTIFICATE`. If you have already imported the certificate, however, you cannot delete it immediately. You must first disable the CA by calling the UpdateCertificateAuthority operation and set the `Status` parameter to `DISABLED`. You can then use the `PermanentDeletionTimeInDays` parameter in the `DeleteCertificateAuthority` operation to specify the number of days, from 7 to 30. During that period the private CA can be restored to `disabled` status. By default, if you do not set the `PermanentDeletionTimeInDays` parameter, the restoration period is 30 days. After this period expires, the private CA is permanently deleted and cannot be restored. For more information, see Restore a CA.

For a Java example that shows you how to use the RestoreCertificateAuthority operation, see RestoreCertificateAuthority.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.DeleteCertificateAuthorityRequest;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.RequestFailedException;

public class DeleteCertificateAuthority {
```

```
   public static void main(String[] args) throws Exception{

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a requrest object and set the ARN of the private CA to delete.
      DeleteCertificateAuthorityRequest req = new DeleteCertificateAuthorityRequest();

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Set the recovery period.
      req.withPermanentDeletionTimeInDays(12);

      // Delete the CA.
      try {
         client.deleteCertificateAuthority(req);
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
```

```
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
    }
}
```

DeletePermission

The following Java sample shows how to use the DeletePermission operation.

The operation deletes permissions that a private CA delegated to an AWS service principal using the CreatePermissions operation. You can find a CA's ARN by calling the ListCertificateAuthorities function. You can inspect the permissions that a CA granted by calling the ListPermissions function.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.DeletePermissionRequest;
import com.amazonaws.services.acmpca.model.DeletePermissionResult;

import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

public class DeletePermission {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
```

```
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object.
      DeletePermissionRequest req =
          new DeletePermissionRequest();

      //  Set the certificate authority ARN.
      req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Set the AWS service principal.
      req.setPrincipal("acm.amazonaws.com");

      // Create a result object.
      DeletePermissionResult result = null;
      try {
         result = client.deletePermission(req);
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      }
```

```
    }
}
```

DeletePolicy

The following Java sample shows how to use the DeletePolicy operation.

The operation delete the resource-based policy attached to a private CA. A resource-based policy is used to enable cross-account CA sharing. You can find the ARN of a private CA by calling the ListCertificateAuthorities action.

Related API actions include PutPolicy and GetPolicy.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.DeletePolicyRequest;
import com.amazonaws.services.acmpca.model.DeletePolicyResult;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.LockoutPreventedException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

public class DeletePolicy {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
```

```
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from file.",
 e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "us-west-2";  // Substitute your Region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create the request object.
        DeletePolicyRequest req = new DeletePolicyRequest();

        // Set the resource ARN.
        req.withResourceArn("arn:aws:acm-pca:us-west-2:111122223333:certificate-
authority/11223344-44ee-aa22-bb33-4cd2d13f1f18");

        // Retrieve a list of your CAs.
        DeletePolicyResult result = null;
        try {
            result = client.deletePolicy(req);
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (LockoutPreventedException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
```

```
        } catch (AWSACMPCAException ex) {
            throw ex;
        }
    }
}
```

DescribeCertificateAuthority

The following Java sample shows how to use the DescribeCertificateAuthority operation.

The operation lists information about your private certificate authority (CA). You must specify the ARN (Amazon Resource Name) of the private CA. The output contains the status of your CA. This can be any of the following:

- `CREATING` – AWS Private CA is creating your private certificate authority.
- `PENDING_CERTIFICATE` – The certificate is pending. You must use your on-premises root or subordinate CA to sign your private CA CSR and then import it into PCA.
- `ACTIVE` – Your private CA is active.
- `DISABLED` – Your private CA has been disabled.
- `EXPIRED` – Your private CA certificate has expired.
- `FAILED` – Your private CA cannot be created.
- `DELETED` – Your private CA is within the restoration period, after which it will be permanently deleted.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.CertificateAuthority;
import com.amazonaws.services.acmpca.model.DescribeCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.DescribeCertificateAuthorityResult;
```

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;

public class DescribeCertificateAuthority {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object
      DescribeCertificateAuthorityRequest req = new
 DescribeCertificateAuthorityRequest();

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Create a result object.
      DescribeCertificateAuthorityResult result = null;
      try {
         result = client.describeCertificateAuthority(req);
```

```
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      }

      // Retrieve and display information about the CA.
      CertificateAuthority PCA = result.getCertificateAuthority();
      String strPCA = PCA.toString();
      System.out.println(strPCA);
   }
}
```

DescribeCertificateAuthorityAuditReport

The following Java sample shows how to use the DescribeCertificateAuthorityAuditReport operation.

The operation lists information about a specific audit report that you created by calling the CreateCertificateAuthorityAuditReport operation. Audit information is created every time the certificate authority (CA) private key is used. The private key is used when you issue a certificate, sign a CRL, or revoke a certificate.

```
package com.amazonaws.samples;

import java.util.Date;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import
 com.amazonaws.services.acmpca.model.DescribeCertificateAuthorityAuditReportRequest;
import
 com.amazonaws.services.acmpca.model.DescribeCertificateAuthorityAuditReportResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
```

```
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

public class DescribeCertificateAuthorityAuditReport {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object.
      DescribeCertificateAuthorityAuditReportRequest req =
            new DescribeCertificateAuthorityAuditReportRequest();

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");
```

```
      // Set the audit report ID.
      req.withAuditReportId("11111111-2222-3333-4444-555555555555");

      // Create waiter to wait on successful creation of the audit report file.
      Waiter<DescribeCertificateAuthorityAuditReportRequest> waiter =
 client.waiters().auditReportCreated();
      try {
         waiter.run(new WaiterParameters<>(req));
      } catch (WaiterUnrecoverableException e) {
          //Explicit short circuit when the recourse transitions into
          //an undesired state.
      } catch (WaiterTimedOutException e) {
          //Failed to transition into desired state even after polling.
      } catch (AWSACMPCAException e) {
          //Unexpected service exception.
      }

      // Create a result object.
      DescribeCertificateAuthorityAuditReportResult result = null;
      try {
         result = client.describeCertificateAuthorityAuditReport(req);
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidArgsException ex) {
         throw ex;
      }

      String status = result.getAuditReportStatus();
      String S3Bucket = result.getS3BucketName();
      String S3Key = result.getS3Key();
      Date createdAt = result.getCreatedAt();

      System.out.println(status);
      System.out.println(S3Bucket);
      System.out.println(S3Key);
      System.out.println(createdAt);
   }
}
```

Your output should be similar to the following:

```
SUCCESS
your-audit-report-bucket-name
```

```
audit-report/a4119411-8153-498a-a607-2cb77b858043/25211c3d-f2fe-479f-b437-
fe2b3612bc45.json
Tue Jan 16 13:07:58 PST 2018
```

GetCertificate

The following Java sample shows how to use the GetCertificate operation.

The operation retrieves a certificate from your private CA. The ARN of the certificate is returned when you call the IssueCertificate operation. You must specify both the ARN of your private CA and the ARN of the issued certificate when calling the `GetCertificate` operation. You can retrieve the certificate if it is in the `ISSUED` state. You can call the CreateCertificateAuthorityAuditReport operation to create a report that contains information about all of the certificates issued and revoked by your private CA.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.RequestFailedException ;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import com.amazonaws.services.acmpca.model.AWSACMPCAException;
```

```
public class GetCertificate {

   public static void main(String[] args) throws Exception{

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object.
      GetCertificateRequest req = new GetCertificateRequest();

      // Set the certificate ARN.
      req.withCertificateArn("arn:aws:acm-pca:region:account:certificate-
authority/CA_ID/certificate/certificate_ID");

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Create waiter to wait on successful creation of the certificate file.
      Waiter<GetCertificateRequest> waiter = client.waiters().certificateIssued();
      try {
         waiter.run(new WaiterParameters<>(req));
```

```
      } catch (WaiterUnrecoverableException e) {
          //Explicit short circuit when the recourse transitions into
          //an undesired state.
      } catch (WaiterTimedOutException e) {
          //Failed to transition into desired state even after polling.
      } catch (AWSACMPCAException e) {
          //Unexpected service exception.
      }

      // Retrieve the certificate and certificate chain.
      GetCertificateResult result = null;
      try {
         result = client.getCertificate(req);
      } catch (RequestInProgressException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      }

      // Get the certificate and certificate chain and display the result.
      String strCert = result.getCertificate();
      System.out.println(strCert);
   }
}
```

Your output should be a certificate chain similar to the following for the certificate authority (CA) and certificate that you specified.

```
-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----

-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----

-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----
```

GetCertificateAuthorityCertificate

The following Java sample shows how to use the GetCertificateAuthorityCertificate operation.

This operation retrieves the certificate and certificate chain for your private certificate authority (CA). Both the certificate and the chain are base64-encoded strings in PEM format. The chain does not include the CA certificate. Each certificate in the chain signs the one before it.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;

public class GetCertificateAuthorityCertificate {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
```

```
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

      // Create a request object
      GetCertificateAuthorityCertificateRequest req =
            new GetCertificateAuthorityCertificateRequest();

      // Set the certificate authority ARN,
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Create a result object.
      GetCertificateAuthorityCertificateResult result = null;
      try {
         result = client.getCertificateAuthorityCertificate(req);
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      }

      // Retrieve and display the certificate information.
      String strPcaCert = result.getCertificate();
      System.out.println(strPcaCert);
      String strPCACChain = result.getCertificateChain();
      System.out.println(strPCACChain);
   }
}
```

Your output should be a certificate and chain similar to the following for the certificate authority (CA) that you specified.

```
-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----

-----BEGIN CERTIFICATE----- base64-encoded certificate -----END CERTIFICATE-----
```

GetCertificateAuthorityCsr

The following Java sample shows how to use the GetCertificateAuthorityCsr operation.

This operation retrieves the certificate signing request (CSR) for your private certificate authority (CA). The CSR is created when you call the CreateCertificateAuthority operation. Take the CSR to your on-premises X.509 infrastructure and sign it using your root or a subordinate CA. Then import the signed certificate back into ACM PCA by calling the ImportCertificateAuthorityCertificate operation. The CSR is returned as a base64-encoded string in PEM format.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

public class GetCertificateAuthorityCsr {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
```

```
      try {
         credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
         throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create the request object and set the CA ARN.
      GetCertificateAuthorityCsrRequest req = new GetCertificateAuthorityCsrRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Create waiter to wait on successful creation of the CSR file.
      Waiter<GetCertificateAuthorityCsrRequest> waiter =
 client.waiters().certificateAuthorityCSRCreated();
      try {
         waiter.run(new WaiterParameters<>(req));
      } catch (WaiterUnrecoverableException e) {
          //Explicit short circuit when the recourse transitions into
          //an undesired state.
      } catch (WaiterTimedOutException e) {
          //Failed to transition into desired state even after polling.
      } catch (AWSACMPCAException e) {
          //Unexpected service exception.
      }

      // Retrieve the CSR.
      GetCertificateAuthorityCsrResult result = null;
      try {
         result = client.getCertificateAuthorityCsr(req);
```

```
        } catch (RequestInProgressException ex) {
           throw ex;
        } catch (ResourceNotFoundException ex) {
           throw ex;
        } catch (InvalidArnException ex) {
           throw ex;
        } catch (RequestFailedException ex) {
           throw ex;
        }

        // Retrieve and display the CSR;
        String Csr = result.getCsr();
        System.out.println(Csr);
    }
}
```

Your output should be similar to the following for the certificate authority (CA) that you specify. The certificate signing request (CSR) is base64-encoded in PEM format. Save it to a local file, take it to your on-premises X.509 infrastructure, and sign it by using your root or a subordinate CA.

```
-----BEGIN CERTIFICATE REQUEST----- base64-encoded request -----END CERTIFICATE
 REQUEST-----
```

GetPolicy

The following Java sample shows how to use the GetPolicy operation.

The operation retrieves the resource-based policy attached to a private CA. A resource-based policy is used to enable cross-account CA sharing. You can find the ARN of a private CA by calling the ListCertificateAuthorities action.

Related API actions include PutPolicy and DeletePolicy.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
```

```
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.GetPolicyRequest;
import com.amazonaws.services.acmpca.model.GetPolicyResult;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

public class GetPolicy {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from file.",
 e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create the request object.
        GetPolicyRequest req = new GetPolicyRequest();

        // Set the resource ARN.
```

```
        req.withResourceArn("arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566");

        // Retrieve a list of your CAs.
        GetPolicyResult result= null;
        try {
            result = client.getPolicy(req);
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (AWSACMPCAException ex) {
            throw ex;
        }

        // Display the policy.
        System.out.println(result.getPolicy());
    }
}
```

ImportCertificateAuthorityCertificate

The following Java sample shows how to use the ImportCertificateAuthorityCertificate operation.

This operation imports your signed private CA certificate into AWS Private CA. Before you can call this operation, you must create the private certificate authority by calling the CreateCertificateAuthority operation. You must then generate a certificate signing request (CSR) by calling the GetCertificateAuthorityCsr operation. Take the CSR to your on-premises CA and use your root certificate or a subordinate certificate to sign it. Create a certificate chain and copy the signed certificate and the certificate chain to your working directory.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
```

```
import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.RequestFailedException;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Objects;

public class ImportCertificateAuthorityCertificate {

   public static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
      }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
```

```
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create the request object and set the signed certificate, chain and CA ARN.
      ImportCertificateAuthorityCertificateRequest req =
            new ImportCertificateAuthorityCertificateRequest();

      // Set the signed certificate.
      String strCertificate =
            "-----BEGIN CERTIFICATE-----\n" +
            "base64-encoded certificate\n" +
            "-----END CERTIFICATE-----\n";
      ByteBuffer certByteBuffer = stringToByteBuffer(strCertificate);
      req.setCertificate(certByteBuffer);

      // Set the certificate chain.
      String strCertificateChain =
            "-----BEGIN CERTIFICATE-----\n" +
            "base64-encoded certificate\n" +
            "-----END CERTIFICATE-----\n";
      ByteBuffer chainByteBuffer = stringToByteBuffer(strCertificateChain);
      req.setCertificateChain(chainByteBuffer);

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Import the certificate.
      try {
         client.importCertificateAuthorityCertificate(req);
      } catch (CertificateMismatchException ex) {
         throw ex;
      } catch (MalformedCertificateException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
```

```
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (RequestInProgressException ex) {
         throw ex;
      } catch (ConcurrentModificationException ex) {
         throw ex;
      } catch (RequestFailedException ex) {
         throw ex;
      }
   }
}
```

IssueCertificate

The following Java sample shows how to use the IssueCertificate operation.

This operation uses your private certificate authority (CA) to issue an end-entity certificate. This operation returns the Amazon Resource Name (ARN) of the certificate. You can retrieve the certificate by calling the GetCertificate and specifying the ARN.

> **Note**
>
> The IssueCertificate operation requires you to specify a certificate template. This example uses the `EndEntityCertificate/V1` template. For information about all of the available templates, see Use AWS Private CA certificate templates.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;
```

```
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

public class IssueCertificate {
   public static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
      }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
```

```
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();

      // Set the CA ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
      String strCSR =
      "-----BEGIN CERTIFICATE REQUEST-----\n" +
      "base64-encoded certificate\n" +
      "-----END CERTIFICATE REQUEST-----\n";
      ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
      req.setCsr(csrByteBuffer);

      // Specify the template for the issued certificate.
      req.withTemplateArn("arn:aws:acm-pca:::template/EndEntityCertificate/V1");

      // Set the signing algorithm.
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(<<3650L>>);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");

      // Issue the certificate.
      IssueCertificateResult result = null;
      try {
         result = client.issueCertificate(req);
      } catch (LimitExceededException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
```

```
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidArgsException ex) {
         throw ex;
      } catch (MalformedCSRException ex) {
         throw ex;
      }

      // Retrieve and display the certificate ARN.
      String arn = result.getCertificateArn();
      System.out.println(arn);
   }
}
```

Your output should be similar to the following:

```
arn:aws:acm-pca:region:account:certificate-authority/CA_ID/certificate/certificate_ID
```

ListCertificateAuthorities

The following Java sample shows how to use the ListCertificateAuthorities operation.

This operation lists the private certificate authorities (CAs) that you created using the CreateCertificateAuthority operation.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ListCertificateAuthoritiesRequest;
import com.amazonaws.services.acmpca.model.ListCertificateAuthoritiesResult;
import com.amazonaws.services.acmpca.model.InvalidNextTokenException;

public class ListCertificateAuthorities {
```

```
   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from file.",
 e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create the request object.
      ListCertificateAuthoritiesRequest req = new ListCertificateAuthoritiesRequest();
      req.withMaxResults(10);

      // Retrieve a list of your CAs.
      ListCertificateAuthoritiesResult result= null;
      try {
          result = client.listCertificateAuthorities(req);
      } catch (InvalidNextTokenException ex) {
          throw ex;
      }

      // Display the CA list.
      System.out.println(result.getCertificateAuthorities());
   }
}
```

If you have any certificate authorities to list, your output should be similar to the following:

```
[{
 Arn: arn: aws: acm-pca: region: account: certificate-
authority/12345678-1234-1234-1234-123456789012,
 CreatedAt: TueNov0712: 05: 39PST2017,
 LastStateChangeAt: WedJan1012: 35: 39PST2018,
 Type: SUBORDINATE,
 Serial: 4109,
 Status: DISABLED,
 NotBefore: TueNov0712: 19: 15PST2017,
 NotAfter: FriNov0513: 19: 15PDT2027,
 CertificateAuthorityConfiguration: {
  KeyType: RSA2048,
  SigningAlgorithm: SHA256WITHRSA,
  Subject: {
   Organization: ExampleCorp,
   OrganizationalUnit: HR,
   State: Washington,
   CommonName: www.example.com,
   Locality: Seattle,

  }
 },
 RevocationConfiguration: {
  CrlConfiguration: {
   Enabled: true,
   ExpirationInDays: 3650,
   CustomCname: your-custom-name,
   S3BucketName: your-bucket-name
  }
 }
},
{
 Arn: arn: aws: acm-pca: region: account>: certificate-
authority/12345678-1234-1234-1234-123456789012,
 CreatedAt: WedSep1312: 54: 52PDT2017,
 LastStateChangeAt: WedSep1312: 54: 52PDT2017,
 Type: SUBORDINATE,
 Serial: 4100,
 Status: ACTIVE,
 NotBefore: WedSep1314: 11: 19PDT2017,
 NotAfter: SatSep1114: 11: 19PDT2027,
 CertificateAuthorityConfiguration: {
```

```
  KeyType: RSA2048,
  SigningAlgorithm: SHA256WITHRSA,
  Subject: {
   Country: US,
   Organization: ExampleCompany,
   OrganizationalUnit: Sales,
   State: Washington,
   CommonName: www.example.com,
   Locality: Seattle,

  }
 },
 RevocationConfiguration: {
  CrlConfiguration: {
   Enabled: false,
   ExpirationInDays: 5,
   CustomCname: your-custom-name,
   S3BucketName: your-bucket-name
  }
 }
},
{
 Arn: arn: aws: acm-pca: region: account>: certificate-
authority/12345678-1234-1234-1234-123456789012,
 CreatedAt: FriJan1213: 57: 11PST2018,
 LastStateChangeAt: FriJan1213: 57: 11PST2018,
 Type: SUBORDINATE,
 Status: PENDING_CERTIFICATE,
 CertificateAuthorityConfiguration: {
  KeyType: RSA2048,
  SigningAlgorithm: SHA256WITHRSA,
  Subject: {
   Country: US,
   Organization: Examples-R-Us Ltd.,
   OrganizationalUnit: corporate,
   State: WA,
   CommonName: www.examplesrus.com,
   Locality: Seattle,

  }
 },
 RevocationConfiguration: {
  CrlConfiguration: {
   Enabled: true,
```

```
   ExpirationInDays: 365,
   CustomCname: your-custom-name,
   S3BucketName: your-bucket-name
  }
 }
},
{
 Arn: arn: aws: acm-pca: region: account>: certificate-
authority/12345678-1234-1234-1234-123456789012,
 CreatedAt: FriJan0511: 14: 21PST2018,
 LastStateChangeAt: FriJan0511: 14: 21PST2018,
 Type: SUBORDINATE,
 Serial: 4116,
 Status: ACTIVE,
 NotBefore: FriJan0512: 12: 56PST2018,
 NotAfter: MonJan0312: 12: 56PST2028,
 CertificateAuthorityConfiguration: {
  KeyType: RSA2048,
  SigningAlgorithm: SHA256WITHRSA,
  Subject: {
   Country: US,
   Organization: ExamplesLLC,
   OrganizationalUnit: CorporateOffice,
   State: WA,
   CommonName: www.example.com,
   Locality: Seattle,

  }
 },
 RevocationConfiguration: {
  CrlConfiguration: {
   Enabled: true,
   ExpirationInDays: 3650,
   CustomCname: your-custom-name,
   S3BucketName: your-bucket-name
  }
 }
}]
```

ListPermissions

The following Java sample shows how to use the ListPermissions operation.

This operation lists the permissions, if any, that your private CA has assigned. Permissions, including `IssueCertificate`, `GetCertificate`, and `ListPermissions`, can be assigned to an AWS service principal with the CreatePermission operation, and revoked with the DeletePermissions operation.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ListPermissionsRequest;
import com.amazonaws.services.acmpca.model.ListPermissionsResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidNextTokenException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RequestFailedException;

public class ListPermissions {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
```

```
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
         .withEndpointConfiguration(endpoint)
         .withCredentials(new AWSStaticCredentialsProvider(credentials))
         .build();

      // Create a request object and set the CA ARN.
      ListPermissionsRequest req = new ListPermissionsRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // List the tags.
      ListPermissionsResult result = null;
      try {
         result = client.listPermissions(req);
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch(RequestFailedException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      }

      // Retrieve and display the permissions.
      System.out.println(result);
   }
}
```

If the designated private CA has assigned permissions to a service principal, your output should be similar to the following:

```
[{
       Arn: arn:aws:acm-
pca:region:account:permission/12345678-1234-1234-1234-123456789012,
       CreatedAt: WedFeb0317: 05: 39PST2019,
       Prinicpal: acm.amazonaws.com,
```

```
       Permissions: {
            ISSUE_CERTIFICATE,
            GET_CERTIFICATE,
            DELETE,CERTIFICATE
       },
       SourceAccount: account
}]
```

ListTags

The following Java sample shows how to use the ListTags operation.

This operation lists the tags, if any, that are associated with your private CA. Tags are labels that you can use to identify and organize your CAs. Each tag consists of a key and an optional value. Call the TagCertificateAuthority operation to add one or more tags to your CA. Call the UntagCertificateAuthority operation to remove tags.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ListTagsRequest;
import com.amazonaws.services.acmpca.model.ListTagsResult;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;

public class ListTags {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
```

```
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a request object and set the CA ARN.
      ListTagsRequest req = new ListTagsRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // List the tags
      ListTagsResult result = null;
      try {
          result = client.listTags(req);
      } catch (InvalidArnException ex) {
          throw ex;
      } catch (ResourceNotFoundException ex) {
          throw ex;
      }

      // Retrieve and display the tags.
      System.out.println(result);
   }
}
```

If you have any tags to list, your output should be similar to the following:

```
{Tags: [{Key: Admin,Value: Alice}, {Key: Purpose,Value: WebServices}],}
```

PutPolicy

The following Java sample shows how to use the PutPolicy operation.

The operation attaches a resource-based policy to a private CA, enabling cross-account sharing. When authorized by a policy, a principal residing in another AWS account can issue and renew private end-entity certificates using a private CA that it does not own. You can find the ARN of a private CA by calling the ListCertificateAuthorities action. For examples of policies, see the AWS Private CA guidance on Resource-Based Policies.

Once a policy is attached to a CA, you can inspect it with the GetPolicy action or delete it with the DeletePolicy action.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.PutPolicyRequest;
import com.amazonaws.services.acmpca.model.PutPolicyResult;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.LockoutPreventedException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class PutPolicy {

    public static void main(String[] args) throws Exception {
```

```
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from file.",
 e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Create the request object.
        PutPolicyRequest req = new PutPolicyRequest();

        // Set the resource ARN.
        req.withResourceArn("arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566");

        // Import and set the policy.
        // Note: This code assumes the file "ShareResourceWithAccountPolicy.json" is in
 a folder titled policy.
        String policy = new String(Files.readAllBytes(Paths.get("policy",
 "ShareResourceWithAccountPolicy.json")));
        req.withPolicy(policy);

        // Retrieve a list of your CAs.
        PutPolicyResult result = null;
        try {
            result = client.putPolicy(req);
```

```
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LockoutPreventedException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (AWSACMPCAException ex) {
            throw ex;
        }
    }
}
```

RestoreCertificateAuthority

The following Java sample shows how to use the RestoreCertificateAuthority operation. A private CA can be restored at any time during its restoration period. Currently, this period can last 7 to 30 days from the date of deletion and can be defined when you delete the CA. For more information, see Restore a CA. See also the DeleteCertificateAuthority Java example.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.RestoreCertificateAuthorityRequest;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
```

```
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;

public class RestoreCertificateAuthority {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from file.",
 e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create the request object.
      RestoreCertificateAuthorityRequest req = new
 RestoreCertificateAuthorityRequest();

      // Set the certificate authority ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Restore the CA.
      try {
          client.restoreCertificateAuthority(req);
      } catch (InvalidArnException ex) {
          throw ex;
      } catch (InvalidStateException ex) {
```

```
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        }
    }
}
```

RevokeCertificate

The following Java sample shows how to use the RevokeCertificate operation.

This operation revokes a certificate that you issued by calling the IssueCertificate operation. If you enabled a certificate revocation list (CRL) when you created or updated your private CA, information about the revoked certificates is included in the CRL. AWS Private CA writes the CRL to an Amazon S3 bucket that you specify. For more information, see the CrlConfiguration structure.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.AmazonClientException;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.RevokeCertificateRequest;
import com.amazonaws.services.acmpca.model.RevocationReason;

import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestAlreadyProcessedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;

public class RevokeCertificate {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
```

```
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a request object.
      RevokeCertificateRequest req = new RevokeCertificateRequest();

      // Set the certificate authority ARN.
      req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");

      // Set the certificate serial number.
      req.setCertificateSerial("79:3f:0d:5b:6a:04:12:5e:2c:9c:fb:52:37:35:98:fe");

      // Set the RevocationReason.
      req.withRevocationReason(RevocationReason.<<KEY_COMPROMISE>>);

      // Revoke the certificate.
      try {
          client.revokeCertificate(req);
      } catch (InvalidArnException ex) {
          throw ex;
      } catch (InvalidStateException ex) {
          throw ex;
      } catch (ResourceNotFoundException ex) {
          throw ex;
```

```
        } catch (RequestAlreadyProcessedException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
    }
}
```

TagCertificateAuthorities

The following Java sample shows how to use the TagCertificateAuthority operation.

This operation adds one or more tags to your private CA. Tags are labels that you can use to identify and organize your AWS resources. Each tag consists of a key and an optional value. When you call this operation, you specify the private CA by its Amazon Resource Name (ARN). You specify the tag by using a key-value pair. To identify a specific characteristic of that CA, you can apply a tag to just one private CA. Or, to filter for a common relationship among those CAs, you can apply the same tag to multiple private CAs. To remove one or more tags, use the UntagCertificateAuthority operation. Call the ListTags operation to see what tags are associated with your CA.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.TagCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.Tag;

import java.util.ArrayList;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidTagException;
```

```
import com.amazonaws.services.acmpca.model.TooManyTagsException;

public class TagCertificateAuthorities {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a tag - method 1
      Tag tag1 = new Tag();
      tag1.withKey("Administrator");
      tag1.withValue("Bob");

      // Create a tag - method 2
      Tag tag2 = new Tag()
          .withKey("Purpose")
          .withValue("WebServices");

      // Add the tags to a collection.
      ArrayList<Tag> tags = new ArrayList<Tag>();
      tags.add(tag1);
      tags.add(tag2);
```

```
      // Create a request object and specify the certificate authority ARN.
      TagCertificateAuthorityRequest req = new TagCertificateAuthorityRequest();
      req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");
      req.setTags(tags);

      // Add a tag
      try {
          client.tagCertificateAuthority(req);
      } catch (InvalidArnException ex) {
          throw ex;
      } catch (ResourceNotFoundException ex) {
          throw ex;
      } catch (InvalidTagException ex) {
          throw ex;
      } catch (TooManyTagsException ex) {
          throw ex;
      }
   }
}
```

UntagCertificateAuthority

The following Java sample shows how to use the UntagCertificateAuthority operation.

This operation removes one or more tags from your private CA. A tag consists of a key-value pair. If you do not specify the value portion of the tag when calling this operation, the tag is removed regardless of value. If you specify a value, the tag is removed only if it is associated with the specified value. To add tags to a private CA, use the TagCertificateAuthority operation. Call the ListTags operation to see what tags are associated with your CA.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.util.ArrayList;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;
```

```
import com.amazonaws.services.acmpca.model.UntagCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.Tag;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidTagException;

public class UntagCertificateAuthority {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a Tag object with the tag to delete.
      Tag tag = new Tag();
      tag.withKey("Administrator");
      tag.withValue("Bob");

      // Add the tags to a collection.
      ArrayList<Tag> tags = new ArrayList<Tag>();
      tags.add(tag);
```

```
      // Create a request object and specify the certificate authority ARN.
      UntagCertificateAuthorityRequest req = new UntagCertificateAuthorityRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");
      req.withTags(tags);

      // Delete the tag
      try {
          client.untagCertificateAuthority(req);
      } catch (InvalidArnException ex) {
          throw ex;
      } catch (ResourceNotFoundException ex) {
          throw ex;
      } catch (InvalidTagException ex) {
          throw ex;
      }
   }
}
```

UpdateCertificateAuthority

The following Java sample shows how to use the UpdateCertificateAuthority operation.

The operation updates the status or configuration of a private certificate authority (CA). Your private CA must be in the ACTIVE or DISABLED state before you can update it. You can disable a private CA that is in the ACTIVE state or make a CA that is in the DISABLED state active again.

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.UpdateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CertificateAuthorityStatus;

import com.amazonaws.AmazonClientException;
```

```
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;

public class UpdateCertificateAuthority {

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from file.",
 e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create the request object.
      UpdateCertificateAuthorityRequest req = new UpdateCertificateAuthorityRequest();

      // Set the ARN of the private CA that you want to update.
      req.setCertificateAuthorityArn("arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566");
```

```
        // Define the certificate revocation list configuration. If you do not want to
        // update the CRL configuration, leave the CrlConfiguration structure alone and
        // do not set it on your UpdateCertificateAuthorityRequest object.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname("your-custom-name");
        crlConfigure.withS3BucketName("your-bucket-name");

        // Set the CRL configuration onto your UpdateCertificateAuthorityRequest object.
        // If you do not want to change your CRL configuration, do not use the
        // setCrlConfiguration method.
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);
        req.setRevocationConfiguration(revokeConfig);

        // Set the status.
        req.withStatus(CertificateAuthorityStatus.<<ACTIVE>>);

        // Create the result object.
        try {
            client.updateCertificateAuthority(req);
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        }
    }
}
```

Create CAs and certificates with custom subject names

The CustomAttribute object allows administrators to pass custom object identifiers (OIDs) to private CAs and certificates. Custom OIDs can be used to create specialized subject-name hierarchies that reflect the structure and needs of your organization. Customized certificates must

be created using one of the `ApiPassthrough` templates. For more information about templates, see AWS Private CA template varieties. For more information about using custom attrributes, see Issue private end-entity certificates and Create a private CA in AWS Private CA.

You cannot use `StandardAttributes` in conjunction with `CustomAttributes`. However, you can pass standard OIDs as part of a `CustomAttributes`. The default subject name OIDs are listed in the following table:

Subject name	Object ID
Country	2.5.4.6
CommonName	2.5.4.3
DistinguishedNameQualifier	2.5.4.46
GenerationQualifier	2.5.4.44
GivenName	2.5.4.42
Initials	2.5.4.43
Locality	2.5.4.7
Organization	2.5.4.10
OrganizationalUnit	2.5.4.11
Pseudonym	2.5.4.65
SerialNumber	2.5.4.5
State	2.5.4.8
Surname	2.5.4.4
Title	2.5.4.12

Topics

- Create CA with CustomAttribute

- Issue a certificate with CustomAttribute

Create CA with CustomAttribute

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;


public class CreateCertificateAuthorityWithCustomAttributes {

    public static void main(String[] args) throws Exception {

        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
```

```
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                   "Cannot load the credentials from the credential profiles file. " +
                   "Please make sure that your credentials file is at the correct " +
                   "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
 format.",
                   e);
        }

        // Define the endpoint for your sample.
        String endpointRegion = "us-west-2";  // Substitute your region here, e.g. "us-
west-2"
        String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
 endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.6") // Country
                .withValue("US"),
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.3") // CommonName
                .withValue("CommonName"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1") // CustomOID
                .withValue("ABCDEFG"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1") // CustomOID
                .withValue("BCDEFGH")
        );
```

```
        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
        configCA.withSubject(subject);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");

        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Define a certificate authority type: ROOT or SUBORDINATE
        CertificateAuthorityType caType = CertificateAuthorityType.SUBORDINATE;

        // Create a tag - method 1
        Tag tag1 = new Tag();
        tag1.withKey("PrivateCA");
        tag1.withValue("Sample");

        // Create a tag - method 2
        Tag tag2 = new Tag()
            .withKey("Purpose")
            .withValue("WebServices");

        // Add the tags to a collection.
        ArrayList<Tag> tags = new ArrayList<Tag>();
        tags.add(tag1);
        tags.add(tag2);

        // Create the request object.
        CreateCertificateAuthorityRequest req = new
CreateCertificateAuthorityRequest();
        req.withCertificateAuthorityConfiguration(configCA);
        req.withRevocationConfiguration(revokeConfig);
```

```
        req.withIdempotencyToken("1234");
        req.withCertificateAuthorityType(caType);
        req.withTags(tags);


        // Create the private CA.
        CreateCertificateAuthorityResult result = null;
        try {
            result = client.createCertificateAuthority(req);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String arn = result.getCertificateAuthorityArn();
        System.out.println(arn);
    }
}
```

Issue a certificate with CustomAttribute

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;
import com.amazonaws.services.acmpca.model.ASN1Subject;
```

```
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

public class IssueCertificateWithCustomAttributes {
   private static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
   }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "us-west-2";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);
```

```
      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();

      // Set the CA ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:account:" +
         "certificate-authority/12345678-1234-1234-1234-123456789012");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
      String strCSR =
      "-----BEGIN CERTIFICATE REQUEST-----\n" +
      "base64-encoded CSR\n" +
      "-----END CERTIFICATE REQUEST-----\n";
      ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
      req.setCsr(csrByteBuffer);

      // Specify the template for the issued certificate.
      req.withTemplateArn("arn:aws:acm-pca:::template/
EndEntityCertificate_APIPassthrough/V1");

      // Set the signing algorithm.
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(100L);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");

      // Define custom attributes
      List<CustomAttribute> customAttributes = Arrays.asList(
         new CustomAttribute()
               .withObjectIdentifier("2.5.4.6") // Country
               .withValue("US"),
         new CustomAttribute()
```

```
                .withObjectIdentifier("2.5.4.3") // CommonName
                .withValue("CommonName"),
          new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1") // CustomOID
                .withValue("ABCDEFG"),
          new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1") // CustomOID
                .withValue("BCDEFGH")
      );

      // Define certificate subject.
      ASN1Subject subject = new ASN1Subject();
      subject.setCustomAttributes(customAttributes);

      // Add subject to api-passthrough
      ApiPassthrough apiPassthrough = new ApiPassthrough();
      apiPassthrough.setSubject(subject);
      req.setApiPassthrough(apiPassthrough);

      // Issue the certificate.
      IssueCertificateResult result = null;
      try {
         result = client.issueCertificate(req);
      } catch (LimitExceededException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidArgsException ex) {
         throw ex;
      } catch (MalformedCSRException ex) {
         throw ex;
      }

      // Retrieve and display the certificate ARN.
      String arn = result.getCertificateArn();
      System.out.println(arn);
   }
}
```

Create certificates with custom extensions

The CustomExtension object allows administrators to set custom X.509 extensions in private certificates. Customized certificates must be created using one of the ApiPassthrough templates. For more information about templates, see AWS Private CA template varieties. For more information about using custom extensions, see Issue private end-entity certificates.

Topics

- Activate a subordinate CA with the NameConstraints extension
- Issue a certificate with the QC statement extension

Activate a subordinate CA with the NameConstraints extension

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;
import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;
import com.amazonaws.AmazonClientException;
```

```
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;

import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.GeneralSubtree;
import org.bouncycastle.asn1.x509.NameConstraints;

import lombok.SneakyThrows;

public class SubordinateCAActivationWithNameConstraints {
   public static void main(String[] args) throws Exception {
      // Place your own Root CA ARN here.
      String rootCAArn = "arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012";
```

```
      // Define the endpoint region for your sample.
      String endpointRegion = "us-west-2";  // Substitute your region here, e.g. "us-
west-2"

      // Define a CA subject.
      ASN1Subject subject = new ASN1Subject();
      subject.setOrganization("Example Organization");
      subject.setOrganizationalUnit("Example");
      subject.setCountry("US");
      subject.setState("Virginia");
      subject.setLocality("Arlington");
      subject.setCommonName("SubordinateCA");

      // Define the CA configuration.
      CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
      configCA.withKeyAlgorithm(KeyAlgorithm.RSA_2048);
      configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);
      configCA.withSubject(subject);

      // Define a certificate revocation list configuration.
      CrlConfiguration crlConfigure = new CrlConfiguration();
      crlConfigure.withEnabled(true);
      crlConfigure.withExpirationInDays(365);
      crlConfigure.withCustomCname(null);
      crlConfigure.withS3BucketName("your-bucket-name");

      // Define a certificate authority type
      CertificateAuthorityType caType = CertificateAuthorityType.SUBORDINATE;

      // ** Execute core code samples for Subordinate CA activation in sequence **
      AWSACMPCA client = ClientBuilder(endpointRegion);
      String rootCertificate = GetCertificateAuthorityCertificate(rootCAArn, client);
      String subordinateCAArn = CreateCertificateAuthority(configCA, crlConfigure,
 caType, client);
      String csr = GetCertificateAuthorityCsr(subordinateCAArn, client);
      String subordinateCertificateArn = IssueCertificate(rootCAArn, csr, client);
      String subordinateCertificate = GetCertificate(subordinateCertificateArn,
 rootCAArn, client);
      ImportCertificateAuthorityCertificate(subordinateCertificate, rootCertificate,
 subordinateCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
```

```
      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException(
                  "Cannot load the credentials from the credential profiles file. " +
                  "Please make sure that your credentials file is at the correct " +
                  "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                  e);
      }

      String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      return client;
  }

  private static String GetCertificateAuthorityCertificate(String rootCAArn, AWSACMPCA
client) {
      // ** GetCertificateAuthorityCertificate **

      // Create a request object and set the certificate authority ARN,
      GetCertificateAuthorityCertificateRequest getCACertificateRequest =
           new GetCertificateAuthorityCertificateRequest();
      getCACertificateRequest.withCertificateAuthorityArn(rootCAArn);

      // Create a result object.
      GetCertificateAuthorityCertificateResult getCACertificateResult = null;
      try {
          getCACertificateResult =
client.getCertificateAuthorityCertificate(getCACertificateRequest);
      } catch (ResourceNotFoundException ex) {
```

```
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        }

        // Retrieve and display the certificate information.
        String rootCertificate = getCACertificateResult.getCertificate();
        System.out.println("Root CA Certificate / Certificate Chain:");
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType caType, AWSACMPCA
client) {
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withRevocationConfiguration(revokeConfig);
        createCARequest.withIdempotencyToken("1234");
        createCARequest.withCertificateAuthorityType(caType);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String subordinateCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Subordinate CA Arn: " + subordinateCAArn);
```

```
        return subordinateCAArn;
    }

    private static String GetCertificateAuthorityCsr(String subordinateCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch(AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("Subordinate CSR:");
        System.out.println(csr);
```

```
        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the issuing CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
SubordinateCACertificate_PathLen0_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(100L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Generate Base64 encoded Nameconstraints extension value
        String base64EncodedExtValue = getNameConstraintExtensionValue();

        // Generate custom extension
        CustomExtension customExtension = new CustomExtension();
        customExtension.setCritical(true);
        customExtension.setObjectIdentifier("2.5.29.30"); // NameConstraints Extension
 OID
        customExtension.setValue(base64EncodedExtValue);

        // Add custom extension to api-passthrough
        ApiPassthrough apiPassthrough = new ApiPassthrough();
```

```
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String subordinateCertificateArn = issueResult.getCertificateArn();
        System.out.println("Subordinate Certificate Arn: " + subordinateCertificateArn);

        return subordinateCertificateArn;
    }

    @SneakyThrows
    private static String getNameConstraintExtensionValue() {
       // Generate Base64 encoded Nameconstraints extension value
       GeneralSubtree dnsPrivate = new GeneralSubtree(new
GeneralName(GeneralName.dNSName, ".private"));
       GeneralSubtree dnsLocal = new GeneralSubtree(new GeneralName(GeneralName.dNSName,
".local"));
       GeneralSubtree dnsCorp = new GeneralSubtree(new GeneralName(GeneralName.dNSName,
".corp"));
       GeneralSubtree dnsSecretCorp = new GeneralSubtree(new
GeneralName(GeneralName.dNSName, ".secret.corp"));
       GeneralSubtree dnsExample = new GeneralSubtree(new
GeneralName(GeneralName.dNSName, ".example.com"));
```

```
      GeneralSubtree[] permittedSubTree = new GeneralSubtree[] { dnsPrivate, dnsLocal,
dnsCorp };
      GeneralSubtree[] excludedSubTree = new GeneralSubtree[] { dnsSecretCorp,
dnsExample };
      NameConstraints nameConstraints = new NameConstraints(permittedSubTree,
excludedSubTree);

      return new String(Base64.getEncoder().encode(nameConstraints.getEncoded()));
  }

  private static String GetCertificate(String subordinateCertificateArn, String
rootCAArn, AWSACMPCA client) {

       // Create a request object.
       GetCertificateRequest certificateRequest = new GetCertificateRequest();

       // Set the certificate ARN.
       certificateRequest.withCertificateArn(subordinateCertificateArn);

       // Set the certificate authority ARN.
       certificateRequest.withCertificateAuthorityArn(rootCAArn);

       // Create waiter to wait on successful creation of the certificate file.
       Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
       try {
           getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
       } catch (WaiterUnrecoverableException e) {
           //Explicit short circuit when the recourse transitions into
           //an undesired state.
       } catch (WaiterTimedOutException e) {
           //Failed to transition into desired state even after polling.
       } catch (AWSACMPCAException e) {
           //Unexpected service exception.
       }

       // Retrieve the certificate and certificate chain.
       GetCertificateResult certificateResult = null;
       try {
           certificateResult = client.getCertificate(certificateRequest);
       } catch (RequestInProgressException ex) {
           throw ex;
       } catch (RequestFailedException ex) {
           throw ex;
```

```
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String subordinateCertificate = certificateResult.getCertificate();
        System.out.println("Subordinate CA Certificate:");
        System.out.println(subordinateCertificate);

        return subordinateCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String
subordinateCertificate, String rootCertificate, String subordinateCAArn, AWSACMPCA
client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(subordinateCertificate);
        importRequest.setCertificate(certByteBuffer);

        ByteBuffer rootCACertByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificateChain(rootCACertByteBuffer);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
```

```
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
        System.out.println("Subordinate CA certificate successfully imported.");
        System.out.println("Subordinate CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

Issue a certificate with the QC statement extension

```
package com.amazonaws.samples;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
```

```
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

import org.bouncycastle.asn1.ASN1EncodableVector;
import org.bouncycastle.asn1.ASN1ObjectIdentifier;
import org.bouncycastle.asn1.DERSequence;
import org.bouncycastle.asn1.DERUTF8String;
import org.bouncycastle.asn1.x509.qualified.ETSIQCObjectIdentifiers;
import org.bouncycastle.asn1.x509.qualified.QCStatement;

import lombok.SneakyThrows;

public class IssueCertificateWithQCStatement {
   private static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
   }

   @SneakyThrows
   private static String generateQCStatementBase64ExtValue() {
      DERSequence qcTypeSeq = new DERSequence(ETSIQCObjectIdentifiers.id_etsi_qct_web);
      QCStatement qcType = new QCStatement(ETSIQCObjectIdentifiers.id_etsi_qcs_QcType,
 qcTypeSeq);

      ASN1EncodableVector pspAIVector = new ASN1EncodableVector(2);
      pspAIVector.add(new ASN1ObjectIdentifier("0.4.0.19495.1.3"));
      pspAIVector.add(new DERUTF8String("PSP_AI"));
      DERSequence pspAISeq = new DERSequence(pspAIVector);

      ASN1EncodableVector pspASVector = new ASN1EncodableVector(2);
      pspASVector.add(new ASN1ObjectIdentifier("0.4.0.19495.1.1"));
      pspASVector.add(new DERUTF8String("PSP_AS"));
      DERSequence pspASSeq = new DERSequence(pspASVector);
```

```
      ASN1EncodableVector pspPIVector = new ASN1EncodableVector(2);
      pspPIVector.add(new ASN1ObjectIdentifier("0.4.0.19495.1.2"));
      pspPIVector.add(new DERUTF8String("PSP_PI"));
      DERSequence pspPISeq = new DERSequence(pspPIVector);

      ASN1EncodableVector pspICVector = new ASN1EncodableVector(2);
      pspICVector.add(new ASN1ObjectIdentifier("0.4.0.19495.1.4"));
      pspICVector.add(new DERUTF8String("PSP_IC"));
      DERSequence pspICSeq = new DERSequence(pspICVector);

      ASN1EncodableVector pspSeqVector = new ASN1EncodableVector(4);
      pspSeqVector.add(pspPISeq);
      pspSeqVector.add(pspICSeq);
      pspSeqVector.add(pspASSeq);
      pspSeqVector.add(pspAISeq);
      DERSequence pspSeq = new DERSequence(pspSeqVector);

      ASN1EncodableVector pspVector = new ASN1EncodableVector(3);
      pspVector.add(pspSeq);
      pspVector.add(new DERUTF8String("Your Financial Authority"));
      pspVector.add(new DERUTF8String("AB-CD"));
      DERSequence psp = new DERSequence(pspVector);
      QCStatement qcPSP = new QCStatement(new ASN1ObjectIdentifier("0.4.0.19495.2"),
psp);

      DERSequence qcSeq = new DERSequence(new QCStatement[] { qcType, qcPSP });

      byte[] qcExtValueInBytes = qcSeq.getEncoded();
      return Base64.getEncoder().encodeToString(qcExtValueInBytes);
   }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
```

```
      String endpointRegion = "us-west-2";  // Substitute your region here, e.g. "us-
west-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();

      // Set the CA ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:account:" +
         "certificate-authority/12345678-1234-1234-1234-123456789012");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
      String strCSR =
      "-----BEGIN CERTIFICATE REQUEST-----\n" +
      "base64-encoded CSR\n" +
      "-----END CERTIFICATE REQUEST-----\n";
      ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
      req.setCsr(csrByteBuffer);

      // Specify the template for the issued certificate.
      req.withTemplateArn("arn:aws:acm-pca:::template/
EndEntityCertificate_APIPassthrough/V1");

      // Set the signing algorithm.
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHRSA);

      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(30L);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");
```

```
      // Generate Base64 encoded extension value for QC Statement
      String base64EncodedExtValue = generateQCStatementBase64ExtValue();

      // Generate custom extension
      CustomExtension customExtension = new CustomExtension();
      customExtension.setObjectIdentifier("1.3.6.1.5.5.7.1.3"); // QC Statement
 Extension OID
      customExtension.setValue(base64EncodedExtValue);

      // Add custom extension to api-passthrough
      ApiPassthrough apiPassthrough = new ApiPassthrough();
      Extensions extensions = new Extensions();
      extensions.setCustomExtensions(Arrays.asList(customExtension));
      apiPassthrough.setExtensions(extensions);
      req.setApiPassthrough(apiPassthrough);

      // Issue the certificate.
      IssueCertificateResult result = null;
      try {
         result = client.issueCertificate(req);
      } catch (LimitExceededException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidArgsException ex) {
         throw ex;
      } catch (MalformedCSRException ex) {
         throw ex;
      }

      // Retrieve and display the certificate ARN.
      String arn = result.getCertificateArn();
      System.out.println(arn);
   }
}
```

Use AWS Private CA to implement Matter certificates

You can use the AWS Private Certificate Authority API to create certificates that conform to the Matter connectivity standard. Matter specifies certificate configurations that improve the security and consistency of internet of things (IoT) devices across multiple engineering platforms. For more information about Matter, see buildwithmatter.com.

Matter 1.2, released in October 2023, supports DAC revocation using Certificate Revocation Lists (CRLs). To help you conform to the current Matter standard, when you enable CRL revocation for CAs that issue Matter certificates, in the `CrlConfiguration` object, in the `CrlDistributionPointExtensionConfiguration` structure, set `OmitExtension` to `true`.

Typically, CAs embed the CRL Distribution Point (CDP) in the certificates they issue so that the relying parties performing certificate chain validation can fetch the CRL and check the certificate status. In Matter, the CDP URI is not written to certificates. Instead, users fetch CDPs from the Matter Distributed Compliance Ledger (DCL), the trusted Matter data store. You must upload the CDP URI to the Matter DCL so that it can be discovered when validating DACs. For more information about determining the CDP URI, see Determining the CRL Distribution Point (CDP) URI . For more information about Matter, see the Matter standard home page.

Topics

- Activate a Product Attestation Authority (PAA)
- Activate an Product Attestation Intermediate (PAI)
- Create a Device Attestation Certificate (DAC)
- Activate a Root CA for Node Operational Certificates (NOC).
- Activate a Subordinate CA for Node Operational Certificates (NOC)
- Create a Node Operational Certificate (NOC)

Activate a Product Attestation Authority (PAA)

This Java sample shows how to use the RootCACertificate_APIPassthrough/V1 definition template to create and install a Matter Root CA (PAA) certificate for product attestation. The AuthorityKeyIdentifier (AKI) extension is optional for PAAs. To set an AKI, you must generate a Base64-encoded AKI value and pass it through a CustomExtension.

The example calls the following AWS Private CA API actions:

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

If you encounter problems, see Troubleshoot AWS Private CA Matter-compliant certificate errors in the Troubleshooting section.

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.samples.GetCertificateAuthorityCertificate;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
```

```
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CrlDistributionPointExtensionConfiguration;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.cert.jcajce.JcaX509ExtensionUtils;
import org.bouncycastle.openssl.PEMParser;
import org.bouncycastle.pkcs.PKCS10CertificationRequest;
import org.bouncycastle.util.io.pem.PemReader;
```

```
import lombok.SneakyThrows;

public class ProductAttestationAuthorityActivation {

    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.3") // CommonName
                .withValue("Matter Test PAA"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.2.1") // Vendor ID
                .withValue("FFF1")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);

        // Define a CRL distribution point extension configuration
        CrlDistributionPointExtensionConfiguration CDPConfigure = new
 CrlDistributionPointExtensionConfiguration();
        CDPConfigure.withOmitExtension(true);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");
        crlConfigure.withS3ObjectAcl("BUCKET_OWNER_FULL_CONTROL");
        crlConfigure.withCrlDistributionPointExtensionConfiguration(CDPConfigure);
```

```
        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.ROOT;

        // ** Execute core code samples for Root CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCAArn = CreateCertificateAuthority(configCA, crlConfigure, CAtype,
client);
        String csr = GetCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String rootCertificate = GetCertificate(rootCertificateArn, rootCAArn, client);
        ImportCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }
```

```
   private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType CAtype, AWSACMPCA
client) {
       RevocationConfiguration revokeConfig = new RevocationConfiguration();
       revokeConfig.setCrlConfiguration(crlConfigure);

       // Create the request object.
       CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
       createCARequest.withCertificateAuthorityConfiguration(configCA);
       createCARequest.withIdempotencyToken("123987");
       createCARequest.withCertificateAuthorityType(CAtype);
       createCARequest.withRevocationConfiguration(revokeConfig);

       // Create the private CA.
       CreateCertificateAuthorityResult createCAResult = null;
       try {
           createCAResult = client.createCertificateAuthority(createCARequest);
       } catch (InvalidArgsException ex) {
           throw ex;
       } catch (InvalidPolicyException ex) {
           throw ex;
       } catch (LimitExceededException ex) {
           throw ex;
       }

       // Retrieve the ARN of the private CA.
       String rootCAArn = createCAResult.getCertificateAuthorityArn();
       System.out.println("Product Attestation Authority (PAA) Arn: " + rootCAArn);

       return rootCAArn;
   }

   private static String GetCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

       // Create the CSR request object and set the CA ARN.
       GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
       csrRequest.withCertificateAuthorityArn(rootCAArn);

       // Create waiter to wait on successful creation of the CSR file.
       Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
```

```
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println(csr);

        return csr;
    }

    @SneakyThrows
    private static String generateAuthorityKeyIdentifier(final String csrPEM) {
        PKCS10CertificationRequest csr = getPKCS10CertificationRequest(csrPEM);
        SubjectPublicKeyInfo spki = csr.getSubjectPublicKeyInfo();

        JcaX509ExtensionUtils extensionUtils = new JcaX509ExtensionUtils();
        byte[] akiBytes =
extensionUtils.createAuthorityKeyIdentifier(spki).getEncoded();

        return Base64.getEncoder().encodeToString(akiBytes);
    }
```

```
    @SneakyThrows
    private static PKCS10CertificationRequest getPKCS10CertificationRequest(final
 String csrPEM) {
        ByteArrayInputStream bais = new ByteArrayInputStream(csrPEM.getBytes());
        PemReader pemReader = new PemReader(new InputStreamReader(bais));
        PEMParser parser = new PEMParser(pemReader);
        Object o = parser.readObject();
        if (o instanceof PKCS10CertificationRequest) {
            return (PKCS10CertificationRequest) o;
        }
        return null;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
RootCACertificate_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(3650L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Generate Base64 encoded extension value for AuthorityKeyIdentifier
        String base64EncodedExtValue = generateAuthorityKeyIdentifier(csr);
```

```
        // Generate custom extension
        CustomExtension customExtension = new CustomExtension();
        customExtension.setObjectIdentifier("2.5.29.35"); // AuthorityKeyIdentifier
Extension OID
        customExtension.setValue(base64EncodedExtValue);

        // Add custom extension to api-passthrough
        ApiPassthrough apiPassthrough = new ApiPassthrough();
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("Product Attestation Authority (PAA) Certificate Arn: " +
rootCertificateArn);

        return rootCertificateArn;
    }

    private static String GetCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();
```

```
        // Set the certificate ARN.
        certificateRequest.withCertificateArn(rootCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }
```

```
   private static void ImportCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {

       // Create the request object and set the signed certificate, chain and CA ARN.
       ImportCertificateAuthorityCertificateRequest importRequest =
           new ImportCertificateAuthorityCertificateRequest();

       ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
       importRequest.setCertificate(certByteBuffer);

       importRequest.setCertificateChain(null);

       // Set the certificate authority ARN.
       importRequest.withCertificateAuthorityArn(rootCAArn);

       // Import the certificate.
       try {
           client.importCertificateAuthorityCertificate(importRequest);
       } catch (CertificateMismatchException ex) {
           throw ex;
       } catch (MalformedCertificateException ex) {
           throw ex;
       } catch (InvalidArnException ex) {
           throw ex;
       } catch (ResourceNotFoundException ex) {
           throw ex;
       } catch (RequestInProgressException ex) {
           throw ex;
       } catch (ConcurrentModificationException ex) {
           throw ex;
       } catch (RequestFailedException ex) {
           throw ex;
       }

       System.out.println("Product Attestation Authority (PAA) certificate
successfully imported.");
       System.out.println("Product Attestation Authority (PAA) activated
successfully.");
   }

   private static ByteBuffer stringToByteBuffer(final String string) {
       if (Objects.isNull(string)) {
           return null;
       }
```

```
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

Activate an Product Attestation Intermediate (PAI)

This Java sample shows how to use the BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1 definition template to create and install a Matter Subordinate CA (PAI) certificate for product attestation. You must generate a Base64-encoded KeyUsage value and pass it through a CustomExtension.

The example calls the following AWS Private CA API actions:

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate
- GetCertificateAuthorityCertificate

If you encounter problems, see Troubleshoot AWS Private CA Matter-compliant certificate errors in the Troubleshooting section.

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
```

```
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CrlDistributionPointExtensionConfiguration;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
```

```
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import lombok.SneakyThrows;

public class ProductAttestationIntermediateActivation {

    public static void main(String[] args) throws Exception {
        // Place your own Root CA ARN here.
        String paaArn = "arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012";

        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("2.5.4.3") // CommonName
                .withValue("Matter Test PAI"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.2.1") // Vendor ID
                .withValue("FFF1"),
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.2.2") // Product ID
                .withValue("8000")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
```

```
        configCA.withSubject(subject);

        // Define a CRL distribution point extension configuration
        CrlDistributionPointExtensionConfiguration CDPConfigure = new
CrlDistributionPointExtensionConfiguration();
        CDPConfigure.withOmitExtension(true);

        // Define a certificate revocation list configuration.
        CrlConfiguration crlConfigure = new CrlConfiguration();
        crlConfigure.withEnabled(true);
        crlConfigure.withExpirationInDays(365);
        crlConfigure.withCustomCname(null);
        crlConfigure.withS3BucketName("your-bucket-name");
        crlConfigure.withS3ObjectAcl("BUCKET_OWNER_FULL_CONTROL");
        crlConfigure.withCrlDistributionPointConfiguration(CDPConfigure);

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.SUBORDINATE;

        // ** Execute core code samples for Subordinate CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCertificate = GetCertificateAuthorityCertificate(paaArn, client);
        String subordinateCAArn = CreateCertificateAuthority(configCA, crlConfigure,
CAtype, client);
        String csr = GetCertificateAuthorityCsr(subordinateCAArn, client);
        String subordinateCertificateArn = IssueCertificate(paaArn, csr, client);
        String subordinateCertificate = GetCertificate(subordinateCertificateArn,
paaArn, client);
        ImportCertificateAuthorityCertificate(subordinateCertificate, rootCertificate,
subordinateCAArn, client);

    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
```

```
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String GetCertificateAuthorityCertificate(String rootCAArn,
AWSACMPCA client) {
        // ** GetCertificateAuthorityCertificate **

        // Create a request object and set the certificate authority ARN,
        GetCertificateAuthorityCertificateRequest getCACertificateRequest =
        new GetCertificateAuthorityCertificateRequest();
        getCACertificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create a result object.
        GetCertificateAuthorityCertificateResult getCACertificateResult = null;
        try {
            getCACertificateResult =
client.getCertificateAuthorityCertificate(getCACertificateRequest);
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        }

        // Retrieve and display the certificate information.
        String rootCertificate = getCACertificateResult.getCertificate();
```

```
        System.out.println("Product Attestation Authority (PAA) Certificate /
Certificate Chain:");
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CrlConfiguration crlConfigure, CertificateAuthorityType CAtype, AWSACMPCA
client) {
        RevocationConfiguration revokeConfig = new RevocationConfiguration();
        revokeConfig.setCrlConfiguration(crlConfigure);

        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);
        createCARequest.withRevocationConfiguration(revokeConfig);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String subordinateCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Product Attestation Intermediate (PAI) Arn: " +
subordinateCAArn);

        return subordinateCAArn;
    }

    private static String GetCertificateAuthorityCsr(String subordinateCAArn, AWSACMPCA
client) {
```

```
        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch(AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("Subordinate CSR:");
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
client) {
```

```
        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the issuing CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(730L); // Approximately two years
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        ApiPassthrough apiPassthrough = new ApiPassthrough();

        // Generate Base64 encoded extension value for ExtendedKeyUsage
        String base64EncodedKUValue = generateKeyUsageValue();

        // Generate custom extension
        CustomExtension customKeyUsageExtension = new CustomExtension();
        customKeyUsageExtension.setObjectIdentifier("2.5.29.15");
        customKeyUsageExtension.setValue(base64EncodedKUValue);
        customKeyUsageExtension.setCritical(true);

        // Set KeyUsage extension to api passthrough
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customKeyUsageExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
```

```
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String subordinateCertificateArn = issueResult.getCertificateArn();
        System.out.println("Subordinate Certificate Arn: " +
subordinateCertificateArn);

        return subordinateCertificateArn;
    }

    @SneakyThrows
    private static String generateKeyUsageValue() {
       KeyUsage keyUsage = new KeyUsage(X509KeyUsage.keyCertSign |
X509KeyUsage.cRLSign);
       byte[] kuBytes = keyUsage.getEncoded();
       return Base64.getEncoder().encodeToString(kuBytes);
    }

    private static String GetCertificate(String subordinateCertificateArn, String
rootCAArn, AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(subordinateCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);
```

```
        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String subordinateCertificate = certificateResult.getCertificate();
        System.out.println("Subordinate CA Certificate:");
        System.out.println(subordinateCertificate);

        return subordinateCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String
subordinateCertificate, String rootCertificate, String subordinateCAArn, AWSACMPCA
client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
```

```
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(subordinateCertificate);
        importRequest.setCertificate(certByteBuffer);

        ByteBuffer rootCACertByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificateChain(rootCACertByteBuffer);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
        System.out.println("Product Attestation Intermediate (PAI) certificate
 successfully imported.");
        System.out.println("Product Attestation Intermediate (PAI) activated
 successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

Create a Device Attestation Certificate (DAC)

This Java sample shows how to use the BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1 template to create a Matter Device Attestation Certificate. You must generate a Base64-encoded KeyUsage value and pass it through a CustomExtension.

The example calls the following AWS Private CA API action:

- IssueCertificate

If you encounter problems, see Troubleshoot AWS Private CA Matter-compliant certificate errors in the Troubleshooting section.

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;
```

```
import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import lombok.SneakyThrows;

public class IssueDeviceAttestationCertificate {
   public static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
   }

   @SneakyThrows
   private static String generateKeyUsageValue() {
      KeyUsage keyUsage = new KeyUsage(X509KeyUsage.digitalSignature);
      byte[] kuBytes = keyUsage.getEncoded();
      return Base64.getEncoder().encodeToString(kuBytes);
   }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"
```

```
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();

      // Set the CA ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
      String strCSR =
      "-----BEGIN CERTIFICATE REQUEST-----\n" +
      "base64-encoded certificate\n" +
      "-----END CERTIFICATE REQUEST-----\n";
      ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
      req.setCsr(csrByteBuffer);

      // Specify the template for the issued certificate.
      req.withTemplateArn("arn:aws:acm-pca:::template/
BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1");

      // Set the signing algorithm.
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(10L);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");

      // Define custom attributes
```

```
      List<CustomAttribute> customAttributes = Arrays.asList(
         new CustomAttribute()
            .withObjectIdentifier("2.5.4.3")
            .withValue("Matter Test DAC 0001"),
         new CustomAttribute()
            .withObjectIdentifier("1.3.6.1.4.1.37244.2.1")
            .withValue("FFF1"),
         new CustomAttribute()
            .withObjectIdentifier("1.3.6.1.4.1.37244.2.2")
            .withValue("8000")
      );

      // Define a cert subject.
      ASN1Subject subject = new ASN1Subject();
      subject.setCustomAttributes(customAttributes);

      ApiPassthrough apiPassthrough = new ApiPassthrough();
      apiPassthrough.setSubject(subject);

      // Generate Base64 encoded extension value for ExtendedKeyUsage
      String base64EncodedKUValue = generateKeyUsageValue();

      // Generate custom extension
      CustomExtension customKeyUsageExtension = new CustomExtension();
      customKeyUsageExtension.setObjectIdentifier("2.5.29.15"); // KeyUsage Extension
OID
      customKeyUsageExtension.setValue(base64EncodedKUValue);
      customKeyUsageExtension.setCritical(true);

      Extensions extensions = new Extensions();
      extensions.setCustomExtensions(Arrays.asList(customKeyUsageExtension));
      apiPassthrough.setExtensions(extensions);
      req.setApiPassthrough(apiPassthrough);

      // Issue the certificate.
      IssueCertificateResult result = null;
      try {
         result = client.issueCertificate(req);
      } catch (LimitExceededException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
```

```
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String arn = result.getCertificateArn();
        System.out.println(arn);
    }
}
```

Activate a Root CA for Node Operational Certificates (NOC).

This Java sample shows how to use the RootCACertificate_APIPassthrough/V1 definition template to create and install a Matter Root CA certificate to issue NOCs. The AuthorityKeyIdentifier (AKI) extension is optional for NOC Root CA certificates. To set an AKI, you must generate a Base64-encoded AKI value and pass it through a CustomExtension.

The example calls the following AWS Private CA API actions:

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

If you encounter problems, see Troubleshoot AWS Private CA Matter-compliant certificate errors in the Troubleshooting section.

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.samples.GetCertificateAuthorityCertificate;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
```

```
import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CrlConfiguration;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Tag;

import java.io.ByteArrayInputStream;
import java.io.InputStreamReader;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
```

```
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.RevocationConfiguration;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import org.bouncycastle.asn1.x509.SubjectPublicKeyInfo;
import org.bouncycastle.cert.jcajce.JcaX509ExtensionUtils;
import org.bouncycastle.openssl.PEMParser;
import org.bouncycastle.pkcs.PKCS10CertificationRequest;
import org.bouncycastle.util.io.pem.PemReader;

import lombok.SneakyThrows;

public class RootCAActivation {
    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.1.4")
                .withValue("CACACACA00000001")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
```

```
        CertificateAuthorityConfiguration configCA = new
CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.ROOT;

        // ** Execute core code samples for Root CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCAArn = CreateCertificateAuthority(configCA, CAtype, client);
        String csr = GetCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String rootCertificate = GetCertificate(rootCertificateArn, rootCAArn, client);
        ImportCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Retrieve your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();
```

```
        return client;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CertificateAuthorityType CAtype, AWSACMPCA client) {
        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String rootCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Root CA Arn: " + rootCAArn);

        return rootCAArn;
    }

    private static String GetCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
```

```
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Retrieve and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println(csr);

        return csr;
    }

    @SneakyThrows
    private static String generateAuthorityKeyIdentifier(final String csrPEM) {
        PKCS10CertificationRequest csr = getPKCS10CertificationRequest(csrPEM);
        SubjectPublicKeyInfo spki = csr.getSubjectPublicKeyInfo();

        JcaX509ExtensionUtils extensionUtils = new JcaX509ExtensionUtils();
        byte[] akiBytes =
extensionUtils.createAuthorityKeyIdentifier(spki).getEncoded();

        return Base64.getEncoder().encodeToString(akiBytes);
    }

    @SneakyThrows
```

```
    private static PKCS10CertificationRequest getPKCS10CertificationRequest(final
 String csrPEM) {
        ByteArrayInputStream bais = new ByteArrayInputStream(csrPEM.getBytes());
        PemReader pemReader = new PemReader(new InputStreamReader(bais));
        PEMParser parser = new PEMParser(pemReader);
        Object o = parser.readObject();
        if (o instanceof PKCS10CertificationRequest) {
            return (PKCS10CertificationRequest) o;
        }
        return null;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
RootCACertificate_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(3650L);
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        // Generate Base64 encoded extension value for AuthorityKeyIdentifier
        String base64EncodedExtValue = generateAuthorityKeyIdentifier(csr);

        // Generate custom extension
```

```
        CustomExtension customExtension = new CustomExtension();
        customExtension.setObjectIdentifier("2.5.29.35"); // AuthorityKeyIdentifier
Extension OID
        customExtension.setValue(base64EncodedExtValue);

        // Add custom extension to api-passthrough
        ApiPassthrough apiPassthrough = new ApiPassthrough();
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Retrieve and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("Root Certificate Arn: " + rootCertificateArn);

        return rootCertificateArn;
    }

    private static String GetCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
```

```
        certificateRequest.withCertificateArn(rootCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Retrieve the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static void ImportCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {
```

```
        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificate(certByteBuffer);

        importRequest.setCertificateChain(null);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(rootCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Root CA certificate successfully imported.");
        System.out.println("Root CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
```

```
}
```

Activate a Subordinate CA for Node Operational Certificates (NOC)

This Java sample shows how to use the BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1 definition template to issue and install a Matter Subordinate CA certificate to issue NOCs. You must generate a Base64-encoded KeyUsage value and pass it through a CustomExtension.

The example calls the following AWS Private CA API actions:

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate
- GetCertificateAuthorityCertificate

If problems occur, see Troubleshoot AWS Private CA Matter-compliant certificate errors in the Troubleshooting section.

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;


import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
```

```
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCertificateResult;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;

import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
```

```
import com.amazonaws.waiters.WaiterUnrecoverableException;

import lombok.SneakyThrows;

public class IntermediateCAActivation {

    public static void main(String[] args) throws Exception {
        // Place your own Root CA ARN here.
        String rootCAArn = "arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012";

        // Define the endpoint region for your sample.
        String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"

        // Define custom attributes
        List<CustomAttribute> customAttributes = Arrays.asList(
            new CustomAttribute()
                .withObjectIdentifier("1.3.6.1.4.1.37244.1.3")
                .withValue("CACACACA00000003")
        );

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject();
        subject.setCustomAttributes(customAttributes);

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration();
        configCA.withKeyAlgorithm(KeyAlgorithm.EC_prime256v1);
        configCA.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);
        configCA.withSubject(subject);

        // Define a certificate authority type
        CertificateAuthorityType CAtype = CertificateAuthorityType.SUBORDINATE;

        // ** Execute core code samples for Subordinate CA activation in sequence **
        AWSACMPCA client = ClientBuilder(endpointRegion);
        String rootCertificate = GetCertificateAuthorityCertificate(rootCAArn, client);
        String subordinateCAArn = CreateCertificateAuthority(configCA, CAtype, client);
        String csr = GetCertificateAuthorityCsr(subordinateCAArn, client);
        String subordinateCertificateArn = IssueCertificate(rootCAArn, csr, client);
        String subordinateCertificate = GetCertificate(subordinateCertificateArn,
 rootCAArn, client);
```

```
        ImportCertificateAuthorityCertificate(subordinateCertificate, rootCertificate,
subordinateCAArn, client);

    }

    private static AWSACMPCA ClientBuilder(String endpointRegion) {
        // Get your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (C:\\Users\\joneps\\.aws\\credentials), and is in valid
format.",
                    e);
        }

        String endpointProtocol = "https://acm-pca." + endpointRegion +
".amazonaws.com/";
        EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol,
endpointRegion);

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

        return client;
    }

    private static String GetCertificateAuthorityCertificate(String rootCAArn,
AWSACMPCA client) {
        // ** GetCertificateAuthorityCertificate **

        // Create a request object and set the certificate authority ARN,
        GetCertificateAuthorityCertificateRequest getCACertificateRequest =
        new GetCertificateAuthorityCertificateRequest();
        getCACertificateRequest.withCertificateAuthorityArn(rootCAArn);
```

```
        // Create a result object.
        GetCertificateAuthorityCertificateResult getCACertificateResult = null;
        try {
            getCACertificateResult =
client.getCertificateAuthorityCertificate(getCACertificateRequest);
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        }

        // Get and display the certificate information.
        String rootCertificate = getCACertificateResult.getCertificate();
        System.out.println("Root CA Certificate / Certificate Chain:");
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static String CreateCertificateAuthority(CertificateAuthorityConfiguration
configCA, CertificateAuthorityType CAtype, AWSACMPCA client) {
        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest();
        createCARequest.withCertificateAuthorityConfiguration(configCA);
        createCARequest.withIdempotencyToken("123987");
        createCARequest.withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }

        // Retrieve the ARN of the private CA.
        String subordinateCAArn = createCAResult.getCertificateAuthorityArn();
```

```
        System.out.println("Subordinate CA Arn: " + subordinateCAArn);

        return subordinateCAArn;
    }

    private static String GetCertificateAuthorityCsr(String subordinateCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest();
        csrRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch(AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Get the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        // Get and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("Subordinate CSR:");
```

```
        System.out.println(csr);

        return csr;
    }

    private static String IssueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {

        // Create a certificate request:
        IssueCertificateRequest issueRequest = new IssueCertificateRequest();

        // Set the issuing CA ARN.
        issueRequest.withCertificateAuthorityArn(rootCAArn);

        // Set the template ARN.
        issueRequest.withTemplateArn("arn:aws:acm-pca:::template/
BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1");

        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the signing algorithm.
        issueRequest.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity();
        validity.withValue(730L); // Approximately two years
        validity.withType("DAYS");
        issueRequest.withValidity(validity);

        // Set the idempotency token.
        issueRequest.setIdempotencyToken("1234");

        ApiPassthrough apiPassthrough = new ApiPassthrough();

        // Generate base64 encoded extension value for ExtendedKeyUsage
        String base64EncodedKUValue = generateKeyUsageValue();

        // Generate custom extension
        CustomExtension customKeyUsageExtension = new CustomExtension();
        customKeyUsageExtension.setObjectIdentifier("2.5.29.15");
        customKeyUsageExtension.setValue(base64EncodedKUValue);
        customKeyUsageExtension.setCritical(true);
```

```
        // Set KeyUsage extension to api passthrough
        Extensions extensions = new Extensions();
        extensions.setCustomExtensions(Arrays.asList(customKeyUsageExtension));
        apiPassthrough.setExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Get and display the certificate ARN.
        String subordinateCertificateArn = issueResult.getCertificateArn();
        System.out.println("Subordinate Certificate Arn: " +
subordinateCertificateArn);

        return subordinateCertificateArn;
    }

    @SneakyThrows
    private static String generateKeyUsageValue() {
       KeyUsage keyUsage = new KeyUsage(X509KeyUsage.keyCertSign |
X509KeyUsage.cRLSign);
       byte[] kuBytes = keyUsage.getEncoded();
       return Base64.getEncoder().encodeToString(kuBytes);
    }

    private static String GetCertificate(String subordinateCertificateArn, String
rootCAArn, AWSACMPCA client) {

        // Create a request object.
```

```
        GetCertificateRequest certificateRequest = new GetCertificateRequest();

        // Set the certificate ARN.
        certificateRequest.withCertificateArn(subordinateCertificateArn);

        // Set the certificate authority ARN.
        certificateRequest.withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            //Explicit short circuit when the recourse transitions into
            //an undesired state.
        } catch (WaiterTimedOutException e) {
            //Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            //Unexpected service exception.
        }

        // Get the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String subordinateCertificate = certificateResult.getCertificate();
        System.out.println("Subordinate CA Certificate:");
        System.out.println(subordinateCertificate);

        return subordinateCertificate;
```

```
    }

    private static void ImportCertificateAuthorityCertificate(String
subordinateCertificate, String rootCertificate, String subordinateCAArn, AWSACMPCA
client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
            new ImportCertificateAuthorityCertificateRequest();

        ByteBuffer certByteBuffer = stringToByteBuffer(subordinateCertificate);
        importRequest.setCertificate(certByteBuffer);

        ByteBuffer rootCACertByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificateChain(rootCACertByteBuffer);

        // Set the certificate authority ARN.
        importRequest.withCertificateAuthorityArn(subordinateCAArn);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
        System.out.println("Subordinate CA certificate successfully imported.");
        System.out.println("Subordinate CA activated successfully.");
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
```

```
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

Create a Node Operational Certificate (NOC)

This Java sample shows how to use the BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1 template to create a Matter Node Operational Certificate. You must generate a Base64-encoded KeyUsage value and pass it through a CustomExtension.

The example calls the following AWS Private CA API action:

- IssueCertificate

If you encounter problems, see Troubleshoot AWS Private CA Matter-compliant certificate errors in the Troubleshooting section.

```
package com.amazonaws.samples.matter;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.client.builder.AwsClientBuilder.EndpointConfiguration;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.List;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CustomAttribute;
import com.amazonaws.services.acmpca.model.CustomExtension;
```

```
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

import org.bouncycastle.asn1.x509.ExtendedKeyUsage;
import org.bouncycastle.asn1.x509.KeyPurposeId;
import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import lombok.SneakyThrows;

public class IssueNodeOperatingCertificate {
   public static ByteBuffer stringToByteBuffer(final String string) {
      if (Objects.isNull(string)) {
          return null;
      }
      byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
      return ByteBuffer.wrap(bytes);
   }

   @SneakyThrows
   private static String generateExtendedKeyUsageValue() {
      KeyPurposeId[] keyPurposeIds = new KeyPurposeId[]
 { KeyPurposeId.id_kp_clientAuth, KeyPurposeId.id_kp_serverAuth };
      ExtendedKeyUsage eku = new ExtendedKeyUsage(keyPurposeIds);
      byte[] ekuBytes = eku.getEncoded();
      return Base64.getEncoder().encodeToString(ekuBytes);
   }

   @SneakyThrows
   private static String generateKeyUsageValue() {
      KeyUsage keyUsage = new KeyUsage(X509KeyUsage.digitalSignature);
      byte[] kuBytes = keyUsage.getEncoded();
      return Base64.getEncoder().encodeToString(kuBytes);
```

```
   }

   public static void main(String[] args) throws Exception {

      // Retrieve your credentials from the C:\Users\name\.aws\credentials file
      // in Windows or the .aws/credentials file in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider("default").getCredentials();
      } catch (Exception e) {
          throw new AmazonClientException("Cannot load your credentials from disk", e);
      }

      // Define the endpoint for your sample.
      String endpointRegion = "region";  // Substitute your region here, e.g. "ap-
southeast-2"
      String endpointProtocol = "https://acm-pca." + endpointRegion +
 ".amazonaws.com/";
      EndpointConfiguration endpoint =
          new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

      // Create a client that you can use to make requests.
      AWSACMPCA client = AWSACMPCAClientBuilder.standard()
          .withEndpointConfiguration(endpoint)
          .withCredentials(new AWSStaticCredentialsProvider(credentials))
          .build();

      // Create a certificate request:
      IssueCertificateRequest req = new IssueCertificateRequest();

      // Set the CA ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:123456789012:certificate-
authority/12345678-1234-1234-1234-123456789012");

      // Specify the certificate signing request (CSR) for the certificate to be signed
 and issued.
      String strCSR =
      "-----BEGIN CERTIFICATE REQUEST-----\n" +
      "base64-encoded certificate\n" +
      "-----END CERTIFICATE REQUEST-----\n";
      ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
      req.setCsr(csrByteBuffer);

      // Specify the template for the issued certificate.
```

```
      req.withTemplateArn("arn:aws:acm-pca:::template/
BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1");

      // Set the signing algorithm.
      req.withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA);

      // Set the validity period for the certificate to be issued.
      Validity validity = new Validity();
      validity.withValue(10L);
      validity.withType("DAYS");
      req.withValidity(validity);

      // Set the idempotency token.
      req.setIdempotencyToken("1234");

      // Define custom attributes
      List<CustomAttribute> customAttributes = Arrays.asList(
         new CustomAttribute()
            .withObjectIdentifier("1.3.6.1.4.1.37244.1.1")
            .withValue("DEDEDEDE00010001"),
         new CustomAttribute()
            .withObjectIdentifier("1.3.6.1.4.1.37244.1.5")
            .withValue("FAB000000000001D")
      );

      // Define a cert subject.
      ASN1Subject subject = new ASN1Subject();
      subject.setCustomAttributes(customAttributes);

      ApiPassthrough apiPassthrough = new ApiPassthrough();
      apiPassthrough.setSubject(subject);

      // Generate Base64 encoded extension value for ExtendedKeyUsage
      String base64EncodedKUValue = generateKeyUsageValue();

      // Generate custom extension
      CustomExtension customKeyUsageExtension = new CustomExtension();
      customKeyUsageExtension.setObjectIdentifier("2.5.29.15");
      customKeyUsageExtension.setValue(base64EncodedKUValue);
      customKeyUsageExtension.setCritical(true);

      // Generate Base64 encoded extension value for ExtendedKeyUsage
      String base64EncodedEKUValue = generateExtendedKeyUsageValue();
```

```
      CustomExtension customExtendedKeyUsageExtension = new CustomExtension();
      customExtendedKeyUsageExtension.setObjectIdentifier("2.5.29.37"); //
 ExtendedKeyUsage Extension OID
      customExtendedKeyUsageExtension.setValue(base64EncodedEKUValue);
      customExtendedKeyUsageExtension.setCritical(true);

      // Set KeyUsage and ExtendedKeyUsage extension to api-passthrough
      Extensions extensions = new Extensions();
      extensions.setCustomExtensions(Arrays.asList(customKeyUsageExtension,
 customExtendedKeyUsageExtension));
      apiPassthrough.setExtensions(extensions);
      req.setApiPassthrough(apiPassthrough);

      // Issue the certificate.
      IssueCertificateResult result = null;
      try {
         result = client.issueCertificate(req);
      } catch (LimitExceededException ex) {
         throw ex;
      } catch (ResourceNotFoundException ex) {
         throw ex;
      } catch (InvalidStateException ex) {
         throw ex;
      } catch (InvalidArnException ex) {
         throw ex;
      } catch (InvalidArgsException ex) {
         throw ex;
      } catch (MalformedCSRException ex) {
         throw ex;
      }

      // Retrieve and display the certificate ARN.
      String arn = result.getCertificateArn();
      System.out.println(arn);
   }
}
```

Use AWS Private CA to implement mDL certificates

You can use the AWS Private Certificate Authority API to create certificates that conform to the ISO/IEC standard for mobile driving license (mDL) . This standard establishes interface

specifications for the implementation of a driving license in association with a mobile device, including certificate configurations.

Topics

- Activate an issuing authority certificate authority (IACA) certificate
- Create a document signer certificate

Activate an issuing authority certificate authority (IACA) certificate

This Java sample shows how to use the BlankRootCACertificate_PathLen0_APIPassthrough/V1 definition template to create and install an ISO/IEC mDL standard-compliant issuing authority certificate authority (IACA) certificate. You must generate base64-encoded values for `KeyUsage`, `IssuerAlternativeName`, and `CRLDistributionPoint`, and pass them through `CustomExtensions`.

> **Note**
>
> The IACA link certificate establishes a trust path from the old IACA root certificate to the new IACA root certificate. The issuing authority can generate and distribute an IACA link certificate during the IACA re-key process. You cannot issue an IACA link certificate by using an IACA root certificate with `pathLen=0` set.

The example calls the following AWS Private CA API actions:

- CreateCertificateAuthority
- GetCertificateAuthorityCsr
- IssueCertificate
- GetCertificate
- ImportCertificateAuthorityCertificate

```
package com.amazonaws.samples.mdl;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
```

```
import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CustomExtension;
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.Objects;

import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrRequest;
import com.amazonaws.services.acmpca.model.GetCertificateAuthorityCsrResult;
import com.amazonaws.services.acmpca.model.GetCertificateRequest;
import com.amazonaws.services.acmpca.model.GetCertificateResult;
import
 com.amazonaws.services.acmpca.model.ImportCertificateAuthorityCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.CertificateMismatchException;
import com.amazonaws.services.acmpca.model.ConcurrentModificationException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidPolicyException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.MalformedCertificateException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;
import com.amazonaws.services.acmpca.model.RequestFailedException;
import com.amazonaws.services.acmpca.model.RequestInProgressException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.AWSACMPCAException;
```

```
import com.amazonaws.waiters.Waiter;
import com.amazonaws.waiters.WaiterParameters;
import com.amazonaws.waiters.WaiterTimedOutException;
import com.amazonaws.waiters.WaiterUnrecoverableException;

import org.bouncycastle.asn1.x509.GeneralNames;
import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.CRLDistPoint;
import org.bouncycastle.asn1.x509.DistributionPoint;
import org.bouncycastle.asn1.x509.DistributionPointName;
import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

import lombok.SneakyThrows;

public class IssuingAuthorityCertificateAuthorityActivation {
    public static void main(String[] args) throws Exception {
        // Define the endpoint region for your sample.
        String endpointRegion = null;  // Substitute your region here, e.g. "ap-
southeast-2"
        if (endpointRegion == null) throw new Exception("Region cannot be null");

        // Define a CA subject.
        ASN1Subject subject = new ASN1Subject()
                .withCountry("US") // mDL spec requires ISO 3166-1-alpha-2 country code
 e.g. "US"
                .withCommonName("mDL Test IACA");

        // Define the CA configuration.
        CertificateAuthorityConfiguration configCA = new
 CertificateAuthorityConfiguration()
                .withKeyAlgorithm(KeyAlgorithm.EC_prime256v1)
                .withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA)
                .withSubject(subject);

        // Define a certificate authority type
        CertificateAuthorityType CAType = CertificateAuthorityType.ROOT;

        // Execute core code samples for Root CA activation in sequence
        AWSACMPCA client = buildClient(endpointRegion);
        String rootCAArn = createCertificateAuthority(configCA, CAType, client);
        String csr = getCertificateAuthorityCsr(rootCAArn, client);
        String rootCertificateArn = issueCertificate(rootCAArn, csr, client);
```

```
        String rootCertificate = getCertificate(rootCertificateArn, rootCAArn, client);
        importCertificateAuthorityCertificate(rootCertificate, rootCAArn, client);
    }

    private static AWSACMPCA buildClient(String endpointRegion) {
        // Get your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from disk",
e);
        }

        // Create a client that you can use to make requests.
        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
                .withRegion(endpointRegion)
                .withCredentials(new AWSStaticCredentialsProvider(credentials))
                .build();

        return client;
    }

    private static String createCertificateAuthority(CertificateAuthorityConfiguration
configCA, CertificateAuthorityType CAtype, AWSACMPCA client) {
        // Create the request object.
        CreateCertificateAuthorityRequest createCARequest = new
CreateCertificateAuthorityRequest()
                .withCertificateAuthorityConfiguration(configCA)
                .withIdempotencyToken("123987")
                .withCertificateAuthorityType(CAtype);

        // Create the private CA.
        CreateCertificateAuthorityResult createCAResult = null;
        try {
            createCAResult = client.createCertificateAuthority(createCARequest);
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (InvalidPolicyException ex) {
            throw ex;
        } catch (LimitExceededException ex) {
            throw ex;
        }
```

```
        // Get the ARN of the private CA.
        String rootCAArn = createCAResult.getCertificateAuthorityArn();
        System.out.println("Issuing Authority Certificate Authority (IACA) Arn: " +
rootCAArn);

        return rootCAArn;
    }

    private static String getCertificateAuthorityCsr(String rootCAArn, AWSACMPCA
client) {

        // Create the CSR request object and set the CA ARN.
        GetCertificateAuthorityCsrRequest csrRequest = new
GetCertificateAuthorityCsrRequest()
                .withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the CSR file.
        Waiter<GetCertificateAuthorityCsrRequest> getCSRWaiter =
client.waiters().certificateAuthorityCSRCreated();
        try {
            getCSRWaiter.run(new WaiterParameters<>(csrRequest));
        } catch (WaiterUnrecoverableException e) {
            // Explicit short circuit when the recourse transitions into
            // an undesired state.
        } catch (WaiterTimedOutException e) {
            // Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            // Unexpected service exception.
        }

        // Get the CSR.
        GetCertificateAuthorityCsrResult csrResult = null;
        try {
            csrResult = client.getCertificateAuthorityCsr(csrRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }
```

```
        // Get and display the CSR;
        String csr = csrResult.getCsr();
        System.out.println("CSR:");
        System.out.println(csr);

        return csr;
    }

    @SneakyThrows
    private static String issueCertificate(String rootCAArn, String csr, AWSACMPCA
 client) {
        IssueCertificateRequest issueRequest = new IssueCertificateRequest()
                .withCertificateAuthorityArn(rootCAArn)
                .withTemplateArn("arn:aws:acm-pca:::template/
BlankRootCACertificate_PathLen0_APIPassthrough/V1")
                .withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA)
                .withIdempotencyToken("1234");

        // Set the CSR.
        ByteBuffer csrByteBuffer = stringToByteBuffer(csr);
        issueRequest.setCsr(csrByteBuffer);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity()
                .withValue(3650L)
                .withType("DAYS");
        issueRequest.setValidity(validity);

        // Generate base64 encoded extension value for KeyUsage
        KeyUsage keyUsage = new KeyUsage(X509KeyUsage.keyCertSign +
 X509KeyUsage.cRLSign);
        byte[] kuBytes = keyUsage.getEncoded();
        String base64EncodedKUValue = Base64.getEncoder().encodeToString(kuBytes);

        CustomExtension keyUsageCustomExtension = new CustomExtension()
                .withObjectIdentifier("2.5.29.15") // KeyUsage Extension OID
                .withValue(base64EncodedKUValue)
                .withCritical(true);

        // Generate base64 encoded extension value for IssuerAlternativeName
        GeneralNames issuerAlternativeName = new GeneralNames(new
 GeneralName(GeneralName.uniformResourceIdentifier, "https://issuer-alternative-
name.com"));
```

```
        String base64EncodedIANValue =
Base64.getEncoder().encodeToString(issuerAlternativeName.getEncoded());

        CustomExtension ianCustomExtension = new CustomExtension()
                .withValue(base64EncodedIANValue)
                .withObjectIdentifier("2.5.29.18"); // IssuerAlternativeName Extension
OID

        // Generate base64 encoded extension value for CRLDistributionPoint
        CRLDistPoint crlDistPoint = new CRLDistPoint(new DistributionPoint[]{new
DistributionPoint(new DistributionPointName(
                new GeneralNames(new GeneralName(GeneralName.uniformResourceIdentifier,
"dummycrl.crl"))), null, null)});
        String base64EncodedCDPValue =
Base64.getEncoder().encodeToString(crlDistPoint.getEncoded());

        CustomExtension cdpCustomExtension = new CustomExtension()
                .withValue(base64EncodedCDPValue)
                .withObjectIdentifier("2.5.29.31"); // CRLDistributionPoint Extension
OID

        // Add custom extension to api-passthrough
        Extensions extensions = new Extensions()
                .withCustomExtensions(Arrays.asList(keyUsageCustomExtension,
ianCustomExtension, cdpCustomExtension));
        ApiPassthrough apiPassthrough = new ApiPassthrough()
                .withExtensions(extensions);
        issueRequest.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult issueResult = null;
        try {
            issueResult = client.issueCertificate(issueRequest);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
```

```
            throw ex;
        }

        // Get and display the certificate ARN.
        String rootCertificateArn = issueResult.getCertificateArn();
        System.out.println("mDL IACA Certificate Arn: " + rootCertificateArn);

        return rootCertificateArn;
    }

    private static String getCertificate(String rootCertificateArn, String rootCAArn,
AWSACMPCA client) {

        // Create a request object.
        GetCertificateRequest certificateRequest = new GetCertificateRequest()
                .withCertificateArn(rootCertificateArn)
                .withCertificateAuthorityArn(rootCAArn);

        // Create waiter to wait on successful creation of the certificate file.
        Waiter<GetCertificateRequest> getCertificateWaiter =
client.waiters().certificateIssued();
        try {
            getCertificateWaiter.run(new WaiterParameters<>(certificateRequest));
        } catch (WaiterUnrecoverableException e) {
            // Explicit short circuit when the recourse transitions into
            // an undesired state.
        } catch (WaiterTimedOutException e) {
            // Failed to transition into desired state even after polling.
        } catch (AWSACMPCAException e) {
            // Unexpected service exception.
        }

        // Get the certificate and certificate chain.
        GetCertificateResult certificateResult = null;
        try {
            certificateResult = client.getCertificate(certificateRequest);
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
```

```
        } catch (InvalidStateException ex) {
            throw ex;
        }

        // Get the certificate and certificate chain and display the result.
        String rootCertificate = certificateResult.getCertificate();
        System.out.println(rootCertificate);

        return rootCertificate;
    }

    private static void importCertificateAuthorityCertificate(String rootCertificate,
String rootCAArn, AWSACMPCA client) {

        // Create the request object and set the signed certificate, chain and CA ARN.
        ImportCertificateAuthorityCertificateRequest importRequest =
                new ImportCertificateAuthorityCertificateRequest()
                        .withCertificateChain(null)
                        .withCertificateAuthorityArn(rootCAArn);

        ByteBuffer certByteBuffer = stringToByteBuffer(rootCertificate);
        importRequest.setCertificate(certByteBuffer);

        // Import the certificate.
        try {
            client.importCertificateAuthorityCertificate(importRequest);
        } catch (CertificateMismatchException ex) {
            throw ex;
        } catch (MalformedCertificateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (RequestInProgressException ex) {
            throw ex;
        } catch (ConcurrentModificationException ex) {
            throw ex;
        } catch (RequestFailedException ex) {
            throw ex;
        }

        System.out.println("Root CA certificate successfully imported.");
        System.out.println("Root CA activated successfully.");
```

```
    }

    private static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }
}
```

Create a document signer certificate

This Java sample shows how to use the BlankEndEntityCertificate_APIPassthrough/V1 template to create a ISO/IEC mDL standard-compliant document signer certificate. You must generate base64-encoded values for `KeyUsage`, `IssuerAlternativeName`, and `CRLDistributionPoint` and pass them through `CustomExtensions`.

The example calls the following AWS Private CA API action:

- IssueCertificate

```
package com.amazonaws.samples.mdl;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.auth.AWSStaticCredentialsProvider;

import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.Arrays;
import java.util.Base64;
import java.util.Objects;

import com.amazonaws.services.acmpca.AWSACMPCA;
import com.amazonaws.services.acmpca.AWSACMPCAClientBuilder;

import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.ApiPassthrough;
import com.amazonaws.services.acmpca.model.ExtendedKeyUsage;
import com.amazonaws.services.acmpca.model.CustomExtension;
```

```
import com.amazonaws.services.acmpca.model.Extensions;
import com.amazonaws.services.acmpca.model.IssueCertificateRequest;
import com.amazonaws.services.acmpca.model.IssueCertificateResult;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.Validity;

import com.amazonaws.AmazonClientException;
import com.amazonaws.services.acmpca.model.LimitExceededException;
import com.amazonaws.services.acmpca.model.ResourceNotFoundException;
import com.amazonaws.services.acmpca.model.InvalidStateException;
import com.amazonaws.services.acmpca.model.InvalidArnException;
import com.amazonaws.services.acmpca.model.InvalidArgsException;
import com.amazonaws.services.acmpca.model.MalformedCSRException;

import org.bouncycastle.asn1.x509.GeneralNames;
import org.bouncycastle.asn1.x509.GeneralName;
import org.bouncycastle.asn1.x509.CRLDistPoint;
import org.bouncycastle.asn1.x509.DistributionPoint;
import org.bouncycastle.asn1.x509.DistributionPointName;
import org.bouncycastle.asn1.x509.KeyUsage;
import org.bouncycastle.jce.X509KeyUsage;

public class IssueDocumentSignerCertificate {
    public static ByteBuffer stringToByteBuffer(final String string) {
        if (Objects.isNull(string)) {
            return null;
        }
        byte[] bytes = string.getBytes(StandardCharsets.UTF_8);
        return ByteBuffer.wrap(bytes);
    }

    public static void main(String[] args) throws Exception {

        // Get your credentials from the C:\Users\name\.aws\credentials file
        // in Windows or the .aws/credentials file in Linux.
        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException("Cannot load your credentials from disk",
 e);
        }

        // Create a client that you can use to make requests.
```

```
        String endpointRegion = null;  // Substitute your region here, e.g. "ap-
southeast-2"
        if (endpointRegion == null) throw new Exception("Region cannot be null");

        AWSACMPCA client = AWSACMPCAClientBuilder.standard()
                .withRegion(endpointRegion)
                .withCredentials(new AWSStaticCredentialsProvider(credentials))
                .build();

        // Create a certificate request:
        String caArn = null;
        if (caArn == null) throw new Exception("Certificate authority ARN cannot be
 null");

        IssueCertificateRequest req = new IssueCertificateRequest()
                .withCertificateAuthorityArn(caArn)
                .withTemplateArn("arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APIPassthrough/V1")
                .withSigningAlgorithm(SigningAlgorithm.SHA256WITHECDSA)
                .withIdempotencyToken("1234");

        // Specify the certificate signing request (CSR) for the certificate to be
 signed and issued.
        // Format: "-----BEGIN CERTIFICATE REQUEST-----\n" +
        //         "base64-encoded certificate\n" +
        //         "-----END CERTIFICATE REQUEST-----\n";
        String strCSR = null;
        if (strCSR == null) throw new Exception("CSR string cannot be null");

        ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
        req.setCsr(csrByteBuffer);

        // Set the validity period for the certificate to be issued.
        Validity validity = new Validity()
                .withValue(365L)
                .withType("DAYS");
        req.setValidity(validity);

        // Define a cert subject.
        ASN1Subject subject = new ASN1Subject()
                .withCountry("US") // mDL spec requires ISO 3166-1-alpha-2 country code
 e.g. "US"
                .withCommonName("mDL Test DS");
```

```
        ApiPassthrough apiPassthrough = new ApiPassthrough()
                .withSubject(subject);

        // Generate base64 encoded extension value for KeyUsage
        KeyUsage keyUsage = new KeyUsage(X509KeyUsage.digitalSignature);
        byte[] kuBytes = keyUsage.getEncoded();
        String base64EncodedKUValue = Base64.getEncoder().encodeToString(kuBytes);

        CustomExtension customKeyUsageExtension = new CustomExtension()
                .withObjectIdentifier("2.5.29.15") // KeyUsage Extension OID
                .withValue(base64EncodedKUValue)
                .withCritical(true);

        // Generate base64 encoded extension value for IssuerAlternativeName
        GeneralNames issuerAlternativeName = new GeneralNames(new
 GeneralName(GeneralName.uniformResourceIdentifier, "https://issuer-alternative-
name.com"));
        String base64EncodedIANValue =
 Base64.getEncoder().encodeToString(issuerAlternativeName.getEncoded());

        CustomExtension ianCustomExtension = new CustomExtension()
                .withValue(base64EncodedIANValue)
                .withObjectIdentifier("2.5.29.18"); // IssuerAlternativeName Extension
 OID

        // Generate base64 encoded extension value for CRLDistributionPoint
        CRLDistPoint crlDistPoint = new CRLDistPoint(new DistributionPoint[]{new
 DistributionPoint(new DistributionPointName(
                new GeneralNames(new GeneralName(GeneralName.uniformResourceIdentifier,
 "dummycrl.crl"))), null, null)});
        String base64EncodedCDPValue =
 Base64.getEncoder().encodeToString(crlDistPoint.getEncoded());

        CustomExtension cdpCustomExtension = new CustomExtension()
                .withValue(base64EncodedCDPValue)
                .withObjectIdentifier("2.5.29.31"); // CRLDistributionPoint Extension
 OID

        // Generate EKU
        ExtendedKeyUsage eku = new ExtendedKeyUsage()
                .withExtendedKeyUsageObjectIdentifier("1.0.18013.5.1.2"); // EKU value
 reserved for mDL DS
```

```
        // Set KeyUsage, ExtendedKeyUsage, IssuerAlternativeName, CRL Distribution
 Point extensions to api-passthrough
        Extensions extensions = new Extensions()
                .withCustomExtensions(Arrays.asList(customKeyUsageExtension,
 ianCustomExtension, cdpCustomExtension))
                .withExtendedKeyUsage(Arrays.asList(eku));
        apiPassthrough.setExtensions(extensions);
        req.setApiPassthrough(apiPassthrough);

        // Issue the certificate.
        IssueCertificateResult result = null;
        try {
            result = client.issueCertificate(req);
        } catch (LimitExceededException ex) {
            throw ex;
        } catch (ResourceNotFoundException ex) {
            throw ex;
        } catch (InvalidStateException ex) {
            throw ex;
        } catch (InvalidArnException ex) {
            throw ex;
        } catch (InvalidArgsException ex) {
            throw ex;
        } catch (MalformedCSRException ex) {
            throw ex;
        }

        // Get and display the certificate ARN.
        String arn = result.getCertificateArn();
        System.out.println("mDL DS Certificate Arn: " + arn);
    }
}
```

Architect your solution for AWS Private CA

AWS Private CA gives you complete, cloud-based control over your organization's private PKI (public key infrastructure), extending from a root certificate authority (CA), through subordinate CAs, to end-entity certificates. Thorough planning is essential for a PKI that is secure, maintainable, extensible, and suited to your organization's needs. This section provides guidance on designing a CA hierarchy, managing your private CA and private end-entity certificate lifecycles, and applying best practices for security.

This section describes how to prepare AWS Private CA for use before you create a private certificate authority (CA). It also explains the option to add revocation support through Online Certificate Status Protocol (OCSP) or a certificate revocation list (CRL).

In addition, you should determine whether your organization prefers to host its private root CA credentials on premises rather than with AWS. In that case, you need to set up and secure a self-managed private PKI before using AWS Private CA. In this scenario, you then create a subordinate CA in AWS Private CA backed by a parent CA outside of AWS Private CA. For more information, see Installing a subordinate CA certificate signed by an external parent CA.

Topics

- Design a CA hierarchy
- Manage the private CA lifecycle
- Plan your AWS Private CA certificate revocation method
- Understand AWS Private CA CA modes
- Plan for resilience in AWS Private CA

Design a CA hierarchy

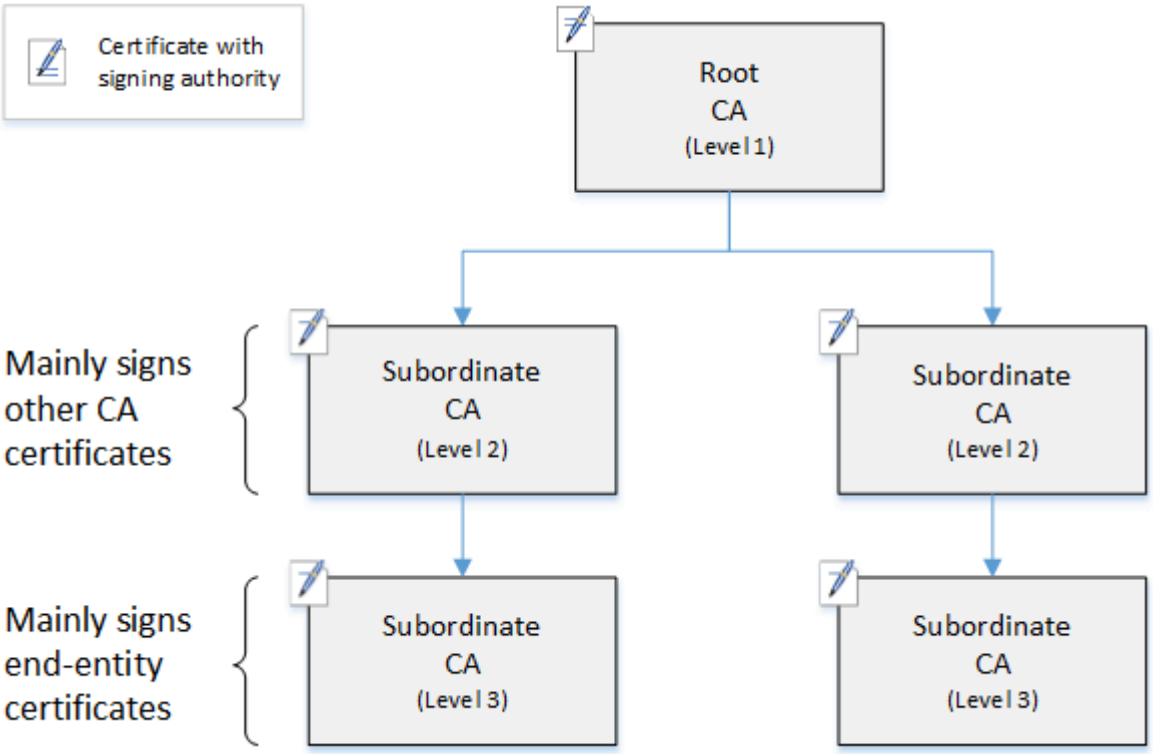
With AWS Private CA, you can create a hierarchy of certificate authorities with up to five levels. The root CA, at the top of a hierarchy tree, can have any number of branches. The root CA can have as many as four levels of subordinate CAs on each branch. You can also create multiple hierarchies, each with its own root.

A well-designed CA hierarchy offers the following benefits:

- Granular security controls appropriate to each CA

- Division of administrative tasks for better load balancing and security
- Use of CAs with limited, revocable trust for daily operations
- Validity periods and certificate path limits

The following diagram illustrates a simple, three-level CA hierarchy.



Each CA in the tree is backed by an X.509 v3 certificate with signing authority (symbolized by the pen-and-paper icon). This means that as CAs, they can sign other certificates subordinate to them. When a CA signs a lower-level CA's certificate, it confers limited, revocable authority on the signed certificate. The root CA in level 1 signs high-level subordinate CA certificates in level 2. These CAs, in turn, sign certificates for CAs in level 3 that are used by PKI (public key infrastructure) administrators who manage end-entity certificates.

Security in a CA hierarchy should be configured to be strongest at the top of the tree. This arrangement protects the root CA certificate and its private key. The root CA anchors trust for all of the subordinate CAs and the end-entity certificates below it. While localized damage can result from the compromise of an end-entity certificate, compromise of the root destroys trust in the entire PKI. Root and high-level subordinate CAs are used only infrequently (usually to sign other CA certificates). Consequently, they are tightly controlled and audited to ensure a lower risk of compromise. At the lower levels of the hierarchy, security is less restrictive. This approach allows

the routine administrative tasks of issuing and revoking end-entity certificates for users, computer hosts, and software services.

> **Note**
>
> Using a root CA to sign a subordinate certificate is a rare event that occurs in only a handful of circumstances:
>
> - When the PKI is created
> - When a high-level certificate authority needs to be replaced
> - When a certificate revocation list (CRL) or Online Certificate Status Protocol (OCSP) responder needs to be configured
>
> Root and other high-level CAs require highly secure operational processes and access-control protocols.

Topics

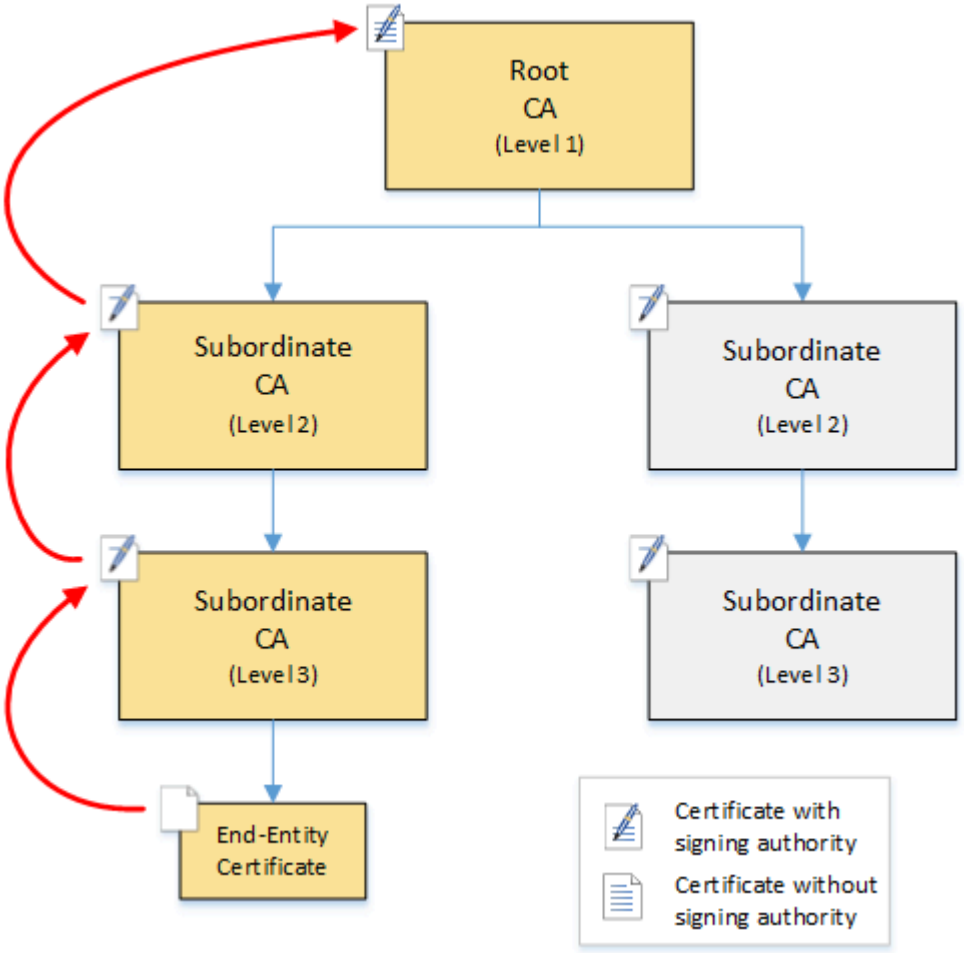
- Validate end-entity certificates
- Plan the structure of a CA hierarchy
- Set length constraints on the certification path

Validate end-entity certificates

End-entity certificates derive their trust from a certification path leading back through the subordinate CAs to a root CA. When a web browser or other client is presented with an end-entity certificate, it attempts to construct a chain of trust. For example, it may check to see that the certificate's *issuer distinguished name* and *subject distinguished name* match with the corresponding fields of the issuing CA certificate. Matching would continue at each successive level up the hierarchy until the client reaches a trusted root that is contained in its trust store.

The trust store is a library of trusted CAs that the browser or operating system contains. For a private PKI, your organization's IT must ensure that each browser or system has previously added the private root CA to its trust store. Otherwise, the certification path cannot be validated, resulting in client errors.

The next diagram shows the validation path that a browser follows when presented with an end-entity X.509 certificate. Note that the end-entity certificate lacks signing authority and serves only to authenticate the entity that owns it.



The browser inspects the end-entity certificate. The browser finds that the certificate offers a signature from subordinate CA (level 3) as its trust credential. The certificates for the subordinate CAs must be included in the same PEM file. Alternatively, they can also be in a separate file that contains the certificates that make up the trust chain. Upon finding these, the browser checks the certificate of subordinate CA (level 3) and finds that it offers a signature from subordinate CA (level 2). In turn, subordinate CA (level 2) offers a signature from root CA (level 1) as its trust credential. If the browser finds a copy of the private root CA certificate preinstalled in its trust store, it validates the end-entity certificate as trusted.

Typically, the browser also checks each certificate against a certificate revocation list (CRL). An expired, revoked, or misconfigured certificate is rejected and validation fails.

Plan the structure of a CA hierarchy

In general, your CA hierarchy should reflect the structure of your organization. Aim for a *path length* (that is, number of CA levels) no greater than necessary to delegate administrative and security roles. Adding a CA to the hierarchy means increasing the number of certificates in the certification path, which increases validation time. Keeping the path length to a minimum also reduces the number of certificates sent from the server to the client when validating an end-entity certificate.

In theory, a root CA, which has no pathLenConstraint parameter, can authorize unlimited levels of subordinate CAs. A subordinate CA can have as many child subordinate CAs as are allowed by its internal configuration. AWS Private CA managed hierarchies support CA certification paths up to five levels deep.

Well designed CA structures have several benefits:

- Separate administrative controls for different organizational units
- The ability to delegate access to subordinate CAs
- A hierarchical structure that protects higher-level CAs with additional security controls

Two common CA structures accomplish all of this:

- **Two CA levels: root CA and subordinate CA**

  This is the simplest CA structure that allows separate administration, control, and security policies for the root CA and a subordinate CA. You can maintain restrictive controls and policies for your root CA while allowing more permissive access for the subordinate CA. The latter is used for bulk issuance of end-entity certificates.

- **Three CA levels: root CA and two layers of subordinate CA**

  Similar to the above, this structure adds an additional CA layer to further separate the root CA from low-level CA operations. The middle CA layer is only used to sign subordinate CAs that carry out the issuance of end-entity certificates.

Less common CA structures include the following:

- **Four or more CA levels**

Though less common than three-level hierarchies, CA hierarchies with four or more levels are possible and may be required to allow administrative delegation.

- **One CA level: root CA only**

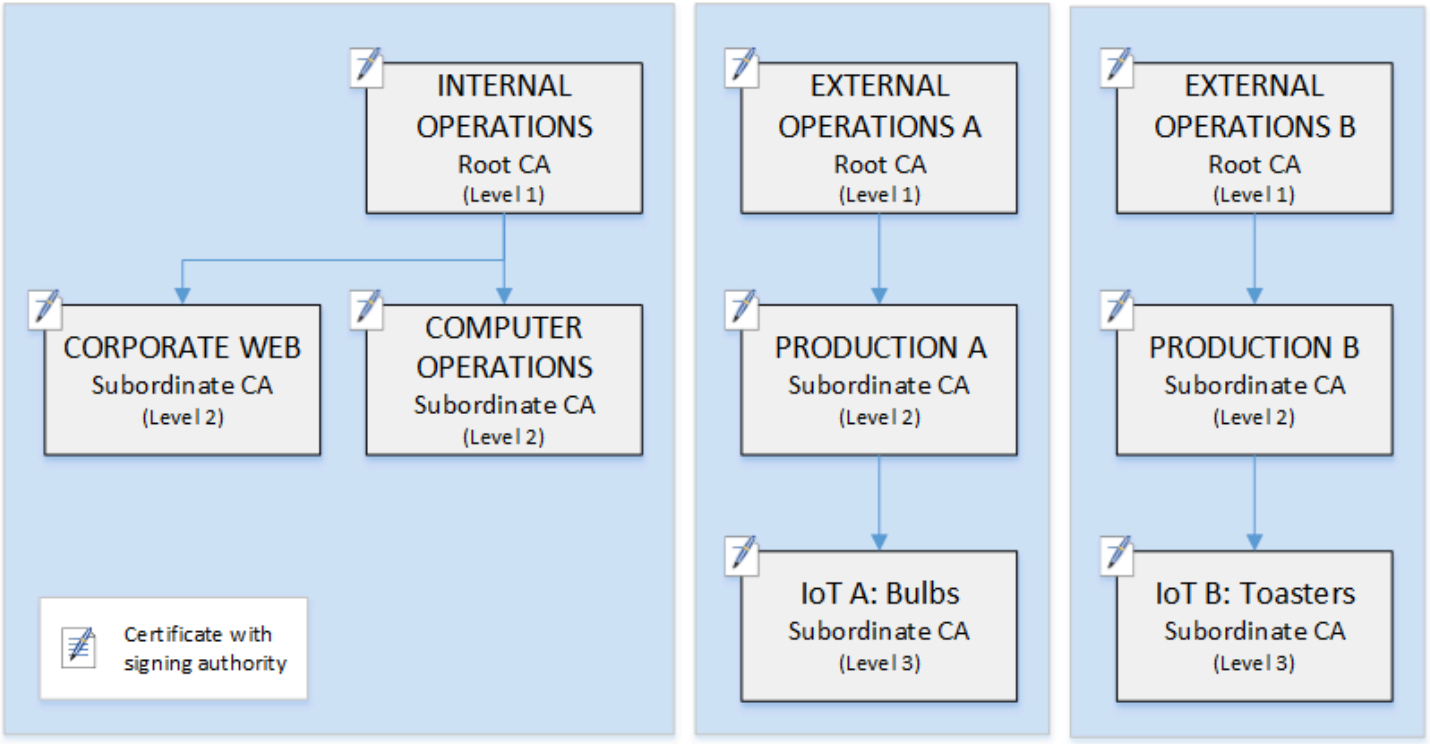
  This structure is commonly used for development and testing when a full chain of trust is not required. Used in production, it is atypical. Moreover, it violates the best practice of maintaining separate security policies for the root CA and the CAs that issue end-entity certificates.

  However, if you are already issuing certificates directly from a root CA, you can migrate to AWS Private CA. Doing so provides security and control advantages over using a root CA managed with OpenSSL or other software.

Example of a private PKI for a manufacturer

In this example, a hypothetical technology company manufactures two Internet of Things (IoT) products, a smart light bulb and a smart toaster. During production, each device is issued an end-entity certificate so it can communicate securely over the internet with the manufacturer. The company's PKI also secures its computer infrastructure, including the internal website and various self-hosted computer services that run finance and business operations.

Consequently, the CA hierarchy closely models these administrative and operational aspects of the business.



This hierarchy contains three roots, one for Internal Operations and two for External Operations (one root CA for each product line). It also illustrates multiple certification path length, with two levels of CA for Internal Operations and three levels for External Operations.

The use of separated root CAs and additional subordinate CA layers on the External Operations side is a design decision serving business and security needs. With multiple CA trees, the PKI is future-proofed against corporate reorganizations, divestitures, or acquisitions. When changes occur, an entire root CA hierarchy can move cleanly with the division it secures. And with two levels of subordinate CA, the roots CAs have a high level of isolation from the level 3 CAs that are responsible for bulk-signing the certificates for thousands or millions of manufactured items.

On the internal side, corporate web and internal computer operations complete a two-level hierarchy. These levels allow web administrators and operations engineers to manage certificate issuance independently for their own work domains. The compartmentalization of PKI into distinct functional domains is a security best practice and protects each from a compromise that might affect the other. Web administrators issue end-entity certificates for use by web browsers throughout the company, authenticating and encrypting communications on the internal website. Operations engineers issue end-entity certificates that authenticate data center hosts and computer services to one another. This system helps keep sensitive data secure by encrypting it on the LAN.

Set length constraints on the certification path

The structure of a CA hierarchy is defined and enforced by the *basics constraints* extension that each certificate contains. The extension defines two constraints:

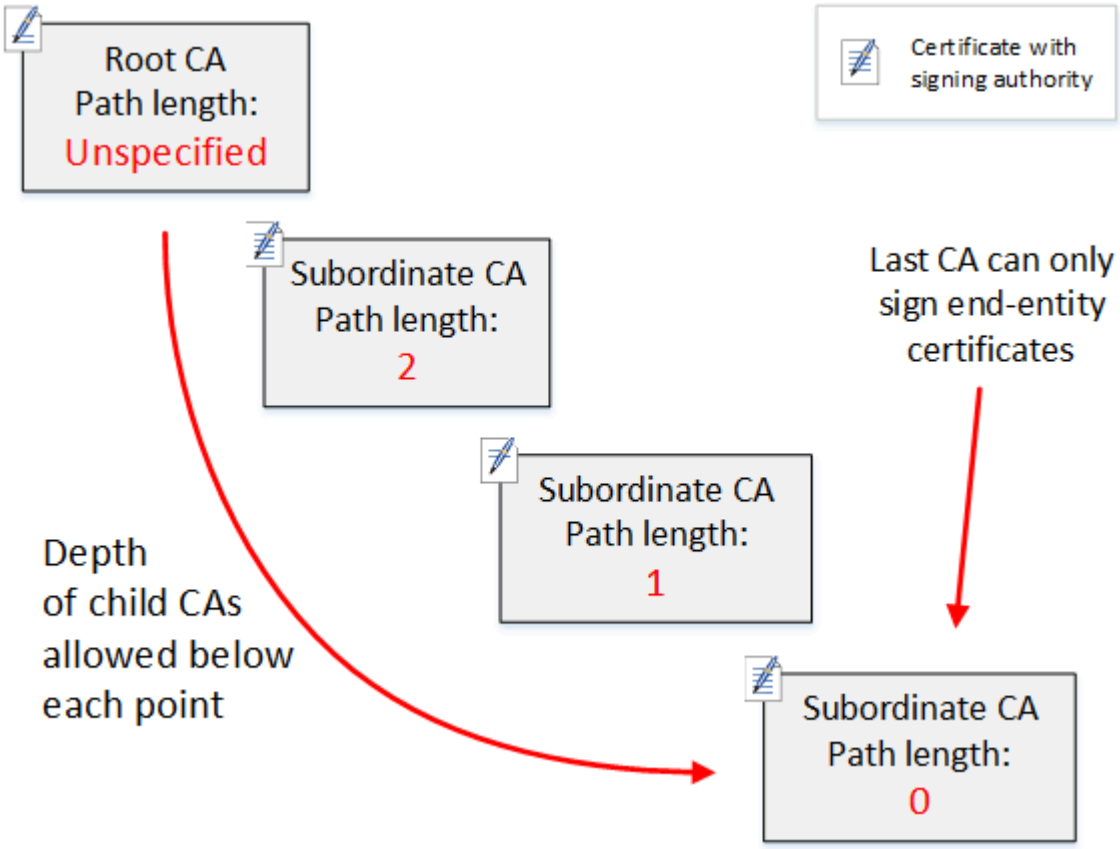
- `cA` – Whether the certificate defines a CA. If this value is *false* (the default), then the certificate is an end-entity certificate.
- `pathLenConstraint` – The maximum number of lower-level subordinate CAs that can exist in a valid chain of trust. The end-entity certificate is not counted because it is not a CA certificate.

A root CA certificate needs maximum flexibility and does not include a path length constraint. This allows the root to define a certification path of any length.

> **Note**
>
> AWS Private CA limits the certification path to five levels.

Subordinate CAs have `pathLenConstraint` values equal to or greater than zero, depending on location in the hierarchy placement and desired features. For example, in a hierarchy with three CAs, no path constraint is specified for the root CA. The first subordinate CA has a path length of 1 and can therefore sign child CAs. Each of these child CAs must necessarily have a `pathLenConstraint` value of zero. This means that they can sign end-entity certificates but cannot issue additional CA certificates. Limiting the power to create new CAs is an important security control.

The following diagram illustrates this propagation of limited authority down the hierarchy.



In this four-level hierarchy, the root is unconstrained (as always). But the first subordinate CA has a `pathLenConstraint` value of 2, which limits its child CAs from going more than two levels deeper. Consequently, for a valid certification path, the constraint value must decrement to zero in the next two levels. If a web browser encounters an end-entity certificate from this branch that has a path length greater than four, validation fails. Such a certificate could be the result of an accidentally created CA, a misconfigured CA, or a unauthorized issuance.

Manage path length with templates

AWS Private CA provides templates for issuing root, subordinate, and end-entity certificates. These templates encapsulate best practices for the basic constraints values, including path length. The templates include the following:

- RootCACertificate/V1
- SubordinateCACertificate_PathLen0/V1
- SubordinateCACertificate_PathLen1/V1
- SubordinateCACertificate_PathLen2/V1
- SubordinateCACertificate_PathLen3/V1

- EndEntityCertificate/V1

The `IssueCertificate` API will return an error if you attempt to create a CA with a path length greater than or equal to the path length of its issuing CA certificate.

For more information about certificate templates, see Use AWS Private CA certificate templates.

Automate CA hierarchy setup with AWS CloudFormation

When you have settled on a design for your CA hierarchy, you can test it and put it into production using a AWS CloudFormation template. For an example of such a template, see Declaring a Private CA Hierarchy in the *CloudFormation User Guide*.

Manage the private CA lifecycle

CA certificates have a fixed lifetime, or validity period. When a CA certificate expires, all of the certificates issued directly or indirectly by subordinate CAs below it in the CA hierarchy become invalid. You can avoid CA certificate expiration by planning in advance.

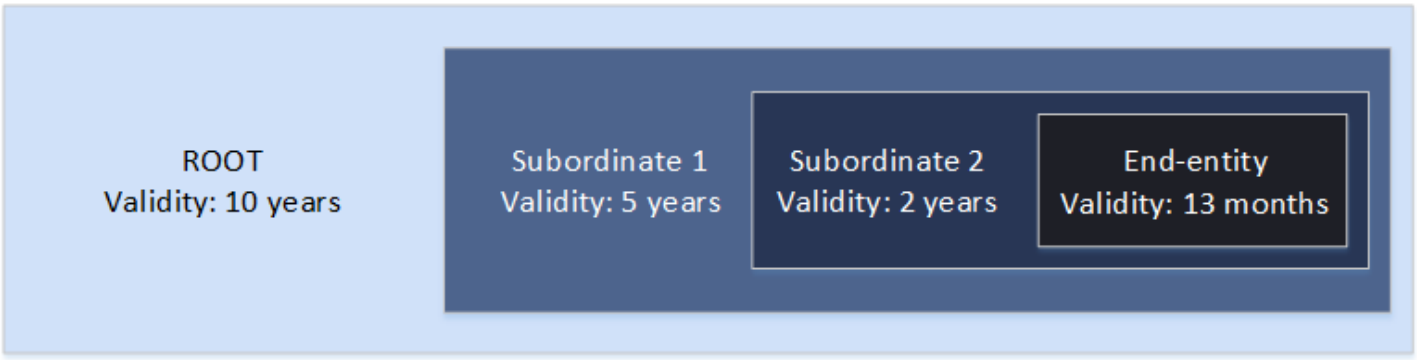
Choose validity periods

The validity period of an X.509 certificate is a required basic certificate field. It determines the time-range during which the issuing CA certifies that the certificate can be trusted, barring revocation. (A root certificate, being self-signed, certifies its own validity period.)

AWS Private CA and AWS Certificate Manager assist with the configuration of certificate validity periods subject to the following constraints:

- A certificate managed by AWS Private CA must have a validity period shorter than or equal to the validity period of the CA that issued it. In other words, child CAs and end-entity certificates cannot outlive their parent certificates. Attempting to use the `IssueCertificate` API to issue a CA certificate with a validity period greater than or equal to the parent's CA fails.
- Certificates issued and managed by AWS Certificate Manager (those for which ACM generates the private key) have a validity period of 13 months (395 days). ACM manages the renewal process for these certificates. If you use AWS Private CA to issue certificates directly, you can choose any validity period.

The following diagram shows a typical configuration of nested validity periods. The root certificate is the most long-lived; end-entity certificates are relatively short-lived; and subordinate CAs range between these extremes.



When you plan your CA hierarchy, determine the optimal lifetime for your CA certificates. Work backwards from the desired lifetime of the end-entity certificates that you want to issue.

End-entity certificates

End-entity certificates should have a validity period appropriate to the use case. A short lifetime minimizes the exposure of a certificate in the event that its private key is lost or stolen. However, short lifetimes mean frequent renewals. Failure to renew an expiring certificate can result in downtime.

The distributed use of end-entity certificates can also present logistical problems if there is a security breach. Your planning should account for renewal and distribution certificates, revocation of compromised certificates, and how quickly revocations propagate to clients that rely on the certificates.

The default validity period for an end-entity certificate issued through ACM is 13 months (395 days). In AWS Private CA, you can use the `IssueCertificate` API to apply any validity period so long as it is less than that of the issuing CA.

Subordinate CA Certificates

Subordinate CA certificates should have significantly longer validity periods than the certificates they issue. A good range for a CA certificate's validity is two to five times the period of any child CA certificate or end-entity certificate it issues. For example, assume you have a two-level CA hierarchy (root CA and one subordinate CA). If you want to issue end-entity certificates with a one-year lifetime, you could configure the subordinate issuing CA lifetime to be three years. This is the default validity period for a subordinate CA certificate in AWS Private CA. Subordinate CA certificates can be changed without replacing the root CA certificate.

Root Certificates

Changes to a root CA certificate affect the entire PKI (public key infrastructure) and require you to update all the dependent client operating system and browser trust stores. To minimize operational impact, you should choose a long validity period for the root certificate. The AWS Private CA default for root certificates is ten years.

Manage CA succession

You have two ways to manage CA succession: Replace the old CA, or reissue the CA with a new validity period.

Replace an old CA

To replace an old CA, you create a new CA and chain it to the same parent CA. Afterward, you issue certificates from the new CA.

Certificates issued from the new CA have a new CA chain. Once the new CA is established, you can disable the old CA to prevent it from issuing new certificates. While disabled, the old CA supports revocation for old certificates issued from the CA, and, if configured to do so, it continues to validate certificates by means of OCSP and/or certificate revocation lists (CRLs). When the last certificate issued from the old CA expires, you can delete the old CA. You can generate an audit report for all of the certificates issued from the CA to confirm that all of the certificates issued have expired. If the old CA has subordinate CAs, you must also replace them, because subordinate CAs expire at the same time or before their parent CA. Start by replacing the highest CA in the hierarchy that needs to be replaced. Then create new replacement subordinate CAs at each subsequent lower level.

AWS recommends that you include a CA generation identifier in the names of CAs as needed. For example, assume that you name the first generation CA “Corporate Root CA." When you create the second generation CA, name it “Corporate Root CA G2." This simple naming convention can help avoid confusion when both CAs are unexpired.

This method of CA succession is preferred because it rotates the private key of the CA. Rotating the private key is a best practice for CA keys. The frequency of rotation should be proportional to the frequency of key use: CAs that issue more certificates should be rotated more frequently.

> **Note**
>
> Private certificates issued through ACM cannot be renewed if you replace the CA. If you use ACM for issuance and renewal, you must re-issue the CA certificate to extend the lifetime of the CA.

Reissue an old CA

When a CA nears expiration, an alternative method of extending its life is to reissue the CA certificate with a new expiration date. Reissuance leaves all of the CA metadata in place and preserves the existing private and public keys. In this scenario, the existing certificate chain and unexpired end-entity certificates issued by the CA remain valid until they expire. New certificate issuance can also continue without interruption. To update a CA with a reissued certificate, follow the usual installation procedures described in Installing the CA certificate.

> **Note**
>
> We recommend replacing an expiring CA rather than reissuing its certificate because of the security advantages gained by rotating to a new key pair.

Revoke a CA

You revoke a CA by revoking its underlying certificate. This also effectively revokes all of the certificates issued by the CA. Revocation information is distributed to clients by means of OCSP or a CRL. You should revoke a CA certificate only if you want to revoke all of its issued end-entity and subordinate CA certificates.

Plan your AWS Private CA certificate revocation method

As you plan your private PKI with AWS Private CA, you should consider how to handle situations where you no longer wish endpoints to trust an issued certificate, such as when the private key of an endpoint is exposed. The common approaches to this problem are to use short-lived certificates or to configure certificate revocation. Short-lived certificates expire in such a short period of time, in hours or days, that revocation makes no sense, with the certificate becoming invalid in about the same time it takes to notify an endpoint of revocation. This section describes the revocation options for AWS Private CA customers, including configuration and best practices.

Customers looking for a revocation method can choose Online Certificate Status Protocol (OCSP), certificate revocation lists (CRLs), or both.

> **Note**
>
> If you create your CA without configuring revocation, you can always configure it later. For more information, see Update a private CA in AWS Private Certificate Authority.

- **Online Certificate Status Protocol (OCSP)**

  AWS Private CA provides a fully managed OCSP solution to notify endpoints that certificates have been revoked without the need for customers to operate infrastructure themselves. Customers can enable OCSP on new or existing CAs with a single operation using the AWS Private CA console, the API, the CLI, or through CloudFormation. Whereas CRLs are stored and processed on the endpoint and can become stale, OCSP storage and processing requirements are handled synchronously on the responder backend.

  When you enable OCSP for a CA, AWS Private CA includes the URL of the OCSP responder in the *Authority Information Access* (AIA) extension of each new certificate issued. The extension allows clients such as web browsers to query the responder and determine whether an end-entity or subordinate CA certificate can be trusted. The responder returns a status message that is cryptographically signed to assure its authenticity.

  The AWS Private CA OCSP responder is compliant with RFC 5019.

  OCSP considerations

  - OCSP status messages are signed using the same signing algorithm that the issuing CA was configured to use. CAs created in the AWS Private CA console use the SHA256WITHRSA signature algorithm by default. Other supported algorithms can be found in the CertificateAuthorityConfiguration API documentation.
  - APIPassthrough and CSRPassthrough certificate templates will not work with the AIA extension if the OCSP responder is enabled.
  - The endpoint of the managed OCSP service is accessible on the public internet. Customers who want OCSP but prefer not to have a public endpoint will need to operate their own OCSP infrastructure.

- **Certificate Revocation Lists (CRLs)**

A certificate revocation list (CRL) is a file that contains a list of certificates revoked before their scheduled expiration date. The CRL contains a list of certificates that should no longer be trusted, the reason for revocation, and other relevant information.

When you configure your certificate authority (CA), you can choose whether AWS Private CA creates a complete or partitioned CRL. Your choice determines the maximum number of certificates that the certificate authority can issue and revoke. For more information, see AWS Private CA quotas.

CRL considerations

- Memory and bandwidth considerations: CRLs require more memory than OCSP due to local download and processing requirements. However, CRLs might reduce network bandwidth compared to OCSP by caching revocation lists instead of checking status per connection. For memory-constrained devices, such as certain IoT devices, consider using partitioned CRLs.
- Changing the CRL type: When changing from a complete to partitioned CRL, AWS Private CA creates new partitions as needed and adds the IDP extension to all CRLs, including the original. Changing from partitioned to complete updates only a single CRL and prevents future revocation of certificates associated with previous partitions.

Note

Both OCSP and CRLs exhibit some delay between revocation and the availability of the status change.

- OCSP responses may take up to 60 minutes to reflect the new status when you revoke a certificate. In general, OCSP tends to support faster distribution of revocation information because, unlike CRLs which can be cached by clients for days, OCSP responses are typically not cached by clients.
- A CRL is typically updated approximately 30 minutes after a certificate is revoked. If for any reason a CRL update fails, AWS Private CA makes further attempts every 15 minutes.

General requirements for revocation configurations

The following requirements apply to all revocation configurations.

- A configuration disabling CRLs or OCSP must contain only the `Enabled=False` parameter, and will fail if other parameters such as `CustomCname` or `ExpirationInDays` are included.
- In a CRL configuration, the `S3BucketName` parameter must conform to Amazon Simple Storage Service bucket naming rules.
- A configuration containing a custom Canonical Name (CNAME) parameter for CRLs or OCSP must conform to RFC7230 restrictions on the use of special characters in a CNAME.
- In a CRL or OCSP configuration, the value of a CNAME parameter must not include a protocol prefix such as "http://" or "https://".

Topics

- Set up a CRL for AWS Private CA
- Customize OCSP URL for AWS Private CA

Set up a CRL for AWS Private CA

Before you can configure a certificate revocation list (CRL) as part of the CA creation process, some prior setup may be necessary. This section explains the prerequisites and options that you should understand before creating a CA with a CRL attached.

For information about using Online Certificate Status Protocol (OCSP) as an alternative or a supplement to a CRL, see Certificate revocation options and Customize OCSP URL for AWS Private CA.

Topics

- CRL types
- CRL structure
- Access policies for CRLs in Amazon S3
- Enable S3 Block Public Access (BPA) with CloudFront
- Determining the CRL Distribution Point (CDP) URI

CRL types

- **Complete** - The default setting. AWS Private CA maintains a single, unpartitioned CRL file for all unexpired certificates issued by a CA that have been revoked. Each certificate that AWS Private CA issues is bound to a specific CRL through its CRL distribution point (CDP) extension, as defined in RFC 5280. You can have up to 1 million private certificates for each CA with complete CRL enabled. For more information, see the AWS Private CA quotas.
- **Partitioned** - Compared to complete CRLs, partitioned CRLs dramatically increase the number of certificates your private CA can issue, and saves you from frequently rotating your CAs.

> **Important**
>
> When using partitioned CRLs, you must validate that the CRL's associated issuing distribution point (IDP) URI matches the certificate's CDP URI to ensure the right CRL has been fetched. AWS Private CA marks the IDP extension as critical, which your client must be able to process.

CRL structure

Each CRL is a DER encoded file. To download the file and use OpenSSL to view it, use a command similar to the following:

```
openssl crl -inform DER -in path-to-crl-file -text -noout
```

CRLs have the following format:

```
Certificate Revocation List (CRL):
          Version 2 (0x1)
      Signature Algorithm: sha256WithRSAEncryption
          Issuer: /C=US/ST=WA/L=Seattle/O=Example Company CA/OU=Corporate/
CN=www.example.com
          Last Update: Feb 26 19:28:25 2018 GMT
          Next Update: Feb 26 20:28:25 2019 GMT
          CRL extensions:
              X509v3 Authority Key Identifier:
                  keyid:AA:6E:C1:8A:EC:2F:8F:21:BC:BE:80:3D:C5:65:93:79:99:E7:71:65

              X509v3 CRL Number:
                  1519676905984
```

```
Revoked Certificates:
    Serial Number: E8CBD2BEDB122329F97706BCFEC990F8
        Revocation Date: Feb 26 20:00:36 2018 GMT
        CRL entry extensions:
            X509v3 CRL Reason Code:
                Key Compromise
    Serial Number: F7D7A3FD88B82C6776483467BBF0B38C
        Revocation Date: Jan 30 21:21:31 2018 GMT
        CRL entry extensions:
            X509v3 CRL Reason Code:
                Key Compromise
    Signature Algorithm: sha256WithRSAEncryption
         82:9a:40:76:86:a5:f5:4e:1e:43:e2:ea:83:ac:89:07:49:bf:
         c2:fd:45:7d:15:d0:76:fe:64:ce:7b:3d:bb:4c:a0:6c:4b:4f:
         9e:1d:27:f8:69:5e:d1:93:5b:95:da:78:50:6d:a8:59:bb:6f:
         49:9b:04:fa:38:f2:fc:4c:0d:97:ac:02:51:26:7d:3e:fe:a6:
         c6:83:34:b4:84:0b:5d:b1:c4:25:2f:66:0a:2e:30:f6:52:88:
         e8:d2:05:78:84:09:01:e8:9d:c2:9e:b5:83:bd:8a:3a:e4:94:
         62:ed:92:e0:be:ea:d2:59:5b:c7:c3:61:35:dc:a9:98:9d:80:
         1c:2a:f7:23:9b:fe:ad:6f:16:7e:22:09:9a:79:8f:44:69:89:
         2a:78:ae:92:a4:32:46:8d:76:ee:68:25:63:5c:bd:41:a5:5a:
         57:18:d7:71:35:85:5c:cd:20:28:c6:d5:59:88:47:c9:36:44:
         53:55:28:4d:6b:f8:6a:00:eb:b4:62:de:15:56:c8:9c:45:d7:
         83:83:07:21:84:b4:eb:0b:23:f2:61:dd:95:03:02:df:0d:0f:
         97:32:e0:9d:38:de:7c:15:e4:36:66:7a:18:da:ce:a3:34:94:
         58:a6:5d:5c:04:90:35:f1:8b:55:a9:3c:dd:72:a2:d7:5f:73:
         5a:2c:88:85
```

Note

The CRL will only be deposited in Amazon S3 after a certificate has been issued that refers to it. Prior to that, there will only be an `acm-pca-permission-test-key` file visible in the Amazon S3 bucket.

Access policies for CRLs in Amazon S3

If you plan to create a CRL, you need to prepare an Amazon S3 bucket to store it in. AWS Private CA automatically deposits the CRL in the Amazon S3 bucket you designate and updates it periodically. For more information, see Creating a bucket.

Your S3 bucket must be secured by an attached IAM permissions policy. Authorized users and service principals require `Put` permission to allow AWS Private CA to place objects in the bucket, and `Get` permission to retrieve them.

Note

The IAM policy configuration depends on the AWS Regions involved. Regions fall into two categories:

- **Default-enabled Regions** – Regions that are *enabled* by default for all AWS accounts.
- **Default-disabled Regions** – Regions that are *disabled* by default, but may be manually enabled by the customer.

For more information and a list of the default-disabled Regions, see Managing AWS Regions. For a discussion of service principals in the context of IAM, see AWS service principals in opt-in Regions.
When you configure CRLs as the certificate revocation method, AWS Private CA creates a CRL and publishes it to an S3 bucket. The S3 bucket requires an IAM policy that allows the AWS Private CA service principal to write to the bucket. The name of the service principal varies according to the Regions used, and not all possibilities are supported.

PCA	S3	Service principal
Both in same Region		`acm-pca.amazonaws.com`
Enabled	Enabled	`acm-pca.amazonaws.com`
Disabled	Enabled	`acm-pca.`*`Region`*`.amazonaws.com`
Enabled	Disabled	Not supported

The default policy applies no `SourceArn` restriction on the CA. We recommend that you apply a less permissive policy such as the following, which restricts access to both a specific AWS account and a specific private CA. Alternatively, you can use the aws:SourceOrgID condition key to constrain

access to a specific organization in AWS Organizations. For more information about bucket policies, see Bucket policies for Amazon Simple Storage Service.

If you choose to allow the default policy, you can always modify it later.

Enable S3 Block Public Access (BPA) with CloudFront

New Amazon S3 buckets are configured by default with the Block Public Access (BPA) feature activated. Included in the Amazon S3 security best practices, BPA is a set of access controls that customers can use to fine-tune access to objects in their S3 buckets and to the buckets as a whole. When BPA is active and correctly configured, only authorized and authenticated AWS users have access to a bucket and its contents.

AWS recommends the use of BPA on all S3 buckets to avoid exposure of sensitive information to potential adversaries. However, additional planning is required if your PKI clients retrieve CRLs across the public internet (that is, while not logged into an AWS account). This section describes how to configure a private PKI solution using Amazon CloudFront, a content delivery network (CDN), to serve CRLs without requiring authenticated client access to an S3 bucket.

> **Note**
>
> Using CloudFront incurs additional costs on your AWS account. For more information, see Amazon CloudFront Pricing.
> If you choose to store your CRL in an S3 bucket with BPA enabled, and you do not use CloudFront, you must build another CDN solution to ensure that your PKI client has access to your CRL.

Set up CloudFront for BPA

Create a CloudFront distribution that will have access to your private S3 bucket, and can serve CRLs to unauthenticated clients.

To configure a CloudFront distribution for the CRL

1. Create a new CloudFront distribution using the procedure in Creating a Distribution in the *Amazon CloudFront Developer Guide*.

   While completing the procedure, apply the following settings:

- In **Origin Domain Name**, choose your S3 bucket.
- Choose **Yes** for **Restrict Bucket Access**.
- Choose **Create a New Identity** for **Origin Access Identity**.
- Choose **Yes, Update Bucket Policy** under **Grant Read Permissions on Bucket**.

> **Note**
>
> In this procedure, CloudFront modifies your bucket policy to allow it to access bucket objects. Consider editing this policy to allow access only to objects under the `crl` folder.

2. After the distribution has initialized, locate its domain name in the CloudFront console and save it for the next procedure.

> **Note**
>
> If your S3 bucket was newly created in a Region other than us-east-1, you might get an HTTP 307 temporary redirect error when you access your published application through CloudFront. It might take several hours for the address of the bucket to propagate.

Set up your CA for BPA

While configuring your new CA, include the alias to your CloudFront distribution.

To configure your CA with a CNAME for CloudFront

- Create your CA using Create a private CA in AWS Private CA.

  When you perform the procedure, the revocation file `revoke_config.txt` should include the following lines to specify a non-public CRL object and to provide a URL to the distribution endpoint in CloudFront:

```
"S3ObjectAcl":"BUCKET_OWNER_FULL_CONTROL",
 "CustomCname":"abcdef012345.cloudfront.net"
```

  Afterward, when you issue certificates with this CA, they will contain a block like the following:

```
X509v3 CRL Distribution Points:
 Full Name:
 URI:http://abcdef012345.cloudfront.net/crl/01234567-89ab-
cdef-0123-456789abcdef.crl
```

> **Note**
>
> If you have older certificates that were issued by this CA, they will be unable to access the CRL.

Determining the CRL Distribution Point (CDP) URI

If you need to use the CRL Distribution Point (CDP) URI in your workflow, you can either issue a certificate use the CRL URI on that certificate or use the following method. This only works for complete CRLs. Partitioned CRLs have a random GUID appended to them.

If you use the S3 bucket as the CRL Distribution Point (CDP) for your CA, the CDP URI can be in one of the following formats.

- `http://`*`amzn-s3-demo-bucket`*`.s3.`*`region-code`*`.amazonaws.com/crl/`*`CA-ID`*`.crl`
- `http://s3.`*`region-code`*`.amazonaws.com/`*`amzn-s3-demo-bucket`*`/crl/`*`CA-ID`*`.crl`

If you have configured your CA with a custom CNAME, the CDP URI will include the CNAME, for example, `http://`*`alternative.example.com`*`/crl/`*`CA-ID`*`.crl`

By default, AWS Private CA writes CDP extensions using regional, IPv4-only `amazonaws.com` endpoints. To use CRLs over IPv6, do one of the following steps so that CDPs are written with URLs that point to S3's dualstack endpoints:

- Set your CRL custom name to the S3 dualstack endpoint domain. For example, *`bucketname`*`.s3.dualstack.`*`region-code`*`.amazonaws.com`
- Set up your own CNAME DNS record pointing at the relevant S3 dualstack endpoint, then use it as your CRL custom name

Customize OCSP URL for AWS Private CA

> **Note**
>
> This topic is for customers who want to customize the public URL of the Online Certificate Status Protocol (OCSP) responder endpoint for branding or other purposes. If you plan to use the default configuration of AWS Private CA managed OCSP, you can skip this topic and follow the configuration instructions in Configure revocation.

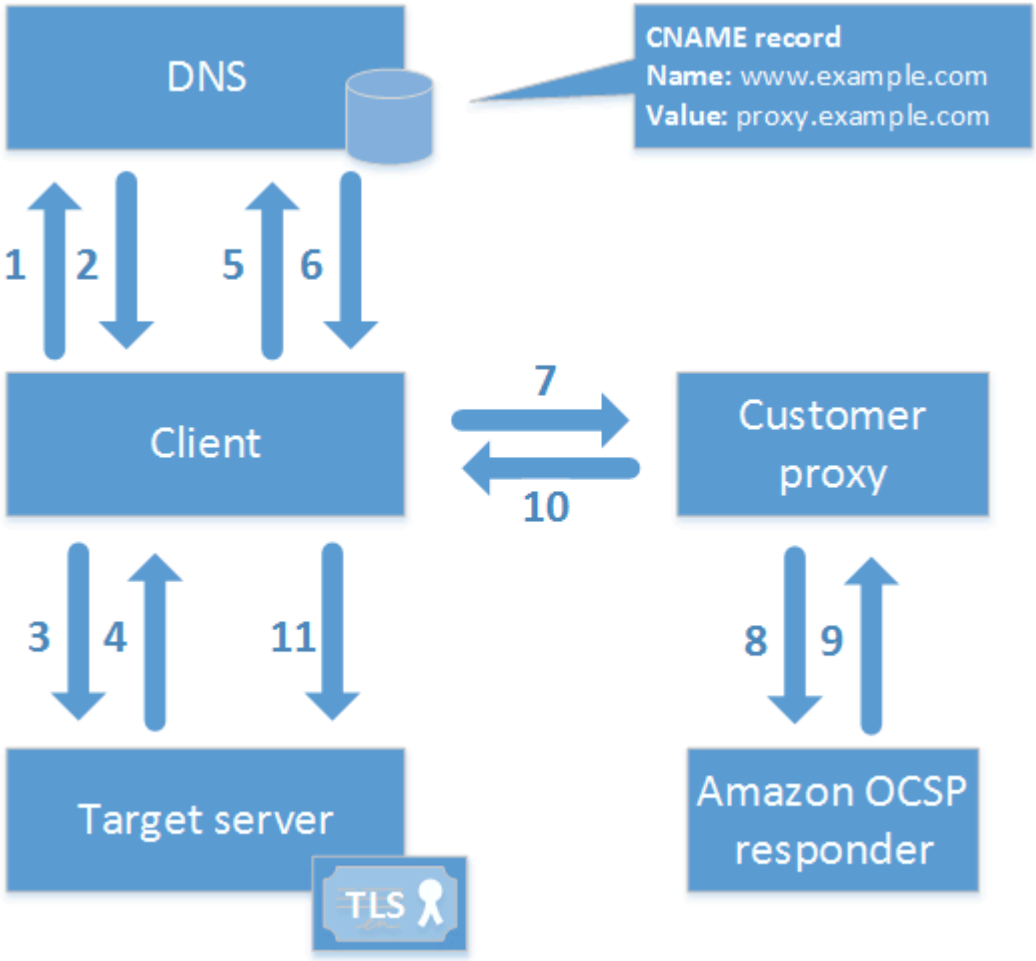
By default, when you enable OCSP for AWS Private CA, each certificate that you issue contains the URL for the AWS OCSP responder. This allows clients requesting a cryptographically secure connection to send OCSP validation queries directly to AWS. However, in some cases it might be preferable to state a different URL in your certificates while still ultimately submitting OCSP queries to AWS.

> **Note**
>
> For information about using a certificate revocation list (CRL) as an alternative or a supplement to OCSP, see Configure revocation and Planning a certificate revocation list (CRL).

Three elements are involved in configuring a custom URL for OCSP.

- **CA configuration** – Specify a custom OCSP URL in the `RevocationConfiguration` for your CA as described in Example 2: Create a CA with OCSP and a custom CNAME enabled in Create a private CA in AWS Private CA.
- **DNS** – Add a CNAME record to your domain configuration to map the URL appearing in the certificates to a proxy server URL. For more information, see Example 2: Create a CA with OCSP and a custom CNAME enabled in Create a private CA in AWS Private CA.
- **Forwarding proxy server** – Set up a proxy server that can transparently forward OCSP traffic that it receives to the AWS OCSP responder.

The following diagram illustrates how these elements work together.



As shown in the diagram, the customized OCSP validation process involves the following steps:

1. Client queries DNS for the target domain.
2. Client receives the target IP.
3. Client opens a TCP connection with target.
4. Client receives target TLS certificate.
5. Client queries DNS for the OCSP domain listed in the certificate.
6. Client receives proxy IP.
7. Client sends OCSP query to proxy.
8. Proxy forwards query to the OCSP responder.
9. Responder returns certificate status to the proxy.
10. Proxy forwards certificate status to the client.
11. If certificate is valid, client begins TLS handshake.

Tip

This example can be implemented using Amazon CloudFront and Amazon Route 53 after you have configured a CA as described above.

1. In CloudFront, create a distribution and configure it as follows:
   - Create an alternate name that matches your custom CNAME.
   - Bind your certificate to it.
   - Set `ocsp.acm-pca.<region>.amazonaws.com` as the origin.
     - To use IPv6 connections, use the dualstack endpoint `acm-pca-ocsp.<region>.api.aws`
   - Apply the `Managed-CachingDisabled` policy.
   - Set **Viewer protocol policy** to **HTTP and HTTPS**.
   - Set **Allowed HTTP methods** to **GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE**.
2. In Route 53, create a DNS record that maps your custom CNAME to the URL of the CloudFront distribution.

Using OCSP over IPv6

The default AWS Private CA OCSP responder URL is IPv4-only. To use OCSP over IPv6, configure a custom OCSP URL for your CA. The URL can be either:

- The FQDN of the dualstack PCA OCSP responder, which takes the form `acm-pca-ocsp.region-name.api.aws`
- A CNAME record that you have configured to point at the dualstack OCSP responder, as explained above.

Understand AWS Private CA CA modes

AWS Private CA supports the creation of a certificate authority (CA) in either of two modes. The modes, general-purpose and short-lived certificate, affect the allowed validity period of the certificates issued by the CA.

> **Note**
>
> AWS Private CA does not perform validity checks on root CA certificates.

General-purpose (default)

This mode permits the CA to issue certificates with any validity period. Most applications use certificates of this type. Typically, the CA also specifies a revocation mechanism.

Short-lived certificate

This mode defines a CA that exclusively issues certificates with a maximum validity period of seven days. These short-lived certificates expire so quickly that they can be deployed without a revocation mechanism in place. For some applications, it makes more sense to frequently deploy short-lived certificates than to incur the network and processing overhead of revocation.

Short-lived certificates must be the last CA in the certificate hierarchy. There is significant overhead because the private CA must be renewed every seven days.

CAs with short-lived certificate mode cost less than general-purpose CAs. For more information, see AWS Private Certificate Authority Pricing.

To create a CA that issues short-lived certificates, set the `UsageMode` parameter to short-lived certificate using the create a CA procedure for creating a CA.

> **Note**
>
> AWS Certificate Manager cannot issue certificates signed by a private CA with short-lived mode.

Use of short-lived certificates is supported by the following AWS services:

- Amazon AppStream
- Amazon WorkSpaces

Plan for resilience in AWS Private CA

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.

Redundancy and disaster recovery

Consider redundancy and DR when planning your CA hierarchy. AWS Private CA is available in multiple Regions, which allows you to create redundant CAs in multiple Regions. The AWS Private CA service operates with a service level agreement (SLA) of 99.9% availability. There are at least two approaches that you can consider for redundancy and disaster recovery. You can configure redundancy at the root CA or at the highest subordinate CA. Each approach has pros and cons.

1. You can create two root CAs in two different AWS Regions for redundancy and disaster recovery. With this configuration, each root CA operates independently in an AWS Region, protecting you in the event of a single-Region disaster. Creating redundant root CAs does, however, increase operational complexity: You will need to distribute both root CA certificates to the trust stores of browsers and operating systems in your environment.
2. You can also create redundant subordinate CAs to deploy in each of your AWS Regions, and chain them to the same unique root CA in a single AWS Region. The benefit of this approach is that you need to distribute only a single root CA certificate to the trust stores in your environment. The limitation is that you don't have a redundant root CA in the event of a disaster that affects the AWS Region in which your root CA exists.

Certificate authorities in AWS Private CA

Using AWS Private Certificate Authority, you can create an entirely AWS hosted hierarchy of root and subordinate certificate authorities (CAs) for internal use by your organization. To manage certificate revocation, you can enable Online Certificate Status Protocol (OCSP), certificate revocation lists (CRLs), or both. AWS Private CA stores and manages your CA certificates, CRLs, and OCSP responses, and the private keys for your root authorities are securely stored by AWS.

> **Note**
>
> The OCSP implementation in AWS Private CA does not support OCSP request extensions. If you submit an OCSP batch query containing multiple certificates, the AWS OCSP responder processes only the first certificate in the queue and drops the others. A revocation might take up to an hour to appear in OCSP responses.

You can access AWS Private CA using the AWS Management Console, the AWS CLI, and the AWS Private CA API. The following topics show you how to use the console and the CLI. To learn more about the API, see the AWS Private Certificate Authority API Reference. For Java examples that show you how to use the API, see Use AWS Private CA with the AWS SDK for Java.

After you create an active private CA and configured access to it, you can issue and retrieve certificates, as described in Issue and manage certificates in AWS Private CA.

Topics

- Set up to use AWS Private CA
- Create a private CA in AWS Private CA
- Installing the CA certificate
- Control access to the private CA
- List private CAs
- View a private CA
- Add tags for your private CA
- Understand AWS Private CA CA status
- Update a private CA in AWS Private Certificate Authority
- Delete your private CA

- Restore a private CA
- Use externally signed private CA certificates

Set up to use AWS Private CA

If you're not already an Amazon Web Services (AWS) customer, you must sign up to be able to use AWS Private CA. Your account automatically has access to all available services, but you are charged only for services that you use.

Topics

- Sign up for an AWS account
- Create a user with administrative access
- Install the AWS Command Line Interface

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*.

Install the AWS Command Line Interface

If you have not installed the AWS CLI but want to use it, follow the directions at AWS Command Line Interface. In this guide, we assume that you have configured your endpoint, Region, and authentication details, and we omit these parameters from the sample commands.

Create a private CA in AWS Private CA

You can use the procedures in this section to create either root CAs or subordinate CAs, resulting in an auditable hierarchy of trust relationships that matches your organizational needs. You can create a CA using the AWS Management Console, the PCA portion of the AWS CLI, or AWS CloudFormation.

For information about updating the configuration of a CA that you have already created, see Update a private CA in AWS Private Certificate Authority.

For information about using a CA to sign end-entity certificates for your users, devices, and applications, see Issue private end-entity certificates.

> **Note**
>
> Your account is charged a monthly price for each private CA starting from the time that you create it.
> For the latest AWS Private CA pricing information, see AWS Private Certificate Authority Pricing. You can also use the AWS pricing calculator to estimate costs.

Topics

- CLI examples for creating a private CA

Console

To create a private CA using the console

1. Complete the following steps to create a private CA using the AWS Management Console.

   To get started using the console

   Sign in to your AWS account and open the AWS Private CA console at **https://console.aws.amazon.com/acm-pca/home**.

   - If you are opening the console in a Region where you have no private CAs, the introductory page appears. Choose **Create a private CA**.
   - If you are opening the console in a Region where you have already created a CA, the **Private certificate authorities** page opens with a list of your CAs. Choose **Create CA**.

2. Under **Mode options**, choose the expiration mode of the certificates that your CA issues.

   - **General-purpose** – Issues certificates that can be configured with any expiration date. This is the default.
   - **Short-lived certificate** – Issues certificates with a maximum validity period of seven days. A short validity period can substitute in some cases for a revocation mechanism.

3. On the **Type options** section of the console, choose the type of private certificate authority that you want to create.

   - Choosing **Root** establishes a new CA hierarchy. This CA is backed by a self-signed certificate. It serves as the ultimate signing authority for other CAs and end-entity certificates in the hierarchy.
   - Choosing **Subordinate** creates a CA that must be signed by a parent CA above it in the hierarchy. Subordinate CAs are typically used to create other subordinate CAs or to issue end-entity certificates to users, computers, and applications.

     > **Note**
     >
     > AWS Private CA provides an automated signing process when your subordinate CA's parent CA is also hosted by AWS Private CA. All you do is choose the parent CA to use.

Your subordinate CA might need to be signed by an external trust services provider. If so, AWS Private CA provides you with a certificate signing request (CSR) that you must download and use to obtain a signed CA certificate. For more information, see Install a subordinate CA certificate signed by an external parent CA.

4. Under **Subject distinguished name options**, configure the subject name of your private CA. You must enter a value for at least one of the following options:

   - **Organization (O)** – For example, a company name
   - **Organization Unit (OU)** – For example, a division within a company
   - **Country name (C)** – A two-letter country code
   - **State or province name** – Full name of a state or province
   - **Locality name** – The name of a city
   - **Common Name (CN)** – A human-readable string to identify the CA.

   Note

   You can further customize the subject name of a certificate by applying an APIPassthrough template at the time of issue. For more information and a detailed example, see Issue a certificate with a custom subject name using an APIPassthrough template.

   Because the backing certificate is self-signed, the subject information that you provide for a private CA is probably more sparse than what a public CA would contain. For more information about each of the values that make up a subject distinguished name, see RFC 5280.

5. Under **Key algorithm options**, choose the key algorithm and the algorithm strength. The default value is RSA 2048. You can choose from the following algorithms:

   - ML-DSA-44
   - ML-DSA-65
   - ML-DSA-87

- RSA 2048
- RSA 3072
- RSA 4096
- ECDSA P256
- ECDSA P384
- ECDSA P521

6. Under **Certificate revocation options**, you can select from two methods of sharing revocation status with clients that use your certificates:

   - **Activate CRL distribution**
   - **Turn on OCSP**

   You can configure either, neither, or both of these revocation options for your CA. Although optional, managed revocation is recommended as a best practice. Before completing this step, see Plan your AWS Private CA certificate revocation method for information about the advantages of each method, the preliminary setup that might be required, and additional revocation features.

   > **Note**
   >
   > If you create your CA without configuring revocation, you can always configure it later. For more information, see Update a private CA in AWS Private Certificate Authority.

   To configure **Certificate revocation options**, perform the following steps.

   a. Under **Certificate revocation options**, choose **Activate CRL distribution**.

   b. Under **S3 bucket URI**, choose an existing bucket from the list.

      When you specify an existing bucket, you must ensure that BPA is disabled for the account and for the bucket. Otherwise, the operation to create the CA fail. If the CA is created successfully, you must still manually attach a policy to it before you can begin generating CRLs. Use one of the policy patterns described in Access policies for CRLs

in Amazon S3 . For more information, see Adding a bucket policy using the Amazon S3 console.

c. Expand **CRL settings** for additional configuration options.

- Choose **Enable partitioning** to enable partitioning of CRLs. If you don't enable partitioning, your CA is subject to the maximum number of revoked certificates. For more information, see AWS Private Certificate Authority quotas. For more information about partitioned CRLs, see CRL types.
- Add a **Custom CRL Name** to create an alias for your Amazon S3 bucket. This name is contained in certificates issued by the CA in the “CRL Distribution Points" extension that is defined by RFC 5280. To use CRLs over IPv6, set this to your bucket's dualstack S3 endpoint as described in Using CRLs over IPv6.
- Add a **Custom path** to create a DNS alias for the file path in your Amazon S3 bucket.
- Type the **Validity in days** your CRL will remain valid. The default value is 7 days. For online CRLs, a validity period of 2-7 days is common. AWS Private CA tries to regenerate the CRL at the midpoint of the specified period.

7. For **Certificate revocation options**, choose **Turn on OCSP**.

- In the **Custom OCSP endpoint - *optional*** field, you can provide a fully qualified domain name (FQDN) for a non-Amazon OCSP endpoint. To use OCSP over IPv6, set this field to a dualstack endpoint as described in Using OCSP over IPv6.

  When you provide an FQDN in this field, AWS Private CA inserts the FQDN into the *Authority Information Access* extension of each issued certificate in place of the default URL for the AWS OCSP responder. When an endpoint receives a certificate containing the custom FQDN, it queries that address for an OCSP response. For this mechanism to work, you need to take two additional actions:

  - Use a proxy server to forward traffic that arrives at your custom FQDN to the AWS OCSP responder.
  - Add a corresponding CNAME record to your DNS database.

> **Tip**
>
> For more information about implementing a complete OCSP solution using a custom CNAME, see Customize OCSP URL for AWS Private CA.

For example, here is a CNAME record for customized OCSP as it would appear in Amazon Route 53.

Record name	Type	Routing policy	Differentiator	Value/Route traffic to
alternative.example.com	CNAME	Simple	-	proxy.example.com

> **Note**
>
> The value of the CNAME must not include a protocol prefix such as "http://" or "https://".

8. Under **Add tags**, you can optionally tag your CA. Tags are key-value pairs that serve as metadata for identifying and organizing AWS resources. For a list of AWS Private CA tag parameters and for instructions on how to add tags to CAs after creation, see Add tags for your private CA.

> **Note**
>
> To attach tags to a private CA during the creation procedure, a CA administrator must first associate an inline IAM policy with the `CreateCertificateAuthority` action and explicitly allow tagging. For more information, see Tag-on-create: Attaching tags to a CA at the time of creation.

9. Under **CA permissions options**, you can optionally delegate automatic renewal permissions to the AWS Certificate Manager service principal. ACM can only automatically renew private end-entity certificates generated by this CA if this permission is granted. You can assign renewal permissions at any time with the AWS Private CA CreatePermission API or create-permission CLI command.

   The default is to enable these permissions.

> **Note**
>
> AWS Certificate Manager does not support the automatic renewal of short-lived certificates.

10. Under **Pricing**, confirm that you understand the pricing for a private CA.

> **Note**
>
> For the latest AWS Private CA pricing information, see AWS Private Certificate Authority Pricing. You can also use the AWS pricing calculator to estimate costs.

11. Choose **Create CA** after you have checked all of the entered information for accuracy. The details page for the CA opens and displays its status as **Pending certificate**.

> **Note**
>
> While on the details page, you can finish configuring your CA by choosing **Actions**, **Install CA certificate**, or you can return later to the **Private certificate authorities** list and complete the installation procedure that applies in your case:
>
> - Install a root CA certificate
> - Install a subordinate CA certificate hosted by AWS Private CA
> - Install a subordinate CA certificate signed by an external parent CA

CLI

Use the create-certificate-authority command to create a private CA. You must specify the CA configuration (containing algorithm and subject-name information), the revocation configuration (if you plan to use OCSP and/or a CRL), and the CA type (root or subordinate). The configuration and revocation configuration details are contained in two files that you supply as arguments to the command. Optionally, you can also configure the CA usage mode (for issuing standard or short-lived certificates), attach tags, and provide an idempotency token.

If you are configuring a CRL, you must have a secured Amazon S3 bucket in place *before* you issue the **create-certificate-authority** command. For more information, see Access policies for CRLs in Amazon S3 .

The CA configuration file specifies the following information:

- The name of the algorithm
- The key size to be used to create the CA private key
- The type of signing algorithm that the CA uses to sign its own Certificate Signing Request, CRLs, and OCSP responses
- X.500 subject information

The revocation configuration for OCSP defines an `OcspConfiguration` object with the following information:

- The `Enabled` flag set to "true".
- (Optional) A custom CNAME declared as a value for `OcspCustomCname`.

The revocation configuration for a CRL defines a `CrlConfiguration` object with the following information:

- The `Enabled` flag set to "true".
- The CRL expiration period in days (the validity period of the CRL).
- The Amazon S3 bucket that will contain the CRL.
- (Optional) An S3ObjectAcl value that determines whether the CRL is publicly accessible. In the example presented here, public access is blocked. For more information, see Enable S3 Block Public Access (BPA) with CloudFront.
- (Optional) A CNAME alias for the S3 bucket that is included in certificates issued by the CA. If the CRL is not publicly accessible, this will point to a distribution mechanism such as Amazon CloudFront.
- (Optional) A `CrlDistributionPointExtensionConfiguration` object with the following information:
  - The `OmitExtension` flag set to "true" or "false". This controls whether the default value for the CDP extension will be written to a certificate issued by the CA. For more

information about the CDP extension, see Determining the CRL Distribution Point (CDP) URI . A CustomCname cannot be set if OmitExtension is "true".

- (Optional) A custom path for the CRL in the S3 bucket.
- (Optional) A CrlType value that determines whether the CRL will be complete or partitioned. If not supplied, the CRL will default to complete.

> **Note**
>
> You can enable both revocation mechanisms on the same CA by defining both an `OcspConfiguration` object and a `CrlConfiguration` object. If you supply no **--revocation-configuration** parameter, both mechanisms are disabled by default. If you need revocation validation support later, see Updating a CA (CLI).

See the following section for CLI examples.

CLI examples for creating a private CA

The following examples assume that you have set up your `.aws` configuration directory with a valid default Region, endpoint, and credentials. For information about configuring your AWS CLI environment, see Configuration and credential file settings. For readability, we supply the CA configuration and revocation input as JSON files in the example commands. Modify the example files as needed for your use.

All of the examples use the following `ca_config.txt` configuration file unless otherwise stated.

File: ca_config.txt

```
{
   "KeyAlgorithm":"RSA_2048",
   "SigningAlgorithm":"SHA256WITHRSA",
   "Subject":{
      "Country":"US",
      "Organization":"Example Corp",
      "OrganizationalUnit":"Sales",
      "State":"WA",
      "Locality":"Seattle",
      "CommonName":"www.example.com"
```

```
   }
}
```

Example 1: Create a CA with OCSP enabled

In this example, the revocation file enables default OCSP support, which uses the AWS Private CA responder to check certificate status.

File: revoke_config.txt for OCSP

```
{
   "OcspConfiguration":{
      "Enabled":true
   }
}
```

Command

```
$ aws acm-pca create-certificate-authority \
     --certificate-authority-configuration file://ca_config.txt \
     --revocation-configuration file://revoke_config.txt \
     --certificate-authority-type "ROOT" \
     --idempotency-token 01234567 \
     --tags Key=Name,Value=MyPCA
```

If successful, this command outputs the Amazon Resource Name (ARN) of the new CA.

```
{
 "CertificateAuthorityArn":"arn:aws:acm-pca:region:account:
       certificate-authority/CA_ID"
}
```

Command

```
$ aws acm-pca create-certificate-authority \
 --certificate-authority-configuration file://ca_config.txt \
 --revocation-configuration file://revoke_config.txt \
 --certificate-authority-type "ROOT" \
 --idempotency-token 01234567 \
 --tags Key=Name,Value=MyPCA-2
```

If successful, this command outputs the Amazon Resource Name (ARN) of the CA.

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

Use the following command to inspect the configuration of your CA.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

This description should contain the following section.

```
"RevocationConfiguration": {
   ...
   "OcspConfiguration": {
      "Enabled": true
   }
   ...
}
```

Example 2: Create a CA with OCSP and a custom CNAME enabled

In this example, the revocation file enables customized OCSP support. The `OcspCustomCname` parameter takes a fully qualified domain name (FQDN) as its value.

When you provide an FQDN in this field, AWS Private CA inserts the FQDN into the *Authority Information Access* extension of each issued certificate in place of the default URL for the AWS OCSP responder. When an endpoint receives a certificate containing the custom FQDN, it queries that address for an OCSP response. For this mechanism to work, you need to take two additional actions:

- Use a proxy server to forward traffic that arrives at your custom FQDN to the AWS OCSP responder.
- Add a corresponding CNAME record to your DNS database.

> **Tip**
>
> For more information about implementing a complete OCSP solution using a custom CNAME, see Customize OCSP URL for AWS Private CA.

For example, here is a CNAME record for customized OCSP as it would appear in Amazon Route 53.

Record name	Type	Routing policy	Differentiator	Value/Route traffic to
alternative.example.com	CNAME	Simple	-	proxy.example.com

> **Note**
>
> The value of the CNAME must not include a protocol prefix such as "http://" or "https://".

File: revoke_config.txt for OCSP

```
{
   "OcspConfiguration":{
      "Enabled":true,
      "OcspCustomCname":"alternative.example.com"
   }
}
```

Command

```
$ aws acm-pca create-certificate-authority \
 --certificate-authority-configuration file://ca_config.txt \
 --revocation-configuration file://revoke_config.txt \
 --certificate-authority-type "ROOT" \
 --idempotency-token 01234567 \
 --tags Key=Name,Value=MyPCA-3
```

If successful, this command outputs the Amazon Resource Name (ARN) of the CA.

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

Use the following command to inspect the configuration of your CA.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

This description should contain the following section.

```
"RevocationConfiguration": {
   ...
   "OcspConfiguration": {
      "Enabled": true,
      "OcspCustomCname": "alternative.example.com"
   }
   ...
}
```

Example 3: Create a CA with an attached CRL

In this example, the revocation configuration defines CRL parameters.

File: revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
      "ExpirationInDays":7,
      "S3BucketName":"amzn-s3-demo-bucket"
   }
}
```

Command

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config.txt \
```

```
      --revocation-configuration file://revoke_config.txt \
      --certificate-authority-type "ROOT" \
      --idempotency-token 01234567 \
      --tags Key=Name,Value=MyPCA-1
```

If successful, this command outputs the Amazon Resource Name (ARN) of the CA.

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

Use the following command to inspect the configuration of your CA.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

This description should contain the following section.

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket"
   },
   ...
}
```

Example 4: Create a CA with an attached CRL and a custom CNAME enabled

In this example, the revocation configuration defines CRL parameters that include a custom CNAME.

File: revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
```

```
      "ExpirationInDays":7,
      "CustomCname": "alternative.example.com",
      "S3BucketName":"amzn-s3-demo-bucket"
   }
}
```

Command

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config.txt \
      --revocation-configuration file://revoke_config.txt \
      --certificate-authority-type "ROOT" \
      --idempotency-token 01234567 \
      --tags Key=Name,Value=MyPCA-1
```

If successful, this command outputs the Amazon Resource Name (ARN) of the CA.

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

Use the following command to inspect the configuration of your CA.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

This description should contain the following section.

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "CustomCname": "alternative.example.com",
      "S3BucketName": "amzn-s3-demo-bucket",
   ...
   }
}
```

Example 5: Create a CA and specify the usage mode

In this example, the CA usage mode is specified when creating a CA. If unspecified, the usage mode parameter defaults to GENERAL_PURPOSE. In this example, the parameter is set to SHORT_LIVED_CERTIFICATE, which means that the CA will issue certificates with a maximum validity period of seven days. In situations where it is inconvenient to configure revocation, a short-lived certificate that has been compromised quickly expires as part of normal operations. Consequently, this example CA lacks a revocation mechanism.

> **Note**
>
> AWS Private CA does not perform validity checks on root CA certificates.

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config.txt \
      --certificate-authority-type "ROOT" \
      --usage-mode SHORT_LIVED_CERTIFICATE \
      --tags Key=usageMode,Value=SHORT_LIVED_CERTIFICATE
```

Use the **describe-certificate-authority** command in the AWS CLI to display details about the resulting CA, as shown in the following command:

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn arn:aws:acm:region:account:certificate-
authority/CA_ID
```

```
{
    "CertificateAuthority":{
       "Arn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
       "CreatedAt":"2022-09-30T09:53:42.769000-07:00",
       "LastStateChangeAt":"2022-09-30T09:53:43.784000-07:00",
       "Type":"ROOT",
       "UsageMode":"SHORT_LIVED_CERTIFICATE",
       "Serial":"serial_number",
       "Status":"PENDING_CERTIFICATE",
       "CertificateAuthorityConfiguration":{
          "KeyAlgorithm":"RSA_2048",
          "SigningAlgorithm":"SHA256WITHRSA",
          "Subject":{
```

```
            "Country":"US",
            "Organization":"Example Corp",
            "OrganizationalUnit":"Sales",
            "State":"WA",
            "Locality":"Seattle",
            "CommonName":"www.example.com"
         }
      },
      "RevocationConfiguration":{
         "CrlConfiguration":{
            "Enabled":false
         },
         "OcspConfiguration":{
            "Enabled":false
         }
      },
...
```

Example 6: Create a CA for Active Directory login

You can create a private CA suitable for use in the Enterprise NTAuth store of Microsoft Active Directory (AD), where it can issue card-logon or domain-controller certificates. For information about importing a CA certificate into AD, see How to import third-party certification authority (CA) certificates into the Enterprise NTAuth store.

The Microsoft certutil tool can be used to publish CA certificates in AD by invoking the **-dspublish** option. A certificate published to AD with certutil is trusted across the entire forest. Using group policy, you can also limit trust to a subset of the entire forest, for example, a single domain or a group of computers in a domain. For logon to work, the issuing CA must also be published in the NTAuth store. For more information, see Distribute Certificates to Client Computers by Using Group Policy.

This example uses the following `ca_config_AD.txt` configuration file.

File: ca_config_AD.txt

```
{
   "KeyAlgorithm":"RSA_2048",
   "SigningAlgorithm":"SHA256WITHRSA",
   "Subject":{
      "CustomAttributes":[
         {
```

```
            "ObjectIdentifier":"2.5.4.3",
            "Value":"root CA"
         },
         {
            "ObjectIdentifier":"0.9.2342.19200300.100.1.25",
            "Value":"example"
         },
         {
            "ObjectIdentifier":"0.9.2342.19200300.100.1.25",
            "Value":"com"
         }
      ]
   }
}
```

Command

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config_AD.txt \
      --certificate-authority-type "ROOT" \
      --tags Key=application,Value=ActiveDirectory
```

If successful, this command outputs the Amazon Resource Name (ARN) of the CA.

```
{
 "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
 }
```

Use the following command to inspect the configuration of your CA.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

This description should contain the following section.

```
...

"Subject":{
```

```
   "CustomAttributes":[
      {
         "ObjectIdentifier":"2.5.4.3",
         "Value":"root CA"
      },
      {
         "ObjectIdentifier":"0.9.2342.19200300.100.1.25",
         "Value":"example"
      },
      {
         "ObjectIdentifier":"0.9.2342.19200300.100.1.25",
         "Value":"com"
      }
   ]
}
...
```

Example 7: Create a Matter CA with an attached CRL and the CDP extension omitted from issued certificates

You can create a private CA suitable for issuing certificates for the Matter smart home standard. In this example, the CA configuration in `ca_config_PAA.txt` defines a Matter Product Attestation Authority (PAA) with the Vendor ID (VID) set to FFF1.

File: ca_config_PAA.txt

```
{
   "KeyAlgorithm":"EC_prime256v1",
   "SigningAlgorithm":"SHA256WITHECDSA",
   "Subject":{
      "Country":"US",
      "Organization":"Example Corp",
      "OrganizationalUnit":"SmartHome",
      "State":"WA",
      "Locality":"Seattle",
      "CommonName":"Example Corp Matter PAA",
   "CustomAttributes":[
      {
        "ObjectIdentifier":"1.3.6.1.4.1.37244.2.1",
        "Value":"FFF1"
      }
    ]
  }
```

```
}
```

The revocation configuration enables CRLs, and configures the CA to omit the default CDP URL from any issued certificates.

File: revoke_config.txt

```
{
   "CrlConfiguration":{
      "Enabled":true,
      "ExpirationInDays":7,
      "S3BucketName":"amzn-s3-demo-bucket",
   "CrlDistributionPointExtensionConfiguration":{
  "OmitExtension":true
   }
   }
}
```

Command

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration file://ca_config_PAA.txt \
      --revocation-configuration file://revoke_config.txt \
      --certificate-authority-type "ROOT" \
      --idempotency-token 01234567 \
      --tags Key=Name,Value=MyPCA-1
```

If successful, this command outputs the Amazon Resource Name (ARN) of the CA.

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
}
```

Use the following command to inspect the configuration of your CA.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

This description should contain the following section.

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket",
   "CrlDistributionPointExtensionConfiguration":{
  "OmitExtension":true
   }
   },
   ...
}
...
```

Installing the CA certificate

Complete the following procedures to create and install your private CA certificate. Your CA will then be ready to use.

AWS Private CA supports three scenarios for installing a CA certificate:

- Installing a certificate for a root CA hosted by AWS Private CA
- Installing a subordinate CA certificate whose parent authority is hosted by AWS Private CA
- Installing a subordinate CA certificate whose parent authority is externally hosted

The following sections describe procedures for each scenario. The console procedures begin on the console page **Private CAs**.

Compatible signing algorithms

Signing algorithm support for CA certificates depends on the signing algorithm of the parent CA and on the AWS Region. The following constraints apply to both console and AWS CLI operations.

- A parent CA with the RSA key algorithm can issue certificates with the following signing algorithms:
  - SHA256 RSA

- SHA384 RSA
- SHA512 RSA

- In a legacy AWS Region, a parent CA with the EDCSA key algorithm can issue certificates with the following signing algorithms:
  - SHA256 ECDSA
  - SHA384 ECDSA
  - SHA512 ECDSA

  Legacy AWS Regions include:

Region name	Geographical location
eu-north-1	Europe (Stockholm)
me-south-1	Middle East (Bahrain)
ap-south-1	Asia Pacific (Mumbai)
eu-west-3	Europe (Paris)
us-east-2	US East (Ohio)
af-south-1	Africa (Cape Town)
eu-west-1	Europe (Ireland)
eu-central-1	Europe (Frankfurt)
sa-east-1	South America (São Paulo)
ap-east-1	Asia Pacific (Hong Kong)

Region name	Geographical location
us-east-1	US East (N. Virginia)
ap-northeast-2	Asia Pacific (Seoul)
eu-west-2	Europe (London)
ap-northeast-1	Asia Pacific (Tokyo)
us-gov-east-1	AWS GovCloud (US-East)
us-gov-west-1	AWS GovCloud (US-West)
us-west-2	US West (Oregon)
us-west-1	US West (N. California)
ap-southeast-1	Asia Pacific (Singapore)
ap-southeast-2	Asia Pacific (Sydney)

- In a non-legacy AWS Region, the following rules apply for EDCSA:
  - A parent CA with the EC_prime256v1 signing algorithm can issue certificates with ECDSA P256.
  - A parent CA with the EC_secp384r1 signing algorithm can issue certificates with ECDSA P384.
- In every AWS Region, the following rules apply for EDCSA:
  - A parent CA with the EC_secp521r1 signing algorithm can issue certificates with ECDSA P521.

Install a root CA certificate

You can install a root CA certificate from the AWS Management Console or the AWS CLI.

To create and install a certificate for your private root CA (console)

1. (Optional) If you are not already on the CA's details page, open the AWS Private CA console at https://console.aws.amazon.com/acm-pca/home. On the **Private certificate authorities** page, choose a root CA with status **Pending certificate** or **Active**.
2. Choose **Actions**, **Install CA certificate** to open the **Install root CA certificate** page.
3. Under **Specify the root CA certificate parameters**, specify the following certificate parameters:

   - **Validity** — Specifies the expiration date and time for the CA certificate. The AWS Private CA default validity period for a root CA certificate is 10 years.
   - **Signature algorithm** — Specifies the signing algorithm to use when the root CA issues new certificates. Available options vary according to the AWS Region where you are creating the CA. For more information, see Compatible signing algorithms, Supported cryptographic algorithms in AWS Private Certificate Authority, and **SigningAlgorithm** in CertificateAuthorityConfiguration.
     - SHA256 RSA
     - SHA384 RSA
     - SHA512 RSA

   Review your settings for correctness, then choose **Confirm and install**. AWS Private CA exports a CSR for your CA, generates a certificate using a root CA certificate template, and self-signs the certificate. AWS Private CA then imports the self-signed root CA certificate.
4. The details page for the CA displays the status of the installation (success or failure) at the top. If the installation was successful, the newly completed root CA displays a status of **Active** in the **General** pane.

To create and install a certificate for your private root CA (AWS CLI)

1. Generate a certificate signing request (CSR).

```
$ aws acm-pca get-certificate-authority-csr \
```

```
     --certificate-authority-arn arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
     --output text \
     --region region > ca.csr
```

The resulting file `ca.csr`, a PEM file encoded in base64 format, has the following appearance.

```
-----BEGIN CERTIFICATE REQUEST-----
MIIC1DCCAbwCAQAwbTELMAkGA1UEBhMCVVMxFTATBgNVBAoMDEV4YW1wbGUgQ29y
cDEOMAwGA1UECwwFU2FsZXMxCzAJBgNVBAgMAldBMRgwFgYDVQQDDA93d3cuZXhh
bXBsZS5jb20xEDAOBgNVBAcMB1NlYXR0bGUwggEiMA0GCSqGSIb3DQEBAQUAA4IB
DwAwggEKAoIBAQDD+7eQChWUO2m6pHslI7AVSFkWvbQofKIHvbvy7wm8VO9/BuI7
LE/jrnd1jGoyI7jaMHKXPtEP3uNlCzv+oEza07OjgjqPZVehtA6a3/3vdQ1qCoD2
rXpv6VIzcq2onx2X7m+Zixwn2oY1l1ELXP7I5g0GmUStymq+pY5VARPy3vTRMjgC
JEiz8w7VvC15uIsHFAWa2/NvKyndQMPaCNft238wesV5s2cXOUS173jghIShg99o
ymf0TRUgvAGQMCXvsW07MrP5VDmBU7k/AZ9ExsUfMe2OB++fhfQWr2N7/lpC4+DP
qJTfXTEexLfRTLeLuGEaJL+c6fMyG+Yk53tZAgMBAAGgIjAgBgkqhkiG9w0BCQ4x
EzARMA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBAA7xxLVI5s1B
qmXMMT44y1DZtQx3RDPanMNGLGO1TmLtyqqnUH49Tla+2p7nrl0tojUf/3PaZ52F
QN09SrFk8qtYSKnMGd5PZL0A+NFsNW+w4BAQNKlg9m617YEsnkztbfKRloaJNYoA
HZaRvbA0lMQ/tU2PKZR2vnao444Ugm0O/t3jx5rj817b31hQcHHQ0lQuXV2kyTrM
ohWeLf2fL+K0xJ9ZgXD4KYnY0zarpreA5RBeO5xs3Ms+oGWc13qQfMBx33vrrz2m
dw5iKjg71uuUUmtDV6ewwGa/VO5hNinYAfogdu5aGuVbnTFT3n45B8WHz2+9r0dn
bA7xUel1SuQ=
-----END CERTIFICATE REQUEST-----
```

You can use OpenSSL to view and verify the contents of the CSR.

```
openssl req -text -noout -verify -in ca.csr
```

This yields output similar to the following.

```
verify OK
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, O=Example Corp, OU=Sales, ST=WA, CN=www.example.com,
 L=Seattle
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
```

```
                    00:c3:fb:b7:90:0a:15:94:3b:69:ba:a4:7b:25:23:
                    b0:15:48:59:16:bd:b4:28:7c:a2:07:bd:bb:f2:ef:
                    09:bc:54:ef:7f:06:e2:3b:2c:4f:e3:ae:77:75:8c:
                    6a:32:23:b8:da:30:72:97:3e:d1:0f:de:e3:65:0b:
                    3b:fe:a0:4c:da:d3:b3:a3:82:3a:8f:65:57:a1:b4:
                    0e:9a:df:fd:ef:75:0d:6a:0a:80:f6:ad:7a:6f:e9:
                    52:33:72:ad:a8:9f:1d:97:ee:6f:99:8b:1c:27:da:
                    86:35:97:51:0b:5c:fe:c8:e6:0d:06:99:44:ad:ca:
                    6a:be:a5:8e:55:01:13:f2:de:f4:d1:32:38:02:24:
                    48:b3:f3:0e:d5:bc:2d:79:b8:8b:07:14:05:9a:db:
                    f3:6f:2b:29:dd:40:c3:da:08:d7:ed:db:7f:30:7a:
                    c5:79:b3:67:17:39:44:b5:ef:78:e0:84:84:a1:83:
                    df:68:ca:67:f4:4d:15:20:bc:01:90:30:25:ef:b1:
                    6d:3b:32:b3:f9:54:39:81:53:b9:3f:01:9f:44:c6:
                    c5:1f:31:ed:8e:07:ef:9f:85:f4:16:af:63:7b:fe:
                    5a:42:e3:e0:cf:a8:94:df:5d:31:1e:c4:b7:d1:4c:
                    b7:8b:b8:61:1a:24:bf:9c:e9:f3:32:1b:e6:24:e7:
                    7b:59
                Exponent: 65537 (0x10001)
        Attributes:
        Requested Extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
         0e:f1:c4:b5:48:e6:cd:41:aa:65:cc:31:3e:38:cb:50:d9:b5:
         0c:77:44:33:da:9c:c3:46:2c:63:b5:4e:62:ed:ca:aa:a7:50:
         7e:3d:4e:56:be:da:9e:e7:ae:5d:2d:a2:35:1f:ff:73:da:67:
         9d:85:40:dd:3d:4a:b1:64:f2:ab:58:48:a9:cc:19:de:4f:64:
         bd:00:f8:d1:6c:35:6f:b0:e0:10:10:34:a9:60:f6:6e:b5:ed:
         81:2c:9e:4c:ed:6d:f2:91:96:86:89:35:8a:00:1d:96:91:bd:
         b0:34:94:c4:3f:b5:4d:8f:29:94:76:be:76:a8:e3:8e:14:82:
         6d:0e:fe:dd:e3:c7:9a:e3:f3:5e:db:df:58:50:70:71:d0:d2:
         54:2e:5d:5d:a4:c9:3a:cc:a2:15:9e:2d:fd:9f:2f:e2:b4:c4:
         9f:59:81:70:f8:29:89:d8:d3:36:ab:a6:b7:80:e5:10:5e:3b:
         9c:6c:dc:cb:3e:a0:65:9c:d7:7a:90:7c:c0:71:df:7b:eb:af:
         3d:a6:77:0e:62:2a:38:3b:d6:eb:94:52:6b:43:57:a7:b0:c0:
         66:bf:54:ee:61:36:29:d8:01:fa:20:76:ee:5a:1a:e5:5b:9d:
         31:53:de:7e:39:07:c5:87:cf:6f:bd:af:47:67:6c:0e:f1:51:
         e9:75:4a:e4
```

2. Using the CSR from the previous step as the argument for the `--csr` parameter, issue the root certificate.

> **Note**
>
> If you are using AWS CLI version 1.6.3 or later, use the prefix `fileb://` when specifying the required input file. This ensures that AWS Private CA parses the Base64-encoded data correctly.

```
$ aws acm-pca issue-certificate \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --csr file://ca.csr \
     --signing-algorithm SHA256WITHRSA \
     --template-arn arn:aws:acm-pca:::template/RootCACertificate/V1 \
     --validity Value=365,Type=DAYS
```

3. Retrieve the root certificate.

```
$ aws acm-pca get-certificate \
 --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
 --certificate-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID \
 --output text > cert.pem
```

The resulting file `cert.pem`, a PEM file encoded in base64 format, has the following appearance.

```
-----BEGIN CERTIFICATE-----
MIIDpzCCAo+gAwIBAgIRAIIuOarlQETlUQEOZJGZYdIwDQYJKoZIhvcNAQELBQAw
bTELMAkGA1UEBhMCVVMxFTATBgNVBAoMDEV4YW1wbGUgQ29ycDEOMAwGA1UECwwF
U2FsZXMxCzAJBgNVBAgMAldBMRgwFgYDVQQDDA93d3cuZXhhbXBsZS5jb20xEDAO
BgNVBAcMB1NlYXR0bGUwHhcNMjEwMzA4MTU0NjI3WhcNMjIwMzA4MTY0NjI3WjBt
MQswCQYDVQQGEwJVUzEVMBMGA1UECgwMRXhhbXBsZSBDb3JwMQ4wDAYDVQQLDAVT
YWxlczELMAkGA1UECAwCV0ExGDAWBgNVBAMMD3d3dy5leGFtcGxlLmNvbTEQMA4G
A1UEBwwHU2VhdHRsZTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMP7
t5AKFZQ7abqkeyUjsBVIWRa9tCh8oge9u/LvCbxU738G4jssT+Oud3WMajIjuNow
cpc+0Q/e42ULO/6gTNrTs6OCOo9lV6G0Dprf/e91DWoKgPatem/pUjNyraifHZfu
b5mLHCfahjWXUQtc/sjmDQaZRK3Kar6ljlUBE/Le9NEyOAIkSLPzDtW8LXm4iwcU
BZrb828rKd1Aw9oI1+3bfzB6xXmzZxc5RLXveOCEhKGD32jKZ/RNFSC8AZAwJe+x
bTsys/lUOYFTuT8Bn0TGxR8x7Y4H75+F9BavY3v+WkLj4M+olN9dMR7Et9FMt4u4
```

```
YRokv5zp8zIb5iTne1kCAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAdBgNVHQ4E
FgQUaW3+r328uTLokog2TklmoBK+yt4wDgYDVR0PAQH/BAQDAgGGMA0GCSqGSIb3
DQEBCwUAA4IBAQAXjd/7UZ8RDE+PLWSDNGQdLemOBTcawF+tK+PzA4Evlmn9VuNc
g+x3oZvVZSDQBANUz0b9oPeo54aE38dW1zQm2qfTab8822aqeWMLyJ1dMsAgqYX2
t9+u6w3NzRCw8Pvz18V69+dFE5AeXmNP0Z5/gdz8H/NSpctjlzopbScRZKCSlPid
Rf3ZOPm9QP92YpWyYDkfAU04xdDo1vR0MYjKPkl4LjRqSU/tcCJnPMbJiwq+bWpX
2WJoEBXB/p15Kn6JxjI0ze2SnSI48JZ8it4fvxrhOo0VoLNIuCuNXJOwU17Rdl1W
YJidaq7je6k18AdgPA0Kh8y1XtfUH3fTaVw4
-----END CERTIFICATE-----
```

You can use OpenSSL to view and verify the contents of the certificate.

```
openssl x509 -in cert.pem -text -noout
```

This yields output similar to the following.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            82:2e:39:aa:e5:40:44:e5:51:01:0e:64:91:99:61:d2
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, O=Example Corp, OU=Sales, ST=WA, CN=www.example.com,
 L=Seattle
        Validity
            Not Before: Mar  8 15:46:27 2021 GMT
            Not After : Mar  8 16:46:27 2022 GMT
        Subject: C=US, O=Example Corp, OU=Sales, ST=WA, CN=www.example.com,
 L=Seattle
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c3:fb:b7:90:0a:15:94:3b:69:ba:a4:7b:25:23:
                    b0:15:48:59:16:bd:b4:28:7c:a2:07:bd:bb:f2:ef:
                    09:bc:54:ef:7f:06:e2:3b:2c:4f:e3:ae:77:75:8c:
                    6a:32:23:b8:da:30:72:97:3e:d1:0f:de:e3:65:0b:
                    3b:fe:a0:4c:da:d3:b3:a3:82:3a:8f:65:57:a1:b4:
                    0e:9a:df:fd:ef:75:0d:6a:0a:80:f6:ad:7a:6f:e9:
                    52:33:72:ad:a8:9f:1d:97:ee:6f:99:8b:1c:27:da:
                    86:35:97:51:0b:5c:fe:c8:e6:0d:06:99:44:ad:ca:
                    6a:be:a5:8e:55:01:13:f2:de:f4:d1:32:38:02:24:
```

```
                    48:b3:f3:0e:d5:bc:2d:79:b8:8b:07:14:05:9a:db:
                    f3:6f:2b:29:dd:40:c3:da:08:d7:ed:db:7f:30:7a:
                    c5:79:b3:67:17:39:44:b5:ef:78:e0:84:84:a1:83:
                    df:68:ca:67:f4:4d:15:20:bc:01:90:30:25:ef:b1:
                    6d:3b:32:b3:f9:54:39:81:53:b9:3f:01:9f:44:c6:
                    c5:1f:31:ed:8e:07:ef:9f:85:f4:16:af:63:7b:fe:
                    5a:42:e3:e0:cf:a8:94:df:5d:31:1e:c4:b7:d1:4c:
                    b7:8b:b8:61:1a:24:bf:9c:e9:f3:32:1b:e6:24:e7:
                    7b:59
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Subject Key Identifier:
                69:6D:FE:AF:7D:BC:B9:32:E8:92:88:36:4E:49:66:A0:12:BE:CA:DE
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         17:8d:df:fb:51:9f:11:0c:4f:8f:2d:64:83:34:64:1d:2d:e9:
         8e:05:37:1a:c0:5f:ad:2b:e3:f3:03:81:2f:96:69:fd:56:e3:
         5c:83:ec:77:a1:9b:d5:65:20:d0:04:03:54:cf:46:fd:a0:f7:
         a8:e7:86:84:df:c7:56:d7:34:26:da:a7:d3:69:bf:3c:db:66:
         aa:79:63:0b:c8:9d:5d:32:c0:20:a9:85:f6:b7:df:ae:eb:0d:
         cd:cd:10:b0:f0:fb:f3:d7:c5:7a:f7:e7:45:13:90:1e:5e:63:
         4f:d1:9e:7f:81:dc:fc:1f:f3:52:a5:cb:63:97:3a:29:6d:27:
         11:64:a0:92:94:f8:9d:45:fd:d9:38:f9:bd:40:ff:76:62:95:
         b2:60:39:1f:01:4d:38:c5:d0:e8:d6:f4:74:31:88:ca:3e:49:
         78:2e:34:6a:49:4f:ed:70:22:67:3c:c6:c9:8b:0a:be:6d:6a:
         57:d9:62:68:10:15:c1:fe:9d:79:2a:7e:89:c6:32:34:cd:ed:
         92:9d:22:38:f0:96:7c:8a:de:1f:bf:1a:e1:3a:8d:15:a0:b3:
         48:b8:2b:8d:5c:93:b0:53:5e:d1:76:5d:56:60:98:9d:6a:ae:
         e3:7b:a9:35:f0:07:60:3c:0d:0a:87:cc:b5:5e:d7:d4:1f:77:
         d3:69:5c:38
```

4. Import the root CA certificate to install it on the CA.

> **Note**
>
> If you are using AWS CLI version 1.6.3 or later, use the prefix `fileb://` when specifying the required input file. This ensures that AWS Private CA parses the Base64-encoded data correctly.

```
$ aws acm-pca import-certificate-authority-certificate \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --certificate file://cert.pem
```

Inspect the new status of the CA.

```
$ aws acm-pca describe-certificate-authority \
 --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
 --output json
```

The status now appears as ACTIVE.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-05T14:24:12.867000-08:00",
        "LastStateChangeAt": "2021-03-08T12:37:14.235000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T07:46:27-08:00",
        "NotAfter": "2022-03-08T08:46:27-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
```

```
                "ExpirationInDays": 7,
                "CustomCname": "alternative.example.com",
                "S3BucketName": "amzn-s3-demo-bucket"
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

Install a subordinate CA certificate hosted by AWS Private CA

You can use the AWS Management Console to create and install a certificate for your AWS Private CA hosted subordinate CA.

To create and install a certificate for your AWS Private CA hosted subordinate CA

1. (Optional) If you are not already on the CA's details page, open the AWS Private CA console at https://console.aws.amazon.com/acm-pca/home. On the **Private certificate authorities** page, choose a subordinate CA with status **Pending certificate** or **Active**.
2. Choose **Actions**, **Install CA Certificate** to open the **Install subordinate CA certificate** page.
3. On the **Install subordinate CA certificate** page, under **Select CA type**, choose **AWS Private CA** to install a certificate that is managed by AWS Private CA.
4. Under **Select parent CA**, choose a CA from the **Parent private CA** list. The list is filtered to display CAs that meet the following criteria:

   - You have permission to use the CA.
   - The CA would not be signing itself.
   - The CA is in state `ACTIVE`.
   - The CA mode is `GENERAL_PURPOSE`.
5. Under **Specify the subordinate CA certificate parameters**, specify the following certificate parameters:

   - **Validity** — Specifies the expiration date and time for the CA certificate.
   - **Signature algorithm** — Specifies the signing algorithm to use when the root CA issues new certificates. Options are:
     - SHA256 RSA

- SHA384 RSA
- SHA512 RSA

- **Path length** — The number of trust layers that the subordinate CA can add when signing new certificates. A path length of zero (the default) means that only end-entity certificates, and not CA certificates, can be created. A path length of one or more means that the subordinate CA may issue certificates to create additional CAs subordinate to it.
- **Template ARN** — Displays the ARN of the configuration template for this CA certificate. The template changes if you change the specified **Path length**. If you create a certificate using the CLI issue-certificate command or API IssueCertificate action, you must specify the ARN manually. For information about available CA certificate templates, see Use AWS Private CA certificate templates.

6. Review your settings for correctness, then choose **Confirm and install**. AWS Private CA exports a CSR, generates a certificate using a subordinate CA certificate template, and signs the certificate it with the selected parent CA. AWS Private CA then imports the signed subordinate CA certificate.
7. The details page for the CA displays the status of the installation (success or failure) at the top. If the installation was successful, the newly completed subordinate CA displays a status of **Active** in the **General** pane.

Install a subordinate CA certificate signed by an external parent CA

After you create a subordinate private CA as described in Create a private CA in AWS Private CA, you have the option of activating it by installing a CA certificate signed by an external signing authority. Signing your subordinate CA certificate with an external CA requires that you first set up an external trust services provider as your signing authority, or arrange for the use of a third-party provider.

> **Note**
>
> Procedures for creating or obtaining an external trust services provider are outside the scope of this guide.

After you have created a subordinate CA and you have access to an external signing authority, complete the following tasks:

1. Obtain a certificate signing request (CSR) from AWS Private CA.
2. Submit the CSR to your external signing authority and obtain a signed CA certificate along with any chain certificates.
3. Import the CA certificate and chain into AWS Private CA to activate your subordinate CA.

For detailed procedures, see Use externally signed private CA certificates .

Control access to the private CA

Any user with the necessary permissions on a private CA from AWS Private CA can use that CA to sign other certificates. The CA owner can issue certificates or delegate the required permissions for issuing certificates to an AWS Identity and Access Management (IAM) user that resides in the same AWS account. A user that resides in a different AWS account can also issue certificates if authorized by the CA owner through a resource-based policy.

Authorized users, whether single-account or cross-account, can use AWS Private CA or AWS Certificate Manager resources when issuing certificates. Certificates that are issued from the AWS Private CA IssueCertificate API or issue-certificate CLI command are unmanaged. Such certificates require manual installation on target devices and manual renewal when they expire. Certificates issued from the ACM console, the ACM RequestCertificate API, or the request-certificate CLI command are managed. Such certificates can easily be installed in services that are integrated with ACM. If the CA administrator permits it and the issuer's account has a service-linked role in place for ACM, managed certificates are renewed automatically when they expire.

Topics

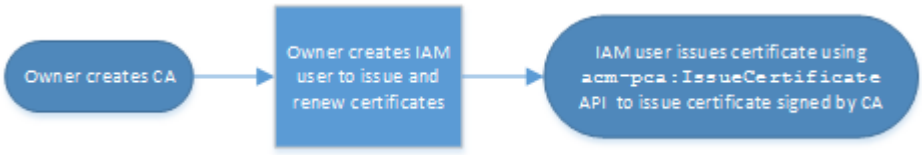
- Create single-account permissions for an IAM user
- Attach a policy for cross-account access

Create single-account permissions for an IAM user

When the CA administrator (that is, the owner of the CA) and the certificate issuer reside in a single AWS account, a best practice is to separate the issuer and administrator roles by creating an AWS Identity and Access Management (IAM) user with limited permissions. For information about using IAM with AWS Private CA, along with example permissions, see Identity and Access Management (IAM) for AWS Private Certificate Authority.

Single-account case 1: Issuing an unmanaged certificate

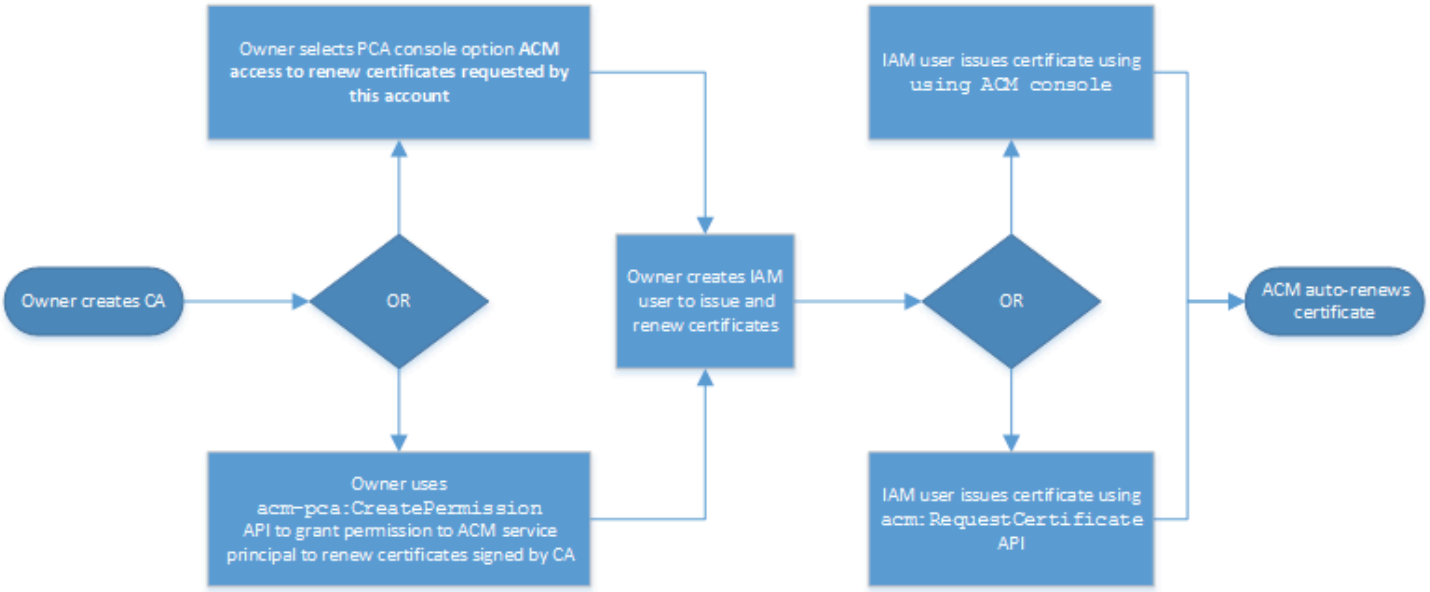
In this case, the account owner creates a private CA and then creates an IAM user with permission to issue certificates signed by the private CA. The IAM user issues a certificate by calling the AWS Private CA `IssueCertificate` API.



Certificates issued in this manner are unmanaged, which means that an administrator must export them and install them on devices where they are intended to be used. They also must be manually renewed when they expire. Issuing a certificate using this API requires a certificate signing request (CSR) and key pair that is generated outside of AWS Private CA by OpenSSL or a similar program. For more information, see the `IssueCertificate` documentation.

Single-account case 2: Issuing a managed certificate through ACM

This second case involves API operations from both ACM and PCA. The account owner creates a private CA and IAM user as before. The account owner then grants permission to the ACM service principal to renew automatically any certificates that are signed by this CA. The IAM user again issues the certificate, but this time by calling the ACM `RequestCertificate` API, which handles CSR and key generation. When the certificate expires, ACM automates the renewal workflow.



The account owner has the option of granting renewal permission through the management console during or after CA creation or using the PCA `CreatePermission` API. The managed

certificates created from this workflow are available for use on with AWS services that are integrated with ACM.

The following section contains procedures for granting renewal permissions.

Assign certificate renewal permissions to ACM

With managed renewal in AWS Certificate Manager (ACM), you can automate the certificate renewal process for both public and private certificates. In order for ACM to automatically renew the certificates generated by a private CA, the ACM service principal must be given all possible permissions *by the CA itself*. If these renewal permissions are not present for ACM, the CA's owner (or an authorized representative) must manually reissue each private certificate when it expires.

> **Important**
>
> These procedures for assigning renewal permissions apply only when the CA owner and the certificate issuer reside in the same AWS account. For cross-account scenarios, see Attach a policy for cross-account access.

Renewal permissions can be delegated during private CA creation or altered anytime after as long as the CA is in the `ACTIVE` state.

You can manage private CA permissions from the AWS Private CA Console, the AWS Command Line Interface (AWS CLI), or the AWS Private CA API:

To assign private CA permissions to ACM (console)

1. Sign in to your AWS account and open the AWS Private CA console at https://console.aws.amazon.com/acm-pca/home.
2. On the **Private certificate authorities page**, choose your private CA from the list.
3. Choose **Actions**, **Configure CA permissions**.
4. Select **Authorize ACM access to renew certificates requested by this account**.
5. Choose **Save**.

To manage ACM permissions in AWS Private CA (AWS CLI)

Use the create-permission command to assign permissions to ACM. You must assign the necessary permissions (`IssueCertificate`, `GetCertificate`, and `ListPermissions`) in order for ACM to automatically renew your certificates.

```
$ aws acm-pca create-permission \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --actions IssueCertificate GetCertificate ListPermissions \
     --principal acm.amazonaws.com
```

Use the list-permissions command to list the permissions delegated by a CA.

```
$ aws acm-pca list-permissions \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID
```

Use the delete-permission command to revoke permissions assigned by a CA to an AWS service principal.

```
$ aws acm-pca delete-permission \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --principal acm.amazonaws.com
```

Attach a policy for cross-account access

When the CA administrator and the certificate issuer reside in different AWS accounts, the CA administrator must share CA access. This is accomplished by attaching a resource-based policy to the CA. The policy grants issuance permissions to a specific principal, which can be an AWS account owner, an IAM user, an AWS Organizations ID, or an organizational unit ID.

A CA administrator can attach and manage policies in the following ways:

- In the management console, using AWS Resource Access Manager (RAM), which is a standard method for sharing AWS resources across accounts. When you share a CA resource in AWS RAM with a principal in another account, the required resource-based policy is attached to the CA automatically. For more information about RAM, see the AWS RAM User Guide.

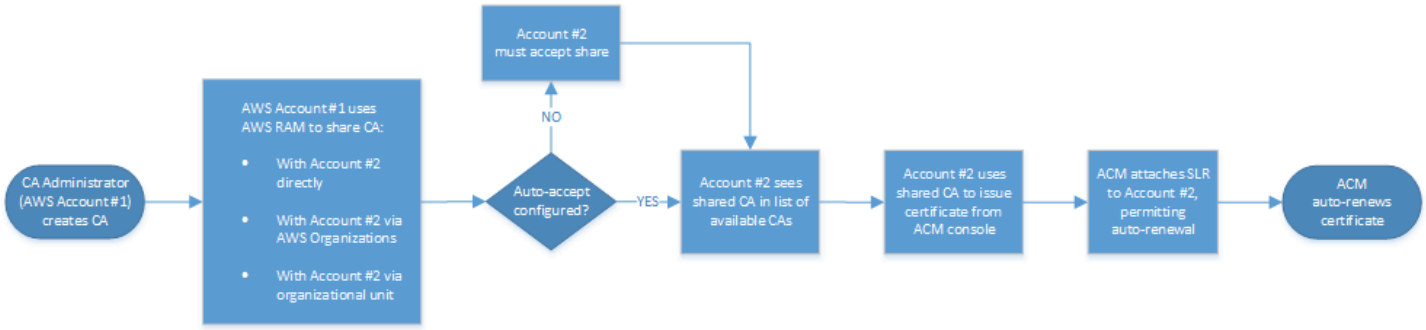
> **Note**
>
> You can easily open the RAM console by choosing a CA and then choosing **Actions**, **Manage resource shares**.

- Programmatically, using the PCA APIs PutPolicy, GetPolicy, and DeletePolicy.
- Manually, using the PCA commands put-policy, get-policy, and delete-policy in the AWS CLI.

Only the console method requires RAM access.

Cross-account case 1: Issuing a managed certificate from the console

In this case, the CA administrator uses AWS Resource Access Manager (AWS RAM) to share CA access with another AWS account, which allows that account to issue managed ACM certificates. The diagram shows that AWS RAM can share the CA directly with the account, or indirectly through an AWS Organizations ID in which the account is a member.

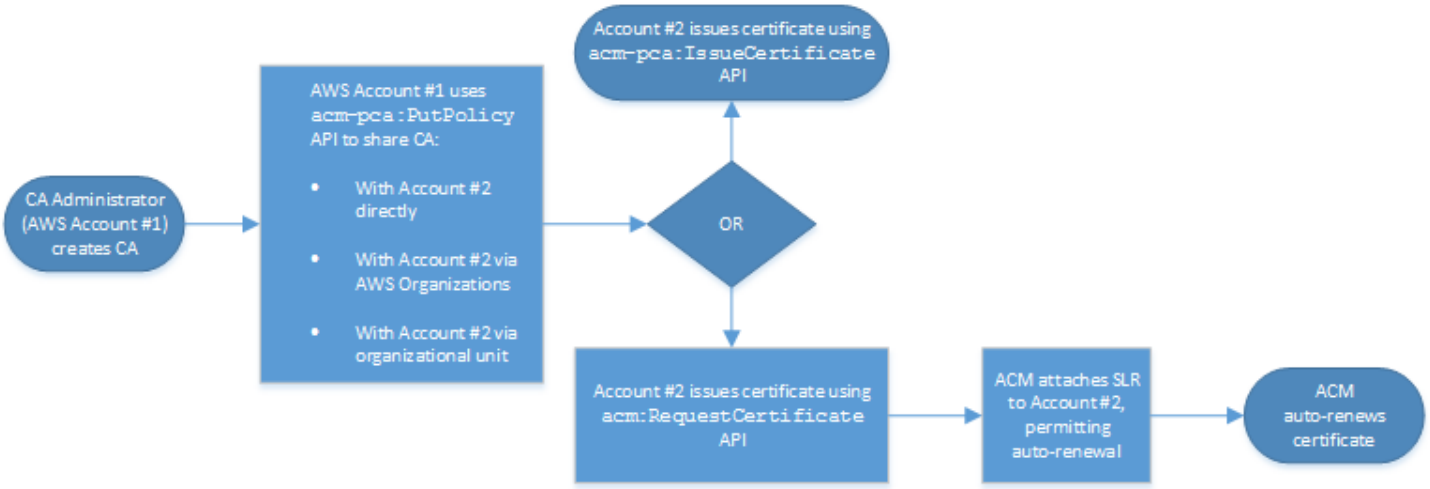


After RAM shares a resource through AWS Organizations, the recipient principal must accept the resource for it to take effect. The recipient can configure AWS Organizations to accept offered shares automatically.

> **Note**
>
> The recipient account is responsible for configuring autorenewal in ACM. Typically, on the first occasion a shared CA is used, ACM installs a service-linked role that permits it to make unattended certificate calls on AWS Private CA. If this fails (usually due to a missing permission), certificates from the CA are not renewed automatically. Only the ACM user can resolve the problem, not the CA administrator. For more information, see Using a Service Linked Role (SLR) with ACM.

Cross-account case 2: Issuing managed and unmanaged certificates using the API or CLI

This second case demonstrates the sharing and issuance options that are possible using the AWS Certificate Manager and AWS Private CA API. All of these operations can also be carried out using the corresponding AWS CLI commands.



Because the API operations are being used directly in this example, the certificate issuer has a choice of two API operations to issue a certificate. The PCA API action `IssueCertificate` results in an unmanaged certificate that will not be automatically renewed and must be exported and manually installed. The ACM API action RequestCertificate results in a managed certificate that can be easily installed on ACM integrated services and renews automatically.

> **Note**
>
> The recipient account is responsible for configuring auto-renewal in ACM. Typically, on the first occasion a shared CA is used, ACM installs a service-linked role that allows it to make unattended certificate calls on AWS Private CA. If this fails (usually due to a missing permission), certificates from the CA will not renew automatically, and only the ACM user can resolve the problem, not the CA administrator. For more information, see Using a Service Linked Role (SLR) with ACM.

List private CAs

You can use the AWS Private CA console or AWS CLI to list private CAs that you own or have access to.

To list available CAs using the console

1. Sign in to your AWS account and open the AWS Private CA console at https://console.aws.amazon.com/acm-pca/home.
2. Review the information in the **Private certificate authorities** list. You can navigate through multiple pages of CAs using the page numbers at upper-right. Each CA occupies a row with some or all of the following columns displayed for each one:

- **Subject** – Summary of distinguished name information for the CA.
- **Id** – 32-byte hexadecimal unique identifier of the CA.
- **Status** – CA status. Possible values are **Creating**, **Pending certificate**, **Active**, **Deleted**, **Disabled**, **Expired**, and **Failed**.
- **Type** – The type of CA. Possible values are **Root** and **Subordinate**.
- **Mode** – The mode of the CA. Possible values are **General-purpose** (issues certificates that can be configured with any expiration date) and **Short-lived certificate** (issues certificates with a maximum validity period of seven days). A short validity period can substitute in some cases for a revocation mechanism. The default is **General-purpose**.
- **Owner** – The AWS account that owns the CA. This may be your account or an account that has delegated CA management permissions to you.
- **Key algorithm** – The public key algorithm supported by the CA. Possible values are **ML_DSA_44**, **ML_DSA_65**, **ML_DSA_87**, **RSA_2048**, **RSA_3072**, **RSA_4096**, **EC_prime256v1**, **EC_secp384r1**, and **EC_secp521r1**.
- **Signing algorithm** – The algorithm that the CA uses to sign certificate requests. (Not to be confused with the `SigningAlgorithm` parameter used to sign certificates when they are issued.) Possible values are **ML_DSA_44**, **ML_DSA_65**, **ML_DSA_87**, **SHA256WITHRSA**, **SHA384WITHRSA**, **SHA512WITHRSA**, **SHA256WITHECDSA**, **SHA384WITHECDSA**, and **SHA512WITHECDSA**.

> **Note**
>
> You can customize the columns that you want to display, as well as other settings, by choosing the settings icon in the upper-right corner of the console.

To list available CAs using the AWS CLI

Use the list-certificate-authorities command to list available CAs as shown in the following example:

```
$ aws acm-pca list-certificate-authorities --max-items 10
```

The command returns information similar to the following:

```
{
   "CertificateAuthorities":[
      {
         "Arn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
         "CreatedAt":"2022-05-02T11:59:02.022000-07:00",
         "LastStateChangeAt":"2022-05-02T11:59:18.498000-07:00",
         "Type":"ROOT",
         "Serial":"serial_number",
         "Status":"ACTIVE",
         "NotBefore":"2022-05-02T10:59:17-07:00",
         "NotAfter":"2032-05-02T11:59:17-07:00",
         "CertificateAuthorityConfiguration":{
            "KeyAlgorithm":"RSA_2048",
            "SigningAlgorithm":"SHA256WITHRSA",
            "Subject":{
               "Organization":"testing_com"
            }
         },
         "RevocationConfiguration":{
            "CrlConfiguration":{
               "Enabled":false
            }
         }
      }
 ...
   ]
}
```

View a private CA

You can use the ACM console or the AWS CLI to view detailed metadata about a private CA and change several of the values as needed. For detailed information about updating CAs, see Update a private CA in AWS Private Certificate Authority.

To view CA details in the console

1. Sign in to your AWS account and open the AWS Private CA console at https://console.aws.amazon.com/acm-pca/home.
2. Review the **Private certificate authorities** list. You can navigate through multiple pages of CAs using the page numbers at upper-right.
3. To show detailed metadata for a listed CA, choose the radio button by the CA that you want to inspect. This opens a details pane with the following tabbed views:

   - **Subject** tab– Information about the distinguished name for the CA. For more information, see Subject distinguished name. The fields displayed include:
     - **Subject** – Summary of provided name information fields
     - **Organization (O)** – For example, a company name
     - **Organization Unit (OU)** – For example, a division within a company
     - **Country name (C)** – A two-letter country code
     - **State or province name** – Full name of a state or province
     - **Locality name** – The name of a city
     - **Common Name (CN)** – A human-readable string to identify the CA.
   - **CA certificate** tab – Information about the validity of the CA certificate
     - **Valid until** – The date and time until the CA certificate is valid
     - **Expires in** – The number of days until expiration
   - **Revocation configuration** tab – Your current selections for certificate revocation options. Choose **Edit** to update.
     - **Certificate Revocation List (CRL) distribution** – Status of **Enabled** or **Disabled**
     - **Online Certificate Status Protocol (OCSP)** – Status of **Enabled** or **Disabled**
   - **Permissions** tab – Your current selection of certificate renewal permisisons for this CA through AWS Certificate Manager (ACM). Choose **Edit** to update.
   - **ACM authorization for renewals** – Status of authorized or unauthorized
   - **Tags** tab – Your current assignment of customizable labels for this CA. Choose the **Manage tags** to update.
   - **Resource shares** tab – Your current assignment of resource shares for this CA through AWS Resource Access Manager (RAM). Choose **Manage resource shares** to update.
     - **Name** – Name of the resource share

- **Status** – status of the resource share

4. Choose the **ID** field of the CA that you want to inspect to open the **General** pane. The CA's 32-byte hexadecimal unique identifier appears at the top. The pane provides the following additional information:

   - **Status** – CA status. Possible values are **Creating**, **Pending certificate**, **Active**, **Deleted**, **Disabled**, **Expired**, and **Failed**.
   - **ARN** – The Amazon Resource Name for the CA.
   - **Owner** – The AWS account that owns the CA. This may be your account (**Self**) or an account that has delegated CA management permissions to you.
   - **CA type** – The type of CA. Possible values are **Root** and **Subordinate**.
   - **Created at** – The date and time when the CA was created.
   - **Expiration date** – The date and time when the CA certificate expires.
   - **Mode** – The mode of the CA. Possible values are **General-purpose** (certificates that can be configured with any expiration date) and **Short-lived certificate** (certificates with a maximum validity period of seven days). A short validity period can substitute in some cases for a revocation mechanism. The default is **General-purpose**.
   - **Key algorithm** – The public key algorithm supported by the CA. Possible values are **ML-DSA-44**, **ML-DSA-65**, **ML-DSA-87**, **RSA 2048**, **RSA 3072**, **RSA 4096**, **ECDSA P256**, **ECDSA P384**, and **ECDSA P521**.
   - **Signing algorithm** – The algorithm that the CA uses to sign its own Certificate Signing Request, CRLs, and OCSP responses (Not to be confused with the `SigningAlgorithm` parameter used in the IssueCertificate API.) Possible values are **ML-DSA-44**, **ML-DSA-65**, **ML-DSA-87**, **SHA256 RSA**, **SHA384 RSA**, and **SHA512 RSA**, **SHA256 ECDSA**, **SHA384 ECDSA**, **SHA512 ECDSA**
   - **Key storage security standard** – Level of Federal Information Processing Standards (FIPS) conformance. You can choose from these values: **FIPS 140-2 level 2 or higher**, **FIPS 140-2 level 3 or higher**, and **CCPC Level 1 or higher**. This parameter varies by AWS Region.

     > **Note**
     >
     > Starting January 26, 2023, AWS Private CA protects all CA private keys in non-China regions using hardware security modules (HSMs) that comply with FIPS PUB 140-2 Level 3.

To view and modify CA details using the AWS CLI

Use the **describe-certificate-authority** command in the AWS CLI to display details about a CA, as shown in the following command:

```
$ aws acm-pca describe-certificate-authority --certificate-authority-arn
 arn:aws:acm:region:account:certificate-authority/CA_ID
```

The command returns information similar to the following:

```
{
   "CertificateAuthority":{
      "Arn":"arn:aws:acm:region:account:certificate-authority/CA_ID",
      "CreatedAt":"2022-05-02T11:59:02.022000-07:00",
      "LastStateChangeAt":"2022-05-02T11:59:18.498000-07:00",
      "Type":"ROOT",
      "Serial":"serial_number",
      "Status":"ACTIVE",
      "NotBefore":"2022-05-02T10:59:17-07:00",
      "NotAfter":"2031-05-02T11:59:17-07:00",
      "CertificateAuthorityConfiguration":{
         "KeyAlgorithm":"RSA_2048",
         "SigningAlgorithm":"SHA256WITHRSA",
         "Subject":{
            "Organization":"testing_com"
         }
      },
      "RevocationConfiguration":{
         "CrlConfiguration":{
            "Enabled":false
         }
      }
   }
}
```

For information about updating a private CA from the command line, see Updating a CA (CLI).

Add tags for your private CA

Tags are words or phrases that act as metadata for identifying and organizing AWS resources. Each tag consists of a **key** and a **value**. You can use the AWS Private CA console, AWS Command Line Interface (AWS CLI), or the PCA API to add, view, or remove tags for private CAs.

You can add or remove custom tags for your private CA at any time. For example, you could tag private CAs with key-value pairs like `Environment=Prod` or `Environment=Beta` to identify which environment the CA is intended for. For more information, see Create a Private CA.

> **Note**
>
> To attach tags to a private CA during the creation procedure, a CA administrator must first associate an inline IAM policy with the `CreateCertificateAuthority` action and explicitly allow tagging. For more information, see Tag-on-create: Attaching tags to a CA at the time of creation.

Other AWS resources also support tagging. You can assign the same tag to different resources to indicate that those resources are related. For example, you can assign a tag such as `Website=example.com` to your CA, the Elastic Load Balancing load balancer, and other related resources. For more information on tagging AWS resources, see Tagging your Amazon EC2 Resources in the Amazon EC2 User Guide.

The following basic restrictions apply to AWS Private CA tags:

- The maximum number of tags per private CA is 50.
- The maximum length of a tag key is 128 characters.
- The maximum length of a tag value is 256 characters.
- The tag key and value can contain the following characters: A-Z, a-z, and .:+=@_%-(hyphen).
- Tag keys and values are case-sensitive.
- The `aws:` and `rds:` prefixes are reserved for AWS use; you cannot add, edit, or delete tags whose key begins with `aws:` or `rds:`. Default tags that begin with `aws:` and `rds:` do not count against your tags-per-resource quota.
- If you plan to use your tagging schema across multiple services and resources, remember that other services might have different restrictions for allowed characters. Refer to the documentation for that service.
- AWS Private CA tags are not available for use in the Resource Groups and Tag Editor in the AWS Management Console.

You can tag a private CA from the AWS Private CA Console, the AWS Command Line Interface (AWS CLI), or the AWS Private CA API.

To tag a private CA (console)

1. Sign in to your AWS account and open the AWS Private CA console at https://console.aws.amazon.com/acm-pca/home.
2. On the **Private certificate authorities page**, choose your private CA from the list.
3. In the details area below the list, choose the **Tags** tab. A list of existing tags is displayed.
4. Choose **Manage tags**.
5. Choose **Add new tag**.
6. Type a key and value pair.
7. Choose **Save**.

To tag a private CA (AWS CLI)

Use the tag-certificate-authority command to add tags to your private CA.

```
$ aws acm-pca tag-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --tags Key=Admin,Value=Alice
```

Use the list-tags command to list the tags for a private CA.

```
$ aws acm-pca list-tags \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --max-results 10
```

Use the untag-certificate-authority command to remove tags from a private CA.

```
$ aws acm-pca untag-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:aregion:account:certificate-
authority/CA_ID \
     --tags Key=Purpose,Value=Website
```

Understand AWS Private CA CA status

The status of a CA that is managed by AWS Private CA results from a user action or, in some cases, from a service action. For example, a CA status changes when it expires. The status options available to CA administrators vary depending on the current status of the CA.

AWS Private CA can report the following status values. The table shows the CA capabilities available in each state.

> **Note**
>
> For all status values except `DELETED` and `FAILED`, you are billed for the CA.

Status	Issue certificates	Validat certs with OCSP	Genera CRLs	Genera audits	You can update the CA cert	Certificates can be revoked	You are billed for the CA
`CREATING` – The CA is being created.	No	No	No	No	No	No	Yes
`PENDING_CERTIFICATE` – The CA has been created and needs a certificate to be operational.*	No	No	No	No	No	No	Yes
`ACTIVE`	Yes	Yes	Yes	Yes	Yes	Yes	Yes
`DISABLED` – You have manually disabled the CA.	No	Yes	Yes	Yes	No	Yes	Yes
`EXPIRED` – The CA certificate has expired.**	No	No	No	No	Yes	No	Yes

Status	Issue certificates	Validate certs with OCSP	Generate CRLs	Generate audits	You can update the CA cert	Certificates can be revoked	You are billed for the CA
FAILED	The CreateCertificateAuthority action failed. This can occur because of a network outage, backend AWS failure, or other errors. A failed CA cannot be recovered. Delete the CA and create a new one.						No
DELETED	Your CA is within the restoration period, which can have a length of 7-30 days. After this period, it is permanently deleted. • If you call the RestoreCertificateAuthority API on a CA with DELETED status and an expired certificate, the CA will be set to EXPIRED. • For more information about deleting a CA, see Delete your private CA.						No

To complete activation, you need to generate a CSR, get a signed CA certificate from a CA, and import the certificate into AWS Private CA. The CSR can be submitted either to your new CA (for self-signing), or to an on-premises root or subordinate CA. For more information, see Installing the CA certificate.

You cannot directly change the status of an expired CA. If you import a new certificate for the CA, AWS Private CA resets the status to ACTIVE unless it was set to DISABLED before the certificate expired.

Additional considerations about expired CA certificates:

- CA certificates are not automatically renewed. For information about automating renewal through AWS Certificate Manager, see Assign certificate renewal permissions to ACM.

- If you attempt to issue a new certificate with an expired CA, the `IssueCertificate` API returns `InvalidStateException`. An expired root CA must self-sign a new root CA certificate before it can issue new subordinate certificates.
- The `ListCertificateAuthorities` and `DescribeCertificateAuthority` APIs return a status of `EXPIRED` if the CA certificate is expired, regardless of whether the CA status is set to `ACTIVE` or `DISABLED`. However, if the expired CA has been set to `DELETED`, the status returned is `DELETED`.
- The `UpdateCertificateAuthority` API cannot update the status of an expired CA.
- The `RevokeCertificate` API cannot be used to revoke any expired certificate, including a CA certificate.

Relation between CA status and CA lifecycle

The following diagram illustrates the CA lifecycle as an interaction of management actions with CA status.

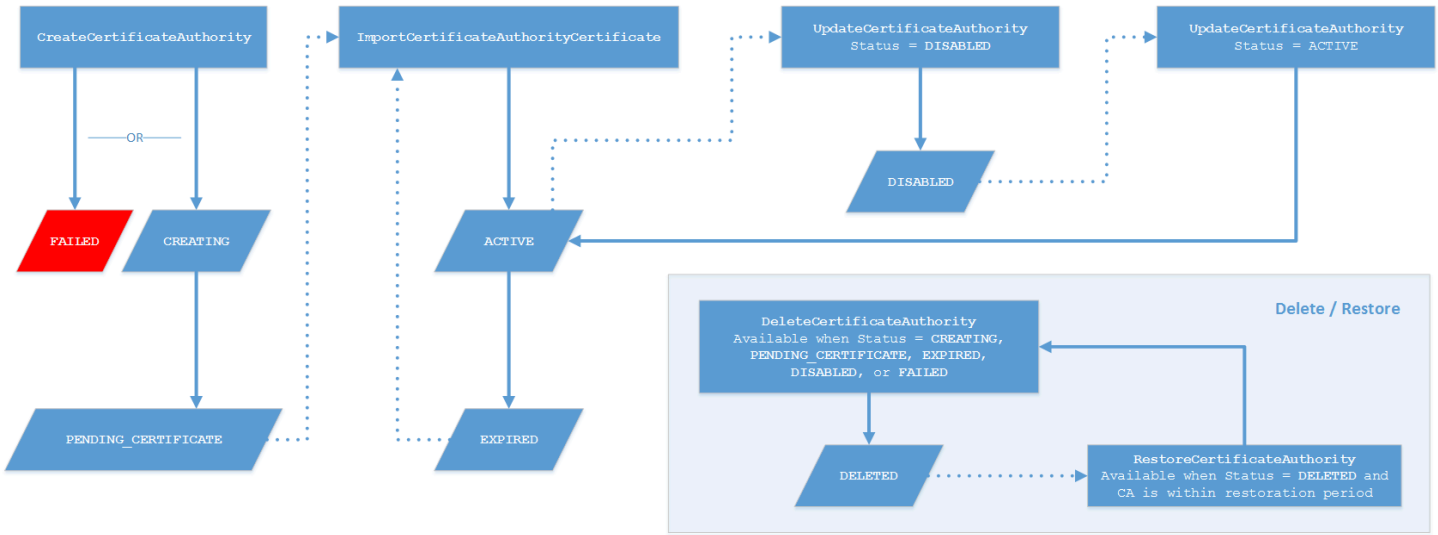


Diagram key

At the top of the diagram, management actions are applied through the AWS Private CA console, CLI, or API. The actions take the CA through creation, activation, expiration and renewal. The CA status changes in response (as shown by the solid lines) to manual actions or automated updates. In most cases, a new status leads to a new possible action (shown by a dotted line) that the CA administrator can apply. The lower-right inset shows the possible status values permitting delete and restore actions.

Update a private CA in AWS Private Certificate Authority

You can update the status of a private CA or change its revocation configuration after creating it. This topic provides details about CA status and the CA lifecycle, along with examples of console and CLI updates to CAs.

Update a CA (console)

The following procedures show how to update existing CA configurations using the AWS Management Console.

Update CA status (console)

In this example, the status of an enabled CA is changed to disabled.

To update the status of a CA

1. Sign in to your AWS account and open the AWS Private CA console at https://console.aws.amazon.com/acm-pca/home
2. On the **Private certificate authorities** page, choose a private CA that is currently active from the list.
3. On the **Actions** menu, choose **Disable** to disable the private CA.

Updating a CA's revocation configuration (console)

You can update the revocation configuration for your private CA, for example, by adding or removing either OCSP or CRL support, or by modifying their settings.

> **Note**
>
> Changes to the revocation configuration of a CA do not affect certificates that were already issued. For managed revocation to work, older certificates must be re-issued.

For OCSP, you change the following settings:

- Enable or disable OCSP.
- Enable or disable a custom OCSP fully qualified domain name (FQDN).
- Change the FQDN.

For a CRL, you can change any of the following settings:

- The CRL type (complete or partitioned)
- Whether the private CA generates a certificate revocation list (CRL)
- The number of days before a CRL expires. Note that AWS Private CA begins trying to regenerate the CRL at ½ the number of days you specify.
- The name of the Amazon S3 bucket where your CRL is saved.
- An alias to hide the name of your Amazon S3 bucket from public view.

> **Important**
>
> Changing any of the preceding parameters can have negative effects. Examples include disabling CRL generation, changing the validity period, or changing the S3 bucket after you have placed your private CA in production. Such changes can break existing certificates that depend on the CRL and the current CRL configuration. Changing the alias can be done safely as long as the old alias remains linked to the correct bucket.

To update the revocation settings

1. Sign in to your AWS account and open the AWS Private CA console at https://console.aws.amazon.com/acm-pca/home.
2. On the **Private certificate authorities** page, choose a private CA from the list. This opens the details panel for the CA.

3. Choose the **Revocation configuration** tab, then choose **Edit**.
4. Under **Certificate revocation options**, two options are displayed:

   - **Activate CRL distribution**
   - **Turn on OCSP**

   You can configure either, neither, or both of these revocation mechanisms for your CA. Although optional, managed revocation is recommended as a best practice. Before completing this step, see Plan your AWS Private CA certificate revocation method for information about the advantages of each method, the preliminary setup that may be required, and additional revocation features.

To configure a CRL

1. Select **Activate CRL distribution**.
2. To create an Amazon S3 bucket for your CRL entries, select **Create a new S3 bucket**. Provide a unique bucket name. (You do not need to include the path to the bucket.) Otherwise, leave this option unselected and choose an existing bucket from the **S3 bucket name** list.

   If you create a new bucket, AWS Private CA creates and attaches the required access policy to it. If you decide to use an existing bucket, you must attach an access policy it before you can begin generating CRLs. Use one of the policy patterns described in Access policies for CRLs in Amazon S3 . For information about attaching a policy, see Adding a bucket policy by using the Amazon S3 console.

   > **Note**
   >
   > When you are using the AWS Private CA console, an attempt to create a CA fails if both of the following conditions apply:
   >
   > - You are enforcing Block Public Access settings on your Amazon S3 bucket or account.
   > - You asked AWS Private CA to create an Amazon S3 bucket automatically.
   >
   > In this situation, the console attempts, by default, to create a publicly accessible bucket, and Amazon S3 rejects this action. Check your Amazon S3 settings if this occurs. For more information, see Blocking public access to your Amazon S3 storage.

3. Expand **Advanced** for additional configuration options.

   - Choose **Enable partitioning** to enable partitioning of CRLs. If you don't enable partitioning, your CA is subject to the maximum number of revoked certificates, shown on the AWS Private Certificate Authority quotas. For more information about partitioned CRLs, see CRL types.
   - Add a **Custom CRL Name** to create an alias for your Amazon S3 bucket. This name is contained in certificates issued by the CA in the “CRL Distribution Points" extension that is defined by RFC 5280. To use CRLs over IPv6, set this to your bucket's dualstack S3 endpoint as described in Using CRLs over IPv6.
   - Add a **Custom path** to create a DNS alias for the file path in your Amazon S3 bucket.
   - Type the **Validity in days** your CRL will remain valid. The default value is 7 days. For online CRLs, a validity period of 2-7 days is common. AWS Private CA tries to regenerate the CRL at the midpoint of the specified period.

4. Choose **Save changes** when done.

To configure OCSP

1. On the **Certificate revocation** page, choose **Turn on OCSP**.
2. (Optional) In the **Custom OCSP endpoint** field, provide a fully qualified domain name (FQDN) for your OCSP endpoint. To use OCSP over IPv6, set this field to a dualstack endpoint as described in Using OCSP over IPv6.

   When you provide an FQDN in this field, AWS Private CA inserts the FQDN into the *Authority Information Access* extension of each issued certificate in place of the default URL for the AWS OCSP responder. When an endpoint receives a certificate containing the custom FQDN, it queries that address for an OCSP response. For this mechanism to work, you need to take two additional actions:

   - Use a proxy server to forward traffic that arrives at your custom FQDN to the AWS OCSP responder.
   - Add a corresponding CNAME record to your DNS database.

> **Tip**
>
> For more information about implementing a complete OCSP solution using a custom CNAME, see Customize OCSP URL for AWS Private CA.

For example, here is a CNAME record for customized OCSP as it would appear in Amazon Route 53.

Record name	Type	Routing policy	Differentiator	Value/Route traffic to
alternative.example.com	CNAME	Simple	-	proxy.example.com

> **Note**
>
> The value of the CNAME must not include a protocol prefix such as "http://" or "https://".

3. Choose **Save changes** when done.

Updating a CA (CLI)

The following procedures show how to update the status and revocation configuration of an existing CA using the AWS CLI.

> **Note**
>
> Changes to the revocation configuration of a CA do not affect certificates that were already issued. For managed revocation to work, older certificates must be re-issued.

To update the status of your private CA (AWS CLI)

Use the update-certificate-authority command.

This is useful when you have an existing CA with status DISABLED that you want to set to ACTIVE. To begin, confirm the initial status of the CA with the following command.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

This results in output similar to the following.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-05T14:24:12.867000-08:00",
        "LastStateChangeAt": "2021-03-08T13:17:40.221000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "DISABLED",
        "NotBefore": "2021-03-08T07:46:27-08:00",
        "NotAfter": "2022-03-08T08:46:27-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "CustomCname": "alternative.example.com",
                "S3BucketName": "amzn-s3-demo-bucket"
   },
            "OcspConfiguration": {
```

```
                "Enabled": false
            }
        }
    }
}
```

The following command sets the status of the private CA to ACTIVE. This is possible only if a valid certificate is installed on the CA.

```
$ aws acm-pca update-certificate-authority \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
      --status "ACTIVE"
```

Inspect the new status of the CA.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566" \
      --output json
```

The status now appears as ACTIVE.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-05T14:24:12.867000-08:00",
        "LastStateChangeAt": "2021-03-08T13:23:09.352000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T07:46:27-08:00",
        "NotAfter": "2022-03-08T08:46:27-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
```

```
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "CustomCname": "alternative.example.com",
                "S3BucketName": "amzn-s3-demo-bucket"
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

In some cases, you might have an active CA with no revocation mechanism configured. If you want to begin using a certificate revocation list (CRL), use the following procedure.

To add a CRL to an existing CA (AWS CLI)

1. Use the following command to inspect the current status of the CA.

```
$ aws acm-pca describe-certificate-authority
 --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
 --output json
```

The output confirms that the CA has status ACTIVE but is not configured to use a CRL.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-08T14:36:26.449000-08:00",
        "LastStateChangeAt": "2021-03-08T14:50:52.224000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T13:46:50-08:00",
```

```
        "NotAfter": "2022-03-08T14:46:50-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": false
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

2. Create and save a file with a name such as `revoke_config.txt` to define your CRL configuration parameters.

```
{
   "CrlConfiguration":{
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket"
   }
}
```

> **Note**
>
> When updating a Matter device attestation CA to enable CRLs, you must configure it to omit the CDP extension from the issued certificates to help conform to the current Matter standard. To do this, define your CRL configuration parameters as illustrated below:

```
{
   "CrlConfiguration":{
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "amzn-s3-demo-bucket"
      "CrlDistributionPointExtensionConfiguration":{
         "OmitExtension": true
      }
   }
}
```

3. Use the update-certificate-authority command and the revocation configuration file to update the CA.

```
$ aws acm-pca update-certificate-authority \
      --certificate-authority-arn arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
      --revocation-configuration file://revoke_config.txt
```

4. Again inspect the status of the CA.

```
$ aws acm-pca describe-certificate-authority
 --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
 --output json
```

The output confirms that CA is now configured to use a CRL.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-08T14:36:26.449000-08:00",
        "LastStateChangeAt": "2021-03-08T14:50:52.224000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T13:46:50-08:00",
        "NotAfter": "2022-03-08T14:46:50-08:00",
        "CertificateAuthorityConfiguration": {
```

```
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": true,
                "ExpirationInDays": 7,
                "S3BucketName": "amzn-s3-demo-bucket",
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

In some cases, you might want to add OCSP revocation support instead of enabling a CRL as in the previous procedure. In that case, use the following steps.

To add OCSP support to an existing CA (AWS CLI)

1. Create and save a file with a name such as `revoke_config.txt` to define your OCSP parameters.

```
{
   "OcspConfiguration":{
      "Enabled":true
   }
}
```

2. Use the update-certificate-authority command and the revocation configuration file to update the CA.

```
$ aws acm-pca update-certificate-authority \
      --certificate-authority-arn arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566 \
      --revocation-configuration file://revoke_config.txt
```

3. Again inspect the status of the CA.

```
$ aws acm-pca describe-certificate-authority
 --certificate-authority-arnarn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
 --output json
```

The output confirms that CA is now configured to use OCSP.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
        "CreatedAt": "2021-03-08T14:36:26.449000-08:00",
        "LastStateChangeAt": "2021-03-08T14:50:52.224000-08:00",
        "Type": "ROOT",
        "Serial": "serial_number",
        "Status": "ACTIVE",
        "NotBefore": "2021-03-08T13:46:50-08:00",
        "NotAfter": "2022-03-08T14:46:50-08:00",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "Country": "US",
                "Organization": "Example Corp",
                "OrganizationalUnit": "Sales",
                "State": "WA",
                "CommonName": "www.example.com",
                "Locality": "Seattle"
            }
        },
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": false
            },
            "OcspConfiguration": {
```

```
                "Enabled": true
            }
        }
    }
}
```

Note

You can also configure both CRL and OCSP support on a CA.

Delete your private CA

You can delete a private CA from the AWS Management Console or AWS CLI permanently. You might want to delete one, for example, to replace it with a new CA that has a new private key. In order to delete a CA safely, follow these steps:

1. Create the replacement CA.
2. Once the new private CA is in production, disable the old one but do not immediately delete it.
3. Keep the old CA disabled until all of the certificates issued by it have expired.
4. Delete the old CA.

AWS Private CA does not check that all of the issued certificates have expired before it processes a delete request. You can generate an audit report to determine which certificates have expired. While the CA is disabled, you can revoke certificates, but you cannot issue new ones.

If you must delete a private CA before all the certificates it has issued have expired, we recommend that you also revoke the CA certificate. The CA certificate will be listed in the CRL of the parent CA, and the private CA will be untrusted by clients.

Important

A private CA can be deleted if it is in the `PENDING_CERTIFICATE`, `CREATING`, `EXPIRED`, `DISABLED`, or `FAILED` state. In order to delete a CA in the `ACTIVE` state, you must first disable it, or else the delete request results in an exception. If you are deleting a private CA in the `PENDING_CERTIFICATE` or `DISABLED` state, you can set the length of its restoration

period from 7-30 days, with 30 being the default. During this period, status is set to DELETED and the CA is restorable. A private CA that is deleted while in the CREATING or FAILED state has no assigned restoration period and cannot be restored. For more information, see Restore a private CA.
You are not charged for a private CA after it has been deleted. However, if a deleted CA is restored, you are charged for the time between deletion and restoration. For more information, see Pricing for AWS Private Certificate Authority.

To delete a private CA (console)

1. Sign in to your AWS account and open the AWS Private CA console at https://console.aws.amazon.com/acm-pca/home.
2. On the **Private certificate authorities** page, choose your private CA from the list.
3. If your CA is in the ACTIVE state, you must first disable it. On the **Actions** menu, choose **Disable**. When prompted, choose **I understand the risk, continue**.
4. For a CA that is not in the ACTIVE state, choose **Actions**, **Delete**.
5. If your CA is in the DISABLED, EXPIRED, or PENDING_CERTIFICATE state, the **Delete CA** page lets you specify a restoration period of 7-30 days. If your private CA is not in one of these states, it cannot be restored later and deletion is permanent.
6. Choose **Delete**.
7. If you are certain that you want to delete the private CA, choose **Permanently delete** when prompted. The status of the private CA changes to DELETED. However, you can restore the private CA before the end of the restoration period. To check the restoration period of a private CA in the DELETED state, call the DescribeCerticateAuthority or ListCertificateAuthorities API operation.

To delete a private CA (AWS CLI)

Use the delete-certificate-authority command to delete a private CA.

```
$ aws acm-pca delete-certificate-authority \
     --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID \
     --permanent-deletion-time-in-days 16
```

Restore a private CA

You can restore a private CA that has been deleted as long as the CA remains within the restoration period that you specified upon deletion. The restoration period is from 7-30 days. At the end of that period, the private CA is permanently deleted. For more information, see Delete your private CA. You cannot restore a private CA that has been permanently deleted.

> **Note**
>
> You are not charged for a private CA after it has been deleted. However, if a deleted CA is restored, you are charged for the time between deletion and restoration. For more information, see Pricing for AWS Private Certificate Authority.

Restoring a private CA (console)

You can use the AWS Management Console to restore a private CA.

To restore a private CA (console)

1. Sign in to your AWS account and open the AWS Private CA console at https://console.aws.amazon.com/acm-pca/home.
2. On the **Private certificate authorities** page, choose your deleted private CA from the list.
3. On the **Actions** menu, choose **Restore**.
4. On the **Restore CA** page, choose **Restore** again.
5. If successful, the status of the private CA is set to its pre-deletion state. Choose **Actions**, **Enable**, and **Enable** again to change its status to `ACTIVE`. If the private CA was in the `PENDING_CERTIFICATE` state at the time of deletion, you must import a CA certificate into the private CA before you can activate it.

Restore a private CA (AWS CLI)

Use the restore-certificate-authority command to restore a deleted private CA that is in the `DELETED` state. The following steps discuss the entire process required to delete, restore, and then reactivate a private CA.

To delete, restore, and reactivate a private CA (AWS CLI)

1. Delete the private CA.

   Run the delete-certificate-authority command to delete the private CA. If the private CA's status is DISABLED or PENDING_CERTIFICATE, you can set the `--permanent-deletion-time-in-days` parameter to specify the private CA's restoration period from 7 -30 days. If you do not specify a restoration period, the default is 30 days. If successful, this command sets the status of the private CA to DELETED.

   > **Note**
   >
   > To be restorable, the private CA's status at the time of deletion must be DISABLED or PENDING_CERTIFICATE.

   ```
   $ aws acm-pca delete-certificate-authority \
        --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID \
        --permanent-deletion-time-in-days 16
   ```

2. Restore the private CA.

   Run the restore-certificate-authority command to restore the private CA. You must run the command before the restoration period that you set with the **delete-certificate-authority** command expires. If successful, the command sets the status of the private CA to its pre-deletion status.

   ```
   $ aws acm-pca restore-certificate-authority \
        --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID
   ```

3. Make the private CA ACTIVE.

   Run the update-certificate-authority command to change the status of the private CA to ACTIVE.

   ```
   $ aws acm-pca update-certificate-authority \
        --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID \
   ```

```
    --status ACTIVE
```

Use externally signed private CA certificates

If your private CA hierarchy's root of trust must be a CA outside of AWS Private CA, you can create and self-sign your own root CA. Alternatively, you can obtain a private CA certificate that is signed by an external private CA operated by your organization. Whatever its source, you can use this externally obtained CA to sign a private subordinate CA certificate that AWS Private CA manages.

> **Note**
>
> Procedures for creating or obtaining an external trust services provider are outside the scope of this guide.

Using an external parent CA with AWS Private CA permits you to enforce CA name constraints as defined in the Name Constraints section of RFC 5280. Name constraints provide a way for CA administrators to restrict subject names in certificates.

If you plan to sign a private subordinate CA certificate with an external CA, there are three tasks to complete before you have a working CA in AWS Private CA:

1. Generate a certificate signing request (CSR).
2. Submit the CSR to your external signing authority and return with a signed certificate and certificate chain.
3. Install a signed certificate in AWS Private CA.

The following procedures describe how to complete these tasks using either the AWS Management Console or the AWS CLI.

To obtain and install an externally signed CA certificate (console)

1. (Optional) If you are not already on the CA's details page, open the AWS Private CA console at https://console.aws.amazon.com/acm-pca/home. On the **Private certificate authorities** page, choose a subordinate CA with status **Pending certificate**, **Active**, **Disabled**, or **Expired**.
2. Choose **Actions**, **Install CA Certificate** to open the **Install subordinate CA certificate** page.

3. On the **Install subordinate CA certificate** page, under **Select CA type**, choose **External private CA**.
4. Under **CSR for this CA**, the console displays the Base64-encoded ASCII text of the CSR. You can copy the text using the **Copy** button or you can choose **Export CSR to a file** and save it locally.

   > **Note**
   >
   > The exact format of the CSR text must be preserved when copying and pasting.

5. If you cannot immediately perform the offline steps to obtain a signed certificate from your external signing authority, you can close the page and return to it once you possess a signed certificate and a certificate chain.

   Otherwise, if you are ready, do either of the following:

   - Paste the Base64-encoded ASCII text of your certificate body and of your certificate chain into their respective text boxes.
   - Choose **Upload** to load the certificate body and certificate chain from local files into their respective text boxes.
6. Choose **Confirm and install**.

To obtain and install an externally signed CA certificate (CLI)

1. Use the get-certificate-authority-csr command to retrieve the certificate signing request (CSR) for your private CA. If you want to send the CSR to your display, use the `--output text` option to eliminate CR/LF characters from the end of each line. To send the CSR to a file, use the redirect option (>) followed by a file name.

   ```
   $ aws acm-pca get-certificate-authority-csr \
   --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
   authority/11223344-1234-1122-2233-112233445566 \
   --output text
   ```

   After saving a CSR as a local file, you can inspect it by using the following OpenSSL command:

   ```
   openssl req -in path_to_CSR_file -text -noout
   ```

This command generates output similar to the following. Notice that the **CA** extension is TRUE, indicating that the CSR is for a CA certificate.

```
Certificate Request:
Data:
Version: 0 (0x0)
Subject: O=ExampleCompany, OU=Corporate Office, CN=Example CA 1
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
            00:d4:23:51:b3:dd:01:09:01:0b:4c:59:e4:ea:81:
            1d:7f:48:36:ef:2a:e9:45:82:ec:95:1d:c6:d7:c9:
            7f:19:06:73:c5:cd:63:43:14:eb:c8:03:82:f8:7b:
            c7:89:e6:8d:03:eb:b6:76:58:70:f2:cb:c3:4c:67:
            ea:50:fd:b9:17:84:b8:60:2c:64:9d:2e:d5:7d:da:
            46:56:38:34:a9:0d:57:77:85:f1:6f:b8:ce:73:eb:
            f7:62:a7:8e:e6:35:f5:df:0c:f7:3b:f5:7f:bd:f4:
            38:0b:95:50:2c:be:7d:bf:d9:ad:91:c3:81:29:23:
            b2:5e:a6:83:79:53:f3:06:12:20:7e:a8:fa:18:d6:
            a8:f3:a3:89:a5:a3:6a:76:da:d0:97:e5:13:bc:84:
            a6:5c:d6:54:1a:f0:80:16:dd:4e:79:7b:ff:6d:39:
            b5:67:56:cb:02:6b:14:c3:17:06:0e:7d:fb:d2:7e:
            1c:b8:7d:1d:83:13:59:b2:76:75:5e:d1:e3:23:6d:
            8a:5e:f5:85:ca:d7:e9:a3:f1:9b:42:9f:ed:8a:3c:
            14:4d:1f:fc:95:2b:51:6c:de:8f:ee:02:8c:0c:b6:
            3e:2d:68:e5:f8:86:3f:4f:52:ec:a6:f0:01:c4:7d:
            68:f3:09:ae:b9:97:d6:fc:e4:de:58:58:37:09:9a:
            f6:27
        Exponent: 65537 (0x10001)
Attributes:
Requested Extensions:
    X509v3 Basic Constraints:
        CA:TRUE
Signature Algorithm: sha256WithRSAEncryption
 c5:64:0e:6c:cf:11:03:0b:b7:b8:9e:48:e1:04:45:a0:7f:cc:
 a7:fd:e9:4d:c9:00:26:c5:6e:d0:7e:69:7a:fb:17:1f:f3:5d:
 ac:f3:65:0a:96:5a:47:3c:c1:ee:45:84:46:e3:e6:05:73:0c:
 ce:c9:a0:5e:af:55:bb:89:46:21:92:7b:10:96:92:1b:e6:75:
 de:02:13:2d:98:72:47:bd:b1:13:1a:3d:bb:71:ae:62:86:1a:
 ee:ae:4e:f4:29:2e:d6:fc:70:06:ac:ca:cf:bb:ee:63:68:14:
 8e:b2:8f:e3:8d:e8:8f:e0:33:74:d6:cf:e2:e9:41:ad:b6:47:
```

```
f8:2e:7d:0a:82:af:c6:d8:53:c2:88:a0:32:05:09:e0:04:8f:
79:1c:ac:0d:d4:77:8e:a6:b2:5f:07:f8:1b:e3:98:d4:12:3d:
28:32:82:b5:50:92:a4:b2:4c:28:fc:d2:73:75:75:ff:10:33:
2c:c0:67:4b:de:fd:e6:69:1c:a8:bb:e8:31:93:07:35:69:b7:
d6:53:37:53:d5:07:dd:54:35:74:50:50:f9:99:7d:38:b7:b6:
7f:bd:6c:b8:e4:2a:38:e5:04:00:a8:a3:d9:e5:06:38:e0:38:
4c:ca:a9:3c:37:6d:ba:58:38:11:9c:30:08:93:a5:62:00:18:
d1:83:66:40
```

2. Submit the CSR to your external signing authority and obtain files containing the Base64 PEM-encoded signed certificate and certificate chain.
3. Use the import-certificate-authority-certificate command to import the private CA certificate file and the chain file into AWS Private CA.

```
$ aws acm-pca import-certificate-authority-certificate \
--certificate-authority-arn arn:aws:acm-pca:region:account:\
certificate-authority/12345678-1234-1234-1234-123456789012 \
--certificate file://C:\example_ca_cert.pem \
--certificate-chain file://C:\example_ca_cert_chain.pem
```

Issue and manage certificates in AWS Private CA

After you have created and activated a private certificate authority (CA) and configured access to it, you or your authorized users can issue and manage certificates. If you have not yet set up AWS Identity and Access Management (IAM) policies for the CA, you can learn more about configuring them in the Identity and Access Management section of this guide. For information about configuring CA access in single-account and cross-account scenarios, see Control access to the private CA.

Topics

- Issue private end-entity certificates
- Retrieve a private certificate
- List private certificates
- Export a private certificate and its secret key
- Revoke a private certificate
- Automate export of a renewed certificate
- Use AWS Private CA certificate templates

Issue private end-entity certificates

With a private CA in place, you can request private end-entity certificates from either AWS Certificate Manager (ACM) or AWS Private CA. The capabilities of both services are compared in the following table.

Capability	ACM	AWS Private CA
Issue end-entity certificates	✓ (using RequestCertificate or the console)	✓ (using IssueCertificate)
Association with load balancers and internet-facing AWS services	✓	Not supported
Managed certificate renewal	✓	Indirectly supported through ACM

Capability	ACM	AWS Private CA
Console support	✓	Not supported
API support	✓	✓
CLI support	✓	✓

When AWS Private CA creates a certificate, it follows a template that specifies the certificate type and path length. If no template ARN is supplied to the API or CLI statement creating the certificate, the EndEntityCertificate/V1 template is applied by default. For more information about available certificate templates, see Use AWS Private CA certificate templates.

While ACM certificates are designed around public trust, AWS Private CA serves the needs of your private PKI. Consequently, you can configure certificates using the AWS Private CA API and CLI in ways not permitted by ACM. These include the following:

- Creating a certificate with any Subject name.
- Using any of the supported private key algorithms and key lengths.
- Using any of the supported signing algorithms.
- Specifying any validity period for your private CA and private certificates.

After creating a private TLS certificate using AWS Private CA, you can import it into ACM and use it with a supported AWS service.

> **Note**
>
> Certificates created with the procedure below, using the **issue-certificate** command, or with the IssueCertificate API action, cannot be directly exported for use outside AWS. However, you can use your private CA to sign certificates issued through ACM, and those certificates can be exported along with their secret keys. For more information, see Requesting a private certificate and Exporting a private certificate in the *ACM User Guide*.

Issue a standard certificate (AWS CLI)

You can use the AWS Private CA CLI command issue-certificate or the API action IssueCertificate to request an end-entity certificate. This command requires the Amazon Resource Name (ARN) of the private CA that you want to use to issue the certificate. You must also generate a certificate signing request (CSR) using a program such as OpenSSL.

If you use the AWS Private CA API or AWS CLI to issue a private certificate, the certificate is unmanaged, meaning that you cannot use the ACM console, ACM CLI, or ACM API to view or export it, and the certificate is not automatically renewed. However, you can use the PCA get-certificate command to retrieve the certificate details, and if you own the CA, you can create an audit report.

Considerations when creating certificates

- In compliance with RFC 5280, the length of the domain name (technically, the Common Name) that you provide cannot exceed 64 octets (characters), including periods. To add a longer domain name, specify it in the Subject Alternative Name field, which supports names up to 253 octets in length.
- If you are using AWS CLI version 1.6.3 or later, use the prefix `fileb://` when specifying base64-encoded input files such as CSRs. This ensures that AWS Private CA parses the data correctly.

The following OpenSSL command generates a CSR and a private key for a certificate:

```
$ openssl req -out csr.pem -new -newkey rsa:2048 -nodes -keyout private-key.pem
```

You can inspect the content of the CSR as follows:

```
$ openssl req -in csr.pem -text -noout
```

The resulting output should resemble the following abbreviated example:

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: C=US, O=Big Org, CN=example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
```

```
                Modulus:
                    00:ca:85:f4:3a:b7:5f:e2:66:be:fc:d8:97:65:3d:
                    a4:3d:30:c6:02:0a:9e:1c:ca:bb:15:63:ca:22:81:
                    00:e1:a9:c0:69:64:75:57:56:53:a1:99:ee:e1:cd:
                    ...
                    aa:38:73:ff:3d:b7:00:74:82:8e:4a:5d:da:5f:79:
                    5a:89:52:e7:de:68:95:e0:16:9b:47:2d:57:49:2d:
                    9b:41:53:e2:7f:e1:bd:95:bf:eb:b3:a3:72:d6:a4:
                    d3:63
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha256WithRSAEncryption
         74:18:26:72:33:be:ef:ae:1d:1e:ff:15:e5:28:db:c1:e0:80:
         42:2c:82:5a:34:aa:1a:70:df:fa:4f:19:e2:5a:0e:33:38:af:
         21:aa:14:b4:85:35:9c:dd:73:98:1c:b7:ce:f3:ff:43:aa:11:
         ....
         3c:b2:62:94:ad:94:11:55:c2:43:e0:5f:3b:39:d3:a6:4b:47:
         09:6b:9d:6b:9b:95:15:10:25:be:8b:5c:cc:f1:ff:7b:26:6b:
         fa:81:df:e4:92:e5:3c:e5:7f:0e:d8:d9:6f:c5:a6:67:fb:2b:
         0b:53:e5:22
```

The following command creates a certificate. Because no template is specified, a base end-entity certificate is issued by default.

```
$ aws acm-pca issue-certificate \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
      --csr fileb://csr.pem \
      --signing-algorithm "SHA256WITHRSA" \
      --validity Value=365,Type="DAYS"
```

The ARN of the issued certificate is returned:

```
{
   "CertificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
}
```

Note

AWS Private CA immediately returns an ARN with a serial number when it receives the **issue-certificate** command. However, certificate processing happens asynchronously and can still fail. If this happens, a **get-certificate** command using the new ARN will also fail.

Issue a certificate with a custom subject name using an APIPassthrough template

In this example, a certificate is issued containing customized subject name elements. In addition to supplying a CSR like the one in Issue a standard certificate (AWS CLI), you pass two additional arguments to the **issue-certificate** command: the ARN of an APIPassthrough template, and a JSON configuration file that specifies the custom attributes and their object identifiers (OIDs). You cannot use `StandardAttributes` in conjunction with `CustomAttributes`. however, you can pass standard OIDs as part of `CustomAttributes`. The default subject name OIDs are listed in the following table (information from RFC 4519 and Global OID reference database):

Subject name	Abbreviation	Object ID
countryName	c	2.5.4.6
commonName	cn	2.5.4.3
dnQualifier [distinguished name qualifier]		2.5.4.46
generationQualifier		2.5.4.44
givenName		2.5.4.42
initials		2.5.4.43
locality	l	2.5.4.7
organizationName	o	2.5.4.10
organizationalUnitName	ou	2.5.4.11

Subject name	Abbreviation	Object ID
pseudonym		2.5.4.65
serialNumber		2.5.4.5
st [state]		2.5.4.8
surname	sn	2.5.4.4
title		2.5.4.12
domainComponent	dc	0.9.2342.19200300.100.1.25
userid		0.9.2342.19200300.100.1.1

The sample configuration file `api_passthrough_config.txt` contains the following code:

```
{
  "Subject": {
    "CustomAttributes": [
      {
        "ObjectIdentifier": "2.5.4.6",
        "Value": "US"
      },
      {
        "ObjectIdentifier": "1.3.6.1.4.1.37244.1.1",
        "Value": "BCDABCDA12341234"
      },
      {
        "ObjectIdentifier": "1.3.6.1.4.1.37244.1.5",
        "Value": "CDABCDAB12341234"
      }
    ]
  }
}
```

Use the following command to issue the certificate:

```
$ aws acm-pca issue-certificate \
      --validity Type=DAYS,Value=10
```

```
      --signing-algorithm "SHA256WITHRSA" \
      --csr fileb://csr.pem \
      --api-passthrough file://api_passthrough_config.txt \
      --template-arn arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APIPassthrough/V1 \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

The ARN of the issued certificate is returned:

```
{
   "CertificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
}
```

Retrieve the certificate locally as follows:

```
$ aws acm-pca get-certificate \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
      --certificate-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID | \
      jq -r .'Certificate' > cert.pem
```

You can inspect the certificate's contents using OpenSSL:

```
$ openssl x509 -in cert.pem -text -noout
```

> **Note**
>
> It is also possible to create a private CA that passes custom attributes to each certificate it issues.

Issue a certificate with custom extensions using an APIPassthrough template

In this example, a certificate is issued that contains customized extensions. For this you need to pass three arguments to the **issue-certificate** command: the ARN of an APIPassthrough template,

and a JSON configuration file that specifies the custom extensions, and a CSR like the one shown in Issue a standard certificate (AWS CLI).

The sample configuration file `api_passthrough_config.txt` contains the following code:

```
{
  "Extensions": {
    "CustomExtensions": [
      {
        "ObjectIdentifier": "2.5.29.30",
        "Value": "MBWgEzARgg8ucGVybWl0dGVkLnRlc3Q=",
        "Critical": true
      }
    ]
  }
}
```

The customized certificate is issued as follows:

```
$ aws acm-pca issue-certificate \
      --validity Type=DAYS,Value=10
      --signing-algorithm "SHA256WITHRSA" \
      --csr fileb://csr.pem \
      --api-passthrough file://api_passthrough_config.txt \
      --template-arn arn:aws:acm-pca:::template/EndEntityCertificate_APIPassthrough/V1
 \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

The ARN of the issued certificate is returned:

```
{
   "CertificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
}
```

Retrieve the certificate locally as follows:

```
$ aws acm-pca get-certificate \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
```

```
      --certificate-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID | \
      jq -r .'Certificate' > cert.pem
```

You can inspect the certificate's contents using OpenSSL:

```
$ openssl x509 -in cert.pem -text -noout
```

Retrieve a private certificate

You can use the AWS Private CA API and AWS CLI to issue a private certificate. If you do, you can use the AWS CLI or AWS Private CA API to retrieve that certificate. If you used ACM to create your private CA and to request certificates, you must use ACM to export the certificate and the encrypted private key. For more information, see Exporting a private certificate.

To retrieve an end-entity certificate

Use the get-certificate AWS CLI command to retrieve a private end-entity certificate. You can also use the GetCertificate API operation. We recommend formatting the output with jq, a sed-like parser.

> **Note**
>
> If you want to revoke a certificate, you can use the **get-certificate** command to retrieve the serial number in hexadecimal format. You can also create an audit report to retrieve the hex serial number. For more information, see Use audit reports with your private CA.

```
$ aws acm-pca get-certificate \
      --certificate-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID \
      --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 | \
      jq -r '.Certificate, .CertificateChain'
```

This command outputs the certificate and certificate chain in the following standard format.

```
-----BEGIN CERTIFICATE-----
...base64-encoded certificate...
```

```
-----END CERTIFICATE----
-----BEGIN CERTIFICATE-----
...base64-encoded certificate...
-----END CERTIFICATE----
-----BEGIN CERTIFICATE-----
...base64-encoded certificate...
-----END CERTIFICATE----
```

To retrieve a CA certificate

You can use the AWS Private CA API and AWS CLI to retrieve the certificate authority (CA) certificate for your private CA. Run the get-certificate-authority-certificate command. You can also call the GetCertificateAuthorityCertificate operation. We recommend formatting the output with jq, a sed-like parser.

```
$ aws acm-pca get-certificate-authority-certificate \
     --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
     | jq -r '.Certificate'
```

This command outputs the CA certificate in the following standard format.

```
-----BEGIN CERTIFICATE-----
...base64-encoded certificate...
-----END CERTIFICATE----
```

List private certificates

To list your private certificates, generate an audit report, retrieve it from its S3 bucket, and parse the report contents as needed. For information about creating AWS Private CA audit reports, see Use audit reports with your private CA. For information about retrieving an object from an S3 bucket, see Downloading an object in the *Amazon Simple Storage Service User Guide*.

The following examples illustrate approaches to creating audit reports and parsing them for useful data. Results are formatted in JSON, and data is filtered using jq, a sed-like parser.

1. Create an audit report.

The following command generates an audit report for a specified CA.

```
$ aws acm-pca create-certificate-authority-audit-report \
```

```
     --region region \
     --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
     --s3-bucket-name bucket_name \
     --audit-report-response-format JSON
```

When successful, the command returns the ID and location of the new audit report.

```
{
   "AuditReportId":"audit_report_ID",
   "S3Key":"audit-report/CA_ID/audit_report_ID.json"
}
```

2. Retrieve and format an audit report.

This command retrieves an audit report, displays its contents in standard output, and filters the results to show only certificates issued on or after 2020-12-01.

```
$ aws s3api get-object \
     --region region \
     --bucket bucket_name \
     --key audit-report/CA_ID/audit_report_ID.json \
     /dev/stdout | jq '.[] | select(.issuedAt >= "2020-12-01")'
```

The returned items resemble the following:

```
{
   "awsAccountId":"account",
   "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial":"serial_number",
   "subject":"CN=pca.alpha.root2.leaf5",
   "notBefore":"2020-12-21T21:28:09+0000",
   "notAfter":"9999-12-31T23:59:59+0000",
   "issuedAt":"2020-12-21T22:28:09+0000",
   "templateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

3. Save an audit report locally.

If you want to perform multiple queries, it is convenient to save an audit report to a local file.

```
$ aws s3api get-object \
     --region region \
     --bucket bucket_name \
     --key audit-report/CA_ID/audit_report_ID.json > my_local_audit_report.json
```

The same filter as before yields the same output:

```
$ cat my_local_audit_report.json | jq '.[] | select(.issuedAt >= "2020-12-01")'
{
   "awsAccountId":"account",
   "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial":"serial_number",
   "subject":"CN=pca.alpha.root2.leaf5",
   "notBefore":"2020-12-21T21:28:09+0000",
   "notAfter":"9999-12-31T23:59:59+0000",
   "issuedAt":"2020-12-21T22:28:09+0000",
   "templateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

4. Query within a date range

You can query for certificates issued within a date range as follows:

```
$ cat my_local_audit_report.json | jq '.[] | select(.issuedAt >= "2020-11-01"
 and .issuedAt <= "2020-11-10")'
```

The filtered content is displayed in standard output:

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf1",
   "notBefore": "2020-11-06T19:18:21+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T20:18:22+0000",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
{
```

```
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.rsa2048sha256",
   "notBefore": "2020-11-06T19:15:46+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T20:15:46+0000",
   "templateArn": "arn:aws:acm-pca:::template/RootCACertificate/V1"
}
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf2",
   "notBefore": "2020-11-06T20:04:39+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T21:04:39+0000",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

5. Search for certificates following a specified template.

The following command filters the report content using a template ARN:

```
$ cat my_local_audit_report.json | jq '.[] | select(.templateArn == "arn:aws:acm-
pca:::template/RootCACertificate/V1")'
```

The output displays matching certificate records:

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.rsa2048sha256",
   "notBefore": "2020-11-06T19:15:46+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T20:15:46+0000",
   "templateArn": "arn:aws:acm-pca:::template/RootCACertificate/V1"
}
```

6. Filter for revoked certificates

To find all revoked certificates, use the following command:

```
$ cat my_local_audit_report.json | jq '.[] | select(.revokedAt != null)'
```

A revoked certificate is displayed as follows:

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf2",
   "notBefore": "2020-11-06T20:04:39+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T21:04:39+0000",
   "revokedAt": "2021-05-27T18:57:32+0000",
   "revocationReason": "UNSPECIFIED",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

7. Filter using a regular expression.

The following command searches for subject names that contain the string "leaf":

```
$ cat my_local_audit_report.json | jq '.[] | select(.subject|test("leaf"))'
```

Matching certificate records are returned as follows:

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.roo2.leaf4",
   "notBefore": "2020-11-16T18:17:10+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-16T19:17:12+0000",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

```
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf5",
   "notBefore": "2020-12-21T21:28:09+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-12-21T22:28:09+0000",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
{
   "awsAccountId": "account",
   "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
   "serial": "serial_number",
   "subject": "CN=pca.alpha.root2.leaf1",
   "notBefore": "2020-11-06T19:18:21+0000",
   "notAfter": "9999-12-31T23:59:59+0000",
   "issuedAt": "2020-11-06T20:18:22+0000",
   "templateArn": "arn:aws:acm-pca:::template/EndEntityCertificate/V1"
}
```

Export a private certificate and its secret key

AWS Private CA cannot directly export a private certificate that it has signed and issued. However, you can use AWS Certificate Manager to export such a certificate along with its encrypted secret key. The certificate is then completely portable for deployment anywhere in your private PKI. For more information, see Exporting a private certificate in the AWS Certificate Manager User Guide.

As an added benefit, AWS Certificate Manager provides managed renewal for private certificates that were issued using the ACM console, the `RequestCertificate` action of the ACM API, or the **request-certificate** command in the ACM section of the AWS CLI. For more information about renewals, see Renewing certificates in a private PKI.

Revoke a private certificate

You can revoke an AWS Private CA certificate using the revoke-certificate AWS CLI command or the RevokeCertificate API action. A certificate may need to be revoked before its scheduled expiration if, for example, its secret key is compromised or its associated domain becomes invalid.

For revocation to be effective, the client using the certificate needs a way to check revocation status whenever it attempts to build a secure network connection.

AWS Private CA provides two fully managed mechanisms to support revocation status checking: Online Certificate Status Protocol (OCSP) and certificate revocation lists (CRLs). With OCSP, the client queries an authoritative revocation database that returns a status in real-time. With a CRL, the client checks the certificate against a list of revoked certificates that it periodically downloads and stores. Clients refuse to accept certificates that have been revoked.

Both OCSP and CRLs depend on validation information embedded in certificates. For this reason, an issuing CA must be configured to support either or both of these mechanisms prior to issuance. For information about selecting and implementing managed revocation through AWS Private CA, see Plan your AWS Private CA certificate revocation method.

Revoked certificates are always recorded in AWS Private CA audit reports.

> **Note**
>
> For cross-account callers, a share with the `AWSRAMRevokeCertificateCertificateAuthority` permission is required. Revocation permissions are not included in `AWSRAMDefaultPermissionCertificateAuthority`. To enable revocation by cross-account issuers, the CA administrator must create two RAM shares, both pointing at the same CA:
>
> 1. A share with the `AWSRAMRevokeCertificateCertificateAuthority` permission.
> 2. A share with the `AWSRAMDefaultPermissionCertificateAuthority` permission.

To revoke a certificate

Use the RevokeCertificate API action or revoke-certificate command to revoke a private PKI certificate. The serial number must be in hexadecimal format. You can retrieve the serial number by calling the get-certificate command. The `revoke-certificate` command does not return a response.

```
$ aws acm-pca revoke-certificate \
    --certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
```

```
    --certificate-serial serial_number \
    --revocation-reason "KEY_COMPROMISE"
```

Revoked certificates and OCSP

OCSP responses may take up to 60 minutes to reflect the new status when you revoke a certificate. In general, OCSP tends to support faster distribution of revocation information because, unlike CRLs which can be cached by clients for days, OCSP responses are typically not cached by clients.

Revoked certificates in a CRL

A CRL is typically updated approximately 30 minutes after a certificate is revoked. If for any reason a CRL update fails, AWS Private CA makes further attempts every 15 minutes.

With Amazon CloudWatch, you can create alarms for the metrics `CRLGenerated` and `MisconfiguredCRLBucket`. For more information, see Supported CloudWatch Metrics. For more information about creating and configuring CRLs, see Set up a CRL for AWS Private CA.

The following example shows a revoked certificate in a certificate revocation list (CRL).

```
Certificate Revocation List (CRL):
        Version 2 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: /C=US/ST=WA/L=Seattle/O=Examples LLC/OU=Corporate Office/
CN=www.example.com
        Last Update: Jan 10 19:28:47 2018 GMT
        Next Update: Jan  8 20:28:47 2028 GMT
        CRL extensions:
            X509v3 Authority key identifier:
                keyid:3B:F0:04:6B:51:54:1F:C9:AE:4A:C0:2F:11:E6:13:85:D8:84:74:67

            X509v3 CRL Number:
                1515616127629
Revoked Certificates:
    Serial Number: B17B6F9AE9309C51D5573BCA78764C23
        Revocation Date: Jan  9 17:19:17 2018 GMT
        CRL entry extensions:
            X509v3 CRL Reason Code:
                Key Compromise
    Signature Algorithm: sha256WithRSAEncryption
         21:2f:86:46:6e:0a:9c:0d:85:f6:b6:b6:db:50:ce:32:d4:76:
         99:3e:df:ec:6f:c7:3b:7e:a3:6b:66:a7:b2:83:e8:3b:53:42:
```

```
        f0:7a:bc:ba:0f:81:4d:9b:71:ee:14:c3:db:ad:a0:91:c4:9f:
        98:f1:4a:69:9a:3f:e3:61:36:cf:93:0a:1b:7d:f7:8d:53:1f:
        2e:f8:bd:3c:7d:72:91:4c:36:38:06:bf:f9:c7:d1:47:6e:8e:
        54:eb:87:02:33:14:10:7f:b2:81:65:a1:62:f5:fb:e1:79:d5:
        1d:4c:0e:95:0d:84:31:f8:5d:59:5d:f9:2b:6f:e4:e6:60:8b:
        58:7d:b2:a9:70:fd:72:4f:e7:5b:e4:06:fc:e7:23:e7:08:28:
        f7:06:09:2a:a1:73:31:ec:1c:32:f8:dc:03:ea:33:a8:8e:d9:
        d4:78:c1:90:4c:08:ca:ba:ec:55:c3:00:f4:2e:03:b2:dd:8a:
        43:13:fd:c8:31:c9:cd:8d:b3:5e:06:c6:cc:15:41:12:5d:51:
        a2:84:61:16:a0:cf:f5:38:10:da:a5:3b:69:7f:9c:b0:aa:29:
        5f:fc:42:68:b8:fb:88:19:af:d9:ef:76:19:db:24:1f:eb:87:
        65:b2:05:44:86:21:e0:b4:11:5c:db:f6:a2:f9:7c:a6:16:85:
        0e:81:b2:76
```

Revoked certificates in an audit report

All certificates, including revoked certificates, are included in the audit report for a private CA. The following example shows an audit report with one issued and one revoked certificate. For more information, see Use audit reports with your private CA.

```
[
   {
      "awsAccountId":"account",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
      "serial":"serial_number",

 "Subject":"1.2.840.113549.1.9.1=#161173616c6573406578616d706c652e636f6d,CN=www.example1.com,OU
 Company,L=Seattle,ST=Washington,C=US",
      "notBefore":"2018-02-26T18:39:57+0000",
      "notAfter":"2019-02-26T19:39:57+0000",
      "issuedAt":"2018-02-26T19:39:58+0000",
      "revokedAt":"2018-02-26T20:00:36+0000",
      "revocationReason":"KEY_COMPROMISE"
   },
   {
      "awsAccountId":"account",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
      "serial":"serial_number",

 "Subject":"1.2.840.113549.1.9.1=#161970726f64407777772e70616c6f75736573616c65732e636f6d,CN=www
 Company,L=Seattle,ST=Washington,C=US",
```

```
      "notBefore":"2018-01-22T20:10:49+0000",
      "notAfter":"2019-01-17T21:10:49+0000",
      "issuedAt":"2018-01-22T21:10:49+0000"
   }
]
```

Automate export of a renewed certificate

When you use AWS Private CA to create a CA, you can import that CA into AWS Certificate Manager and let ACM manage certificate issuance and renewal. If a certificate being renewed is associated with an integrated service, the service seamlessly applies the new certificate. However, if the certificate was originally exported for use elsewhere in your PKI environment (for example, in an on-premises server or appliance), you need to export it again after renewal.

For a sample solution that automates the ACM export process using Amazon EventBridge and AWS Lambda, see Automating export of renewed certificates.

Use AWS Private CA certificate templates

AWS Private CA uses configuration templates to issue both CA certificates and end-entity certificates. When you issue a CA certificate from the PCA console, the appropriate root or subordinate CA certificate template is applied automatically.

If you use the CLI or API to issue a certificate, you can supply a template ARN as a parameter to the `IssueCertificate` action. If you provide no ARN, then the `EndEntityCertificate/V1` template is applied by default. For more information, see the IssueCertificate API and issue-certificate command documentation.

> **Note**
>
> AWS Certificate Manager (ACM) users with cross-account shared access to a private CA can issue managed certificates that are signed by the CA. Cross-account issuers are constrained by a resource-based policy and have access only to the following end-entity certificate templates:
>
> - EndEntityCertificate/V1
> - EndEntityClientAuthCertificate/V1
> - EndEntityServerAuthCertificate/V1

- BlankEndEntityCertificate_APIPassthrough/V1
- BlankEndEntityCertificate_APICSRPassthrough/V1
- SubordinateCACertificate_PathLen0/V1

For more information, see Resource-based policies.

Topics

- AWS Private CA template varieties
- AWS Private CA template order of operations
- AWS Private CA template definitions

AWS Private CA template varieties

AWS Private CA supports four varieties of template.

- **Base templates**

  Pre-defined templates in which no passthrough parameters are allowed.
- **CSRPassthrough templates**

  Templates that extend their corresponding base template versions by allowing CSR passthrough. Extensions in the CSR that is used to issue the certificate are copied over to the issued certificate. In cases where the CSR contains extension values that conflict with the template definition, the template definition will always have the higher priority. For more details about priority, see AWS Private CA template order of operations.
- **APIPassthrough templates**

  Templates that extend their corresponding base template versions by allowing API passthrough. Dynamic values that are known to the administrator or other intermediate systems may not be known by the entity requesting the certificate, may be impossible to define in a template, and may not be available in the CSR. The CA administrator, however, can retrieve additional information from another data source, such as an Active Directory, to complete the request. For example, if a machine doesn't know what organization unit it belongs to, the administrator can look up the information in Active Directory and add it to the certificate request by including the information in a JSON structure.

Values in the `ApiPassthrough` parameter of the `IssueCertificate` action are copied over to the issued certificate. In cases where the `ApiPassthrough` parameter contains information that conflicts with the template definition, the template definition will always have the higher priority. For more details about priority, see AWS Private CA template order of operations.

- **APICSRPassthrough templates**

  Templates that extend their corresponding base template versions by allowing both API and CSR passthrough. Extensions in the CSR used to issue the certificate are copied over to the issued certificate, and values in the `ApiPassthrough` parameter of the `IssueCertificate` action are also copied over . In cases where the template definition, API passthrough values, and CSR passthrough extensions exhibit a conflict, the template definition has highest priority, followed by the API passthrough values, followed by the CSR passthrough extensions. For more details about priority, see AWS Private CA template order of operations.

The tables below list all of the template types supported by AWS Private CA with links to their definitions.

> **Note**
>
> For information about template ARNs in GovCloud regions, see AWS Private Certificate Authority in the *AWS GovCloud (US) User Guide*.

Base templates

Template Name	Template ARN	Certificate Type
CodeSigningCertificate/V1	`arn:aws:acm-pca:::template/CodeSigningCertificate/V1`	Code signing
EndEntityCertificate/V1	`arn:aws:acm-pca:::template/EndEntityCertificate/V1`	End-entity
EndEntityClientAuthCertificate/V1	`arn:aws:acm-pca:::template/EndEntity`	End-entity

Template Name	Template ARN	Certificate Type
	ClientAuthCertificate/V1	
EndEntityServerAuthCertificate/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate/V1	End-entity
OCSPSigningCertificate/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate/V1	OCSP signing
RootCACertificate/V1	arn:aws:acm-pca:::template/RootCACertificate/V1	CA
SubordinateCACertificate_PathLen0/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0/V1	CA
SubordinateCACertificate_PathLen1/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1/V1	CA
SubordinateCACertificate_PathLen2/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2/V1	CA
SubordinateCACertificate_PathLen3/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3/V1	CA

CSRPassthrough templates

Template Name	Template ARN	Certificate Type
BlankEndEntityCertificate_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CSRPassthrough/V1	End-entity
BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1	End-entity
BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen2_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen2_CSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen3_CSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertifica	CA

Template Name	Template ARN	Certificate Type
	te_PathLen3_CSRPas sthrough/V1	
CodeSigningCertificate_CSRP assthrough/V1	arn:aws:acm-pca::: template/CodeSigni ngCertificate_CSRP assthrough/V1	Code signing
EndEntityCertificate_CSRPas sthrough/V1	arn:aws:acm-pca::: template/EndEntity Certificate_CSRPas sthrough/V1	End-entity
EndEntityClientAuthCertific ate_CSRPassthrough/V1	arn:aws:acm-pca::: template/EndEntity ClientAuthCertific ate_CSRPassthrough/V1	End-entity
EndEntityServerAuthCertific ate_CSRPassthrough/V1	arn:aws:acm-pca::: template/EndEntity ServerAuthCertific ate_CSRPassthrough/V1	End-entity
OCSPSigningCertificate_CSRP assthrough/V1	arn:aws:acm-pca::: template/OCSPSigni ngCertificate_CSRP assthrough/V1	OCSP signing
SubordinateCACertificate_Pa thLen0_CSRPassthrough/V1	arn:aws:acm-pca::: template/Subordina teCACertificate_Pa thLen0_CSRPassthrough/ V1	CA

Template Name	Template ARN	Certificate Type
SubordinateCACertificate_PathLen1_CSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1_CSRPassthrough/V1	CA
SubordinateCACertificate_PathLen2_CSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_CSRPassthrough/V1	CA
SubordinateCACertificate_PathLen3_CSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_CSRPassthrough/V1	CA

APIPassthrough templates

Template Name	Template ARN	Certificate Type
BlankEndEntityCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_APIPassthrough/V1	End-entity
BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1	End-entity

Template Name	Template ARN	Certificate Type
CodeSigningCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/CodeSigningCertificate_APIPassthrough/V1	Code signing
EndEntityCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/EndEntityCertificate_APIPassthrough/V1	End-entity
EndEntityClientAuthCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/EndEntityClientAuthCertificate_APIPassthrough/V1	End-entity
EndEntityServerAuthCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertificate_APIPassthrough/V1	End-entity
OCSPSigningCertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate_APIPassthrough/V1	OCSP signing
RootCACertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/RootCACertificate_APIPassthrough/V1	CA
BlankRootCACertificate_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_APIPassthrough/V1	CA

Template Name	Template ARN	Certificate Type
BlankRootCACertificate_PathLen0_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen0_APIPassthrough/V1	CA
BlankRootCACertificate_PathLen1_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen1_APIPassthrough/V1	CA
BlankRootCACertificate_PathLen2_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen2_APIPassthrough/V1	CA
BlankRootCACertificate_PathLen3_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankRootCACertificate_PathLen3_APIPassthrough/V1	CA
SubordinateCACertificate_PathLen0_APIPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0_APIPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1	CA

Template Name	Template ARN	Certificate Type
SubordinateCACertificate_PathLen1_APIPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1_APIPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1	CA
SubordinateCACertificate_PathLen2_APIPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_APIPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1	CA
SubordinateCACertificate_PathLen3_APIPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_APIPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1	CA

APICSRPassthrough templates

Template Name	Template ARN	Certificate Type
BlankEndEntityCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_APICSRPassthrough/V1	End-entity
BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1	End-entity
CodeSigningCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/CodeSigningCertificate_APICSRPassthrough/V1	Code signing
EndEntityCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/EndEntityCertificate_APICSRPassthrough/V1	End-entity
EndEntityClientAuthCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/EndEntityClientAuthCertificate_APICSRPassthrough/V1	End-entity
EndEntityServerAuthCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/EndEntityServerAuthCertific	End-entity

Template Name	Template ARN	Certificate Type
	ate_APICSRPassthrough/V1	
OCSPSigningCertificate_APICSRPassthrough/V1	arn:aws:acm-pca:::template/OCSPSigningCertificate_APICSRPassthrough/V1	OCSP signing
SubordinateCACertificate_PathLen0_APICSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0_APICSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1	CA
SubordinateCACertificate_PathLen1_APICSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1_APICSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1	CA

Template Name	Template ARN	Certificate Type
SubordinateCACertificate_PathLen2_APICSRPassthrough/PathLen3_APIPassthroughV1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_APICSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1	CA
SubordinateCACertificate_PathLen3_APICSRPassthrough/V1	arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_APICSRPassthrough/V1	CA
BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/V1	arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/V1	CA

AWS Private CA template order of operations

Information contained in an issued certificate can come from four sources: the template definition, API passthrough, CSR passthrough, and the CA configuration.

API passthrough values are only respected when you use an API passthrough or APICSR passthrough template. CSR passthrough is only respected when you use a CSRPassthrough or APICSR passthrough template. When these sources of information are in conflict, a general rule usually applies: For each extension value, the template definition has highest priority, followed by API passthrough values, followed by CSR passthrough extensions.

Examples

1. The template definition for EndEntityClientAuthCertificate_APIPassthrough defines the ExtendedKeyUsage extension with a value of "TLS web server authentication, TLS web client authentication". If ExtendedKeyUsage is defined in the CSR or in the `IssueCertificate ApiPassthrough` parameter, the `ApiPassthrough` value for ExtendedKeyUsage will be ignored because the template definition takes priority, and the CSR value for ExtendedKeyUsage value will be ignored because the template is not a CSR passthrough variety.

   > **Note**
   >
   > The template definition nonetheless copies over other values from the CSR, such as Subject and Subject Alternative Name. These values are still taken from the CSR even though the template is not a CSR passthrough variety, because the template definition always takes highest priority.

2. The template definition for EndEntityClientAuthCertificate_APICSRPassthrough defines the Subject Alternative Name (SAN) extension as being copied from the API or CSR. If the SAN extension is defined in the CSR and provided in the `IssueCertificate ApiPassthrough` parameter, the API passthrough value will take priority because API passthrough values take priority over CSR passthrough values.

AWS Private CA template definitions

The following sections provide configuration details about supported AWS Private CA certificate templates.

BlankEndEntityCertificate_APIPassthrough/V1 definition

With blank end-entity certificate templates, you can issue end-entity certificates with only X.509 Basic constraints present. This is the simplest end-entity certificate that AWS Private CA can issue, but it can be customized using the API structure. The Basic constraints extension defines whether or not the certificate is a CA certificate. A blank end-entity certificate template enforces a value of FALSE for Basic constraints to ensure that an end-entity certificate is issued and not a CA certificate.

You can use blank passthrough templates to issue smart card certificates that require specific values for Key usage (KU) and Extended key usage (EKU). For example, Extended key usage may

require Client Authentication and Smart Card Logon, and Key usage may require Digital Signature, Non Repudiation, and Key Encipherment. Unlike other passthrough templates, blank end-entity certificate templates allow the configuration of KU and EKU extensions, where KU can be any of the nine supported values (digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment, keyAgreement, keyCertSign, cRLSign, encipherOnly, and decipherOnly) and EKU can be any of the supported values (serverAuth, clientAuth, codesigning, emailProtection, timestamping, and OCSPSigning) plus custom extensions.

BlankEndEntityCertificate_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankEndEntityCertificate_APICSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankEndEntityCertificate_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	CA:FALSE

X509v3 Parameter	Value
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankEndEntityCertificate_CriticalBasicConstraints_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration, API, or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankEndEntityCertificate_CriticalBasicConstraints_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration or API]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankEndEntityCertificate_CriticalBasicConstraints_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]

X509v3 Parameter	Value
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankEndEntityCertificate_CSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankEndEntityCertificate_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 0`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 0`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankSubordinateCACertificate_PathLen0_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 0`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration]

BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankSubordinateCACertificate_PathLen1_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 1`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]

X509v3 Parameter	Value
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 1`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 1`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankSubordinateCACertificate_PathLen2_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 2`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankSubordinateCACertificate_PathLen2_CSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankSubordinateCACertificate_PathLen2_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 2`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 2`

X509v3 Parameter	Value
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankSubordinateCACertificate_PathLen3_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 3`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankSubordinateCACertificate_PathLen3_CSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankSubordinateCACertificate_PathLen3_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE, pathlen: 3`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition.

BlankSubordinateCACertificate_PathLen3_APICSRPassthrough

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE, pathlen: 3`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

CodeSigningCertificate/V1 definition

This template is used to create certificates for code signing. You can use code-signing certificates from AWS Private CA with any code-signing solution that is based on a private CA infrastructure. For example, customers using Code Signing for AWS IoT can generate a code-signing certificate with AWS Private CA and import it to AWS Certificate Manager. For more information, see What Is Code Signing for AWS IoT? and Obtain and Import a Code Signing Certificate.

CodeSigningCertificate/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	`CA:FALSE`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature
Extended key usage	Critical, code signing
CRL distribution points*	[Passthrough from CA configuration]

*CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

CodeSigningCertificate_APICSRPassthrough/V1 definition

This template extends CodeSigningCertificate/V1 to support API and CSR passthrough values.

CodeSigningCertificate_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	`CA:FALSE`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature
Extended key usage	Critical, code signing
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

CodeSigningCertificate_APIPassthrough/V1 definition

This template is identical to the `CodeSigningCertificate` template with one difference: In this template, AWS Private CA passes additional extensions through the API to the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the API.

CodeSigningCertificate_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	`CA:FALSE`
Authority key identifier	[SKI from CA Certificate]

X509v3 Parameter	Value
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature
Extended key usage	Critical, code signing
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

CodeSigningCertificate_CSRPassthrough/V1 definition

This template is identical to the `CodeSigningCertificate` template with one difference: In this template, AWS Private CA passes additional extensions from the certificate signing request (CSR) into the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the CSR.

CodeSigningCertificate_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	`CA:FALSE`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature
Extended key usage	Critical, code signing
CRL distribution points*	[Passthrough from CA configuration or CSR]

*CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

EndEntityCertificate/V1 definition

This template is used to create certificates for end entities such as operating systems or web servers.

EndEntityCertificate/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, key encipherment
Extended key usage	TLS web server authentication, TLS web client authentication
CRL distribution points*	[Passthrough from CA configuration]

*CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

EndEntityCertificate_APICSRPassthrough/V1 definition

This template extends EndEntityCertificate/V1 to support API and CSR passthrough values.

EndEntityCertificate_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]

X509v3 Parameter	Value
Subject	[Passthrough from API or CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, key encipherment
Extended key usage	TLS web server authentication, TLS web client authentication
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

EndEntityCertificate_APIPassthrough/V1 definition

This template is identical to the `EndEntityCertificate` template with one difference: In this template, AWS Private CA passes additional extensions through the API to the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the API.

EndEntityCertificate_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]

X509v3 Parameter	Value
Key usage	Critical, digital signature, key encipherment
Extended key usage	TLS web server authentication, TLS web client authentication
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

EndEntityCertificate_CSRPassthrough/V1 definition

This template is identical to the `EndEntityCertificate` template with one difference: In this template, AWS Private CA passes additional extensions from the certificate signing request (CSR) into the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the CSR.

EndEntityCertificate_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, key encipherment
Extended key usage	TLS web server authentication, TLS web client authentication
CRL distribution points*	[Passthrough from CA configuration or CSR]

*CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

EndEntityClientAuthCertificate/V1 definition

This template differs from the `EndEntityCertificate` only in the Extended key usage value, which restricts it to TLS web client authentication.

EndEntityClientAuthCertificate/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, key encipherment
Extended key usage	TLS web client authentication
CRL distribution points*	[Passthrough from CA configuration or CSR]

*CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

EndEntityClientAuthCertificate_APICSRPassthrough/V1 definition

This template extends EndEntityClientAuthCertificate/V1 to support API and CSR passthrough values.

EndEntityClientAuthCertificate_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]

X509v3 Parameter	Value
Subject	[Passthrough from API or CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, key encipherment
Extended key usage	TLS web client authentication
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

EndEntityClientAuthCertificate_APIPassthrough/V1 definition

This template is identical to the `EndEntityClientAuthCertificate` template with one difference. In this template, AWS Private CA passes additional extensions through the API into the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the API.

EndEntityClientAuthCertificate_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, key encipherment

X509v3 Parameter	Value
Extended key usage	TLS web client authentication
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

EndEntityClientAuthCertificate_CSRPassthrough/V1 definition

This template is identical to the `EndEntityClientAuthCertificate` template with one difference. In this template, AWS Private CA passes additional extensions from the certificate signing request (CSR) into the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the CSR.

EndEntityClientAuthCertificate_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, key encipherment
Extended key usage	TLS web client authentication
CRL distribution points*	[Passthrough from CA configuration or CSR]

*CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

EndEntityServerAuthCertificate/V1 definition

This template differs from the `EndEntityCertificate` only in the Extended key usage value, which restricts it to TLS web server authentication.

EndEntityServerAuthCertificate/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, key encipherment
Extended key usage	TLS web server authentication
CRL distribution points*	[Passthrough from CA configuration]

*CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

EndEntityServerAuthCertificate_APICSRPassthrough/V1 definition

This template extends EndEntityServerAuthCertificate/V1 to support API and CSR passthrough values.

EndEntityServerAuthCertificate_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	CA:FALSE

X509v3 Parameter	Value
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, key encipherment
Extended key usage	TLS web server authentication
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

EndEntityServerAuthCertificate_APIPassthrough/V1 definition

This template is identical to the `EndEntityServerAuthCertificate` template with one difference. In this template, AWS Private CA passes additional extensions through the API into the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the API.

EndEntityServerAuthCertificate_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, key encipherment
Extended key usage	TLS web server authentication
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

EndEntityServerAuthCertificate_CSRPassthrough/V1 definition

This template is identical to the `EndEntityServerAuthCertificate` template with one difference. In this template, AWS Private CA passes additional extensions from the certificate signing request (CSR) into the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the CSR.

EndEntityServerAuthCertificate_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	CA:FALSE
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, key encipherment
Extended key usage	TLS web server authentication
CRL distribution points*	[Passthrough from CA configuration or CSR]

*CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

OCSPSigningCertificate/V1 definition

This template is used to create certificates for signing OCSP responses. The template is identical to the `CodeSigningCertificate` template, except that the Extended key usage value specifies OCSP signing instead of code signing.

OCSPSigningCertificate/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	`CA:FALSE`
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature
Extended key usage	Critical, OCSP signing
CRL distribution points*	[Passthrough from CA configuration]

*CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

OCSPSigningCertificate_APICSRPassthrough/V1 definition

This template extends the OCSPSigningCertificate/V1 to support API and CSR passthrough values.

OCSPSigningCertificate_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	`CA:FALSE`
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature

X509v3 Parameter	Value
Extended key usage	Critical, OCSP signing
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

OCSPSigningCertificate_APIPassthrough/V1 definition

This template is identical to the `OCSPSigningCertificate` template with one difference. In this template, AWS Private CA passes additional extensions through the API into the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the API.

OCSPSigningCertificate_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	`CA:FALSE`
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature
Extended key usage	Critical, OCSP signing
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

OCSPSigningCertificate_CSRPassthrough/V1 definition

This template is identical to the `OCSPSigningCertificate` template with one difference. In this template, AWS Private CA passes additional extensions from the certificate signing request (CSR) into the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the CSR.

OCSPSigningCertificate_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	`CA:FALSE`
Authority key identifier	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature
Extended key usage	Critical, OCSP signing
CRL distribution points*	[Passthrough from CA configuration or CSR]

*CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

RootCACertificate/V1 definition

This template is used to issue self-signed root CA certificates. CA certificates include a critical basic constraints extension with the CA field set to `TRUE` to designate that the certificate can be used to issue CA certificates. The template does not specify a path length (pathLenConstraint) because this could inhibit future expansion of the hierarchy. Extended key usage is excluded to prevent use of the CA certificate as a TLS client or server certificate. No CRL information is specified because a self-signed certificate cannot be revoked.

RootCACertificate/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE`
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, keyCertSign, CRL sign
CRL distribution points	N/A

RootCACertificate_APIPassthrough/V1 definition

This template extends RootCACertificate/V1 to support API passthrough values.

RootCACertificate_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`
Authority key identifier	[Passthrough from API]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, keyCertSign, CRL sign
CRL distribution points*	N/A

BlankRootCACertificate_APIPassthrough/V1 definition

With blank root certificate templates, you can issue root certificates with only X.509 basic constraints present. This is the simplest root certificate that AWS Private CA can issue, but it can be customized using the API structure. The basic constraints extension defines whether or not the certificate is a CA certificate. A blank root certificate template enforces a value of TRUE for basic constraints to ensure that a root CA certificate is issued.

You can use blank passthrough root templates to issue root certificates that require specific values for key usage (KU). For example, key usage might require `keyCertSign` and `cRLSign`, but not `digitalSignature`. Unlike the other non-blank root passthrough certificate template, blank root certificate templates allow the configuration of the KU extension, where KU can be any of the nine supported values (`digitalSignature`, `nonRepudiation`, `keyEncipherment`, `dataEncipherment`, `keyAgreement`, `keyCertSign`, `cRLSign`, `encipherOnly`, and `decipherOnly`).

BlankRootCACertificate_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`
Subject key identifier	[Derived from CSR]

BlankRootCACertificate_PathLen0_APIPassthrough/V1 definition

For general information about blank root CA templates, see ???.

BlankRootCACertificate_PathLen0_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 0`

X509v3 Parameter	Value
Subject key identifier	[Derived from CSR]

BlankRootCACertificate_PathLen1_APIPassthrough/V1 definition

For general information about blank root CA templates, see ???.

BlankRootCACertificate_PathLen1_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 1`
Subject key identifier	[Derived from CSR]

BlankRootCACertificate_PathLen2_APIPassthrough/V1 definition

For general information about blank root CA templates, see ???.

BlankRootCACertificate_PathLen2_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 2`
Subject key identifier	[Derived from CSR]

BlankRootCACertificate_PathLen3_APIPassthrough/V1 definition

For general information about blank root CA templates, see ???.

BlankRootCACertificate_PathLen3_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE, pathlen: 3`
Subject key identifier	[Derived from CSR]

SubordinateCACertificate_PathLen0/V1 definition

This template is used to issue subordinate CA certificates with a path length of `0`. CA certificates include a critical basic constraints extension with the CA field set to TRUE to designate that the certificate can be used to issue CA certificates. Extended key usage is not included, which prevents the CA certificate from being used as a TLS client or server certificate.

For more information about certification paths, see Setting Length Constraints on the Certification Path.

SubordinateCACertificate_PathLen0/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE, pathlen: 0`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration]

*CRL distribution points are included in certificates that are issued with this template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen0_APICSRPassthrough/V1 definition

This template extends SubordinateCACertificate_PathLen0/V1 to support API and CSR passthrough values.

SubordinateCACertificate_PathLen0_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 0`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen0_APIPassthrough/V1 definition

This template extends SubordinateCACertificate_PathLen0/V1 to support API passthrough values.

SubordinateCACertificate_PathLen0_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]

X509v3 Parameter	Value
Basic constraints	Critical, `CA:TRUE`, `pathlen: 0`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen0_CSRPassthrough/V1 definition

This template is identical to the `SubordinateCACertificate_PathLen0` template with one difference: In this template, AWS Private CA passes additional extensions from the certificate signing request (CSR) into the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the CSR.

> **Note**
>
> A CSR that contains custom additional extensions must be created outside of AWS Private CA.

SubordinateCACertificate_PathLen0_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 0`
Authority key identifier	[SKI from CA Certificate]

X509v3 Parameter	Value
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration or CSR]

*CRL distribution points are included in certificates issued with this template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen1/V1 definition

This template is used to issue subordinate CA certificates with a path length of 1. CA certificates include a critical Basic constraints extension with the CA field set to TRUE to designate that the certificate can be used to issue CA certificates. Extended key usage is not included, which prevents the CA certificate from being used as a TLS client or server certificate.

For more information about certification paths, see Setting Length Constraints on the Certification Path.

SubordinateCACertificate_PathLen1/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 1`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration]

*CRL distribution points are included in certificates issued with this template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen1_APICSRPassthrough/V1 definition

This template extends SubordinateCACertificate_PathLen1/V1 to support API and CSR passthrough values.

SubordinateCACertificate_PathLen1_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 1`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen1_APIPassthrough/V1 definition

This template extends SubordinateCACertificate_PathLen0/V1 to support API passthrough values.

SubordinateCACertificate_PathLen1_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]

X509v3 Parameter	Value
Basic constraints	Critical, `CA:TRUE, pathlen: 1`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen1_CSRPassthrough/V1 definition

This template is identical to the `SubordinateCACertificate_PathLen1` template with one difference: In this template, AWS Private CA passes additional extensions from the certificate signing request (CSR) into the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the CSR.

> **Note**
>
> A CSR that contains custom additional extensions must be created outside of AWS Private CA.

SubordinateCACertificate_PathLen1_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE, pathlen: 1`
Authority key identifier	[SKI from CA Certificate]

X509v3 Parameter	Value
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration or CSR]

*CRL distribution points are included in certificates issued with this template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen2/V1 definition

This template is used to issue subordinate CA certificates with a path length of 2. CA certificates include a critical Basic constraints extension with the CA field set to TRUE to designate that the certificate can be used to issue CA certificates. Extended key usage is not included, which prevents the CA certificate from being used as a TLS client or server certificate.

For more information about certification paths, see Setting Length Constraints on the Certification Path.

SubordinateCACertificate_PathLen2/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 2`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration]

*CRL distribution points are included in certificates issued with this template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen2_APICSRPassthrough/V1 definition

This template extends SubordinateCACertificate_PathLen2/V1 to support API and CSR passthrough values.

SubordinateCACertificate_PathLen2_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 2`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen2_APIPassthrough/V1 definition

This template extends SubordinateCACertificate_PathLen2/V1 to support API passthrough values.

SubordinateCACertificate_PathLen2_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]

X509v3 Parameter	Value
Basic constraints	Critical, `CA:TRUE, pathlen: 2`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen2_CSRPassthrough/V1 definition

This template is identical to the `SubordinateCACertificate_PathLen2` template with one difference: In this template, AWS Private CA passes additional extensions from the certificate signing request (CSR) into the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the CSR.

> **Note**
>
> A CSR that contains custom additional extensions must be created outside of AWS Private CA.

SubordinateCACertificate_PathLen2_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE, pathlen: 2`
Authority key identifier	[SKI from CA Certificate]

X509v3 Parameter	Value
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration or CSR]

*CRL distribution points are included in certificates issued with this template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen3/V1 definition

This template is used to issue subordinate CA certificates with a path length of 3. CA certificates include a critical Basic constraints extension with the CA field set to TRUE to designate that the certificate can be used to issue CA certificates. Extended key usage is not included, which prevents the CA certificate from being used as a TLS client or server certificate.

For more information about certification paths, see Setting Length Constraints on the Certification Path.

SubordinateCACertificate_PathLen3/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 3`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration]

*CRL distribution points are included in certificates issued with this template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen3_APICSRPassthrough/V1 definition

This template extends SubordinateCACertificate_PathLen3/V1 to support API and CSR passthrough values.

SubordinateCACertificate_PathLen3_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 3`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration or CSR]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen3_APIPassthrough/V1 definition

This template extends SubordinateCACertificate_PathLen3/V1 to support API passthrough values.

SubordinateCACertificate_PathLen3_APIPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]
Subject	[Passthrough from API or CSR]

X509v3 Parameter	Value
Basic constraints	Critical, `CA:TRUE`, `pathlen: 3`
Authority key identifier	[SKI from CA Certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration]

* CRL distribution points are included in the template only if the CA is configured with CRL generation enabled.

SubordinateCACertificate_PathLen3_CSRPassthrough/V1 definition

This template is identical to the `SubordinateCACertificate_PathLen3` template with one difference: In this template, AWS Private CA passes additional extensions from the certificate signing request (CSR) into the certificate if the extensions are not specified in the template. Extensions specified in the template always override extensions in the CSR.

> **Note**
>
> A CSR that contains custom additional extensions must be created outside of AWS Private CA.

SubordinateCACertificate_PathLen3_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 3`
Authority key identifier	[SKI from CA Certificate]

X509v3 Parameter	Value
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, `keyCertSign` , CRL sign
CRL distribution points*	[Passthrough from CA configuration or CSR]

*CRL distribution points are included in certificates issued with this template only if the CA is configured with CRL generation enabled.

Security in AWS Private Certificate Authority

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are 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 Private Certificate Authority, 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 Private CA. The following topics show you how to configure AWS Private CA 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 Private CA resources.

Topics

- Identity and Access Management (IAM) for AWS Private Certificate Authority
- Security best practices for Cross-account access to private CAs
- Data protection in AWS Private Certificate Authority
- Compliance validation for AWS Private Certificate Authority
- Infrastructure security in AWS Private Certificate Authority
- AWS Private Certificate Authority Customer CP/CPS Framework

Identity and Access Management (IAM) for AWS Private Certificate Authority

Access to AWS Private CA requires credentials that AWS can use to authenticate your requests. The following topics provide details on how you can use AWS Identity and Access Management (IAM) to help secure your private certificate authorities (CAs) by controlling who can access them.

In AWS Private CA, the primary resource that you work with is a *certificate authority (CA)*. Every private CA that you own or control is identified by an Amazon Resource Name (ARN), which has the following form.

```
arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566
```

A *resource owner* is the *principal entity* of the AWS account in which an AWS resource is created. The following examples illustrate how this works.

- If you use the credentials of your AWS account root user to create a private CA, your AWS account owns the CA.

  > **Important**
  >
  > - We do not advise using an AWS account root user to create CAs.
  > - We strongly recommend the use of multi-factor authentication (MFA) any time you access AWS Private CA.

- If you create an IAM user in your AWS account, you can grant that user permission to create a private CA. However, the account to which that user belongs owns the CA.
- If you create an IAM role in your AWS account and grant it permission to create a private CA, anyone who can assume the role can create the CA. However, the account to which the role belongs will own the private CA.

A *permissions policy* describes who has access to what. The following discussion explains the available options for creating permissions policies.

> **Note**
>
> This documentation discusses using IAM in the context of AWS Private CA. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see the IAM User Guide. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference.

AWS Private CA API operations and permissions

When you set up access control and permissions policies that you plan to attach to an IAM identity (identity-based policies), use the following table as a reference. The first column in the table lists each AWS Private CA API operation. You specify actions in a policy's `Action` element. The remaining columns provide the additional information.

AWS Private CA API operations	Required permissions	Resources
CreateCertificateAuthority	`acm-pca:CreateCertificateAuthority` `acm-pca:TagCertificateAuthority` (Only required when creating a CA with tags.)	`arn:aws:acm-pca:us-east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566`
CreateCertificateAuthorityAuditReport	`acm-pca:CreateCertificateAuthorityAuditReport`	`arn:aws:acm-pca:us-east-1:111122223333:certificate-authority/11223344-1234-1122-2233-112233445566`
CreatePermission	`acm-pca:CreatePermission`	`arn:aws:acm-pca:us-east-1:111122223333:certificate-authority/11223344-`

AWS Private CA API operations	Required permissions	Resources
		1234-1122-2233-112 233445566
DeleteCertificateAuthority	acm-pca:DeleteCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
DeletePermission	acm-pca:DeletePermission	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
DeletePolicy	acm-pca:DeletePolicy	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
DescribeCertificateAuthority	acm-pca:DescribeCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

AWS Private CA API operations	Required permissions	Resources
DescribeCertificateAuthorityAuditReport	acm-pca:DescribeCertificateAuthorityAuditReport	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
GetCertificate	acm-pca:GetCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
GetCertificateAuthorityCertificate	acm-pca:GetCertificateAuthorityCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
GetCertificateAuthorityCsr	acm-pca:GetCertificateAuthorityCsr	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
GetPolicy	acm-pca:GetPolicy	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

AWS Private CA API operations	Required permissions	Resources
ImportCertificateAuthorityCertificate	acm-pca:ImportCertificateAuthorityCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
IssueCertificate	acm-pca:IssueCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
ListCertificateAuthorities	acm-pca:ListCertificateAuthorities	N/A
ListPermissions	acm-pca:ListPermissions	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
ListTags	acm-pca:ListTags	N/A
PutPolicy	acm-pca:PutPolicy	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

AWS Private CA API operations	Required permissions	Resources
RevokeCertificate	acm-pca:RevokeCertificate	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
TagCertificateAuthority	acm-pca:TagCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
UntagCertificateAuthority	acm-pca:UntagCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*
UpdateCertificateAuthority	acm-pca:UpdateCertificateAuthority	arn:*aws*:acm-pca: *us-east-1* :*111122223333* :certificate-authority/ *11223344-1234-1122-2233-112233445566*

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.

- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.

- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

AWS managed policies

AWS Private CA includes a set of predefined AWS managed policies for AWS Private CA administrators, users, and auditors. Understanding these policies can help you implement Customer managed policies.

Choose any of the policies listed below to see details and sample policy code.

AWSPrivateCAFullAccess

Grants unrestricted administrative control.

For a JSON listing of the policy details, see AWSPrivateCAFullAccess.

AWSPrivateCAReadOnly

Grants access limited to read-only API operations.

For a JSON listing of the policy details, see AWSPrivateCAReadOnly.

AWSPrivateCAPrivilegedUser

Grants ability to issue and revoke CA certificates. This policy has no other administrative capabilities and no ability to issue end-entity certificates. Permissions are mutually exclusive with the **User** policy.

For a JSON listing of the policy details, see AWSPrivateCAPrivilegedUser.

AWSPrivateCAUser

Grant ability to issue and revoke end-entity certificates. This policy has no administrative capabilities and no ability to issue CA certificates. Permissions are mutually exclusive with the **PrivilegedUser** policy.

For a JSON listing of the policy details, see AWSPrivateCAUser.

AWSPrivateCAAuditor

Grant access to read-only API operations and permission to generate a CA audit report.

For a JSON listing of the policy details, see AWSPrivateCAAuditor.

AWSPrivateCAConnectorForKubernetesPolicy

Grants essential permissions for the AWS Private CA Connector for Kubernetes.

For a JSON listing of the policy details, see AWSPrivateCAConnectorForKubernetesPolicy.

Updates to AWS managed policies for AWS Private CA

In the following table, view details about updates to AWS managed policies for AWS Private CA since the service began tracking these changes. For automatic alerts about all changes to AWS Private CA, subscribe to the RSS feed on the Document History page.

Managed policy changes

Change	Description	Date
New Policy: AWSPrivateCAConnectorForKubernetesPolicy	New managed policy introduced for use with AWS Private CA Connector for Kubernetes.	May 19, 2025
AWSPrivateCAPrivilegedUser and AWSPrivateCAUser - Updated policy	Replaced `StringLike` with `ArnLike`, and `StringNotLike` with `ArnNotLike`. Updated template arn to include wild cards `arn:aws:acm-pca:::`	January 22, 2025

Change	Description	Date
	`template` to `arn:aws:acm-pca:*:*:template` .	
New policy names: • `AWSPrivateCAFullAccess` • `AWSPrivateCAReadOnly` • `AWSPrivateCAPrivilegedUser` • `AWSPrivateCAAuditor` • `AWSPrivateCAUser`	Policy name prefixes were changed from `AWSCertificateManagerPrivateCA` to `AWSPrivateCA` . Functionality remains unchanged.	February 13, 2023

Customer managed policies

As a best practice, don't use your AWS account root user to interact with AWS, including AWS Private CA. Instead use AWS Identity and Access Management (IAM) to create an IAM user, IAM role, or federated user. Create an administrator group and add yourself to it. Then log in as an administrator. Add additional users to the group as needed.

Another best practice is to create a customer managed IAM policy that you can assign to users. Customer managed policies are standalone identity-based policies that you create and which you can attach to multiple users, groups, or roles in your AWS account. Such a policy restricts users to performing only the AWS Private CA actions that you specify.

The following example customer-managed policy allows a user to create a CA audit report. This is an example only. You can choose any AWS Private CA operations that you want. For more examples, see Inline policies.

To create a customer managed policy

1. Sign in to the IAM console using the credentials of an AWS administrator.
2. In the navigation pane of the console, choose **Policies**.
3. Choose **Create policy**.

4. Choose the **JSON** tab.
5. Copy the following policy and paste it into the editor.

   JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"acm-pca:CreateCertificateAuthorityAuditReport",
         "Resource":"*"
      }
   ]
}
```

6. Choose **Review policy**.
7. For **Name**, type `PcaListPolicy`.
8. (Optional) Type a description.
9. Choose **Create policy**.

An administrator can attach the policy to any IAM user to limit what AWS Private CA actions the user can perform. For ways to apply a permissions policy, see Changing Permissions for an IAM User in the *IAM User Guide*.

Inline policies

Inline policies are policies that you create and manage and embed directly into a user, group, or role. The following policy examples show how to assign permissions to perform AWS Private CA actions. For general information about inline policies, see Working with Inline Policies in the IAM User Guide. You can use the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the IAM API to create and embed inline policies.

> **Important**
>
> We strongly recommend the use of multi-factor authentication (MFA) any time you access AWS Private CA.

Topics

- Listing private CAs
- Retrieving a private CA certificate
- Importing a private CA certificate
- Deleting a private CA
- Tag-on-create: Attaching tags to a CA at the time of creation
- Tag-on-create: Restricted tagging
- Controlling access to Private CA using tags
- Read-only access to AWS Private CA
- Full access to AWS Private CA

Listing private CAs

The following policy allows a user to list all of the private CAs in an account.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"acm-pca:ListCertificateAuthorities",
         "Resource":"*"
      }
   ]
}
```

Retrieving a private CA certificate

The following policy allows a user to retrieve a specific private CA certificate.

JSON

```
{
```

```
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:GetCertificateAuthorityCertificate",
      "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID/certificate/certificate_ID"
   }
}
```

Importing a private CA certificate

The following policy allows a user to import a private CA certificate.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:ImportCertificateAuthorityCertificate",
      "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID/certificate/certificate_ID"
   }
}
```

Deleting a private CA

The following policy allows a user to delete a specific private CA.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:DeleteCertificateAuthority",
      "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID/certificate/certificate_ID"   }
}
```

Tag-on-create: Attaching tags to a CA at the time of creation

The following policy allows a user to apply tags during CA creation.

JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Action": [
            "acm-pca:CreateCertificateAuthority",
            "acm-pca:TagCertificateAuthority"
         ],
         "Effect": "Allow",
         "Resource": "*"
      }
   ]
}
```

Tag-on-create: Restricted tagging

The following tag-on-create policy *prevents* use of the key-value pair Environment=Prod during CA creation. Tagging with other key-value pairs is allowed.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"acm-pca:*",
         "Resource":"*"
      },
      {
         "Effect":"Deny",
         "Action":"acm-pca:TagCertificateAuthority",
         "Resource":"*",
         "Condition":{
```

```
            "StringEquals":{
               "aws:ResourceTag/Environment":[
                  "Prod"
               ]
            }
         }
      }
   ]
}
```

Controlling access to Private CA using tags

The following policy allows access only to CAs with the key-value pair Environment=PreProd. It also requires that new CAs include this tag.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:*"
         ],
         "Resource":"*",
         "Condition":{
            "StringEquals":{
               "aws:ResourceTag/Environment":[
                  "PreProd"
               ]
            }
         }
      }
   ]
}
```

Read-only access to AWS Private CA

The following policy allows a user to describe and list private certificate authorities and to retrieve the private CA certificate and certificate chain.

JSON

```
{
    "Version":"2012-10-17",
    "Statement":{
       "Effect":"Allow",
       "Action":[
          "acm-pca:DescribeCertificateAuthority",
          "acm-pca:DescribeCertificateAuthorityAuditReport",
          "acm-pca:ListCertificateAuthorities",
          "acm-pca:ListTags",
          "acm-pca:GetCertificateAuthorityCertificate",
          "acm-pca:GetCertificateAuthorityCsr",
          "acm-pca:GetCertificate"
       ],
       "Resource":"*"
    }
}
```

Full access to AWS Private CA

The following policy allows a user to perform any AWS Private CA action.

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:*"
         ],
         "Resource":"*"
      }
```

```
    ]
}
```

Security best practices for Cross-account access to private CAs

An AWS Private CA administrator can share a CA with principals (users, roles, etc.) in another AWS account. When a share has been received and accepted, the principal can use the CA to issue end-entity certificates using AWS Private CA or AWS Certificate Manager resources. The principal can use the CA to issue subordinate CA certificates using AWS Private CA.

> **Important**
>
> Charges associated with a certificate issued in a cross-account scenario are billed to the AWS account that issues the certificate.

To share access to a CA, AWS Private CA administrators can choose either of the following methods:

- Use AWS Resource Access Manager (RAM) to share the CA as a resource with a principal in another account or with AWS Organizations. RAM is a standard method for sharing AWS resources across accounts. For more information about RAM, see the AWS RAM User Guide. For more information about AWS Organizations, see the AWS Organizations User Guide.
- Use the AWS Private CA API or CLI to attach a resource-based policy to a CA, thereby granting access to a principal in another account. For more information, see Resource-based policies.

The Control access to the private CA section of this guide provides workflows for granting access to CAs in both single-account and cross-account scenarios.

Resource-based policies

Resource-based policies are permissions policies that you create and manually attach to a resource (in this case, a private CA) rather than to a user identity or role. Or, instead of creating your own policies, you can use AWS managed policies for AWS Private CA. Using AWS RAM to apply a resource-based policy, an AWS Private CA administrator can share access to a CA with a user in a different AWS account directly or through AWS Organizations. Alternatively, an AWS Private CA administrator can use the PCA APIs PutPolicy, GetPolicy, and DeletePolicy, or the corresponding

AWS CLI commands put-policy, get-policy, and delete-policy, to apply and manage resource-based policies.

For general information about resource-based policies, see Identity-Based Policies and Resource-Based Policies and Controlling Access Using Policies.

To view the list of AWS managed resource-based policies for AWS Private CA, navigate to the Managed permissions library in the AWS Resource Access Manager console, and search for **CertificateAuthority**. As with any policy, before you apply it, we recommend applying the policy in a test environment to ensure that it meets your requirements.

AWS Certificate Manager (ACM) users with cross-account shared access to a private CA can issue managed certificates that are signed by the CA. Cross-account issuers are constrained by a resource-based policy and have access only to the following end-entity certificate templates:

- EndEntityCertificate/V1
- EndEntityClientAuthCertificate/V1
- EndEntityServerAuthCertificate/V1
- BlankEndEntityCertificate_APIPassthrough/V1
- BlankEndEntityCertificate_APICSRPassthrough/V1
- SubordinateCACertificate_PathLen0/V1

Policy examples

This section provides example cross-account policies for various needs. In all cases, the following command pattern is used to apply a policy:

```
$ aws acm-pca put-policy \
   --region region \
   --resource-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
   --policy file:///[path]/policyN.json
```

In addition to specifying the ARN of a CA, the administrator provides an AWS account ID or an AWS Organizations ID that will be granted access to the CA. The JSON of each of the following polices is formatted as a file for readability, but can also be supplied as an inline CLI arguments.

> **Note**
>
> The structure of the JSON resource-based polices shown below must be followed precisely. Only the ID fields for the principals (the AWS account number or the AWS Organizations ID) and the CA ARNs can be configured by customers.

1. **File: policy1.json – Sharing access to a CA with a user in a different account**

   Replace *555555555555* with the AWS account ID that's sharing the CA.

   For the resource ARN, replace the following with your own values:

   - *aws* - The AWS partition. For example, `aws`, `aws-us-gov`, `aws-cn`, etc.
   - *us-east-1* - The AWS Region that the resource is available in, such as `us-west-1`.
   - *111122223333* - The AWS account ID of the resource owner.
   - *11223344-1234-1122-2233-112233445566* - The resource ID of the certificate authority.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
            "Sid": "ExampleStatementID",
            "Effect": "Allow",
            "Principal": {
                "AWS": "555555555555"
            },
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListPermissions",
                "acm-pca:ListTags"
            ],
            "Resource": "arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID"
        },
        {
            "Sid": "ExampleStatementID2",
            "Effect": "Allow",
            "Principal": {
```

```
                "AWS": "555555555555"
            },
            "Action": [
                "acm-pca:IssueCertificate"
            ],
            "Resource": "arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID",
            "Condition": {
                "StringEquals": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
EndEntityCertificate/V1"
                }
            }
        }
    ]
}
```

2. **File: policy2.json – Sharing access to a CA through AWS Organizations**

   Replace *o-a1b2c3d4z5* with the AWS Organizations ID.

   For the resource ARN, replace the following with your own values:

   - *aws* - The AWS partition. For example, aws, aws-us-gov, aws-cn, etc.
   - *us-east-1* - The AWS Region that the resource is available in, such as us-west-1.
   - *111122223333* - The AWS account ID of the resource owner.
   - *11223344-1234-1122-2233-112233445566* - The resource ID of the certificate authority.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ExampleStatementID3",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "acm-pca:IssueCertificate",
            "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID",
            "Condition": {
                "StringEquals": {
```

```
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
EndEntityCertificate/V1",
                    "aws:PrincipalOrgID": "o-a1b2c3d4z5"
                },
                "StringNotEquals": {
                    "aws:PrincipalAccount": "111122223333"
                }
            }
        },
        {
            "Sid": "ExampleStatementID4",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListPermissions",
                "acm-pca:ListTags"
            ],
            "Resource":"arn:aws:acm-pca:us-east-1:123456789012:certificate-
authority/CA_ID",
            "Condition": {
                "StringEquals": {
                    "aws:PrincipalOrgID": "o-a1b2c3d4z5"
                },
                "StringNotEquals": {
                    "aws:PrincipalAccount": "111122223333"
                }
            }
        }
    ]
}
```

Data protection in AWS Private Certificate Authority

The AWS shared responsibility model applies to data protection in AWS Private Certificate Authority. 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 Private CA 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.

Storage and security compliance of AWS Private CA private keys

The private keys for private CAs are stored in AWS managed hardware security modules (HSMs). The HSMs comply with FIPS PUB 140-2 Level 3 Security Requirements for Cryptographic Modules.

Data encryption in AWS Private CA Connector for Active Directory

AWS Private CA Connector for AD stores customer configuration data regarding connectors, templates, directory registrations, service principal names, and template group access control entries. This data is encrypted in transit and at rest. Information about certificates issued through

Connector for AD can be discovered using the GetCertificate action in the AWS Private CA API. No information regarding the certificates issued, or regarding the client or machine requesting a certificate, is stored by AWS.

Compliance validation for AWS Private Certificate Authority

Third-party auditors assess the security and compliance of AWS Private Certificate Authority as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see AWS Services in Scope by Compliance Program. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS Private CA is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- For organizations that are required to encrypt their Amazon S3 buckets, the following topics describe how to configure encryption to accommodate AWS Private CA assets:
  - Encrypting Your Audit Reports
  - Encrypting Your CRLs
- Security and Compliance Quick Start Guides — These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on AWS.
- Architecting for HIPAA Security and Compliance Whitepaper — This whitepaper describes how companies can use AWS to create HIPAA-compliant applications.
- AWS Compliance Resources — This collection of workbooks and guides might apply to your industry and location.
- Evaluating Resources with Rules in the *AWS Config Developer Guide* — The AWS Config service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- AWS Security Hub CSPM — This AWS service provides a comprehensive view of your security state within AWS that helps you check your compliance with security industry standards and best practices.

Use audit reports with your private CA

You can create an audit report to list all of the certificates that your private CA has issued or revoked. The report is saved in a new or existing S3 bucket that you specify on input.

For information about adding encryption protection to your audit reports, see Encrypting your audit reports .

The audit report file has the following path and file name. The ARN for an Amazon S3 bucket is the value for `amzn-s3-demo-bucket`. `CA_ID` is the unique identifier of an issuing CA. `UUID` is the unique identifier of an audit report.

```
amzn-s3-demo-bucket/audit-report/CA_ID/UUID.[json|csv]
```

You can generate a new report every 30 minutes and download it from your bucket. The following example shows a CSV-separated report.

```
awsAccountId,requestedByServicePrincipal,certificateArn,serial,subject,notBefore,notAfter,issue
123456789012,,arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/
certificate_ID,00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff,"2.5.4.5=#012345678901,2.5.4.44=
 Company,L=Seattle,ST=Washington,C=US",2020-03-02T21:43:57+0000,2020-04-07T22:43:57+0000,2020-0
pca:::template/EndEntityCertificate/V1
123456789012,acm.amazonaws.com,arn:aws:acm-pca:region:account:certificate-
authority/CA_ID/
certificate/
certificate_ID,ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00,"2.5.4.5=#012345678901,2.5.4.44=
 Company,L=Seattle,ST=Washington,C=US",2020-03-02T20:53:39+0000,2020-04-07T21:53:39+0000,2020-0
pca:::template/EndEntityCertificate/V1
```

The following example shows a JSON-formatted report.

```
[
   {
      "awsAccountId":"123456789012",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
      "serial":"00:11:22:33:44:55:66:77:88:99:aa:bb:cc:dd:ee:ff",

 "subject":"2.5.4.5=#012345678901,2.5.4.44=#0a1b3c4d,2.5.4.65=#0a1b3c4d5e6f,2.5.4.43=#0a1b3c4d5
 Company,L=Seattle,ST=Washington,C=US",
```

```
      "notBefore":"2020-02-26T18:39:57+0000",
      "notAfter":"2021-02-26T19:39:57+0000",
      "issuedAt":"2020-02-26T19:39:58+0000",
      "revokedAt":"2020-02-26T20:00:36+0000",
      "revocationReason":"UNSPECIFIED",
      "templateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
   },
   {
      "awsAccountId":"123456789012",
      "requestedByServicePrincipal":"acm.amazonaws.com",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID",
      "serial":"ff:ee:dd:cc:bb:aa:99:88:77:66:55:44:33:22:11:00",

 "subject":"2.5.4.5=#012345678901,2.5.4.44=#0a1b3c4d,2.5.4.65=#0a1b3c4d5e6f,2.5.4.43=#0a1b3c4d5
 Company,L=Seattle,ST=Washington,C=US",
      "notBefore":"2020-01-22T20:10:49+0000",
      "notAfter":"2021-01-17T21:10:49+0000",
      "issuedAt":"2020-01-22T21:10:49+0000",
      "templateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
   }
]
```


Note

When AWS Certificate Manager renews a certificate, the private CA audit report populates the `requestedByServicePrincipal` field with `acm.amazonaws.com`. This indicates that the AWS Certificate Manager service called the `IssueCertificate` action of the AWS Private CA API on behalf of a customer to renew the certificate.

Prepare an Amazon S3 bucket for audit reports

Important

AWS Private CA doesn't support the use of Amazon S3 Object Lock. If you enable Object Lock on your bucket, AWS Private CA isn't able to write audit reports to the bucket.

To store your audit reports, you need to prepare an Amazon S3 bucket. For more information, see How Do I Create an S3 bucket?

Your S3 bucket must be secured by a permissions policy that allows AWS Private CA to access and write to the S3 bucket that you specify. Authorized users and service principals require `Put` permission to allow AWS Private CA to place objects in the bucket, and `Get` permission to retrieve them. We recommend that you apply the policy shown below, which restricts access to both an AWS account and the ARN of a private CA. Alternatively, you can use the aws:SourceOrgID condition key to constrain access to a specific organization in AWS Organizations. For more information about bucket policies, see Bucket policies for Amazon Simple Storage Service.

Create an audit report

You can create an audit report from either the console or the AWS CLI.

To create an audit report (console)

1. Sign in to your AWS account and open the AWS Private CA console at https://console.aws.amazon.com/acm-pca/home.
2. On the **Private certificate authories** page, choose your private CA from the list.
3. From the **Actions** menu, choose **Generate audit report**.
4. Under **Audit report destination**, for **Create a new S3 bucket?**, choose **Yes** and type a unique bucket name, or choose **No** and choose an existing bucket from the list.

   If you choose **Yes**, AWS Private CA creates and attaches the default policy to your bucket. The default policy includes an `aws:SourceAccount` condition key, which limits access to a specific AWS account. If you wish to further constrain access, you can add other condition keys to the policy such as in the preceding example.

   If you choose **No**, you must attach a policy to your bucket before you can generate an audit report. Use the policy pattern described in Prepare an Amazon S3 bucket for audit reports. For information about attaching a policy, see Adding a bucket policy using the Amazon S3 console.
5. Under **Output format**, choose **JSON** for JavaScript Object Notation or **CSV** for comma-separated values.
6. Choose **Generate audit report**.

To create an audit report (AWS CLI)

1. If you do not already have an S3 bucket to use, create one.

2. Attach a policy to your bucket. Use the policy pattern described in Prepare an Amazon S3 bucket for audit reports. For information about attaching a policy, see Adding a bucket policy using the Amazon S3 console
3. Use the create-certificate-authority-audit-report command to create the audit report and to place it in the prepared S3 bucket.

```
$ aws acm-pca create-certificate-authority-audit-report \
--certificate-authority-arn arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566 \
--s3-bucket-name amzn-s3-demo-bucket \
--audit-report-response-format JSON
```

Retrieve an audit report

To retrieve an audit report for inspection, use the Amazon S3 console, API, CLI, or SDK. For more information, see Downloading an object in the *Amazon Simple Storage Service User Guide*.

Encrypting your audit reports

You can optionally configure encryption on the Amazon S3 bucket containing your audit reports. AWS Private CA supports two encryption modes for assets in S3:

- Automatic server-side encryption with Amazon S3-managed AES-256 keys.
- Customer managed encryption using AWS Key Management Service and an AWS KMS key configured to your specifications.

> **Note**
>
> AWS Private CA does not support using default KMS keys generated automatically by S3.

The following procedures describe how to set up each of the encryption options.

To configure automatic encryption

Complete the following steps to enable S3 server-side encryption.

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.

2. In the **Buckets** table, choose the bucket that will hold your AWS Private CA assets.
3. On the page for your bucket, choose the **Properties** tab.
4. Choose the **Default encryption** card.
5. Choose **Enable**.
6. Choose **Amazon S3 key (SSE-S3)**.
7. Choose **Save Changes**.

To configure custom encryption

Complete the following steps to enable encryption using a custom key.

1. Open the Amazon S3 console at https://console.aws.amazon.com/s3/.
2. In the **Buckets** table, choose the bucket that will hold your AWS Private CA assets.
3. On the page for your bucket, choose the **Properties** tab.
4. Choose the **Default encryption** card.
5. Choose **Enable**.
6. Choose **AWS Key Management Service key (SSE-KMS)**.
7. Choose either **Choose from your AWS KMS keys** or **Enter AWS KMS key ARN**.
8. Choose **Save Changes**.
9. (Optional) If you do not have an KMS key already, create one using the following AWS CLI create-key command:

```
$ aws kms create-key
```

The output contains the key ID and Amazon Resource Name (ARN) of the KMS key. The following is an example output:

```
{
    "KeyMetadata": {
        "KeyId": "01234567-89ab-cdef-0123-456789abcdef",
        "Description": "",
        "Enabled": true,
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "CreationDate": 1478910250.94,
```

```
        "Arn": "arn:aws:kms:us-west-2:123456789012:key/01234567-89ab-
cdef-0123-456789abcdef",
        "AWSAccountId": "123456789012"
    }
}
```

10. Using the following steps, you give the AWS Private CA service principal permission to use the KMS key. By default, all KMS keys are private; only the resource owner can use a KMS key to encrypt and decrypt data. However, the resource owner can grant permissions to access the KMS key to other users and resources. The service principal must be in the same Region as where the KMS key is stored.

    a. First, save the default policy for your KMS key as `policy.json` using the following get-key-policy command:

    ```
    $ aws kms get-key-policy --key-id key-id --policy-name default --output text
     > ./policy.json
    ```

    b. Open the `policy.json` file in a text editor. Select one of the following policy statements and add it to the existing policy.

    If your Amazon S3 bucket key is *enabled*, use the following statement:

    ```
    {
       "Sid":"Allow ACM-PCA use of the key",
       "Effect":"Allow",
       "Principal":{
          "Service":"acm-pca.amazonaws.com"
       },
       "Action":[
          "kms:GenerateDataKey",
          "kms:Decrypt"
       ],
       "Resource":"*",
       "Condition":{
          "StringLike":{
             "kms:EncryptionContext:aws:s3:arn":"arn:aws:s3:::bucket-name"
          }
       }
    }
    ```

    If your Amazon S3 bucket key is *disabled*, use the following statement:

```
{
   "Sid":"Allow ACM-PCA use of the key",
   "Effect":"Allow",
   "Principal":{
      "Service":"acm-pca.amazonaws.com"
   },
   "Action":[
      "kms:GenerateDataKey",
      "kms:Decrypt"
   ],
   "Resource":"*",
   "Condition":{
      "StringLike":{
         "kms:EncryptionContext:aws:s3:arn":[
            "arn:aws:s3:::bucket-name/acm-pca-permission-test-key",
            "arn:aws:s3:::bucket-name/acm-pca-permission-test-key-private",
            "arn:aws:s3:::bucket-name/audit-report/*",
            "arn:aws:s3:::bucket-name/crl/*"
         ]
      }
   }
}
```

c. Finally, apply the updated policy using the following put-key-policy command:

```
$ aws kms put-key-policy --key-id key_id --policy-name default --policy file://
policy.json
```

Infrastructure security in AWS Private Certificate Authority

As a managed service, AWS Private Certificate Authority is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access AWS Private CA through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.

- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

AWS Private CA VPC endpoints (AWS PrivateLink)

You can create a private connection between your VPC and AWS Private CA by configuring an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink, a technology for privately accessing AWS Private CA API operations. AWS PrivateLink routes all network traffic between your VPC and AWS Private CA through the Amazon network, avoiding exposure on the open internet. Each VPC endpoint is represented by one or more elastic network interfaces with private IP addresses in your VPC subnets.

The interface VPC endpoint connects your VPC directly to AWS Private CA without an internet gateway, NAT device, VPN connection, or Direct Connect connection. The instances in your VPC don't need public IP addresses to communicate with the AWS Private CA API.

To use AWS Private CA through your VPC, you must connect from an instance that is inside the VPC. Alternatively, you can connect your private network to your VPC by using an AWS Virtual Private Network (Site-to-Site VPN) or Direct Connect. For information about Site-to-Site VPN, see VPN Connections in the *Amazon VPC User Guide*. For information about Direct Connect, see Creating a Connection in the *Direct Connect User Guide*.

AWS Private CA does not require the use of AWS PrivateLink, but we recommend it as an additional layer of security. For more information about AWS PrivateLink and VPC endpoints, see Accessing Services Through AWS PrivateLink.

Considerations for AWS Private CA VPC endpoints

Before you set up interface VPC endpoints for AWS Private CA, be aware of the following considerations:

- AWS Private CA might not support VPC endpoints in some Availability Zones. When you create a VPC endpoint, first check support in the management console. Unsupported Availability Zones are marked "Service not supported in this Availability Zone."
- VPC endpoints do not support cross-Region requests. Ensure that you create your endpoint in the same Region where you plan to issue your API calls to AWS Private CA.

- VPC endpoints only support Amazon provided DNS through Amazon Route 53. If you want to use your own DNS, you can use conditional DNS forwarding. For more information, see DHCP Options Sets in the *Amazon VPC User Guide*.
- The security group attached to the VPC endpoint must allow incoming connections on port 443 from the private subnet of the VPC.

AWS Private CA API currently supports VPC endpoints in the following AWS Regions:

- US East (Ohio)
- US East (N. Virginia)
- US West (N. California)
- US West (Oregon)
- Africa (Cape Town)
- Asia Pacific (Hong Kong)
- Asia Pacific (Hyderabad)
- Asia Pacific (Jakarta)
- Asia Pacific (Melbourne)
- Asia Pacific (Mumbai)
- Asia Pacific (Osaka)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Canada West (Calgary)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)
- Europe (Milan)
- Europe (Paris)
- Europe (Spain)

- Europe (Stockholm)
- Europe (Zurich)
- Israel (Tel Aviv)
- Middle East (Bahrain)
- Middle East (UAE)
- South America (São Paulo)

Creating the VPC endpoints for AWS Private CA

You can create a VPC endpoint for the AWS Private CA service using either the VPC console at https://console.aws.amazon.com/vpc/ or the AWS Command Line Interface. For more information, see the Creating an Interface Endpoint procedure in the *Amazon VPC User Guide*. AWS Private CA supports making calls to all of its API operations inside your VPC.

If you have enabled private DNS host names for the endpoint, then the default AWS Private CA endpoint now resolves to your VPC endpoint. For a comprehensive list of default service endpoints, see Service Endpoints and Quotas.

If you have not enabled private DNS host names, Amazon VPC provides a DNS endpoint name that you can use in the following format:

```
vpc-endpoint-id.acm-pca.region.vpce.amazonaws.com
```

> **Note**
>
> The value *region* represents the Region identifier for an AWS Region supported by AWS Private CA, such as `us-east-2` for the US East (Ohio) Region. For a list of AWS Private CA, see AWS Certificate Manager Private Certificate Authority Endpoints and Quotas.

For more information, see AWS Private CA VPC endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Create a VPC endpoint policy for AWS Private CA

You can create a policy for Amazon VPC endpoints for AWS Private CA to specify the following:

- 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 Guide*.

Example – VPC endpoint policy for AWS Private CA actions

When attached to an endpoint, the following policy grants access for all principals to the AWS Private CA actions `IssueCertificate`, `DescribeCertificateAuthority`, `GetCertificate`, `GetCertificateAuthorityCertificate`, `ListPermissions`, and `ListTags`. The resource in each stanza is a private CA. The first stanza authorizes the creation of end-entity certificates using the specified private CA and certificate template. If you don't want to control the template being used, the `Condition` section is not needed. However, removing this allows all principals to create CA certificates as well as end-entity certificates.

```
{
      "Statement":[
         {
            "Principal":"*",
            "Effect":"Allow",
            "Action":[
               "acm-pca:IssueCertificate"
            ],
            "Resource":[
               "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
            ],
            "Condition":{
               "StringEquals":{
                  "acm-pca:TemplateArn":"arn:aws:acm-pca:::template/
EndEntityCertificate/V1"
               }
            }
         },
         {
            "Principal":"*",
            "Effect":"Allow",
            "Action":[
               "acm-pca:DescribeCertificateAuthority",
```

```
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListPermissions",
                "acm-pca:ListTags"
            ],
            "Resource":[
                "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
            ]
         }
      ]
   }
```

Dual-stack endpoint support

AWS Private Certificate Authority provides a dual-stack public endpoint that supports both IPv4 and IPv6 clients. A dual-stack endpoint enables clients to communicate with AWS Private CA using either IPv4 or IPv6 addresses. AWS Private CA for Active Directory and AWS Private CA Connector for SCEP also support dual-stack endpoints.

The AWS Private CA dual-stack public endpoint at `https://acm-pca.your-region.api.aws` supports both IPv4 and IPv6 clients. AWS Private CA is also privately accessible over IPv4 and IPv6 from your virtual private cloud (VPC) using AWS PrivateLink. For more information about creating private interface VPC endpoints for AWS Private CA, see AWS Private CA VPC endpoints (AWS PrivateLink).

For more information, see the following resources:

- IP addressing for your VPCs and subnets
- IPv6 support for your VPC

Using IPv6 addresses in IAM and AWS Private CA

Before trying to access AWS Private Certificate Authority over IPv6, ensure any IAM policies containing IP address restrictions are updated to include IPv6 address ranges. IP based policies that are not updated to handle IPv6 addresses may result in clients incorrectly losing or gaining access when they start using IPv6. To learn more about AWS Private CA and dual-stack support, see Dual-stack endpoint support.

Important

These statements do not allow any actions. Use these statements in combination with other statements that allow specific actions.

The following statement explicitly denies access to all AWS Private CA permissions for requests originating from the `192.0.2.*` range of IPv4 addresses. Any IP addresses outside of this range are not explicitly denied AWS Private CA permissions. Since all IPv6 addresses are outside of the denied range, this statement does not explicitly deny AWS Private CA permissions for any IPv6 addresses.

```
{
    "Sid": "DenyPrivateCAPermissions",
    "Effect": "Deny",
    "Action": [
        "acm-pca:*"
    ],
    "Resource": "*",
    "Condition": {
        "NotIpAddress": {
            "aws:SourceIp": [
                "192.0.2.0/24"
            ]
        }
    }
}
```

You can modify the `Condition` element to deny both IPv4 (`192.0.2.0/24`) and IPv6 (`2001:db8::/32`) address ranges as shown in the following example:

```
{
    "Sid": "DenyPrivateCAPermissions",
    "Effect": "Deny",
    "Action": [
        "acm-pca:*"
    ],
    "Resource": "*",
    "Condition": {
```

```
        "NotIpAddress": {
            "aws:SourceIp": [
                "192.0.2.0/24",
                "2001:db8::/32"
            ]
        }
    }
}
```

AWS Private Certificate Authority Customer CP/CPS Framework

AWS Private Certificate Authority provides infrastructure services that enable you to create certificate authority (CA) hierarchies, including root and subordinate CAs, without the investment and maintenance costs of operating an on-premise CA. When you use AWS Private CA to create your CA hierarchies, there is a shared responsibility between you and AWS Private CA. The shared responsibility model can help relieve your operational burden as AWS operates, manages and controls the physical security of the facilities in which the service operates. You assume responsibility and management of the certificate authority (including creation and deletion of CA resources; distributing trust anchors; PKI hierarchy creation; certification policies and practices; configuration for allowing or denying CA sharing across AWS accounts; policies for template usage; auditing; access controls, including separation of duties; and other CA configuration and policies). You should carefully consider the services you choose as your responsibilities vary depending on the services used, the integration of those services into your IT environment, and applicable laws and regulations. For more information, see the AWS Cloud Security Shared Responsibility Model.

Creating a certificate policy (CP) or certification practice statement (CPS) for your private certificate authority is a critical part of managing your public key infrastructure (PKI). A CP defines all the requirements/rules for your PKI and the CPS explains how you meet the CP requirements. You are responsible for creating a CP and CPS as the certificate authority of your PKI. AWS Private CA provides you with AWS control and compliance documentation, such as the AWS System and Organization Controls (SOC) 2 Report, you can use to help create your CP and CPS and to perform your control evaluation and verification procedures as required. AWS SOC Reports are independent third-party examination reports that demonstrate how AWS achieves key compliance controls and objectives. The purpose of the reports is to help you and your auditors understand the AWS controls established to support operations and compliance.

This document presents a framework that aligns to RFC 3647 to assist you with writing your CP and CPS and identifies the shared responsibility between you and AWS Private CA. Sections of

the CP/CPS requirements where AWS Private CA has a compliance responsibility is identified with "Shared" or "AWS Private Certificate Authority" and corresponding "Supplemental Information" is provided to help you understand how AWS Private CA meets the associated CP/CPS requirement. For example, Requirement 5 (4.5.1) is an AWS Private CA responsibility and you can find the corresponding control language in Section D.6 of the AWS SOC 2 Report to help complete your CP/CPS. For more information on AWS SOC Reports and how you can request access to SOC Reports, please visit our SOC FAQs page.

CP/CPS Requirements and Responsibilities

CP/CPS Requirement	Responsibility	Supplemental Information
1. Introduction (All)	You	You are responsible for documenting the overview, document name and identification, PKI participants, certificate usage, policy administration, and definitions and acronyms related to your PKI.
2. Publication and Repository Responsibilities (All)	You	You are responsible for documenting the definitions related to your PKI.
3. Identification and Authentication (All)	You	You are responsible for documenting the procedures used to authenticate the identity and/or other attributes of an end-user certificate applicant to a CA or Registration Authority (RA) prior to certificate issuance.
4. Certificate Life-Cycle Operational Requirements (4.4.1 — 4.4.6, 4.4.9 — 4.4.11)	Shared	You are responsible for specifying requirements imposed upon issuing CA, subject CAs, RAs, subscribe

CP/CPS Requirement	Responsibility	Supplemental Information
		rs, or other participants with respect to the life-cycle of a certificate. AWS Private CA provides you with two fully managed mechanisms to help support revocation status checking: Online Certificate Status Protocol (OCSP) and certifica te revocation lists (CRLs) to help you meet 4.4.9 and 4.4.10.
4. Certificate Life-Cycle Operational Requirements (4.4.7, 4.4.8, 4.4.12)	N/A	AWS Private CA does not support Certificate Re-key, Certificate Modification, or Key Escrow and Recovery.

CP/CPS Requirement	Responsibility	Supplemental Information
5. Facility, Management, and Operational Controls (4.5.1)	AWS Private CA	You inherit access controls that help you meet the requirements in this section that are within the scope of the AWS Private CA SOC 2 Type 2 Report (see Section D.6 Physical Security and Environmental Protection). **Note** You are responsible for the physical security and data classification of CA data exported or transferred out of the AWS environment but not for the physical security of CA data stored on AWS.
5. Facility, Management, and Operational Controls (4.5.2)	Shared	You are responsible for satisfying the requirements in this section specific to defining trusted roles for the operations of your PKI environment. AWS Private CA maintains trusted roles specific to physical access of cryptographic modules.

CP/CPS Requirement	Responsibility	Supplemental Information
5. Facility, Management, and Operational Controls (4.5.3)	Shared	You are responsible for satisfying the requirements in this section specific to background check, training, and disciplinary actions procedures for your trusted persons. You inherit controls related to background checks, training, and disciplinary action procedures for AWS employees that are within the scope of the AWS Private CA SOC 2 Type 2 Report (see Section A. Policies, A.1 Control Environment, B. Communications, and D.1 Security Organization and D.2 Employee User Access).

CP/CPS Requirement	Responsibility	Supplemental Information
5. Facility, Management, and Operational Controls (4.5.4)	Shared	You are responsible for enabling, configuring retention, and protectin g CloudTrail and audit reporting logs and CloudWatc h alerts. Additionally, you are responsible for creating log processing procedures and performing vulnerability assessments of your use of the AWS Private CA service that satisfy the requirements in this section. You inherit controls related to availability of your logs, physical access/site security, CA/RA configura tion management, security of AWS infrastructure logs, and vulnerability assessmen ts of AWS infrastructure that are within the scope of the AWS Private CA SOC 2 Type 2 Report (see Section A.1 Control Environment, Section C.1 Service Commitmen ts, D.2 Employee User Access, D.3 Logical Security, D.6 Physical Security and Environmental Protection, D.7 Change Management, D.8 Data Integrity, Availabil

CP/CPS Requirement	Responsibility	Supplemental Information
		ity, and Redundancy, and E.1 Monitoring Activities).
5. Facility, Management, and Operational Controls (4.5.5)	Shared	You are responsible for configuring backup and retention periods that satisfy the requirements in this section. You inherit controls related to availability of your logs (when you configure) that are within the scope of the AWS Private CA SOC 2 Type 2 Report (see D.8 Data Integrity, Availabil ity, and Redundancy).
5. Facility, Management, and Operational Controls (4.5.6)	N/A	AWS Private CA does not support Key Changeover.

CP/CPS Requirement	Responsibility	Supplemental Information
5. Facility, Management, and Operational Controls (4.5.7)	Shared	You are responsible for implementing incident and compromise handling procedures specific to your use of AWS Private CA that satisfy the requirements in this section. You inherit incident, compromise handling procedures, business continuit y, and disaster recovery procedures specific to physical site housing and infrastructure operations that help you meet the requireme nts in this section that are within the scope of the AWS Private CA SOC 2 Type 2 Privacy Report (see D.8 Data Integrity, Availability, and Redundancy and Section D.10 Privacy).
5. Facility, Management, and Operational Controls (4.5.8)	You	You are required to document requirements relating to procedures for termination and termination of a CA or RA, including the identity of the custodian of CA and RA archival records.

CP/CPS Requirement	Responsibility	Supplemental Information
6. Technical Controls (4.6.1)	Shared	You are responsible for documenting the key generation and installation needs for your PKI. AWS Private CA provides you with cryptographic modules that are FIPS 140-3 level 3 certified for CA key generatio n.
6. Technical Controls (4.6.2)	Shared	You are responsible for documenting private key protection and cryptogra phic module engineering controls such as cryptogra phic standard requirements and multi-person controls. AWS Private CA provides you with cryptographic modules that are FIPS 140-3 level 3 certified for CA key generatio n and two-party physical access controls to HSMs.
6. Technical Controls (4.6.3)	You	You are responsible for documenting other aspects of key pair management such as archival of your public key and operational period of certificates.

CP/CPS Requirement	Responsibility	Supplemental Information
6. Technical Controls (4.6.4)	N/A	AWS AWS Private CA HSMs are always online and have no notion of "activation data". **Note** You are responsible for implementing user access controls to your Private CA to appropriately restrict the ability create CA's and issue certificates.
6. Technical Controls (4.6.5)	Shared	You are responsible for documenting computer security controls for your use of your Private CA. You inherit controls related to logical access of AWS employees, network and computer security controls of the AWS infrastructure, and password parameter controls of AWS employee accounts that are within the scope of the AWS Private CA SOC 2 Type 2 Report (see Section D.2 Employee User Access, D.3 Logical Security, and D.6 Physical Security and Environmental Protection).

CP/CPS Requirement	Responsibility	Supplemental Information
6. Technical Controls (4.6.6)	Shared	You are responsible for documenting security management controls related to your use of your Private CA. You inherit controls related to system development controls of the AWS Private CA service that are within the scope of the AWS Private CA SOC 2 Type 2 Report (see Section D.7 Change Management).
6. Technical Controls (4.6.7)	Shared	You are responsible for documenting network security controls for your use of Private CA if applicable to your PKI environment. You inherit controls related to network security controls of the AWS infrastructure that are within the scope of the AWS Private CA SOC 2 Type 2 Report (see Section C.1 Service Commitments, D.3 Logical Security, and E.1 Monitoring Activities).
6. Technical Controls (4.6.8)	AWS Private CA	AWS Private CA uses trusted time sources to timestamp CA data.

CP/CPS Requirement	Responsibility	Supplemental Information
7. Certificate, CRL, and OCSP Profiles (All)	Shared	You are responsible for documenting profile requirements and certificate input that meet the needs of your PKI environment. AWS Private CA provides you with profile templates to help meet your profile requireme nts.
8. Compliance Audit and Other Assessment (All)	Shared	You are responsible for documenting compliance audit and other assessments. AWS Private CA provides you with a SOC 2 Report to help you and your auditors understand the AWS controls established to support operations and compliance.
9. Other Business and Legal Matters	You	You are responsible for documenting general business and legal matters that cover your Private CA.

Monitor AWS Private CA resources

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Private CA and your other AWS solutions. AWS provides the following monitoring tools to watch AWS Private CA, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and the applications you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the Amazon CloudWatch User Guide.
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from Amazon EC2 instances, CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the Amazon CloudWatch Logs User Guide.
- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.
- *Amazon EventBridge* is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services and routes that data to targets such as Lambda. This enables you to monitor events that happen in services, and build event-driven architectures. For more information, see the Amazon EventBridge User Guide.

The following topics describe AWS cloud-monitoring tools available for use with AWS Private CA.

AWS Private CA CloudWatch metrics

Amazon CloudWatch is a monitoring service for AWS resources. You can use CloudWatch to collect and track metrics, set alarms, and automatically react to changes in your AWS resources. CloudWatch metrics are published at least once.

AWS Private CA supports the following CloudWatch metrics.

Metric	Description
`CRLGenerated`	A certificate revocation list (CRL) was generated. This metric applies only to a private CA.
`MisconfiguredCRLBucket`	The S3 bucket specified for the CRL is not correctly configured. Check the bucket policy. This metric applies only to a private CA.
`Time`	The time in milliseconds between an issuance request and the completion (or failure) of issuance. This metric applies only to the **IssueCertificate** operation.
`Success`	A certificate was successfully issued. This metric applies only to the **IssueCertificate** operation.
`Failure`	An operation failed. This metric applies only to the **IssueCertificate** operation.

For more information about CloudWatch metrics, see the following topics:

- Using Amazon CloudWatch Metrics
- Creating Amazon CloudWatch Alarms

Monitor AWS Private CA with CloudWatch Events

You can use Amazon CloudWatch Events to automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to CloudWatch Events in near-real time. You can write simple rules to indicate which events are of interest to you and the automated actions to take when an event matches a rule. CloudWatch Events are published at least once. For more information, see Creating a CloudWatch Events Rule That Triggers on an Event.

CloudWatch Events are turned into actions using Amazon EventBridge. With EventBridge, you can use events to trigger targets including AWS Lambda functions, AWS Batch jobs, Amazon SNS topics, and many others. For more information, see What Is Amazon EventBridge?

Success or failure when creating a private CA

These events are triggered by the CreateCertificateAuthority operation.

Success

On success, the operation returns the ARN of the new CA.

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Creation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:14:56Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"success"
   }
}
```

Failure

On failure, the operation returns an ARN for the CA. Using the ARN, you can call DescribeCertificateAuthority to determine the status of the CA.

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Creation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:14:56Z",
   "region":"region",
   "resources":[
```

```
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"failure"
   }
}
```

Success or failure when issuing a certificate

These events are triggered by the IssueCertificate operation.

Success

On success, the operation returns the ARNs of the CA and of the new certificate.

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Certificate Issuance",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:57:46Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   ],
   "detail":{
      "result":"success"
   }
}
```

Failure

On failure, the operation returns a certificate ARN and the ARN of the CA. With the certificate ARN, you can call GetCertificate to view the reason for the failure.

```
{
   "version":"0",
   "id":"event_ID",
```

```
   "detail-type":"ACM Private CA Certificate Issuance",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:57:46Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   ],
   "detail":{
      "result":"failure"
   }
}
```

Success when revoking a certificate

This event is triggered by the RevokeCertificate operation.

No event is sent if the revocation fails or if the certificate has already been revoked.

Success

On success, the operation returns the ARNs of the CA and of the revoked certificate.

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Certificate Revocation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-05T20:25:19Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   ],
   "detail":{
      "result":"success"
   }
```

```
}
```

Success or failure when generating a CRL

These events are triggered by the RevokeCertificate operation, which should result in the creation of a certificate revocation list (CRL).

Success

On success, the operation returns the ARN of the CA associated with the CRL.

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA CRL Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T21:07:08Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"success"
   }
}
```

Failure 1 – CRL could not be saved to Amazon S3 because of a permission error

Check your Amazon S3 bucket permissions if this error occurs.

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA CRL Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-07T23:01:25Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
```

```
   ],
   "detail":{
      "result":"failure",
      "reason":"Failed to write CRL to S3. Check your S3 bucket permissions."
   }
}
```

Failure 2 – CRL could not be saved to Amazon S3 because of an internal error

Retry the operation if this error occurs.

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA CRL Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-07T23:01:25Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"failure",
      "reason":"Failed to write CRL to S3. Internal failure."
   }
}
```

Failure 3 – AWS Private CA failed to create a CRL

To troubleshoot this error, check your CloudWatch metrics.

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA CRL Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-07T23:01:25Z",
   "region":"region",
   "resources":[
```

```
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566"
   ],
   "detail":{
      "result":"failure",
      "reason":"Failed to generate CRL. Internal failure."
   }
}
```

Success or failure when creating a CA audit report

These events are triggered by the CreateCertificateAuthorityAuditReport operation.

Success

On success, the operation returns the ARN of the CA and the ID of the audit report.

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Audit Report Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T21:54:20Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "audit_report_ID"
   ],
   "detail":{
      "result":"success"
   }
}
```

Failure

An audit report can fail when AWS Private CA lacks PUT permissions on your Amazon S3 bucket, when encryption is enabled on the bucket, or for other reasons.

```
{
   "version":"0",
   "id":"event_ID",
```

```
   "detail-type":"ACM Private CA Audit Report Generation",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T21:54:20Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "audit_report_ID"
   ],
   "detail":{
      "result":"failure"
   }
}
```

Logging AWS Private Certificate Authority API calls using AWS CloudTrail

AWS Private Certificate Authority is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in AWS Private CA. CloudTrail captures API calls and signing operations for AWS Private CA as events. The calls captured include calls from the AWS Private CA console and code calls to the AWS Private CA API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS Private CA. 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 Private CA, 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.

AWS Private CA information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS Private CA, 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 Private CA, 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

All AWS Private CA actions are logged by CloudTrail and are documented in the AWS Private CA API reference. For example, calls to the `ImportCACertificate`, `IssueCertificate` and `CreateAuditReport` actions generate entries in the CloudTrail log files.

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.

AWS Private CA management events

AWS Private CA integrates with CloudTrail to record API actions made by a user, a role, or an AWS service in AWS Private CA. You can use CloudTrail to monitor AWS Private CA API requests in real time and store logs in Amazon Simple Storage Service, Amazon CloudWatch Logs, and Amazon CloudWatch Events. AWS Private CA supports logging the following actions and operations as events in CloudTrail log files:

- CreateCertificateAuthority
- CreateCertificateAuthorityAuditReport
- CreatePermission
- DeleteCertificateAuthority

- DeletePermission
- DeletePolicy
- DescribeCertificateAuthority
- DescribeCertificateAuthorityReport
- GetCertificate
- GetCertificateAuthorityCertificate
- GetCertificateAuthorityCsr
- GetPolicy
- ImportCertificateAuthorityCertificate
- IssueCertificate
- ListCertificateAuthorities
- ListPermissions
- ListTags
- PutPolicy
- RestoreCertificateAuthority
- RevokeCertificate
- TagCertificateAuthority
- UntagCertificateAuthority
- UpdateCertificateAuthority
- `GenerateOCSPResponse` - Triggered when AWS Private CA generates a OCSP response.
- `SignCertificate` - Generated when your client calls IssueCertificate.
- `SignOCSPResponse` - Generated when AWS Private CA signs an OCSP response.
- `GenerateCRL` - Generated when AWS Private CA generates a certificate revocation list (CRL).
- `SignCACSR` - Generated when AWS Private CA signs a certificate authority (CA) certificate signing request (CSR).
- `SignCRL` - Generated when AWS Private CA signs a CRL.

Example AWS Private CA events

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 are examples of AWS Private CA CloudTrail events.

Example 1: Management event, `IssueCertificate`

The following example shows a CloudTrail log entry that demonstrates the `IssueCertificate` action.

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"ACM Private CA Certificate Issuance",
   "source":"aws.acm-pca",
   "account":"account",
   "time":"2019-11-04T19:57:46Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   ],
   "detail":{
      "result":"success"
   }
}
```

Example 2: Management event, `ImportCertificateAuthorityCertificate`

The following example shows a CloudTrail log entry that demonstrates the `ImportCertificateAuthorityCertificate` action.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
```

```
   },
   "eventTime":"2018-01-26T21:53:28Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"ImportCertificateAuthorityCertificate",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
      "certificate":{
         "hb":[
            45,
            45,
            ...10
         ],
         "offset":0,
         "isReadOnly":false,
         "bigEndian":true,
         "nativeByteOrder":false,
         "mark":-1,
         "position":1257,
         "limit":1257,
         "capacity":1257,
         "address":0
      },
      "certificateChain":{
         "hb":[
            45,
            45,
            ...10
         ],
         "offset":0,
         "isReadOnly":false,
         "bigEndian":true,
         "nativeByteOrder":false,
         "mark":-1,
         "position":1139,
         "limit":1139,
         "capacity":1139,
         "address":0
      }
   },
   "responseElements":null,
```

```
    "requestID":"request_ID",
    "eventID":"event_ID",
    "eventType":"AwsApiCall",
    "recipientAccountId":"account"
}
```

Troubleshoot issues with AWS Private Certificate Authority

Consult the following topics if you have problems using AWS Private Certificate Authority.

Topics

- Troubleshoot AWS Private CA certificate revocation issues
- Troubleshoot AWS Private Certificate Authority exception messages
- Troubleshoot AWS Private CA Matter-compliant certificate errors

Troubleshoot AWS Private CA certificate revocation issues

OCSP response latency

OCSP responsiveness may be slower if the caller is geographically distant from a regional edge cache or from the Region of the issuing CA. For more information about regional edge cache availability, see Global Edge Network. We recommend issuing certificates in a Region near where they will be used.

Revocation of self-signed certificates

You can't revoke a self-signed CA certificate. To functionally revoke the certificate, delete the CA.

Troubleshoot AWS Private Certificate Authority exception messages

An AWS Private CA command might fail for several reasons. For information on each exception and recommendations for resolving them, see the table below.

AWS Private CA Exceptions

Exception Returned by AWS Private CA	Description	Remediation
`AccessDeniedException`	The permissions required to use the given command	For information on delegating permissions in AWS Private

Exception Returned by AWS Private CA	Description	Remediation
	have not been delegated by a private CA to the calling account.	CA, see Assign certificate renewal permissions to ACM.
`InvalidArgsException`	A certificate creation or renewal request was made with invalid parameters.	Check the command's individual documentation to make sure that your input parameters are valid. If you are creating a new certificate, make sure that the requested signing algorithm can be used with the CA's key type.
`InvalidStateException`	The associated private CA cannot renew the certificate because it is not in the `ACTIVE` state.	Attempt to restore the private CA. If the private CA is outside of its restoration period, the CA cannot be restored and the certificate cannot be renewed.
`LimitExceededException`	Each certificate authority (CA) has a quota of certificates that it can issue. The private CA that is associated with the designated certificate has reached its quota. For more information, see Service Quotas in the AWS General Reference Guide.	Contact the AWS Support Center to request a quota increase.
`MalformedCSRException`	The certificate signing request (CSR) that was submitted to AWS Private CA cannot be verified or validated.	Confirm that your CSR was properly generated and configured.

Exception Returned by AWS Private CA	Description	Remediation
`OtherException`	An internal error has caused the request to fail.	Attempt to run the command again. If the problem persists, contact the AWS Support Center.
`RequestFailedException`	A networking problem in your AWS environment caused the request to fail.	Retry the request. If the failure persists, check your Amazon VPC (VPC) configuration.
`ResourceNotFoundException`	The private CA that issued the certificate was deleted and no longer exists.	Request a new certificate from another active CA.
`ThrottlingException`	A requested API action failed because it exceeded a quota.	Confirm that you are not issuing more calls than allowed by AWS Private CA. A `ThrottlingException` error may also occur because you have encountered a transient condition rather than from an exceeded quota. If you encounter the error and you have not been making calls in excess of the quota, try your request again. If you are running up against a quota, you may be able to request an increase. For more information, see Service Quotas in the AWS General Reference Guide.

Exception Returned by AWS Private CA	Description	Remediation
`ValidationException`	The request's input parameters were incorrectly formatted, or the validity period of the root certificate ends before the validity period of the requested certificate.	Check the syntax requirements of the command's input parameters as well as the validity period of your CA's root certificate. For information about changing the validity period, see Update a private CA in AWS Private Certificate Authority.

Troubleshoot AWS Private CA Matter-compliant certificate errors

The Matter connectivity standard specifies certificate configurations that improve the security and consistency of internet of things (IoT) devices. Java samples for creating Matter-compliant root CA, intermediate CA, and end-entity certificates can be found at Use AWS Private CA to implement Matter certificates.

To assist with troubleshooting, the Matter developers provide a certificate verification tool called chip-cert. Errors that the tool reports are listed in the following table with remediations.

Error code	Meaning	Remediation
0x000003 5	`BasicConstraints`, `KeyUsage`, and `ExtensionKeyUsage` extensions must be marked critical.	Ensure that you have selected the correct template for your us
0x000000 0	The authority key identifier extension must be present.	AWS Private CA does not set the authority key identifier exten on root certificates. You must generate a Base64-encoded Au rityKeyIdentifier value using the CSR and then pass it through CustomExtension. For more information, see Activate a Root

Error code	Meaning	Remediation
		Node Operational Certificates (NOC). and Activate a Product A on Authority (PAA).
0x00000C E	Certificate is expired.	Ensure that the certificate you use is unexpired.

Error code	Meaning	Remediation

0x00000C 4	Certificate chain validation failure.	This error may be encountered if you attempt to create a Matt compliant end-entity certificate without using the provided Ja examples, which use the AWS Private CA API to pass a properl configured KeyUsage.
		
		By default, AWS Private CA generates nine-bit KeyUsage exten values, with the ninth bit resulting in an extra byte. Matter ig ores the extra byte during format conversions, causing chain-v ation failures. However, a CustomExtension in the APIPassth rough template can be used to set the exact number of byte the KeyUsage value. For an example, see Create a Node Oper Certificate (NOC).
		
		If you modify the sample code or use an alternative X.509 utili as OpenSSL, you need to perform manual verification in order chain validation errors.
		
		To verify that conversions are lossless
		
		1. Use openssl to verify that a certificate a node (end-entity te contains a valid chain. In this exanple, rcac.pem is the CA certificate, icac.pem is the intermediate CA certifica noc.pem is the node certificate.
		
		`openssl verify -verbose -CAfile <(cat rcac.pem icac noc.pem`
		
		2. Use chip-cert to convert the PEM-formatted node certific TLV (tag, length, value) format and back again.
		
		`./chip-cert convert-cert noc.pem noc.chip -c`
		`./chip-cert convert-cert noc.chip noc_converted.pem`

Error code	Meaning	Remediation
		The files `noc.pem` and `noc_converted.pem` should be the same as confirmed by a string comparison tool.

Secure Kubernetes with AWS Private Certificate Authority

You can use AWS Private Certificate Authority to provide certificates for secure authentication and encryption over TLS and mTLS. AWS Private CA provides an open source plugin, AWS Private CA Connector for Kubernetes, (`aws-privateca-issuer`) for the widely adopted cert-manager add-on to Kubernetes that requests certificates, distributes them to Kubernetes secrets, and automates certificate renewal.

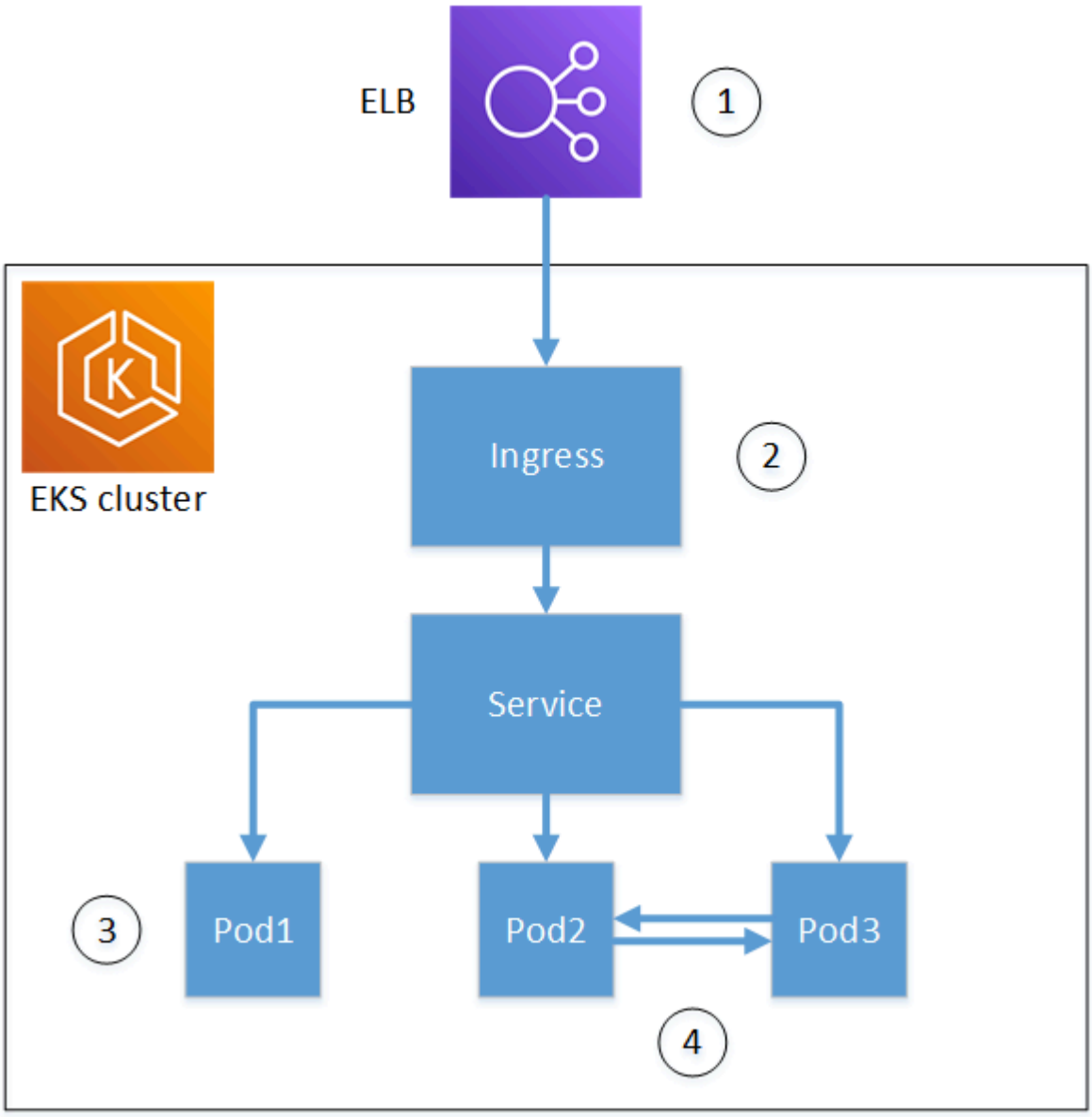
The `aws-privateca-issuer` plugin allows you to issue AWS Private CA certificates through `cert-manager`. You can use the plugin with Amazon Elastic Kubernetes Service (Amazon EKS), a self-managed Kubernetes cluster on AWS, or in an on-premise Kubernetes cluster. The plugin works on both x86 and ARM architectures.

AWS Private CA has HSM backed keys that can't be exported. If you have regulatory requirements for controlling access and auditing your CA operations, you can use AWS Private CA to improve auditability and to support compliance.

> **Note**
>
> If you are running on Amazon EKS, we recommend that you use the `cert-manager` and `aws-privateca-connector-for-kubernetes` add-ons for a managed installation experience. For more information, refer to AWS add-ons.

Concepts

The following diagram shows some of the options available for using TLS in an Amazon EKS cluster. The example cluster sits behind a load balancer. The numbers identify possible endpoints for TLS-secured communications.



1. **Termination at the load balancer**

   Elastic Load Balancing (Elastic Load Balancing) is integrated with the AWS Certificate Manager service. You don't need to install `cert-manager` on the load balancer. You can provision ACM with a private CA, sign a certificate with the private CA, and install the certificate using the Elastic Load Balancing console. AWS Private CA certificates are automatically renewed.

   As an alternative, you can provide a private certificate to a non-AWS load balancer to terminate TLS.

This provides encrypted communication between a remote client and the load balancer. Data after the load balancer is passed unencrypted to the Amazon EKS cluster.

2. **Termination at the Kubernetes ingress controller**

   The ingress controller is inside the Amazon EKS cluster and acts as a load balancer and router. To use the ingress controller as the cluster's endpoint for external communication, you must:

   - Install both `cert-manager` and `aws-privateca-issuer`
   - Provision the controller with a TLS private certificate from the AWS Private CA.

   Communications between the load balancer and the ingress controller are encrypted, data passes unencrypted to the cluster's resources.

3. **Termination at a pod**

   Each pod is a group of one or more containers that share storage and network resources. If you install both `cert-manager` and `aws-privateca-issuer` and provision the cluster with a private CA, Kubernetes can install a signed TLS private certificate on pods as needed. A TLS connection terminating at a pod is unavailable to other pods in the cluster by default.

4. **Secure communications between pods.**

   You can provision multiple pods with certificates to allow them to communicate with one another. The following scenarios are possible:

   - Provisioning with Kubernetes generated self-signed certificates. This secures communications between pods, but self-signed certificates don't satisfy HIPAA or FIPS requirements.
   - Provisioning with certificates signed by AWS Private CA. This requires installing both `cert-manager` and `aws-privateca-issuer`. Kubernetes can then install signed mTLS certificates on the pods as needed.

Considerations

When using AWS Private Certificate Authority with Kubernetes, keep the following considerations in mind.

Cross-account use of cert-manager

Administrators with cross-account access to a CA can use the `cert-manager` add on for Kubernetes to provision certificates for a cluster using the shared CA. For more information, refer to Security best practices for Cross-account access to private CAs.

You can use only certain AWS Private CA certificate templates in cross-account scenarios.

The following table lists AWS Private CA templates that you can use with cert-manager to provision a Kubernetes cluster.

Templates supported for Kubernetes	Support for cross-account use
BlankEndEntityCertificate_CSRPassthrough/V1 definition	No
CodeSigningCertificate/V1 definition	No
EndEntityCertificate/V1 definition	Yes
EndEntityClientAuthCertificate/V1 definition	Yes
EndEntityServerAuthCertificate/V1 definition	Yes
OCSPSigningCertificate/V1 definition	No

Get started with AWS Private CA Connector for Kubernetes.

The following topics show how to use AWS Private CA to secure communications in a Kubernetes cluster. For another example, refer to Encryption in transit for Kubernetes on GitHub.

You can use a private certificate authority to secure communications with your Amazon EKS clusters. Before you begin, ensure that you have the following:

- An AWS account with appropriate permissions scoped to your security policies.

  Amazon EKS clusters

  JSON

```
{
```

```
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "IAM",
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam:AttachRolePolicy",
        "iam:GetRole"
      ],
      "Resource": "*"
    },
    {
      "Sid": "EKS",
      "Effect": "Allow",
      "Action": [
        "eks:CreateAddon",
        "eks:DescribeAddon",
        "eks:CreatePodIdentityAssociation",
        "eks:DescribeCluster"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMPassRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::*:role/CertManagerPrivateCARole"
    }
  ]
}
```

Other clusters

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "GetAndIssuePCACertificates",
      "Effect": "Allow",
```

```
      "Action": [
        "acm-pca:GetCertificate",
        "acm-pca:IssueCertificate"
      ],
      "Resource": "*"
    },
    {
      "Sid": "RolesAnywhere",
      "Effect": "Allow",
      "Action": [
        "rolesanywhere:CreateProfile"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAM",
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam:AttachRolePolicy"
      ],
      "Resource": "*"
    },
    {
      "Sid": "IAMPassRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "arn:aws:iam::*:role/CertManagerPrivateCARole"
    }
  ]
}
```

- A Kubernetes cluster. To create a Amazon Elastic Kubernetes Service cluster, refer to the Amazon EKS quickstart guide. For simplicity, create an environment variable to hold the cluster name:

```
export CLUSTER=aws-privateca-demo
```

- The AWS Region where your CA and Amazon EKS cluster are located. For simplicity, create an environment variable to hold the Region:

```
export REGION=aws-region
```

- The Amazon Resource Name (ARN) of a AWS Private CA private certificate authority. For simplicity, create an environment variable to hold the private CA ARN:

  ```
  export CA_ARN="arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
  certificate/certificate_ID"
  ```

  To create a private CA, refer to https://docs.aws.amazon.com/privateca/latest/userguide/create-CA.htmlCreate a private CA in AWS Private CA
- A computer with the following software installed:
  - AWS CLI v2 configured
  - kubectl v1.13+
  - For non-Amazon EKS clusters, Helm v3

Install cert-manager

To use a private CA, you must install the `cert-manager>` add-on that requests certificates, distributes them, and automates certificate renewal. You must also install the `aws-private-ca-issuer` plugin that allows you to issue private certificates from AWS Private CA. Use the following steps to install the add-on and plugin.

Amazon EKS clusters

Install `cert-manager` as an Amazon EKS add-on:

```
aws eks create-addon \
  --cluster-name $CLUSTER \
  --addon-name cert-manager \
  --region $REGION
```

Other clusters

Install `cert-manager` using Helm:

```
helm repo add jetstack https://charts.jetstack.io
helm repo update
```

```
helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --create-namespace \
  --set crds.enabled=true
```

Configure IAM permissions

The `aws-privateca-issuer` plugin requires permission the interact with AWS Private CA. For Amazon EKS clusters you use the pod identity. For other clusters you use AWS Identity and Access Management Roles Anywhere.

Fist, create an IAM policy. The policy uses the `AWSPrivateCAConnectorForKubernetesPolicy` managed policy. For more information about the policy, refer to AWSPrivateCAConnectorForKubernetesPolicy in the *AWS Managed policy reference guide*.

Amazon EKS clusters

1. Create a file named `trust-policy.json` containing the following trust policy:

   JSON

   ```
   {
     "Version":"2012-10-17",
     "Statement": [
       {
         "Sid": "TrustPolicyForEKSClusters",
         "Effect": "Allow",
         "Principal": {
           "Service": "pods.eks.amazonaws.com"
         },
         "Action": [
           "sts:AssumeRole",
           "sts:TagSession"
         ]
       }
     ]
   }
   ```

2. Run the following commands to create an IAM role:

```
ROLE_ARN=$(aws iam create-role \
  --role-name CertManagerPrivateCARole \
  --assume-role-policy-document file://trust-policy.json \
  --region $REGION \
  --output text \
  --query "Role.Arn")

 aws iam attach-role-policy \
  --role-name CertManagerPrivateCARole \
  --policy-arn arn:aws:iam::aws:policy/AWSPrivateCAConnectorForKubernetesPolicy
```

Other clusters

1. Create a trust anchor that trusts the private CA stored in CA_ARN. For instructions, refer to Getting started with IAM Roles Anywhere. Create an environment variable to store the trust anchor ARN:

   ```
   export TRUST_ANCHOR_ARN=trustAnchorArn
   ```

2. Create a file called `trust-policy.json` containing the following trust policy:

   JSON

   ```
   {
       "Version":"2012-10-17",
       "Statement": [
           {
               "Sid": "TrustPolicyForSelfManagedOrOnPremiseClusters",
               "Effect": "Allow",
               "Principal": {
                   "Service": "rolesanywhere.amazonaws.com"
               },
               "Action": [
                   "sts:AssumeRole",
                   "sts:SetSourceIdentity",
                   "sts:TagSession"
               ],
               "Condition": {
                   "ArnEquals": {
                       "aws:SourceArn": [
   ```

```
                        "arn:aws:rolesanywhere:us-
east-1:123456789012:trust-anchor/TRUST_ANCHOR_ARN"
                    ]
                },
                "StringEquals": {
                    "aws:PrincipalTag/x509Subject/CN": "aws-privateca-
issuer"
                }
            }
        }
    ]
}
```

3. Run the following commands to create an IAM role:

```
ROLE_ARN=$(aws iam create-role \
  --role-name CertManagerPrivateCARole \
  --assume-role-policy-document file://trust-policy.json \
  --query "Role.Arn" \
  --region $REGION \
  --output text)

 aws iam attach-role-policy \
  --role-name CertManagerPrivateCARole \
  --region $REGION \
  --policy-arn arn:aws:iam::aws:policy/AWSPrivateCAConnectorForKubernetesPolicy
```

Install and configure the AWS Private CA cluster issuer

To install the `aws-privateca-connector-for-kubernetes` add-on, use the following commands:

Amazon EKS clusters

Create the add-on:

```
aws eks create-addon --region $REGION \
  --cluster-name $CLUSTER \
  --addon-name aws-privateca-connector-for-kubernetes \
  --pod-identity-associations "[{
    \"serviceAccount\": \"aws-privateca-issuer\",
    \"roleArn\": \"$ROLE_ARN\"
```

```
  }]"
```

Then wait for the add-on to be active:

```
aws eks describe-addon \
  --cluster-name $CLUSTER \
  --addon-name aws-privateca-connector-for-kubernetes \
  --region $REGION \
  --query 'addon.status'
```

Other clusters

1. Create a profile in IAM Roles Anywhere:

   ```
   PROFILE_ARN=$(aws rolesanywhere create-profile \
     --name "privateca-profile" \
     --role-arns "$ROLE_ARN" \
     --region "$REGION" \
     --query 'profile.profileArn' \
     --enabled \
     --output text)
   ```

2. Generate a client certificate for use with the Connector for Kubernetes and IAM Roles Anywhere to authenticate with AWS Private CA:

   a. Generate a private key for the client certificate:

   ```
   openssl genrsa -out client.key 2048
   ```

   b. Generate a certificate signing request (CSR) for the client certificate:

   ```
   openssl req -new \
     -key client.key \
     -out client.csr \
     -subj "/CN=aws-privateca-issuer"
   ```

   c. Issue the client certificate from AWS Private CA:

   ```
   CERT_ARN=$(aws acm-pca issue-certificate \
     --signing-algorithm SHA256WITHRSA \
     --csr fileb://client.csr \
     --validity Value=1,Type=DAYS \
   ```

```
  --certificate-authority-arn "$CA_ARN" \
  --region "$REGION" \
  --query 'CertificateArn' \
  --output text)
```

d. Store the client certificate locally:

```
aws acm-pca get-certificate \
  --certificate-authority-arn $CA_ARN \
  --certificate-arn $CERT_ARN \
  --region $REGION \
  --query 'Certificate'
  --output text > pca-issuer-client-cert.pem
```

3. Install the AWS Private CA issuer in the cluster with the client certificate:

a. Add the `awspca` Helm repository:

```
helm repo add awspca https://cert-manager.github.io/aws-privateca-issuer
helm repo update
```

b. Create a namespace:

```
kubectl create namespace aws-privateca-issuer
```

c. Put the certificate created earlier into a secret:

```
kubectl create secret tls aws-privateca-credentials \
  -n aws-privateca-issuer \
  --cert=pca-issuer-client-cert.pem \
  --key=client.key
```

4. Install the AWS Private CA issuer with IAM Roles Anywhere:

a. Create a file named `values.yaml` to configure the AWS Private CA issuer plugin to use with IAM Roles Anywhere:

```
cat > values.yaml <<EOF
env:
  AWS_EC2_METADATA_SERVICE_ENDPOINT: "http://127.0.0.1:9911"

extraContainers:
  - name: "rolesanywhere-credential-helper"
```

```
    image: "public.ecr.aws/rolesanywhere/credential-helper:latest"
    command: ["aws_signing_helper"]
    args:
      - "serve"
      - "--private-key"
      - "/etc/cert/tls.key"
      - "--certificate"
      - "/etc/cert/tls.crt"
      - "--role-arn"
      - "$ROLE_ARN"
      - "--profile-arn"
      - "$PROFILE_ARN"
      - "--trust-anchor-arn"
      - "$TRUST_ANCHOR_ARN"
    volumeMounts:
      - name: cert
        mountPath: /etc/cert/
        readOnly: true

volumes:
  - name: cert
    secret:
      secretName: aws-privateca-credentials
EOF
```

b. Install the AWS Private CA issuer with IAM Roles Anywhere:

```
helm install aws-privateca-issuer awspca/aws-privateca-issuer \
  -n aws-privateca-issuer \
  -f values.yaml
```

Wait for the issuer to be ready. Use the following command:

```
kubectl wait --for=condition=ready pods --all -n aws-privateca-issuer --timeout=120s
```

And then verify the installation to make sure that all pods have reached the READY state:

```
kubectl -n aws-privateca-issuer get all
```

To configure the `aws-private-ca-cluster-issuer`, create a YAML file named `cluster-issuer.yaml`containing the configuration of the issuer:

```
cat > cluster-issuer.yaml <<EOF
apiVersion: awspca.cert-manager.io/v1beta1
kind: AWSPCAClusterIssuer
metadata:
  name: aws-privateca-cluster-issuer
spec:
  arn: "$CA_ARN"
  region: "$REGION"
EOF
```

Next, apply the cluster configuration:

```
kubectl apply -f cluster-issuer.yaml
```

Check the status of the issuer:

```
kubectl describe awspcaclusterissuer aws-privateca-cluster-issuer
```

You should see a response similar to the following:

```
Status:
  Conditions:
    Last Transition Time:  2025-08-13T21:00:00Z
    Message:               AWS PCA Issuer is ready
    Reason:                Verified
    Status:                True
    Type:                  Ready
```

Manage the AWS Private CA client certificate with cert-manager

If you are not using an Amazon EKS cluster, after you manually bootstrap a trusted certificate in `aws-privateca-issuer` you can transition to a client authentication certificate managed by `cert-manager`. This allows `cert-manager` to automatically renew the client authentication certificate.

1. Create a file called `pca-auth-cert.yaml`:

   ```
   cat > pca-auth-cert.yaml <<EOF
   apiVersion: cert-manager.io/v1
   kind: Certificate
   ```

```
metadata:
  name: aws-privateca-client-cert
  namespace: aws-privateca-issuer
spec:
  secretName: aws-privateca-credentials
  duration: 168h
  renewBefore: 48h
  commonName: aws-privateca-issuer
  privateKey:
    algorithm: ECDSA
    size: 256
    rotationPolicy: Always
  usages:
    - client auth
  issuerRef:
    name: aws-privateca-cluster-issuer
    kind: AWSPCAClusterIssuer
    group: awspca.cert-manager.io
EOF
```

2. Create the new managed client authentication certificate:

```
kubectl apply -f pca-auth-cert.yaml
```

3. Validate that the certificate was created:

```
kubectl get certificate aws-privateca-client-cert -n aws-privateca-issuer
```

You should see a response similar to the following:

```
NAME                        READY   SECRET                       AGE
aws-privateca-client-cert   True    aws-privateca-credentials   19m
```

Issue your first TLS certificate

Now that the `cert-manager` and `aws-privateca-issuer` are installed, you can issue a certificate.

Create a YAML file named `certificate.yaml` containing the certificate resource:

```
cat > certificate.yaml <<EOF
```

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-certificate
  namespace: default
spec:
  secretName: example-certificate-tls
  issuerRef:
    name: aws-privateca-cluster-issuer
    kind: AWSPCAClusterIssuer
    group: awspca.cert-manager.io
  commonName: example.internal
  dnsNames:
    - example.internal
    - api.example.internal
  duration: 2160h # 90 days
  renewBefore: 360h # 15 days
  usages:
    - digital signature
    - key encipherment
    - server auth
EOF
```

Apply the certificate using the following command:

```
kubectl apply -f certificate.yaml
```

You can then check the status of the certificate with the following commands:

```
kubectl get certificate example-certificate
kubectl describe certificate example-certificate
```

You should see a response similar to this:

```
NAME                READY   SECRET                    AGE
example-certificate  True    example-certificate-tls   30s
```

You can inspect the issued certificate with the following command:

```
kubectl get secret example-certificate-tls -o yaml
```

You can also decode and examine the certificate with the following command:

```
kubectl get secret example-certificate-tls -o jsonpath='{.data.tls\.crt}' | base64 -d |
 openssl x509 -text -noout
```

Examples

The following examples show how AWS Private CA can be used with Kubernetes clusters.

- Sample: Encryption in transit for Kubernetes
- TLS-enabled Kubernetes clusters with AWS Private CA and Amazon EKS
- Setting up end-to-end TLS encryption on Amazon EKS with the new AWS Load Balancer Controller

Monitor Kubernetes with AWS Private CA

To monitor the Kubernetes cluster private CA, use the techniques outlined in Monitor AWS Private CA resources. You can use the following to monitor a private CA:

- AWS Private CA CloudWatch metrics
- Monitor AWS Private CA with CloudWatch Events
- Logging AWS Private Certificate Authority API calls using AWS CloudTrail

Troubleshoot Kubernetes with AWS Private CA

You can get the logs for `aws-private-ca-issuer` with the following procedure:

1. Get the name of the pod:

   ```
   kubectl get pods -A
   ```

2. To view the issuer logs, use the following command:

   ```
   kubectl logs -n aws-privateca-issuer <pod-name> aws-privateca-issuer
   ```

3. To view the IAM Roles Anywhere logs, use the following command:

```
kubectl logs -n aws-privateca-issuer <pod-name> rolesanywhere-credentials-helper
```

To check the status of your AWS Private CA issuer, use one of the following:

To check that your issuer is ready, use the following command:

```
kubectl get AWSPCAClusterIssuers -o json | jq '.items[].status
```

The response should be similar to the following:

```
{
  "conditions": [
    {
      "lastTransitionTime": "2024-07-03T13:56:37Z",
      "message": "Issuer verified",
      "reason": "Verified",
      "status": "True",
      "type": "Ready"
    }
  ]
}
```

If the issuer is not in the `Ready` state, the `message` field provides information on why the issuer was unable to reach the `Ready` state.

To check that your certificate is ready, use the following command:

```
kubectl get certificates -o json | jq '.items[].status'
```

The response should be similar to the following:

```
{
  "conditions": [
    {
      "lastTransitionTime": "2024-07-03T13:58:13Z",
      "message": "Certificate is up to date and has not expired",
      "observedGeneration": 1,
      "reason": "Ready",
      "status": "True",
```

```
      "type": "Ready"
    }
  ],
  "notAfter": "2024-10-01T13:58:12Z",
  "notBefore": "2024-07-03T12:58:12Z",
  "renewalTime": "2024-09-16T13:58:12Z",
  "revision": 1
}
```

If the certificate is not in the `Ready` state, the `message` field provides information on why the certificate was not able to reach the `Ready` state.

AWS Private CA Connector for Active Directory

AWS Private CA can issue and manage certificates required by AWS Managed Microsoft AD. Using the AWS Private CA Connector for Active Directory (Connector for AD), you can replace on-premises enterprise or other third-party CAs with a managed private CA that you own, providing certificate enrollment to users, groups, and machines that are managed by your AD.

You can use the Connector for AD with AWS Managed Microsoft AD to eliminate on-premises infrastructure by migrating your AD and public key infrastructure to the cloud. For customers looking to use AWS Private CA with their on-premises AD, this feature also integrates with AWS Managed Microsoft AD Connector.

Topics

- Are You a First-Time Connector for AD User?
- Set up Connector for AD
- Get started with AWS Private CA Connector for Active Directory
- AWS Private CA connectors for Active Directory
- Integrating Connector for AD into event-driven applications using Amazon EventBridge
- Troubleshoot issues with AWS Private CA Connector for Active Directory

Are You a First-Time Connector for AD User?

If you are a first-time user of Connector for AD, we recommend that you begin by reading the following sections:

- What is AWS Private CA?
- What is Directory Service?

Access Connector for AD

You can access Connector for AD through the console, AWS CLI, and APIs. You can get access to the connector in the console from the AWS Private CA console, from your Directory Service console, or by searching for Connector for AD in the AWS Management Console search bar.

Pricing

Connector for AD is offered as a feature of AWS Private CA at no additional cost. You only pay for the private certificate authorities and the certificates you issue through them.

For the latest AWS Private CA pricing information, see AWS Private Certificate Authority Pricing. You can also use the AWS pricing calculator to estimate costs.

Set up Connector for AD

The steps in this section are prerequisites to using Connector for AD. It assumes that you've already created an AWS account. After you complete the steps on this page, you can get started with creating a connector for AD.

Step 1: Create a private CA using AWS Private CA

Set up a private certificate authority (CA) for issuing certificates to your directory objects. For more information, see Certificate authorities in AWS Private CA.

The private CA must be in the `Active` state to create a Connector for AD. The private CA's subject name must include a common name. Connector creation will fail if you try to create a connector using a private CA without a common name.

Step 2: Set up an Active Directory

In addition to a private CA, you need an active directory in a virtual private cloud (VPC). Connector for AD supports the following directory types offered by Directory Service:

- AWS Managed Microsoft Active Directory: With Directory Service you can run Microsoft Active Directory (AD) as a managed service. AWS Directory Service for Microsoft Active Directory also referred to as AWS Managed Microsoft AD, is powered by Windows Server 2019. With AWS Managed Microsoft AD, you can run directory-aware workloads in the AWS Cloud, including Microsoft Sharepoint and custom .Net and SQL Server-based applications.
- Active Directory Connector: AD Connector is a directory gateway that can redirect directory requests to your on-premises Microsoft Active Directory, without caching any information in the cloud. AD Connector supports connecting to a domain hosted on Amazon EC2

(Active Directory Connector only) Step 3: Delegate permissions to service account

> **Note**
>
> If you are using AWS Managed Microsoft AD the additional permissions are delegated automatically when you authorize the Connector for AD service with your directory. You can skip this prerequisite step.

When using the Directory Service AD Connector, you need to delegate additional permissions to the service account. Set access-control list (ACL) on the service account to allow the ability:

- Add and remove a Service Principal Name (SPN) to itself
- Create and update certification authorities in the following containers:

```
#containers
CN=Public Key Services,CN=Services,CN=Configuration
CN=AIA,CN=Public Key Services,CN=Services,CN=Configuration
CN=Certification Authorities,CN=Public Key Services,CN=Services,CN=Configuration
```

- Create and update a NTAuthCertificates Certification Authority (CA) object. Note: if the NTAuthCertificates CA object exists then you must delegate permissions for it. If the object does not exist then you must delegate the ability to create child objects on the Public Key Services container.

```
#objects
CN=NTAuthCertificates,CN=Public Key Services,CN=Services,CN=Configuration
```

The PowerShell script available in the official Connector for Active Directory repository can be used to delegate the additional permissions required for the Directory Service AD Connector service account.

This script creates the NTAuthCertificates certification authority object.

For the latest version of the script and usage details, refer to the README in the GitHub repository.

Step 4: Create IAM Policy

To create a connector for AD, you need an IAM policy that allows you to create connector resources, share your private CA with the Connector for AD service, and authorize the Connector for AD service with your directory.

This is an example a user managed policy:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "pca-connector-ad:*",
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListCertificateAuthorities",
                "acm-pca:ListTags",
                "acm-pca:PutPolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "acm-pca:IssueCertificate",
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APIPassthrough/V*"
                },
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "pca-connector-ad.amazonaws.com"
                }
```

```
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ds:AuthorizeApplication",
                "ds:DescribeDirectories",
                "ds:ListTagsForResource",
                "ds:UnauthorizeApplication",
                "ds:UpdateAuthorizedApplication"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:CreateVpcEndpoint",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcEndpoints",
                "ec2:DescribeVpcs",
                "ec2:DeleteVpcEndpoints"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeTags",
                "ec2:DeleteTags",
                "ec2:CreateTags"
            ],
            "Resource": "arn:*:ec2:*:*:vpc-endpoint/*"
        }
    ]
}
```

Connector for AD requires additional AWS RAM permissions, for both console and command line use.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "ram:CreateResourceShare",
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "ram:Principal": "pca-connector-ad.amazonaws.com",
                    "ram:RequestedResourceType": "acm-pca:CertificateAuthority"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ram:GetResourcePolicies",
                "ram:GetResourceShareAssociations",
                "ram:GetResourceShares",
                "ram:ListPrincipals",
                "ram:ListResources",
                "ram:ListResourceSharePermissions",
                "ram:ListResourceTypes"
            ],
            "Resource": "*"
        }
    ]
}
```

Step 5: Share your private CA with Connector for AD

You will need to share your private CA with the connectors service by using AWS Resource Access Manager service principal sharing.

When you create a connector in the AWS console, the resource share is automatically created for you.

When you create a resource share using the AWS CLI, you will use the AWS RAM **create-resource-share** command.

The following command creates a resource share:

```
$  aws ram create-resource-share \
    --region us-east-1 \
    --name MyPcaConnectorAdResourceShare \
    --permission-arns arn:aws:ram::aws:permission/
AWSRAMBlankEndEntityCertificateAPIPassthroughIssuanceCertificateAuthority \
    --resource-arns arn:aws:acm-pca:region:account:certificate-authority/CA_ID \
    --principals pca-connector-ad.amazonaws.com \
    --sources account
```

The service principal that calls CreateConnector has certificate issuance permissions on the PCA. To prevent service principals that use Connector for AD from having general access to your AWS Private CA resources, restrict their permissions using `CalledVia`.

Step 6: Create directory registration

You authorize the Connector for AD service with your directory so the connector can communicate with your directory. To authorize the Connector for AD service, you create a directory registration. For more information on creating a directory registration, see Manage directory registrations

Step 7: Configure security groups

Communication between your VPC and the Connector for AD connector is through AWS PrivateLink, which requires a security group(s) with inbound rules that open port 443 TCP on your VPC. You will be asked for this security group when you create a connector. You can specify the source as custom and select your VPC's CIDR block. You can choose to restrict this further (i.e. IP, CIDR, and security group ID).

Step 8: Configure network access for directory objects

Directory objects require public internet access to validate Online Certificate Status Protocol (OCSP) and certificate revocation lists (CRLs) from the following domains:

```
*.windowsupdate.com
*.amazontrust.com
```

Minimum required access rules:

- Required for OCSP and CRL communication:

```
TCP 80: (HTTP) to 0.0.0.0/0
```

- Required for Connector for AD:

```
TCP 443: (HTTPS) to 0.0.0.0/0
```

- Required for Active Directory:

```
TCP 88: (Kerberos) to Domain Controller IP range
TCP/UDP 389/636: (LDAP/LDAPS) to Domain Controller IP range, depending on Domain
 Controller configuration
TCP/UDP 53: (DNS) to 0.0.0.0/0
```

If the devices do not have public internet access, certificate issuance will fail intermittently with the error code `WS_E_OPERATION_TIMED_OUT`.

> **Note**
>
> If you are configuring a security group for an Amazon EC2 instance, it does not have to be the same one in Step 7.

Get started with AWS Private CA Connector for Active Directory

With AWS Private CA Connector for Active Directory, you can issue certificates from your private CA to your Active Directory objects for authentication and encryption. When you create a connector, AWS Private Certificate Authority creates an endpoint for you in your VPC for your directory objects to request certificates.

To issue certificates, you create a connector and AD-compatible templates for the connector. When you create a template, you can set enrollment permissions for your AD groups.

Topics

- Before you begin
- Step 1: Create a connector

- Step 2: Configure Microsoft Active Directory policies
- Step 3: Create a template
- Step 4: Configure Microsoft group permissions

Before you begin

The following tutorial guides you through the process of creating a connector for AD and a connector template. To follow this tutorial, you must first fulfill the prerequisites listed in the section.

Step 1: Create a connector

To create a connector, see Creating a connector for Active Directory.

Step 2: Configure Microsoft Active Directory policies

Connector for AD is unable to view or manage the customer's group policy object (GPO) configuration. The GPO controls the routing of AD requests to the customer's AWS Private CA or to other authentication or certificate vending servers. An invalid GPO configuration may result in your requests being routed incorrectly. It is up to customers to configure and test the Connector for AD configuration.

Group Policies are associated with a Connector, and you may choose to create multiple Connectors for a single AD. It is up to you to manage the access control to each connector if its group policy configurations are different.

The security of the data plane calls depends on Kerberos and your VPC configuration. Anyone with access to the VPC can make data plane calls as long as they are authenticated to the corresponding AD. This exists outside of the boundary of AWSAuth and managing authorization and authentication is up to you, the customer.

When using AWS Managed Microsoft AD, use the Directory Service **enable-ca-enrollment-policy** command to configure GPOs on the domain controller of the AWS Managed Microsoft AD instance.

The following command enables enrolling domain controllers:

```
$  aws ds enable-ca-enrollment-policy \
    --pca-connector-arn MyPcaConnectorAdArn \
    --directory-id MyDirectoryId
```

When using AD Connector, use the following steps to create a GPO that points to the URI generated when you created a connector. This step is *required* to use Connector for AD from the console or the command-line.

Configure GPOs.

1. Open **Server Manager** on the DC
2. Go to **Tools** and choose **Group Policy Management** in the upper right corner of the console.
3. Go to **Forest > Domains**. Select your domain name and right click on your domain. Select *Create a GPO in this domain, and link it here …* and enter `PCA GPO` for the name.
4. The newly created GPO will now be listed under your domain name.
5. Choose **PCA GPO** and select **Edit**. If a dialog box opens with the alert message *This is a link and that changes will be globally propagated*, acknowledge the message to continue. The **Group Policy Management Editor** should open.
6. In the **Group Policy Management Editor**, go to **Computer Configuration > Policies > Windows Settings > Security Settings > Public Key Policies (choose the folder)**.
7. Go to **object type** and choose **Certificate Services Client - Certificate Enrollment Policy**
8. In the options, change **Configuration Model** to **Enabled**.
9. Confirm that **Active Directory Enrollment Policy** is **checked** and **Enabled**. Choose **Add**.
10. The **Certificate Enrollment Policy Server** window should open.
11. Enter the certificate enrollment policy server endpoint that was generated when you created your connector in the **Enter enrollment server policy URI** field.
12. Leave the **Authentication Type** as **Windows integrated**.
13. Choose **Validate**. After validation succeeds, select **Add**. The dialog box closes.
14. Go back to **Certificate Services Client - Certificate Enrollment Policy** and check the box beside the newly created connector to ensure that the connector is the default enrollment policy
15. Choose **Active Directory Enrollment Policy** and select **Remove**.
16. In the confirmation dialog box, choose **Yes** to delete the LDAP-based authentication.
17. Choose **Apply** and **OK** on the **Certificate Services Client > Certificate Enrollment Policy** window and close it.
18. Go to the **Public Key Policies** folder and choose **Certificate Services Client - Auto-Enrollment**.
19. Change the **Configuration Model** option to **Enabled**.

20. Confirm that **Renew expired certificates** and **Update Certificates** are both checked. Leave the other settings as they are.
21. Choose **Apply**, then **OK**, and close the dialogue box.

Configure the Public Key Policies for user configuration next. Go to **User Configuration > Policies > Windows Settings > Security Settings > Public Key Policies**. Follow the procedures outlined from step 6 to step 21 to configure the Public Key Policies for user configuration.

Once you've finished configuring GPOs and Public Key Policies, objects in the domain will request certificates from AWS Private CA Connector for AD and get certificates issued by AWS Private CA.

Step 3: Create a template

To create a template, see Create a connector template.

Step 4: Configure Microsoft group permissions

To configure Microsoft group permissions, see Manage Connector for AD template access control entries.

AWS Private CA connectors for Active Directory

The procedures in this section describe how to create Active Directory (AD) connectors, configure templates, and integrate with AWS Private CA and Active Directory. You can perform these operations from the AWS Private CA Connector for AD console, by using the Connector for AD section of the AWS CLI, or by using the AWS Private CA Connector for AD API.

> **Note**
>
> Although AWS Private CA Connector for AD is closely integrated with AWS Private CA, the two services have separate APIs. For more information, see the AWS Private Certificate Authority API Reference and the AWS Private CA Connector for Active Directory API Reference.

Creating a connector for Active Directory

Use the following procedures to create a connector using the console, command line, or API for AWS Private CA Connector for Active Directory.

Console

To create a connector using the console

Sign in to your AWS account and open the AWS Private CA Connector for Active Directory console at **https://console.aws.amazon.com/pca-connector-ad/home**.

1. On the first-time service landing page or the **Connectors for Active Directory** page, choose **Create connector**.
2. On the **Create Private CA Connector for Active Directory** page, provide information in the **Active Directory** section.

   - Under **Select your Active Directory type**, choose one of the two available types:
     - **AWS Directory Service for Microsoft Active Directory** – Specifies an Active Directory managed by Directory Service.
     - **On-premises Active Directory with AWS AD Connector**– Uses AD Connector to access an Active Directory that you host on-premises.
   - Under **Select your directory**, choose your directory from the list.

     Alternatively, you can choose **Create directory**, which opens the Directory Service console in a new window. When you finish creating a new directory, return to the AWS Private CA Connector for Active Directory console and refresh the list of directories. Your new directory should be available for selection.

     > **Note**
     >
     > When creating a directory, note that Connector for AD supports only the following directory types offered in the Directory Service console:
     >
     > - **AWS Managed Microsoft AD**
     > - **AD Connector**

   - Under **Select security groups for VPC endpoint**, choose a security group from the list.

     Alternatively, you can choose **Create security group**, which opens the Amazon EC2 console to the **Create security group** page in a new window. When you finish creating a security group, return to the AWS Private CA Connector for Active Directory console and refresh the list of security groups. Your new security group should be available for selection.

3. In the **IP address type** section, choose from the following options:

   - **IPv4** - Enables IPv4 connectivity to the service. Choose this option only if all subnets hosting your directory have IPv4 address ranges.
   - **Dualstack** - Enables both IPv4 and IPv6 connectivity to the service. Choose this option only if all subnets hosting your directory have both IPv4 and IPv6 address ranges.

4. In the **Private certificate authority** section, choose a private CA from the list.

   Alternatively, you can choose **Create Private CA**, which opens the AWS Private CA console to the **Private certificate authorities** page in a new window. When you finish creating a CA, return to the AWS Private CA Connector for Active Directory console and refresh the list of CAs. Your new CA should be available for selection.

5. In the **Tags – optional** pane, you can apply and remove metadata on your AD resource. Tags are key-value string pairs where the key must be unique to the resource and the value is optional. The pane displays any existing tags for the resource in a table. The following actions are supported.

   - Choose **Manage tags** to open the **Manage tags** page.
   - Choose Add new tag to create a tag. Fill in the **Key** field and, optionally, the **Value** field. Choose **Save changes** to apply the tag.
   - Choose the **Remove** button next to a tag to mark it for deletion, and choose **Save changes** to confirm.

6. After providing the required information and reviewing your choices, choose **Create connector**. This opens the **Connectors for Active Directory** details page where can view the progress of your connector as it is created.

   After the process of creating a connector completes, assign it a service principal name.

API

To create a connector using the API

To create a connector for Active Directory with the API, use the CreateConnector action in the AWS Private CA Connector for Active Directory API.

CLI

To create a connector using the AWS CLI

To create a connector for Active Directory with the CLI, use the create-connector command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Create a connector template

A template is a list of configurations for how the certificate should look once issued, and how the client should handle the certificates. The following procedures explain how to create a template.

Console

To create a template using the console

1. Sign in to your AWS account and open the AWS Private CA Connector for Active Directory console at **https://console.aws.amazon.com/pca-connector-ad/home**.
2. Choose a connector from the **Connectors for Active Directory** list and then choose **View details**.
3. On the details page for the connector, find the **Templates** section and then choose **Create template**.
4. On the **Create template** page, in the **Template creation method** section, choose one of the method options.

   - **Start from a predefined template** (default) – Choose from a list of predefined templates for AD applications:
     - **Code Signing**
     - **Computer**
     - **Domain Controller Authentication**
     - **EFS Recovery Agent**
     - **Enrollment Agent**
     - **Enrollment Agent (Computer)**
     - **IPSec**
     - **Kerberos Authentication**
     - **RAS and IAS Server**
     - **Smartcard Logon**
     - **Trust List Signing**
     - **User Signature**

- **Workstation Authentication**

- **Start from an existing template that you created** – Choose from a list of custom templates that you previously created.
- **Start from a blank template** – Choose this option to begin creating a completely new template.

5. In the **Certificate settings** section, define the following settings for certificates based on this template.

- **Certificate type** – Specify whether to create **User** or **Computer** certificates.
- **Auto-enrollment** – Choose whether to activate auto-enrollment for certificates based on this template.
- **Validity period** – Specify a certificate validity period as an integer value of hours, days, weeks, months, or years. The minimum value is 2 hours.
- **Renewal period** – Specify a certificate renewal period as an integer value of hours, days, weeks, months, or years. The renewal period must be no more than 75% of the validity period.
- **Subject name** – Choose one or more options to be included in the subject name based on information contained in Active Directory.

> **Note**
>
> At least one subject name or subject alternative name option must be specified.

  - **Common name**
  - **DNS as common name**
  - **Directory path**
  - **Email**
- **Subject alternative name** – Choose one or more options to be included in the subject alternative name based on information contained in Active Directory.

> **Note**
>
> At least one subject name or subject alternative name option must be specified.

- **Directory GUID**
- **DNS name**
- **Domain DNS**
- **Email**
- **Service principal name (SPN)**
- **User principal name (UPN)**

6. In the **Certificate request handling and enrollment options** section, specify the purpose of certificates based on the template, choosing one of the following options.

   - **Signature**
   - **Encryption**
   - **Signature and encryption**
   - **Signature and smartcard logon**

   Next, choose which of the following features to activate. Options vary depending on the certificate purpose.

   - **Delete invalid certificates (do not archive)**
   - **Include symmetric algorithms**
   - **Exportable private key**

   Finally, choose a certificate enrollment option. Options vary depending on the certificate purpose.

   - **No user input required**
   - **Prompt user during enrollment**
   - **Prompt user during enrollment and require user input**

7. In the **Application policies** section, choose all of the application policies that apply. The available policies are listed across several pages. Some policies may be preselected because of previous settings.
8. In the **Custom application policies** section, you can add custom OIDs to the template, and specify whether application policy extensions are critical.

9. In the **Cryptography settings** section, choose the following categories of cryptography settings for certificates based on this template.
10. In the **Groups and permissions** section, you can view the templates existing groups and permissions for enrollment, or you can choose the **Add new groups and permissions** button to add a new ones. The button opens a form requiring the following information:
    - **Display name**
    - **Security identifier** (SID)
    - **Enroll**, with options ALLOW | DENY | NOT SET
    - **Auto-enroll**, with options ALLOW | DENY | NOT SET
11. In the **Supersede templates** section, you can notify Active Directory that the current template supersedes one or more templates created in AD. Apply the superseding template by choosing **Add template from Active Directory to supersede** and specifying the common name of the superseding template.
12. In the **Tags – optional** pane, you can apply and remove metadata on your AD resource. Tags are key-value string pairs where the key must be unique to the resource and the value is optional. The pane displays any existing tags for the resource in a table. The following actions are supported.
    - Choose **Manage tags** to open the **Manage tags** page.
    - Choose Add new tag to create a tag. Fill in the **Key** field and, optionally, the **Value** field. Choose **Save changes** to apply the tag.
    - Choose the **Remove** button next to a tag to mark it for deletion, and choose **Save changes** to confirm.
13. After providing the required information and reviewing your choices, choose **Create template**. This opens **Template details**, where you can review the new template's settings, edit or delete the template, manage groups and permissions, manage superseded templates, manage tags, and set automatic re-enrollment for certificate holders.

API

To create a connector template using the API

Use the CreateTemplate action in the AWS Private CA Connector for Active Directory API.

CLI

To create a connector template using the AWS CLI

Use the create-template command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Update a template for Active Directory

Use the following procedures to update a template using the console, command line, or API for AWS Private CA Connector for Active Directory.

Console

To update a template using the console

Sign in to your AWS account and open the AWS Private CA Connector for Active Directory console at **https://console.aws.amazon.com/pca-connector-ad/home**.

1. On the list of your **Connectors for Active Directory**, select the connector whose template that you'd like to update. Choose **Edit** to view and modify the connector's templates.
2. In your connector's template details page, choose **Edit**. Follow the prompts to make your updates. When you're done editing an area, choose **Save** to save your changes.

API

To update a template using the API

To update a template for Active Directory with the API, use the UpdateTemplate action in the AWS Private CA Connector for Active Directory API.

CLI

To update a template using the AWS CLI

To update a connector for Active Directory with the CLI, use the update-template command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

How Connector for Active Directory propagates your template changes

AWS Private CA applies template to your policy when your client refreshes the policy cache, which is every eight hours. This includes changes to template group access control entries. When your client refreshes the cache, it queries the connector for available templates. In the case of auto-enrollment refresh, the client issues certificates that match either or both of the following conditions:

- The certificate is within the renewal period.
- The certificate isn't present on the client device.

For *manual refresh*, the client will query the connector, and you must set the template to issue.

If you're debugging, you can manually clear the policy cache to immediately see the template changes. To do so, run the following Powershell command on your client.

```
certutil -f -user -policyserver * -policycache delete
```

List connectors for Active Directory

You can use the AWS Private CA Connector for Active Directory console or AWS CLI to list the connectors that you own.

Console

To list your connectors using the console

1. Sign in to your AWS account and open the AWS Private CA Connector for Active Directory console at **https://console.aws.amazon.com/pca-connector-ad/home**.
2. Review the information in the **Connectors for Active Directory** list. You can navigate through multiple pages of connectors using the page numbers at upper-right. Each connector occupies a row displaying the following columns of information by default.

- **Connector ID** – The unique ID of the connector.
- **Directory name** – The Active Directory resource associated with the connector.
- **Connector status** – Connector status. Possible values are: **Creating | Active | Deleting | Failed**.

- **Service principal name status** – Status of the service principal name (SPN) associated with the connector. Possible values are: **Creating | Active | Deleting | Failed**.
- **Directory registration status** – Registration status of the associate director. Possible values are:**Creating | Active | Deleting | Failed**.
- **Created at** – Time stamp at the connector's creation.

By choosing the gear icon in the upper-right corner of the console, you can customize the number of connectors shown on a page using the **Page size** preference.

API

To list your connectors using the API

Use the ListConnectors action in the AWS Private CA Connector for Active Directory API.

CLI

To list your connectors using the AWS CLI

Use the list-connectors command to list your connectors.

List connector templates

You can use the AWS Private CA Connector for Active Directory console or AWS CLI to list templates for connectors that you own. Connector templates are based on AWS Private CA BlankEndEntityCertificate_APIPassthrough/V1 templates.

Console

To list your templates using the console

1. Sign in to your AWS account and open the AWS Private CA Connector for Active Directory console at **https://console.aws.amazon.com/pca-connector-ad/home**.
2. Choose a connector from the **Connectors for Active Directory** list and then choose **View details**.
3. On the connector details page, review the information in the **Templates** section. You can navigate through multiple pages of templates using the page numbers at upper-right. Each template occupies a row displaying the following columns of information.

- **Template name** – The human-readable name of the template.
- **Template status** – Status of the template. Possible values are: **Active | Deleting**.
- **Template ID** – The unique identifier of the template.

API

To list your connectors using the API

Use the ListTemplates action in the AWS Private CA Connector for Active Directory API to list templates for the specified connector.

CLI

To list your connectors using the AWS CLI

Use the list-templates command to list templates for the specified connector.

View connector details

Use the following procedures to view the configuration details of a connector in the console, command line, or API for AWS Private CA Connector for Active Directory.

Console

To view details for a connector using the console

1. Sign in to your AWS account and open the AWS Private CA Connector for Active Directory console at **https://console.aws.amazon.com/pca-connector-ad/home**.
2. Choose a connector from the **Connectors for Active Directory** list and then choose **View details**.
3. On the connector details page, review the information in the Connector details, pane, which includes the following:

   - **Connector ID**
   - **Connector status**
   - **Additional status details**
   - **Connector ARN**
   - **Certificate enrollment policy server endpoint**

- **Directory name**
- **Directory ID**
- **AWS Private CA subject**
- **AWS Private CA status**
- **IP address type**
- **VPC endpoint and security groups**

4. In the **Templates** pane, you can create or manage templates associated with the connector.
5. From the **Service principal name (SPN)** pane, you can view the service principle name associated with the connector.
6. From the **Directory Registration** pane, you can view or change the directory registration associated with the connector.
7. From the **Tags — *optional*** pane, you can create or manage tags associated with the connector.

API

To list your connectors using the API

Use the GetConnector action in the AWS Private CA Connector for Active Directory API.

CLI

To list your connectors using the AWS CLI

Use the get-connector command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

View connector template details

Use the following procedures to view the configuration details of a connector template using the console, command line, or API for AWS Private CA Connector for Active Directory

Console

To view details for a connector template using the console

1. Sign in to your AWS account and open the AWS Private CA Connector for Active Directory console at **https://console.aws.amazon.com/pca-connector-ad/home**.

2. Choose a connector from the **Connectors for Active Directory** list and then choose **View details**.
3. On the connector details page, review the information in the **Templates** section, and select the template that you wish to inspect. Then choose **View details**.
4. On the details page, the **Template details** pane displays the following information about the template:

   - **Template name**
   - **Template ID**
   - **Template status**
   - **Template schema version**
   - **Template version**
   - **Template ARN**
   - **Certificate type**
   - **Auto-enrollment turned on**
   - **Validity period**
   - **Renewal period**
   - **Subject name requirements**
   - **Subject alternative name requirements**
   - **Certificate request and enrollment settings**
   - **Cryptography provider category**
   - **Key algorithm**
   - **Minimum key size (bits)**
   - **Hash algorithm**
   - **Cryptography providers**
   - **Key usage extension settings**

   From this pane, you can also perform the following actions using the **Edit**, **Delete**, and **Actions** buttons.

   - **Edit**
   - **Delete**

- **Manage groups and permissions** – For more information, see Configure groups and permissions.
- **Manage superseded templates** – For more information, see Review and create.
- **Manage tags** – For more information, see Tagging Connector for AD resources.
- **Re-enroll all certificate holders** – This setting allows the major version of a template to be increased automatically. All members of Active Directory groups that are allowed to enroll with a template will receive a new certificate issued using that template. For more information, see the UpdateTemplate API.

5. The lower pane displays a row of tabs allowing changes to the configuration of the template.

- **Groups and permissions** – View and manage permissions for Active Directory groups to enroll certificates using this template. For more information, see Configure groups and permissions
- **Application policies** – View and manage template application policies. For more information, see Assign application policies.
- **Superseded templates** – View and manage superseded templates. For more information, see Review and create.
- **Tag*optional*** – View and manage tagging on this template. For more information, see Tagging Connector for AD resources.

API

To list your connectors using the API

Use the GetTemplate action in the AWS Private CA Connector for Active Directory API.

CLI

To list your connectors using the AWS CLI

Use the get-template command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Manage directory registrations

Console

To manage directory registrations using the console

Directory registrations for connectors can be managed from the top level of the AWS Private CA Connector for Active Directory console. This topic walks through the available management options.

1. Sign in to your AWS account and open the AWS Private CA Connector for Active Directory console at https://console.aws.amazon.com/pca-connector-ad/home.
2. In the left navigation area, choose **Directory registrations**.
3. The **Directory registrations** page displays a table of registered directories with the following fields:

   - **Directory ID** – The unique ID of the directory
   - **Directory name** – The directory domain site name
   - **Directory type**
   - **Registered** – The status of the registration. Supported values are CREATING | ACTIVE | DELETING | FAILED.
   - **Directory status** – The status of the directory

   Use can use **Register directory** to create a new registration.
4. You can select one of the listed registrations in order to manage it. This enables the **View registration details** and **Deregister directory** buttons. The **View registration details** button opens the details page for the registration.
5. The **Directory registration details** pane displays the following information:

   - **Directory domain site name**
   - **Directory ID** – The unique ID of the directory. Choosing the link takes you to the AWS Directory Service console.
   - **Directory type**
   - **Status** – Status of the directory
   - **Directory registration ARN** – The Amazon resource name of the directory registration

- **Additional status information**

6. In the **Connectors and service principal name (SPNs)** pane, you can manage SPNs for the connector. For more information, see View connector details.
7. In the **Tags – optional** pane, you can apply and remove metadata on your AD resource. Tags are key-value string pairs where the key must be unique to the resource and the value is optional. The pane displays any existing tags for the resource in a table. The following actions are supported.

   - Choose **Manage tags** to open the **Manage tags** page.
   - Choose Add new tag to create a tag. Fill in the **Key** field and, optionally, the **Value** field. Choose **Save changes** to apply the tag.
   - Choose the **Remove** button next to a tag to mark it for deletion, and choose **Save changes** to confirm.

API

To manage directory registrations using the API

Create: CreateDirectoryRegistration action in the AWS Private CA Connector for Active Directory API.

Retrieve: GetDirectoryRegistration action in the AWS Private CA Connector for Active Directory API.

List: ListDirectoryRegistrations action in the AWS Private CA Connector for Active Directory API.

Delete: DeleteDirectoryRegistration action in the AWS Private CA Connector for Active Directory API.

CLI

To manage directory registrations using the CLI

Create: Use the create-directory-registration command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Retrieve: get-directory-registration command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

List: list-directory-registrations command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Delete: delete-directory-registration command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Manage Connector for AD template access control entries

An access control entry grants controls which Active Directory groups can or cannot enroll certificates for a specific Connector for AD template. When you can create or manage groups and permissions in Connector for AD, you must provide the Security identifier (SID) of the group object from Active Directory. You can obtain the SID using the following PowerShell command. For information about SIDs, see How security identifiers work in the Microsoft Directory Domain Services documentation.

```
$ Get-ADGroup -Identity "my_active_directory_group_name"
```

The following procedures illustrate how to create and manage Connector for AD template access group entries.

Console

To manage template group permissions using the console

You can manage groups and permissions for an existing template can be managed from a template's details page. For more information, see View connector template details.

Set permissions on which groups can or cannot enroll certificates for the specific template. You provide the security identifier (SID) of the group. Then you set the enroll and auto-enroll permissions for the group. For auto-enrollment, both enroll and auto-enroll must be set to "Allow."

API

To manage template group permissions using the API

Create: CreateTemplateGroupAccessControlEntry action in the AWS Private CA Connector for Active Directory API.

Update: UpdateTemplateGroupAccessControlEntry action in the AWS Private CA Connector for Active Directory API.

Retrieve: GetTemplateGroupAccessControlEntry action in the AWS Private CA Connector for Active Directory API.

List: ListTemplateGroupAccessControlEntries action in the AWS Private CA Connector for Active Directory API.

Delete: DeleteTemplateGroupAccessControlEntry action in the AWS Private CA Connector for Active Directory API.

CLI

To manage template group permissions using the CLI

Create: create-template-group-access-control-entry command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Update: update-template-group-access-control-entry command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Retrieve: get-template-group-access-control-entry command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

List: list-template-group-access-control-entries command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Delete: delete-template-group-access-control-entries command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Configuring the service principal name

Learn how to configure the service principal name for the connector.

Console

To manage manage service principal names using the console

The service principal name (SPN) of an existing AD connector can be managed from the details page of the connector. For more information, see Managing directory registration View connector details

API

To manage service principal names using the API

Create: CreateServicePrincipalName action in the AWS Private CA Connector for Active Directory API.

Retrieve: GetServicePrincipalName action in the AWS Private CA Connector for Active Directory API.

List: ListServicePrincipalNames action in the AWS Private CA Connector for Active Directory API.

Delete: DeleteServicePrincipalName action in the AWS Private CA Connector for Active Directory API.

CLI

To manage service principal names using the CLI

Create: create-service-principal-name command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Retrieve: get-service-principal-name command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

List: list-service-principal-names command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Delete: delete-service-principal-name command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Tagging Connector for AD resources

You can apply tags to your connectors, templates, and directory registrations. Tagging adds metadata to a resource that can assist with organization and management.

Console

To manage resource tagging using the console

Tagging of existing resources is managed on the details page for the resource. For more information, see the following procedures:

- View connector template details
- Managing directory registrations

API

To manage resource tagging using the API

Tag: TagResource action in the AWS Private CA Connector for Active Directory API.

List tags: ListTagsForResource action in the AWS Private CA Connector for Active Directory API.

Untag: UntagResource action in the AWS Private CA Connector for Active Directory API.

Important - It is acceptable to use tags to label objects containing confidential data. However, the tags themselves shouldn't contain any personally identifiable information (PII), sensitive, or confidential information.

CLI

To manage resource tagging using the CLI

Tag: tag-resource command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

List tags: list-tags-for-resource command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Untag: untag-resource command in the AWS Private CA Connector for Active Directory section of the AWS CLI.

Integrating Connector for AD into event-driven applications using Amazon EventBridge

You can incorporate Connector for AD into event-driven applications (EDAs) that use events that occur in Connector for AD to communicate between application components and initiate downstream processes.

For example, you could invoke other AWS services or custom components when the following Connector for AD events occur in your account:

- A certificate is created or when creation fails.
- A certificate is enrolled, or enrollment fails.

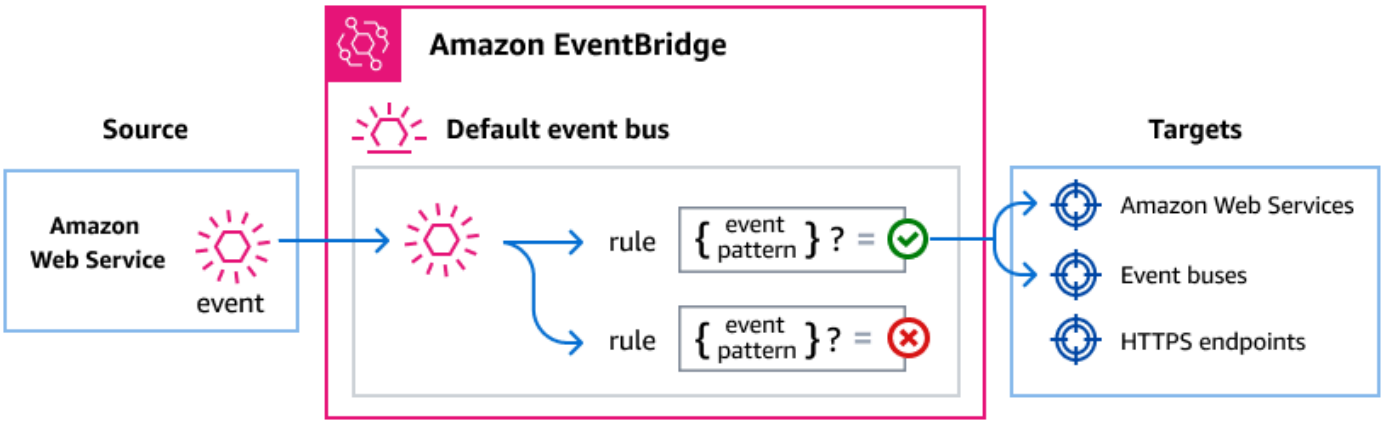
You do this by using Amazon EventBridge to route events from Connector for AD to other software components. Amazon EventBridge is a serverless service that uses events to connect application components together, making it easier for you to integrate AWS services like Connector for AD into event-driven architectures without additional code and operations.

How EventBridge routes Connector for AD events

Here's how EventBridge works with Connector for AD events:

As with many AWS services, Connector for AD generates and sends events to the EventBridge default *event bus*. An event bus is a router that receives events and routes them to the destinations, or *targets*, that you specify. Targets can include other AWS services, custom applications, and SaaS partner applications.

EventBridge routes events according to *rules* you create on the event bus. For each rule, you specify a filter, or *event pattern*, to select only the events you want. Whenever an event is sent to the event bus, EventBridge compares it against each rule. If the event matches the rule, EventBridge routes the event to the specified target(s).



Connector for AD events

For a list of Connector for AD events sent to EventBridge, refer to the Connector for AD topic in the *EventBridge Events Reference*.

Event structure

All events from AWS services contain two types of data:

- A common set of fields containing metadata about the event, such as the AWS service that is the source of the event, the time the event was generated, the account and region in which the event took place, and others. For definitions of these general fields, see Event structure in the *Amazon EventBridge Events Reference*.
- A `detail` field that contains data specific to that particular service event.

Creating event patterns that match Connector for AD events

Event patterns are filters where specify what data the events you want to select should contain.

Each event pattern is a JSON object that contains:

- A `source` attribute that identifies the service sending the event. For Connector for AD events, the source is `aws.pca-connector-ad`.
- (Optional): A `detail-type` attribute that contains an array of the event names to match.
- (Optional): A `detail` attribute containing any other event data on which to match.

For example, the following event pattern would select all Certificate Policy Enrollment Succeeded events from Connector for AD:

```
{
  "source": ["aws.pca-connector-ad"],
  "detail-type": ["Certificate Policy Enrollment Succeeded"]
}
```

For more information on writing event patterns, see Event patterns in the *EventBridge User Guide*.

Receiving events from EventBridge

You can specify Connector for AD certificates as the target for a rule. This enables Connector for AD to receive events from a wide variety of sources, including other AWS services, custom applications, and SaaS partners. For more information, see Creating rules that react to events in the *EventBridge User Guide*.

For a full list of the AWS services that you can specify as targets, see Target types in the *EventBridge Events Reference*.

Troubleshoot issues with AWS Private CA Connector for Active Directory

Use the information here to help you diagnose and fix AWS Private Certificate Authority Connector for AD issues.

Topics

- Troubleshoot Connector for AD error codes
- Troubleshoot Connector for AD connector creation failures
- Troubleshoot Connector for AD SPN creation failure
- Troubleshoot Connector for AD template update issues

Troubleshoot Connector for AD error codes

Connector for AD sends error messages for several reasons. For information on each error and recommendations about resolving them, see the following table. You can receive these errors by subscribing to Amazon EventBridge Scheduler events (event source: `aws.pca-connector-ad`) or by using manual enrollment in Windows.

Error code	Root cause	Remediation
0x8FFFA000	Kerberos authentication failed.	Make sure that your directory is reachable and the client is either a user or computer. If you're using auto-enrollment, then fix your AWS resource service principal. If you're using the Active Directory UI to get a cert, run `gpupdate /force`.
0x8FFFA001	The SOAP message must contain an action header.	Add an action header.

Error code	Root cause	Remediation
0x8FFFA002	The connector does not have access to the private CA it is connected to.	Share your private CA with the connector by creating an AWS Resource Access Manager (RAM) to share between your private CA and the Connector for AD service.
0x8FFFA003	The private CA for this connector is not active.	Move the private CA to Active state. If your private CA is in the pending certificate state, then install the CA certificate.
0x8FFFA004	The private CA for this connector does not exist.	Move your certificate authority to the Active state if it is in the Deleted state. If your private CA is permanently deleted then create a new connector with a different CA.
0x8FFFA005	The template specified the `directoryGuid` attribute for the certificate subject or the subject alternate name, but the attribute was not found in the AD object for the requester.	Active Directory did not generate a `directoryGuid` for your directory. Troubleshoot in Active Directory.
0x8FFFA006	The template specified the `dnsHostName` attribute for the certificate subject or the subject alternate name, but the attribute was not found in the AD object for the requester.	Add the `dnsHostName` attribute to your AD object.

Error code	Root cause	Remediation
0x8FFFA007	The template specified the email attribute to be included in the certificate subject or the subject alternate name, but the attribute was not found in the AD object for the requester.	Add the email attribute to your AD object
0x8FFFA008	The SOAP message must have an action header of either `http://schemas.microsoft.com/windows/pki/2009/01/enrollmentpolicy/IPolicy/GetPolicies` or `http://schemas.microsoft.com/windows/pki/2009/01/enrollment/RST/wstep` .	Update the action header to use one of the specified values.
0x8FFFA009	The BinarySecurityToken must be encoded in `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd#base64binary` .	Update the binary security token type.
0x8FFFA00A	The BinarySecurityToken is invalid.	Check that the CSR is generated correctly.

Error code	Root cause	Remediation
0x8FFFA00B	The BinarySecurityToken must have a value type of either `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd#PKCS7` or `http://schemas.microsoft.com/windows/pki/2009/01/enrollment#PKCS10` .	Update the binary security token value type to a valid value.
0x8FFFA00C	The BinarySecurityToken contained invalid CMS.	The Base64 is valid but the cryptographic message syntax (CMS) is invalid. Review the CMS syntax.
0x8FFFA00D	The BinarySecurityToken contained an invalid CSR.	Check that the CSR was generated correctly.
0x8FFFA00E	The private CA was unable to issue a certificate using the specific template.	Review the validation exception from AWS Private CA. You can view the validation exception in Amazon EventBridge or AWS CloudTrail.
0x8FFFA00F	The SOAP message must have a request type of `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue` .	Set the request type to `http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue` .

Error code	Root cause	Remediation
0x8FFFA010	The SOAP message must have a to header of either the connector's `CertificateEnrollmentPolicyServerEndpoint` field or the URI field in the XCEP response.	Set the header of the request security token to either the `CertificateEnrollmentPolicyServerEndpoint` field or the URI field in the XCEP response.
0x8FFFA011	The SOAP message must have only one action header.	Review the SOAP message header of the request security token and set the header correctly.
0x8FFFA012	The SOAP message must have only one `messageId` header.	Review the SOAP message header of the request security token and set the header correctly.
0x8FFFA013	The SOAP message must have only one to header.	Review the SOAP message header of the request security token and set the header correctly.
0x8FFFA014	The requester does not have access to the requested template.	Allow the requester's group to enroll using the requested template by creating an Access Control Entry.
0x8FFFA015	Either the `CertificateTemplateInformation` or the `CertificateTemplateName` extension must be present in the BinarySecurityToken.	Add the security extension to your CSR.

Error code	Root cause	Remediation
0x8FFFA016	The requested template was not found for the given connector.	Templates are child resources to each connector. Create the template for the connector using `createTemplate` .
0x8FFFA017	The request was denied due to request throttling.	Slow down the rate of requests.
0x8FFFA018	The SOAP message must contain a `to` header.	Review the header of the SOAP message.
0x8FFFA019	Could not process the SOAP message due to an unrecognized header.	Review the header of the SOAP message.
0x8FFFA01A	The template specified the UPN attribute to be included in the certificate subject or the subject alternate name, but the attribute was not found in the AD object for the requester.	Add an UPN to the Active Directory object.

Troubleshoot Connector for AD connector creation failures

Connector for AD connector creation can fail for various reasons. When connector creation fails, you'll receive the failure reason in the API response. If you're using the console, then the failure reason is displayed in the Connector **details** page under the **Additional status details** field within in the Connector **details** container. The following table describes failure reasons and recommended steps for resolution.

Failure status	Description	Remediation
`CA_CERTIFICATE_REGISTRATION_FAILED`	Connector for AD is unable to import CA certificates into your directory.	Review the Prerequisites page and check that your service account has the right permissions. After delegating the correct permissions to your service account, delete the failed connector and create a new one. For information about delegating permissions, see Delegate privileges to your service account in the *AWS Directory Service Administration Guide*.
`DIRECTORY_ACCESS_DENIED`	Connector for AD unable to access your directory.	You must grant Connector for AD access to your directory. Review the Step 4: Create IAM Policy section to make sure that you the IAM policy associated with your AWS account enables you to access and describe directories. After granting the correct permissions to your AWS role, delete the failed connector and create a new one. If using Connector for AD with an AWS Directory Service AD Connector, make sure that the AD Connector service account's password isn't expired and is valid.

Failure status	Description	Remediation
		For information about AD Connector service accounts, see Getting started with AD Connector in the *AD Connector Administration Guide*.
`INTERNAL_FAILURE`	Connector for AD experienced an internal failure.	Try again later. Delete the failed connector and create a new one.
`INSUFFICIENT_FREE_ADDRESSES`	The VPC subnet must have at least one available private IP address.	Ensure that there is an available private IP address in the subnet. Delete the failed connector and create a new one.
`INVALID_SUBNET_IP_PROTOCOL`	Connector for AD is unable to create the endpoint on your VPC because the subnets associated with your directory do not support the specified IP address type.	Ensure the VPC and subnets that host your directory support your chosen IP address type. For more information, see IP address types. Delete the failed connector and create a new one with the supported IP address type.

Failure status	Description	Remediation
PRIVATECA_ACCESS_DENIED	Connector for AD is unable to access your private CA.	Review the Prerequisites page and check that you have the permissions to create a connector. For information, see Step 4: Create IAM Policy. If you're creating a connector through AWS CLI or API, review the Prerequisites page and check that you have shared the private CA with Connector for AD using AWS Resource Access Manager. After checking and fixing IAM permissions and AWS RAM resource sharing, delete the failed connector and create a new one.
PRIVATECA_RESOURCE_NOT_FOUND	Connector for AD can't find the specified private CA.	Make sure that you specify the correct private CA Amazon Resource Name (ARN), then delete the failed connector and create a new one using your intended private CA ARN.

Failure status	Description	Remediation
SECURITY_GROUP_NOT_IN_VPC	The security group isn't in the VPC that hosts your directory.	Use a security group that is in the VPC that hosts your directory. For more information, see Step 7: Configure security groups. Delete the failed connector and create a new one with a security group that is in the VPC.
VPC_ACCESS_DENIED	Connector for AD can't access the Amazon VPC that hosts your directory.	Check your IAM permissions. Delete the failed connector and create a new one. For an example IAM policy that includes access permissions, see Step 4: Create IAM Policy
VPC_ENDPOINT_LIMIT_EXCEEDED	Connector for AD can't create an endpoint in your Amazon VPC. You have reached the limit of VPC endpoints that you can create for your account.	Delete Amazon VPC endpoints, or request a limit increase. Once you've done one of the two steps, delete the failed connector and create a new one. For information about quotas, see Amazon Virtual Private Cloud Service quotas.
VPC_RESOURCE_NOT_FOUND	Connector for AD can't find the specified VPC.	Make sure that you specified the correct VPC and that the VPC exists. Then delete the failed connector and create a new one using the correct VPC ID.

Troubleshoot Connector for AD SPN creation failure

Service principal name (SPN) creation can fail for various reasons. When SPN creation fails you'll receive the failure reason in the API response. If you're using the console, then the failure reason is displayed in the Connector details page under the **Additional status details** field within the **Service principal name (SPN)** container. The following table describes failure reasons and recommended steps for resolution.

Failure status	Description	Remediation
`DIRECTORY_ACCESS_DENIED`	Connector for AD can't access your directory.	Grant Connector for AD access to your directory. For an example IAM policy that includes permissions that grant directory access, see Step 4: Create IAM Policy.
`DIRECTORY_NOT_REACHABLE`	Connector for AD can't access your directory.	Check the network between AWS and your directory, and try creating an SPN again.
`DIRECTORY_RESOURCE_NOT_FOUND`	Connector for AD can't find the specified directory.	Make sure you specify the correct directory ID, then delete the failed connector and create a new one using your intended directory ID.
`INTERNAL_FAILURE`	Connector for AD experienced an internal failure.	Try again later.
`SPN_EXISTS_ON_DIFFERENT_AD_OBJECT`	The service principal name (SPN) exists on a different Active Directory object.	Delete the SPN from the Active Directory object, and try creating the SPN again.
`SPN_LIMIT_EXCEEDED`	Connector for AD can't create the SPN because you've	

Failure status	Description	Remediation
	reached the limit of SPNs per directory. The maximum number of SPNs per directory is 10.	Delete one or more SPNs from your account, and try creating the SPN again.

Troubleshoot Connector for AD template update issues

If you made changes to your template or group access control entry, but you don't see the changes, this might be due to policy caching. AWS Private CA applies template to your policy when your client refreshes the policy cache, which is every eight hours. When your client refreshes the cache, it queries the connector for available templates. In the case of auto-enrollment refresh, the client issues certificates that match either or both of the following conditions:

- The certificate is within the renewal period.
- The certificate isn't present on the client device.

For *manual refresh*, the client will query the connector, and you must set the template to issue.

If you're debugging, you can manually clear the policy cache to immediately see the template changes. To do so, run the following Powershell command on your client.

```
certutil -f -user -policyserver * -policycache delete
```

AWS Private CA Connector for SCEP

Connector for Simple Certificate Enrollment Protocol (SCEP) links AWS Private Certificate Authority to your SCEP-enabled mobile devices and networking equipment. With Connector for SCEP, you can use AWS Private CA to issue certificates and enroll your SCEP devices. Connector for SCEP is available to use with popular mobile device management (MDM) systems and is designed to work with clients or endpoints that supports SCEP.

Topics

- Features
- How to get started with Connector for SCEP
- Related services
- Access Connector for SCEP
- Pricing
- Connector for SCEP concepts
- Understand Connector for SCEP considerations and limitations
- Set up Connector for SCEP
- Get started with Connector for SCEP
- Configure your MDM system for Connector for SCEP
- Monitor Connector for SCEP
- Troubleshoot AWS Private Certificate Authority Connector for SCEP issues

Features

Support for SCEP protocol - SCEP is a widely-adopted protocol for getting digital identity certificates from a certificate authority (CA) and distributing them to mobile devices and networking gear. You can use Connector for SCEP to help you enroll your endpoints using SCEP.

Mobile device enrollment - You can use Connector for SCEP with popular MDM systems including Microsoft Intune and Jamf Pro.

Issue certificates at scale - After you configure your SCEP-enabled devices to request certificates through the connector's SCEP endpoint, your clients can automatically request certificates from AWS Private CA.

How to get started with Connector for SCEP

To get started, launch the guided wizard from the Connector for SCEP management console which helps you create a connector and designate the private CA to use with the connector. After completing these steps, Connector for SCEP provides an endpoint and other configuration parameters that you can enter into your MDM systems or networking equipment. After configuring your MDM systems or networking equipment, your clients will automatically request certificates from AWS Private CA. To learn more about how to get started with Connector for SCEP, see Get started with Connector for SCEP.

Related services

Connector for SCEP is related to the following AWS services.

- **AWS Private Certificate Authority** - AWS Private CA provides you a highly-available private CA service without the upfront investment and ongoing maintenance costs of operating your own private CA.
- **AWS Private CA Connector for Active Directory** - Connector for AD links your Active Directory (AD) to AWS Private CA. The connector brokers the exchange of certificates from AWS Private CA to users and machines managed by your AD.

Access Connector for SCEP

You can create, access, and manage your Connector for SCEP connectors using any of the following interfaces:

- **AWS Management Console** - Provides a web interface that you can use to access Connector for SCEP. See Connector for SCEP management console.
- **AWS Command Line Interface** - Provides commands for a broad set of AWS services, including Connector for SCEP. The AWS CLI is supported on Windows, macOS, and Linux. For more information, see AWS Command Line Interface.
- **AWS SDKs** - Provide language-specific APIs and take care of many of the connection details, such as calculating signatures, handling request retries, and error handling. For more information, see AWS Command Line Interface.
- **Connector for SCEP API** - Provides low-level API actions that you call using HTTPS requests. Using the Connector for SCEP API is the most direct way to access the service. However,

the Connector for SCEP API requires that your application handle low-level details such as generating the hash to sign the request, and error handling. For more information, see Connector for SCEP API reference.

Pricing

Connector for SCEP is offered as a feature of AWS Private CA at no additional cost. You only pay for AWS Private Certificate Authority operations and certificates used to create and update connectors.

For the latest AWS Private CA pricing information, see AWS Private Certificate Authority Pricing. You can also use the AWS pricing calculator to estimate costs.

Connector for SCEP concepts

Connector for SCEP is an add-on feature for AWS Private Certificate Authority.

The following are the key concepts for Connector for SCEP:

Certificate Signing Request (CSR)

The required information provided to a CA in order to have a digital certificate issued. This information contains a public key as well as an identity.

Challenge password

The SCEP protocol uses challenge passwords to authenticate a request before issuing a certificate from a CA. Connector for SCEP handles SCEP challenge passwords based on the connector type. For more information, see Configure your MDM system for Connector for SCEP.

Certificate revocation

Certificate revocation is the process of revoking an issued certificate before its expiration date. You can revoke the private CA certificate associated to a connector by calling RevokeCertificate in the API, AWS SDK, AWS Command Line Interface, or AWS CloudFormation.

Connector for SCEP

A connector for SCEP links AWS Private CA to your SCEP-enabled devices.

Mobile Device Management

Mobile Device Management (MDM) allows IT administrators to control, secure, and enforce policies on smartphones, tablets, and other endpoints or devices. Many MDM systems provide built-in integrations for SCEP-based certificate enrollment.

SCEP

SCEP is a standardized protocol (RFC 8894) to automatically distribute certificates. The protocol provides an endpoint for devices to request certificates from a CA. SCEP uses challenge passwords to authorize certificate issuance to devices. SCEP is commonly applied for mobile device management (MDM) systems and networking equipment. MDM solutions allow IT administrators to control, secure and enforce policies on smartphones, tablets and other entities like Apple workstations. Most MDM solutions support SCEP, such as Microsoft Intune, Apple MDM, and Jamf Pro. Most networking equipment, such as routers, load balancers, Wi-Fi hubs, VPN devices and firewalls, use SCEP for automated certificate enrollment.

SCEP profile

A SCEP profile contains configuration parameters that are used to define the certificate profile. This includes certificate validity period, key size, SCEP configuration name, the challenge password, number of failed attempt retries and retry interval, and other information relevant to the issuance of certificates. MDM systems and certificate management platforms typically send the SCEP profile to the client that will request a certificate for authentication.

Understand Connector for SCEP considerations and limitations

Keep in mind the following considerations and limitations when using Connector for SCEP.

Considerations

CA operating modes

You can only use Connector for SCEP with private CAs that use a general-purpose operating mode. Connector for SCEP defaults to issuing certificates with a validity period of one year. A private CA using a short-lived certificate mode doesn't support issuing certificates with a validity period greater than seven days. For information about operating modes, see Understand AWS Private CA CA modes.

Challenge passwords

- Distribute your challenge passwords very carefully and share only with highly trusted individuals and clients. A single challenge password can be used to issue any certificate, with any subject and SANs, which poses a security risk.
- If using a general-purpose connector, we recommend that you manually rotate your challenge passwords frequently.

Conformance to RFC 8894

Connector for SCEP deviates from the RFC 8894 protocol by providing HTTPS endpoints instead of HTTP endpoints.

CSRs

- If a certificate signing request (CSR) that is sent to Connector for SCEP doesn't include the Extended Key Usage (EKU) extension, we'll set the EKU value to `clientAuthentication`. For information, see 4.2.1.12. Extended Key Usage in RFC 5280.
- We support `ValidityPeriod` and `ValidityPeriodUnits` custom attributes in CSRs. If your CSR doesn't include a `ValidityPeriod`, we issue a certificate that has a one year validity period. Keep in mind that you might not be able to set these attributes in your MDM system. But if you can set them, we support them. For information about these attributes, see szENROLLMENT_NAME_VALUE_PAIR.

Endpoint sharing

Distribute a connector's endpoints to trusted parties only. Treat the endpoints as secret because anyone who can find your unique fully-qualified domain name and path can retrieve your CA certificate.

Limitations

The following limitations apply to Connector for SCEP.

Dynamic challenge passwords

You can only create static challenge passwords with general-purpose connectors. To use dynamic passwords with a general-purpose connector, you must build your own rotation mechanism that employs the connector's static passwords. Connector for SCEP for Microsoft Intune connector types offer support for dynamic passwords, which you manage using Microsoft Intune.

HTTP

Connector for SCEP supports HTTPS only, and creates redirects for HTTP calls. If your system is reliant on HTTP, make sure that it can accommodate the HTTP redirects that Connector for SCEP provides.

Shared private CAs

You can only use Connector for SCEP with private CAs of which you are the owner.

Set up Connector for SCEP

The procedures in this section help you get started with Connector for SCEP. It assumes that you've already created an AWS account. After you complete the steps on this page, you can proceed with creating a connector for SCEP.

Topics

- Step 1: Create an AWS Identity and Access Management policy
- Step 2: Create a private CA
- Step 3: Create a resource share using AWS Resource Access Manager

Step 1: Create an AWS Identity and Access Management policy

To create a connector for SCEP, you need to create an IAM policy that grants Connector for SCEP the ability to create and manage resources needed by the connector, and to issue certificates on your behalf. For more information about IAM see What is IAM? in the *IAM User Guide*.

The following example is a customer managed policy that you can use for Connector for SCEP.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "pca-connector-scep:*",
            "Resource": "*"
        },
```

```
        {
            "Effect": "Allow",
            "Action": [
                "acm-pca:DescribeCertificateAuthority",
                "acm-pca:GetCertificate",
                "acm-pca:GetCertificateAuthorityCertificate",
                "acm-pca:ListCertificateAuthorities",
                "acm-pca:ListTags",
                "acm-pca:PutPolicy"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "acm-pca:IssueCertificate",
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "acm-pca:TemplateArn": "arn:aws:acm-pca:::template/
BlankEndEntityCertificate_APICSRPassthrough/V*"
                },
                "ForAnyValue:StringEquals": {
                    "aws:CalledVia": "pca-connector-scep.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ram:CreateResourceShare",
                "ram:GetResourcePolicies",
                "ram:GetResourceShareAssociations",
                "ram:GetResourceShares",
                "ram:ListPrincipals",
                "ram:ListResources",
                "ram:ListResourceSharePermissions",
                "ram:ListResourceTypes"
            ],
            "Resource": "*"
        }
    ]
}
```

Step 2: Create a private CA

To use Connector for SCEP you need to associate a private CA from AWS Private Certificate Authority to the connector. We recommend that you use a private CA that's only for the connector, due to inherent security vulnerabilities that are present in the SCEP protocol.

The private CA must meet the following requirements:

- It must be in an active state and use the general-purpose operating mode.
- You must own the private CA. You can't use a private CA that was shared with you through cross-account sharing.

Be aware of the following considerations when configuring your private CA to use with Connector for SCEP:

- **DNS name constrains** – Consider using DNS name constraints as a way to control which domains are allowed or prohibited in the certificates issued for your SCEP devices. For more information, see How to enforce DNS name constraints in AWS Private Certificate Authority.
- **Revocation** – Enable OCSP or CRLs on your private CA to allow for revocation. For more information, see Plan your AWS Private CA certificate revocation method.
- **PII** – We advise that you do not add personally identifiable information (PII) or other confidential or sensitive information in your CA certificates. In the event of a security exploit, this helps to limit exposure of sensitive information.
- **Store root certificates in trust stores** – Store your root CA certificates in your device trust stores, so that you can verify certificates and the return values of GetCertificateAuthorityCertificate. For information about trust stores as they relate to AWS Private CA, see Root CA .

For information about how to create a private CA, see Create a private CA in AWS Private CA.

Step 3: Create a resource share using AWS Resource Access Manager

If you're using Connector for SCEP programmatically using the AWS Command Line Interface, AWS SDK, or Connector for SCEP API, you need to share your private CA with Connector for SCEP by using AWS Resource Access Manager service principal sharing. This gives Connector for SCEP shared access to your private CA. When you create a connector in the AWS console, we automatically create the resource share for you. For information about resource sharing, see Create a resource share in the *AWS RAM User Guide*.

To create a resource share using the AWS CLI, you can use the AWS RAM **create-resource-share** command. The following command creates a resource share. Specify the ARN of the private CA that you want to share as the value of *resource-arns*.

```
$  aws ram create-resource-share \
--region us-east-1 \
--name MyPcaConnectorScepResourceShare \
--permission-arns arn:aws:ram::aws:permission/
AWSRAMBlankEndEntityCertificateAPICSRPassthroughIssuanceCertificateAuthority \
--resource-arns arn:aws:acm-pca:Region:account:certificate-authority/CA_ID \
--principals pca-connector-scep.amazonaws.com \
--sources account
```

The service principal that calls `CreateConnector` has certificate issuance permissions on the private CA. To prevent service principals that use Connector for SCEP from having general access to your AWS Private CA resources, restrict their permissions using `CalledVia`.

Get started with Connector for SCEP

With AWS Private Certificate Authority Connector for SCEP, you can issue certificates from your private CA to SCEP-enabled devices and mobile device management (MDM) systems. When you create a connector, AWS Private Certificate Authority creates a public SCEP URL for you to request certificates, and also provides you with information that you can use to integrate into your MDM systems.

To issue certificates, you must create an AWS Private Certificate Authority private CA, create a connector, and then configure your SCEP-enabled MDM systems and devices to request certificates from the connector.

Topics

- Before you begin
- Step 1: Create a connector
- Step 2: Copy connector details into your MDM system

Before you begin

The following tutorial guides you through the process of creating a connector for SCEP.

To follow this tutorial, you'll need a private CA and a SCEP-enabled device. You also must first fulfill the prerequisites listed in the Set up Connector for SCEP section.

The following procedure guides you how to create a connector using the AWS console.

Tasks

- Step 1: Create a connector
- Step 2: Copy connector details into your MDM system

Step 1: Create a connector

You'll create either a connector for general-purpose use or Connector for SCEP for Microsoft Intune. General-purpose connectors are designed for use with SCEP-enabled endpoints, and you manage the SCEP challenge passwords. Connector for SCEP for Microsoft Intune are for use with Microsoft Intune, and you manage the challenge passwords using Microsoft Intune.

General-purpose

To create a connector for general-purpose use

Sign in to your AWS account and open the Connector for SCEP console at **https://console.aws.amazon.com/pca-connector-scep/home**.

1. Choose **Create connector**.
2. In the **Create connector** page, optionally give the connector a friendly name in the **Name tag** field. The name will be displayed in your list of connectors. If you wish, you can add more tags to the connector by selecting **Add more tags**. A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.
3. Under **Connector type**, choose **General-purpose**.
4. Under **Private CA**, choose the private CA to use with this connector. Or, create a new one by selecting **Create private CA**. Due to the inherent vulnerabilities in the SCEP protocol, we recommend using a private CA that's dedicated to this connector. If you created a new CA, when you finish creating it in AWS Private CA, return to the Connector for SCEP console and refresh the list of private CAs. Your new private CA should be available for selection.
5. Under **Challenge password** select **Automatically generate challenge password**. We'll generate a static challenge password for you when we create this connector.

6. Select **Create connector**.

Microsoft Intune

To create Connector for SCEP for Microsoft Intune

Sign in to your AWS account and open the Connector for SCEP console at **https://console.aws.amazon.com/pca-connector-scep/home**.

1. Choose **Create connector**.
2. On the **Create connector** page, optionally give the connector a friendly name in the **Name tag** field. The name will be displayed in your list of connectors. If you wish, you can add more tags to the connector by selecting **Add more tags**. A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.
3. Under **Connector type**, choose **Microsoft Intune**.
   a. For **Application (client) ID**, enter the application (client) ID from your Microsoft Entra ID app registration. For information about using Microsoft Intune with Connector for SCEP, see Configure your MDM system for Connector for SCEP.
   b. For **Directory (tenant) ID or primary domain**, enter either the directory (tenant) ID or primary domain from your Microsoft Entra ID app registration.
4. Under **Private CA**, choose the private CA to use with this connector. Or, create a new one by selecting **Create private CA**. Due to the inherent vulnerabilities in the SCEP protocol, we recommend using a private CA that's dedicated to this connector. If you created a new CA, when you finish creating it in AWS Private CA, return to the Connector for SCEP console and refresh the list of private CAs. Your new private CA should be available for selection.
5. Select **Create connector**.

Step 2: Copy connector details into your MDM system

After you create your connector, you'll need to copy the following details from the connector into your MDM system. To view a connector's details using the console, select the connector from the list on the Connectors for SCEP console page.

- **Public SCEP URL** - This is the connector's endpoint where your SCEP clients will request certificates from. Take care to only provide this endpoint to trusted entities.

- (General-purpose) **Challenge password** - Under **Challenge passwords**, select the password that you automatically generated in the preceding procedure and then select **View password** to view the password. To create an additional password, select **Create password**. Take care to distribute passwords carefully and to only highly trusted individuals and clients. A single challenge password can be used to issue any certificate, with any subject and SANs, and so should be handled with care.
- (Microsoft Intune) **Open ID** values - If you're integrating with Microsoft Intune, you must copy the **Open ID issuer**, **Open ID subject**, and **Open ID audience** into your Microsoft Entra app registration's OpenID Connect (OIDC) credential. For more information, see Configure your MDM system for Connector for SCEP.

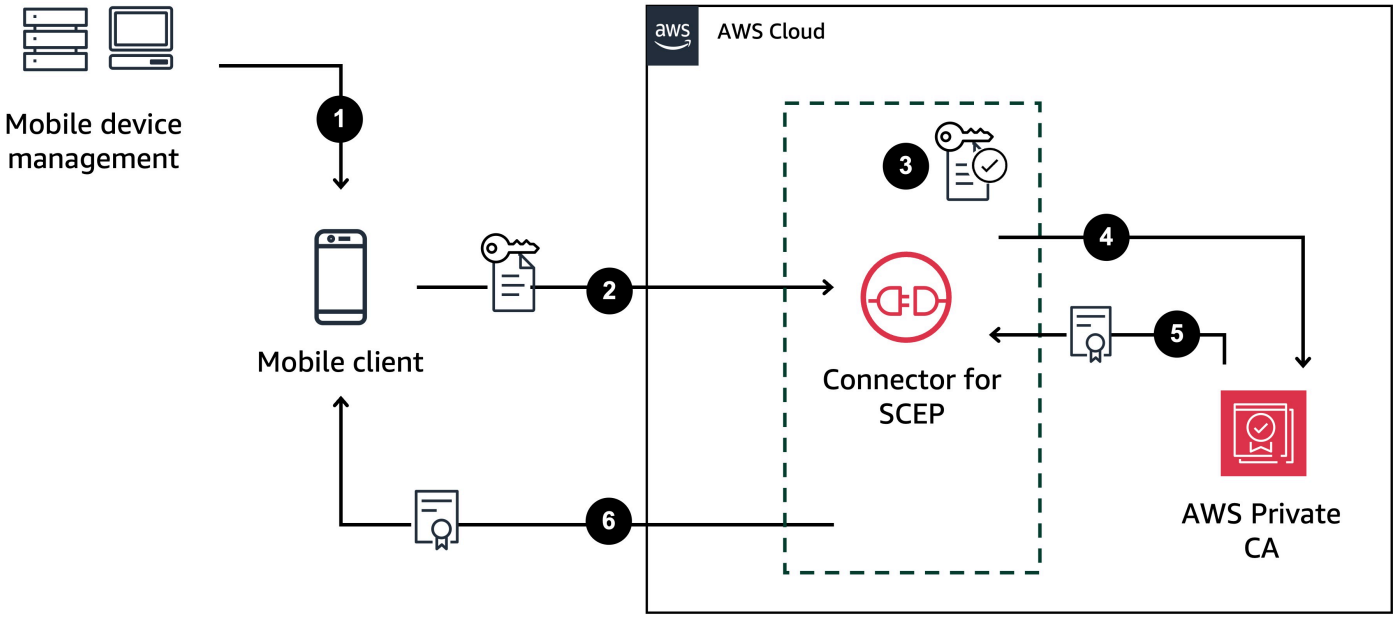
Configure your MDM system for Connector for SCEP

Simple Certificate Enrollment Protocol (SCEP) is a standard protocol used for certificate enrollment and renewal. Connector for SCEP is a RFC 8894-based SCEP server that automatically issues certificates from AWS Private Certificate Authority to your SCEP clients. When you create a connector, Connector for SCEP provides an HTTPS endpoint for SCEP clients to request certificates from. The clients authenticate using a challenge password that's included as part of their certificate signing request (CSR) to the service. You can use Connector for SCEP with popular mobile device management (MDM) systems, including Microsoft Intune, Omnissa Workspace ONE and Jamf Pro, to enroll mobile devices. It's designed to work with any client or endpoint that supports SCEP.

Connector for SCEP offers two types of connectors—general-purpose and Connector for SCEP for Microsoft Intune. The following sections describe how they work, and how to configure your MDM system to use them.

General-purpose connector

A general-purpose connector is designed to work with mobile device endpoints that support SCEP, except for Microsoft Intune, which has a dedicated connector. With general-purpose connectors, such as Jamf Pro or Omnissa Workspace ONE, you manage the SCEP challenge passwords. The following diagram uses a mobile device management (MDM) system as an example, but the same functionality applies to other SCEP-enabled systems or devices.

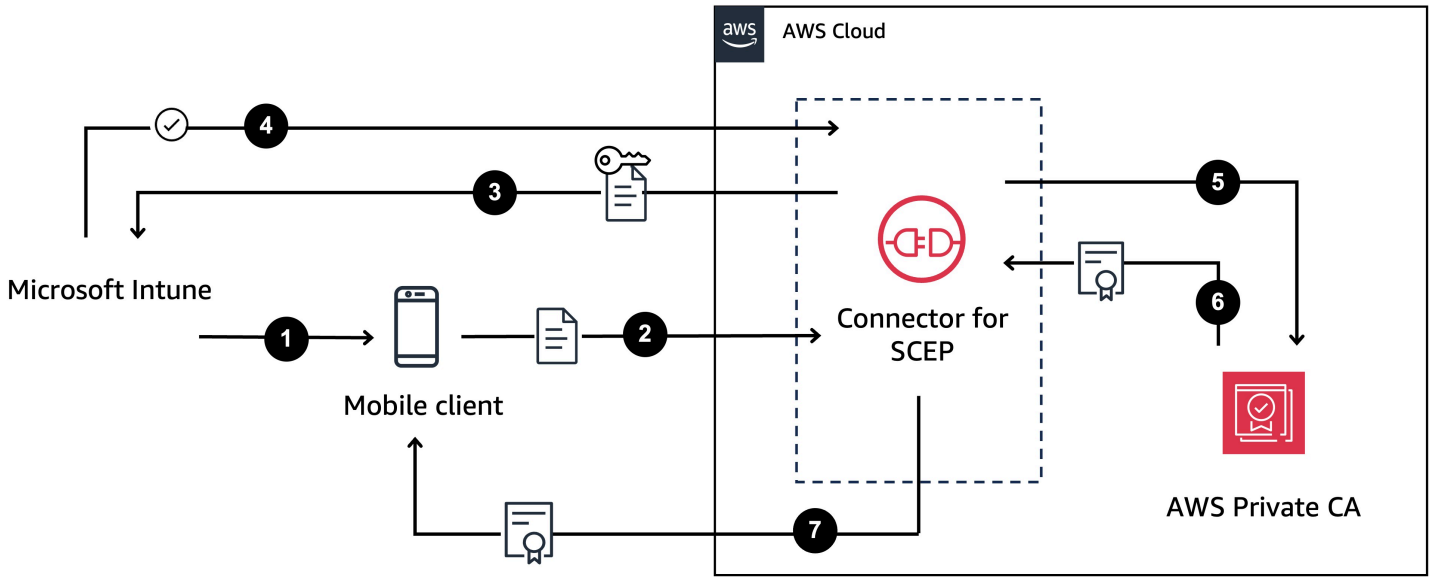


1. The MDM system (or other device or system) sends a SCEP profile to the mobile client. A SCEP profile contains configuration parameters that define the certificate profile, such as certificate validity period, challenge password, and other information relevant to the issuance of certificates.
2. The mobile client requests a certificate and also sends a certificate signing request (CSR) that includes a challenge password.
3. Connector for SCEP validates the challenge password. If it's valid, then the service requests a certificate from AWS Private CA on behalf of the mobile client.
4. AWS Private CA issues the certificate and sends it to Connector for SCEP.
5. Connector for SCEP sends the issued certificate to the mobile client.

AWS Private CA Connector for SCEP for Microsoft Intune

AWS Private CA Connector for SCEP for Microsoft Intune is designed for use with Microsoft Intune. With the Connector for SCEP for Microsoft Intune connector type, you'll use Microsoft Intune to manage your SCEP challenge passwords. For more information about using Connector for SCEP with Microsoft Intune, see Configure Microsoft Intune for Connector for SCEP.

To use Connector for SCEP with Microsoft Intune, you must enable specific functionalities using the Microsoft Intune API, and possess a valid Microsoft Intune license. You should also review the Microsoft Intune® App Protection Policies.



1. Microsoft Intune sends a SCEP profile to the mobile client. The profile contains an encrypted challenge password that the mobile client places into the CSR.
2. The mobile client requests a certificate and sends the CSR to Connector for SCEP.
3. Connector for SCEP sends the CSR to Microsoft Intune for authorization.
4. Microsoft Intune decrypts the challenge password in the CSR. If it's valid, Microsoft Intune sends approval to Connector for SCEP to issue the certificate to the mobile client.
5. Connector for SCEP requests a certificate from AWS Private CA on behalf of the mobile client.
6. AWS Private CA issues the certificate and sends it to Connector for SCEP.
7. Connector for SCEP sends the issued certificate to the mobile client.

Topics

- Configure Jamf Pro for Connector for SCEP
- Configure Microsoft Intune for Connector for SCEP
- Configure Omnissa Workspace ONE for Connector for SCEP

Configure Jamf Pro for Connector for SCEP

You can use AWS Private CA as an external certificate authority (CA) with the Jamf Pro mobile device management (MDM) system. This guide provides instructions on how to configure Jamf Pro after you create a general-purpose connector.

Configure Jamf Pro for Connector for SCEP

This guide provides instructions on how to configure Jamf Pro for use with Connector for SCEP. After you successfully configure Jamf Pro and Connector for SCEP, you'll be able to issue AWS Private CA certificates to your managed devices.

Jamf Pro requirements

Your implementation of Jamf Pro must meet the following requirements.

- You must enable the **Enable certificate-based authentication** setting in Jamf Pro. You can find details on this setting on the Jamf Pro Security Settings page in the Jamf Pro documentation.

Step 1: (Optional - recommended) Obtain your private CA's fingerprint

A fingerprint is a unique identifier for your private CA that can be used to verify the identity of your CA when establishing trust with other systems or applications. Incorporating a certificate authority (CA) fingerprint allows managed devices to authenticate the CA they are connecting to and request certificates solely from the anticipated CA. We recommend using a CA fingerprint with Jamf Pro.

To generate a fingerprint for your private CA

1. Obtain the private CA certificate from either AWS Private CA console or by using the GetCertificateAuthorityCertificate. Save it as `ca.pem` file.
2. Install the OpenSSL Command Line Utilities.
3. In OpenSSL, run the following command to generate the fingerprint:

```
openssl x509 -in ca.pem -sha256 -fingerprint
```

Step 2: Configure AWS Private CA as an external CA in Jamf Pro

After you create a connector for SCEP, you must set AWS Private CA as an external certificate authority (CA) in Jamf Pro. You can set AWS Private CA as a global, external CA. Alternatively, you can use a Jamf Pro configuration profile to issue different certificates from AWS Private CA for different use cases, such as issuing certificates to a subset of devices in your organization. Guidance on implementing Jamf Pro configuration profiles is beyond the scope of this document.

To configure AWS Private CA as an external certificate authority (CA) in Jamf Pro

1. In the Jamf Pro console, go to the **PKI certificates settings** page by going to **Settings** > **Global** > **PKI certificates**.
2. Select the **Management Certificate Template** tab.
3. Select **External CA**.
4. Select **Edit**.
5. (Optional) Select **Enable Jamf Pro as SCEP Proxy for configuration profiles**. You can use Jamf Pro configuration profiles to issue different certificates tailored to specific use-cases. For guidance on how to use configuration profiles in Jamf Pro, see Enabling Jamf Pro as SCEP Proxy for Configuration Profiles in the Jamf Pro documentation.
6. Select **Use a SCEP-enabled external CA for computer and mobile device enrollment**.
7. (Optional) Select **Use Jamf Pro as SCEP Proxy for computer and mobile device enrollment**. If you experience profile installation failures, see Troubleshoot profile installation failures.
8. Copy and paste the Connector for SCEP **public SCEP URL** from the connector's details to the **URL** field in Jamf Pro. To view a connector's details, choose the connector from the Connectors for SCEP list. Alternatively, you can get the URL by calling GetConnector and copy the `Endpoint` value from the response.
9. (Optional) Enter the name of the instance in the **Name** field. For example, you can name it **AWS Private CA**.
10. Select **Static** for the challenge type.
11. Copy a challenge password from your connector, and paste it into the **Challenge** field. A connector can have multiple challenge passwords. To view your connector's challenge passwords, navigate to your connector's details page in the AWS console and select the **View password** button. Alternatively, you can get a connector's challenge password(s) by calling GetChallengePassword and copy a `Password` value from the response. For information about using challenge passwords, see Understand Connector for SCEP considerations and limitations.
12. Paste the challenge password into the **Verify Challenge** field.
13. Choose a **Key Size**. We recommend a key size of 2048 or higher.
14. (Optional) Select **Use as digital signature**. Select this for authentication purposes to grant devices secure access to resources like Wi-Fi and VPN.
15. (Optional) Select **Use for key encipherment**.
16. (Optional - recommended) Enter a hex string in the **Fingerprint** field. We recommend that you add a CA fingerprint to allow managed devices to verify the CA, and only request certificates

from the CA. For instructions on how to generate a fingerprint for your private CA, see Step 1: (Optional - recommended) Obtain your private CA's fingerprint.

17. Select **Save**.

Step 3: Set up a configuration profile signing certificate

To use Jamf Pro with Connector for SCEP, you must provide the signing and CA certificates for the private CA that's associated with your connector. You can do this by uploading a profile signing certificate keystore to Jamf Pro that contains both certificates.

Here are the steps to create a certificate keystore and upload it into Jamf Pro:

- Generate a certificate signing request (CSR) using your internal processes.
- Get the CSR signed by the private CA associated with your connector.
- Create a profile signing certificate keystore that contains both the profile signing and CA certificates.
- Upload the certificate keystore to Jamf Pro.

By following these steps, you can make sure that your devices can validate and authenticate the configuration profile signed by your private CA, enabling the use of Connector for SCEP with Jamf Pro.

1. The following example uses OpenSSL and AWS Certificate Manager, but you can generate a certificate signing request using your preferred method.

   AWS Certificate Manager console

   To create a profile signing certificate using the ACM console

   1. Use ACM to request a private PKI certificate. Include the following:

      - **Type** - Use the same private CA type that's serving as the SCEP certificate authority for your MDM system.
      - In the **Certificate authority details** section, select the **Certificate authority** menu and choose the private CA that serves as the CA for Jamf Pro.
      - **Domain name** - Provide a domain name to be embedded into the certificate. You can use a fully qualified domain name (FQDN), such as `www.example.com`, or a bare or apex domain name such as `example.com` (which excludes `www.`).

2. Use ACM to export the private certificate you created in the preceding step. Choose **Export a file** for the certificate, certificate chain, and encrypted key. Keep the **Passphrase** handy because you'll need it in the next step.
3. In a terminal, run the following command in a folder containing the exported files to write the PKCS#12 bundle into the `output.p12` file encoded by the passphrase you created in the previous step.

```
openssl pkcs12 -export \
  -in "Exported Certificate.txt" \
  -certfile "Certificate Chain.txt" \
  -inkey "Exported Certificate Private Key.txt" \
  -name example \
  -out output.p12 \
  -passin pass:your-passphrase \
  -passout pass:your-passphrase
```

AWS Certificate Manager CLI

To create a profile signing certificate using the ACM CLI

- The following command shows how to create a certificate in ACM, and then export the files as a PKCS#12 bundle.

```
PCA=<Enter your Private CA ARN>

CERTIFICATE=$(aws acm request-certificate \
    --certificate-authority-arn $PCA \
    --domain-name <any valid domain name, such as test.name> \
    | jq -r '.CertificateArn')

while [[ $(aws acm describe-certificate \
  --certificate-arn $CERTIFICATE \
  | jq -r '.Certificate.Status') != "ISSUED" ]] do sleep 1; done

aws acm export-certificate \
  --certificate-arn $CERTIFICATE \
  --passphrase password | jq -r '.Certificate' > Certificate.pem
aws acm export-certificate \
  --certificate-arn $CERTIFICATE \
  --passphrase password | jq -r '.CertificateChain' > CertificateChain.pem
```

```
aws acm export-certificate \
  --certificate-arn $CERTIFICATE \
  --passphrase password | jq -r '.PrivateKey' > PrivateKey.pem

openssl pkcs12 -export \
  -in "Certificate.pem" \
  -certfile "CertificateChain.pem" \
  -inkey "PrivateKey.pem" \
  -name example \
  -out output.p12 \
  -passin pass:passphrase \
  -passout pass:passphrase
```

OpenSSL CLI

To create a profile signing certificate using OpenSSL CLI

1. Using OpenSSL, generate a private key by running the following command.

```
openssl genrsa -out local.key 2048
```

2. Generate a certificate signing request (CSR):

```
openssl req -new -key local.key -sha512 -out local.csr -
subj "/CN=MySigningCertificate/O=MyOrganization" -addext
 keyUsage=critical,digitalSignature,nonRepudiation
```

3. Using the AWS CLI, issue the signing certificate using the CSR you generated in the previous step. Run the following command, and note the certificate ARN in the response.

```
aws acm-pca issue-certificate --certificate-authority-arn <SAME CA AS
 USED ABOVE, SO IT'S TRUSTED> --csr fileb://local.csr --signing-algorithm
 SHA512WITHRSA --validity Value=365,Type=DAYS
```

4. Get the signing certificate by running the following command. Specify the certificate ARN from the previous step.

```
aws acm-pca get-certificate --certificate-authority-arn <SAME CA AS USED
 ABOVE, SO IT'S TRUSTED> --certificate-arn <ARN OF NEW CERTIFICATE> | jq -r
 '.Certificate' >local.crt
```

5. Get the CA certificate by running the following command.

```
aws acm-pca get-certificate-authority-certificate --certificate-authority-
arn <SAME CA AS USED ABOVE, SO IT'S TRUSTED> | jq -r '.Certificate' > ca.crt
```

6. Using OpenSSL, output the signing certificate keystore in p12 format. Use the CRT files that you generated in steps four and five.

```
openssl pkcs12 -export -in local.crt -inkey local.key -certfile ca.crt -name
 "CA Chain" -out local.p12
```

7. When prompted, enter an export password. This password is your keystore password to provide to Jamf Pro.

2. In Jamf Pro, navigate to the **Management Certificate Template** and go to the **External CA** pane.
3. At the bottom of the **External CA** pane, select **Change Signing and CA Certificates**.
4. Follow the onscreen instructions to upload the signing and CA certificates for the external CA.

Step 4: (Optional) Install certificate during user-initiated enrollment

To establish trust between your client devices and your private CA, you must ensure your devices trust the certificates issued by Jamf Pro. You can use Jamf Pro's User-Initiated Enrollment Settings to automatically install your AWS Private CA's CA certificate on the client devices when they request a certificate during the enrolllment process.

Troubleshoot profile installation failures

If you're experiencing profile installation failures after enabling **Use Jamf Pro as SCEP Proxy for computer and mobile device enrollment**, consult your device logs and try the following.

Device log error message	Mitigation
`Profile installation failed. Unable to obtain certificate from SCEP server at "<your-jamf-endpoint>.jamfcloud.com". <MDM-SCEP:15001>`	If you receive this error message while trying to enroll, retry the enrollment. It can take several tries before enrollment succeeds.

Device log error message	Mitigation
`Profile installation failed. Unable to obtain certificate from SCEP server at "<your-jamf-endpoint>.jamfcloud.com". <MDM-SCEP:14006>`	Your challenge password might be misconfigured. Verify that the challenge password in Jamf Pro matches your connector's challenge password.

Configure Microsoft Intune for Connector for SCEP

You can use AWS Private CA as an external certificate authority (CA) with the Microsoft Intune mobile device management (MDM) system. This guide provides instructions on how to configure Microsoft Intune after you create a Connector for SCEP for Microsoft Intune.

Prerequisites

Before you create a Connector for SCEP for Microsoft Intune, you must complete the following prerequisites.

- Create an Entra ID.
- Create a Microsoft Intune Tenant.
- Create an App Registration in your Microsoft Entra ID. See Update an app's requested permissions in Microsoft Entra ID in the Microsoft Entra documentation for information about how to manage application-level permissions for your App Registration. The App Registration must have the following permissions:
  - Under **Intune** set **scep_challenge_provider**.
  - For **Microsoft Graph** set **Application.Read.All** and **User.Read**.
- You must grant the application in your App Registration admin consent. For information, see Grant tenant-wide admin consent to an application in the Microsoft Entra documentation.

> **Tip**
>
> When you create the App Registration, take note of the **Application (client) ID** and **Directory (tenant) ID or primary domain**. When you create your Connector for SCEP for Microsoft Intune, you'll enter these values. For information about how to get these

values, see Create a Microsoft Entra application and service principal that can access resources in the Microsoft Entra documentation.

Step 1: Grant AWS Private CA permission to use your Microsoft Entra ID Application

After you create a Connector for SCEP for Microsoft Intune, you must create a federated credential under the Microsoft App Registration so that Connector for SCEP can communicate with Microsoft Intune.

To configure AWS Private CA as an external CA in Microsoft Intune

1. In the Microsoft Entra ID console, navigate to the **App registrations**.
2. Choose the application that you created to use with Connector for SCEP. The application (client) ID of the application you click must match the ID you specified when you created the connector.
3. Select **Certificates & secrets** from the **Managed** drop-down menu.
4. Select the **Federated credentials** tab.
5. Select **Add a credential**.
6. From the **Federated credential scenario** drop down menu, choose **Other issuer**.
7. Copy and paste the **OpenID issuer** value from your Connector for SCEP for Microsoft Intune details into the **Issuer** field. To view a connector's details, choose the connector from the Connectors for SCEP list in the AWS console. Alternatively, you can get the URL by calling GetConnector and then copy the `Issuer` value from the response.
8. For **Type**, select **Explicit subject identifier**.
9. Copy and paste **OpenID subject** value from your connector into the **Value** field. You can view the OpenID issuer value in the connector details page in the AWS console. Alternatively, you can get the URL by calling GetConnector and then copy the `Audience` value from the response.
10. (Optional) Enter the name of the instance in the **Name** field. For example, you can name it **AWS Private CA**.
11. (Optional) Enter a description into the **Description** field.
12. Copy and paste the **OpenID Audience** value from your Connector for SCEP for Microsoft Intune details into the **Audience** field. To view a connector's details, choose the connector

from the Connectors for SCEP list in the AWS console. Alternatively, you can get the URL by calling GetConnector and then copy the `Subject` value from the response.

13. Select **Add**.

Step 2: Set up a Microsoft Intune configuration profile

After you give AWS Private CA the permission to call Microsoft Intune, you must use Microsoft Intune to create a Microsoft Intune configuration profile that instructs devices to reach out to Connector for SCEP for certificate issuance.

1. Create a trusted certificate configuration profile. You must upload the root CA certificate of the chain that you're using with Connector for SCEP into Microsoft Intune to establish trust. For information on how to create a trusted certificate configuration profile, see Trusted root certificate profiles for Microsoft Intune in the Microsoft Intune documentation.
2. Create a SCEP certificate configuration profile that points your devices to the connector when they require a new certificate. The configuration profile's **Profile type** should be **SCEP Certificate**. For the configuration profile's root certificate, make sure that you use the trusted certificate that you created in the previous step.

   For **SCEP Server URLs**, copy and paste the **Public SCEP URL** from your connector's details into the **SCEP Server URLs** field. To view a connector's details, choose the connector from the Connectors for SCEP list. Alternatively, you can get the URL by calling ListConnectors, and then copy the `Endpoint` value from the response. For guidance on creating configuration profiles in Microsoft Intune, see Create and assign SCEP certificate profiles in Microsoft Intune in the Microsoft Intune documentation.

   > **Note**
   >
   > For non-mac OS and iOS devices, if you don't set a validity period in the configuration profile, Connector for SCEP issues a certificate with a validity of one year. If you don't set an Extended Key Usage (EKU) value in the configuration profile, Connector for SCEP issues a certificate with the EKU set with `Client Authentication (Object Identifier: 1.3.6.1.5.5.7.3.2)`. For macOS or iOS devices, Microsoft Intune doesn't respect `ExtendedKeyUsage` or `Validity` parameters in your configuration profiles. For these devices, Connector for SCEP issues a certificate with a one-year validity period to these devices through client authentication.

Step 3: Verify the connection to Connector for SCEP

After you've created a Microsoft Intune configuration profile that points to the Connector for SCEP endpoint, confirm that an enrolled device can request a certificate. To confirm, make sure that there aren't any policy assignment failures. To confirm, in the Intune portal navigate to **Devices** > **Manage Devices** > **Configuration** and verify that there's nothing listed under **Configuration Policy Assignment Failures**. If there is, confirm your set up with the information from the preceding procedures. If your set up is correct and there still are failures, then consult Collect available data from mobile device.

For information about device enrollment, see What is device enrollment? in the Microsoft Intune documentation.

Configure Omnissa Workspace ONE for Connector for SCEP

You can use AWS Private CA as an external certificate authority (CA) with the Omnissa Workspace ONE UEM (Unified Endpoint Management) system. This guide provides instructions on how to configure Omnissa Workspace ONE after you create a SCEP connector in AWS.

Prerequisites

Before you create a SCEP connector for Omnissa Workspace ONE, you must complete the following prerequisites:

- Create a private CA in the AWS console. For more information, see Create a private CA in AWS Private CA.
- Create a general purpose SCEP connector. For more information, see Create a connector.
- Have an active Omnissa Workspace ONE environment admin account with an Organization Group ID.
- If you are enrolling an Apple device, configure the Apple Push Notification Service (APNs) for MDM. For more information, see APNs Certificates in the Omnissa documentation.

Step 1: Define a certificate authority and template in Omnissa Workspace ONE

After creating a private CA and SCEP connector in the AWS console, define the certificate authority and template in Omnissa Workspace ONE.

Add AWS Private CA as a certificate authority

1. From the **System** menu, choose **Enterprise Integration** and then choose **Certificate Authorities**.
2. Choose **+ ADD** and provide the following information:
   - **Name**: AWS-Private-CA.
   - **Description**: AWS Private CA for device certificate issuance.
   - **Authority Type**: Select **Generic SCEP**.
   - **SCEP URL**: Enter the SCEP URL from AWS Private CA.
   - **Challenge Type**: Select **STATIC**.
   - **Static Challenge**: Enter the SCEP static challenge password from the Connector for SCEP configuration in the AWS console.
   - Enter the **Retry Timeout** and **Max Retries** values.
3. Save the configuration.

Create a certificate template

1. From the **System** menu, choose **Enterprise Integration**, choose **Certificate Authorities**, and then choose **Templates**.
2. Choose **Add Templates** and provide the following information:
   - **Template Name**: Device-Cert-Template.
   - **Certificate Authority**: Choose **AWS-Private-CA**.
   - **Subject Name**: This is a customizable field. You can choose variable values from a list of attributes. For example, CN={DeviceReportedName}, O={DevicePlatform}, OU={CustomAttribute1}
   - **Private Key Length**: 2048 bits.
   - **Private Key Type**: Select **Signing** and **Encryption** as required
   - **Automatic Renewal**: Enabled/Disabled (Based on your needs).
3. Save the template.

Step 2: Set up an Omnissa Workspace ONE UEM profile configuration

Create a profile in Omnissa Workspace ONE UEM that directs devices to Connector for SCEP to issue a certificate.

Create a SCEP device profile for certificate distribution

1. From the **Resources** menu, choose **Profiles & Baselines**, and then choose **Profiles**.
2. Choose **Add** then **Add Profile**
3. Select the device platform (**Android**, **iOS**, **macOS**, **Windows**).
4. Set the **Management type** and **Context** as appropriate.
5. Set the **Name**: Device-Cert-Profile.
6. Scroll to **SCEP Payload**.
7. Select **SCEP** and then choose **+Add**.
8. Use the following configuration:
   - **SCEP**:
     - For **Credential Source** select **Defined Certificate Authority** (Default).
     - For **Certificate Authority** select **AWS-Private-CA**
     - For **Certificate Template** select the **Device-Cert-Template** defined in Step 1.
9. Choose **Next** and in the **Assignment** section select the right smart group from the list (assignment group for the device).
10. Select **Assignment type** as **Auto** to enable auto-renewal.
11. Save and publish the profile.

> **Note**
>
> For more information, see SCEP in the Omnissa documentation.

Step 3: Enroll devices in Omnissa Workspace ONE

Create or verify a smart group

1. From **Groups & Settings** choose **Groups** and then choose **Assignment Groups**.

2. Create or edit the POC-Devices smart group:

   - **Name**: POC-Devices.
   - **Device Type**: Select **All** or a specific platform (Android or iOS, for example).
   - **Criteria**: Use **UserGroup**, **Platform and OS**, **OEM and Model** to specify the criteria to group the target devices.
   - **Ownership**: Select **Any** for personal or corporate devices.
3. Save and verify the target devices appear in the **Preview** tab.

Manual device enrollment

Android

- Download the **Workspace ONE Intelligent Hub** app from Google Play.
- Open the app and enter the enrollment URL or scan a QR code.
- Log in and follow the prompts to enroll as an MDM-managed device.

iOS/macOS

- On the device, open **Safari** and navigate to the enrollment URL (https://<WorkspaceONEUEMHostname>/enroll, for example).
- Log in with user credentials.
- Download and install the **Workspace ONE Intelligent Hub** app from the App Store.
- Follow prompts to install the MDM profile in **Settings** > **General** > **VPN & Device Management** > **Profile** > **Install**.

Windows

- Download the **Workspace ONE Intelligent Hub** from the Workspace ONE server or Microsoft Store.
- Enroll via the Hub using the enrollment URL and credentials.

Assign enrolled devices to the POC-Devices Smart Group in **Devices** > **List View** > **More Actions** > **Assign to Smart Group**.

For more information, see Automated Device Enrollment in the Omnissa documentation.

Verify enrollment

1. In the Omnissa Workspace ONE UEM Console, go to **Devices** and then **List View**.

2. Confirm that your enrolled devices appear with the status set to **Enrolled**.
3. Verify devices are in the POC-Devices smart group in the **Groups** tab of the **Device Details**.

Step 4: Issue a certificate

Trigger issuing a certificate

1. In **Devices List View**, select the enrolled device.
2. Choose on the **Query** button to prompt a check-in.
3. The Device-Cert-Profile should issue a certifcate via AWS Private CA.

Verify certificate installation

Android

Choose **Settings**, then **Security**, then **Trusted Credentials**, and then **User** to verify the certificate.

iOS

Go to **Settings**, then choose **General**, then **VPN & Device Management**, and then **Configuration Profile**. Verify that the certificate from AWS-Private-CA is present.

macOS

Open **Keychain Access** and then **System Keychain** and verify the certificate.

Windows

Open **certmgr.msc**, then **Personal**, and then **Certificates** to verify the certificate.

Troubleshooting

SCEP Errors ("22013 - The SCEP server returned an invalid response" for example)

- Verify the SCEP URL and static challenge password in Workspace ONE match AWS Private CA.
- Test SCEP endpoint connectivity: curl <SCEP_URL>.
- Check AWS CloudTrail logs for AWS Private CA errors (`IssueCertificate` failures, for example).

APNs issues (iOS/macOS)

- Make sure the APNs certificate is valid and assigned to the correct Organization Group.
- Test APNs connectivity: telnet gateway.push.apple.com 2195.

Profile installation failures

- Confirm devices are in the correct Smart Group (**Devices**, then **List View**, and then **Groups**).
- Force a profile sync: **More Actions**, then **Send**, and then **Profile List**.

Logs

- Android: Use **Logcat** or Workspace ONE logs.
- iOS/macOS: log show --predicate 'process == "mdmclient"' --last 1h (via Xcode/Apple Configurator).
- Windows: **Event Viewer**, then **Applications and Services Logs** and then **Microsoft-Windows-DeviceManagement**.
- Workspace ONE UEM: **Monitor**, then **Reports & Analytics**, then **Events**, and then **Device Events**.

For detailed Connector for SCEP monitoring in AWS, see **Monitor Connector for SCEP**.

Security considerations

- Store SCEP URLs and secrets securely. For more information, see the AWS Secrets Manager service.
- Restrict smart group criteria to target devices only.
- Regularly renew Apple Push Notifications (APNs) certificates (valid for 1 year).
- Set short certificate validity periods for proof of concept projects to minimize risk.
- For personal devices, make sure cleanup removes all profiles and certificates.

For information about how to configure Omnissa Workspace ONE UEM and CA integration using a SCEP connector, see the SCEP in Omnissa Workspace ONE documentation.

Monitor Connector for SCEP

Monitoring is an important part of maintaining the reliability, availability, and performance of Connector for SCEP and your other AWS solutions. AWS provides the following monitoring tools

to watch Connector for SCEP, report when something is wrong, and take automatic actions when appropriate:

- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS APIs, the source IP address from which the calls were made, and when the calls occurred.

  If you monitor CloudTrail data events, the logs contain the list of all recent requests from client devices. Data events come with identifying client device information such as IP address, the type of operation performed, and the error code and detailed message if the operation results in a `failed` status. For more information, see the AWS CloudTrail User Guide.
- *Amazon EventBridge* is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services and routes that data to targets such as Lambda and CloudWatch Logs. This enables you to monitor events that happen in services, and build event-driven architectures. For more information, see the Amazon EventBridge User Guide.

Topics

- Automate Connector for SCEP using EventBridge
- Log Connector for SCEP API calls using AWS CloudTrail

Automate Connector for SCEP using EventBridge

You can use Amazon EventBridge to automate your AWS services and respond automatically to system events such as application availability issues or resource changes. Events from AWS services are delivered to EventBridge in near-real time. You can write simple rules to indicate which events are of interest to you and the automated actions to take when an event matches a rule. EventBridge are published at least once. For more information, see Creating rules that react to events in EventBridge.

CloudWatch Events are turned into actions using EventBridge. With EventBridge, you can use events to trigger targets. For more information, see What Is Amazon EventBridge?

Connector for SCEP event types

Certificate Issuance Succeeded

Connector for SCEP sends a `Certificate Issuance Succeeded` event to EventBridge when we issue a certificate in response to a `PkiOperationPost` request.

The following is example data for the event.

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Issuance Succeeded",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "success",
       "requestType": "PkiOperationPost",
       "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   }
}
```

Certificate Issuance Failed

Connector for SCEP sends a `Certificate Issuance Failed` event to EventBridge when we are unable to issue a certificate in response to a `PkiOperationPost` request.

The following is example data for the event.

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Issuance Failed",
   "source": "aws.pca-connector-scep",
```

```
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "failure",
       "requestType": "PkiOperationPost",
       "reason": "The certificate authority is not active."
   }
}
```

Certificate Authority Certificate Retrieval Succeeded

Connector for SCEP sends a `Certificate Authority Certificate Retrieval Succeeded` event to EventBridge when we receive a `GetCACert` request and successfully retrieve the connector's private CA certificate.

The following is example data for the event.

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Authority Certificate Retrieval Succeeded",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "success",
       "requestType": "GetCACert"
   }
}
```

Certificate Authority Certificate Retrieval Failed

Connector for SCEP sends a `Certificate Authority Certificate Retrieval Failed` event to EventBridge when we receive a `GetCACert` request and aren't able to retrieve the connector's private CA certificate. The event includes the reason for the failure.

The following is example data for the event.

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Authority Certificate Retrieval Failed",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "failure",
       "requestType": "GetCACert",
       "reason": "The certificate authority certificate validity must be at least one
 year from today."
   }
}
```

Certificate Authority Certificate Retrieval Succeeded

Connector for SCEP sends a `Certificate Authority Certificate Retrieval Succeeded` event to EventBridge when we receive a `GetCACert` request and successfully retrieve the connector's private CA certificate.

The following is example data for the event.

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Certificate Authority Certificate Retrieval Succeeded",
   "source": "aws.pca-connector-scep",
```

```
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {
       "result": "success",
       "requestType": "GetCACert"
   }
}
```

Certificate Authority Capabilities Retrieval Succeeded

Connector for SCEP sends a `Certificate Authority Capabilities Retrieval Succeeded` event to EventBridge when we receive a SCEP `GetCACaps` request and successfully retrieve the CA's capabilities.

The following is example data for the event.

Certificate Authority Capabilities Retrieval Failed

Connector for SCEP sends a `Certificate Authority Capabilities Retrieval Failed` event to EventBridge when we receive a SCEP `GetCACaps` request and can't retrieve the CA's capabilities. We include the reason for failure in the event.

The following is example data for the event.

```
{
 "resources":
     [
     "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
     "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector11223344-1234-1122-2233-112233445566"
     ],
 "detailType":"Certificate Authority Capabilities Retrieval Failed",
 "detail": {
```

```
    "result":"failure",
    "requestType":"GetCACaps",
    "reason":"The request was denied due to request throttling."
},
"source":"aws.pca-connector-scep","accountId":"111122223333"
}
```

Unsupported Operation Invoked

Unsupported Operation Invoked

Connector for SCEP sends an `Unsupported Operation Invoked` event to EventBridge if the operation sent to the connector endpoint is unsupported or unknown.

```
{
   "version": "0",
   "id": "event_ID",
   "detail-type": "Unsupported Operation Invoked",
   "source": "aws.pca-connector-scep",
   "account": "account",
   "time": "2024-09-12T19:14:56Z",
   "region": "region",
   "resources":[
       "arn:aws:acm-pca:us-east-1:111122223333:certificate-
authority/11223344-1234-1122-2233-112233445566",
       "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/11223344-1234-1122-2233-112233445566"
   ],
   "detail": {}
}
```

Create an EventBridge rule

In EventBridge, you can create rules that responds to events recorded by CloudTrail. To create a rule that includes all events logged by Connector for SCEP, set the source to `aws.pca-connector-scep`. For more information about rules, see Create a rule in Amazon EventBridge.

Log Connector for SCEP API calls using AWS CloudTrail

Connector for Simple Certificate Enrollment Protocol (SCEP) is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, client, or an AWS service. CloudTrail captures all API calls for Connector for SCEP as events. The calls captured include calls from the

Connector for SCEP console and code calls to the Connector for SCEP API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for Connector for SCEP. 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 Connector for SCEP, 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.

Connector for SCEP information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in Connector for SCEP, 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 Connector for SCEP, 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

All Connector for SCEP actions are logged by CloudTrail and are documented in the Connector for SCEP API reference. For example, calls to the `CreateConnector`, `GetConnector` and `CreateChallenge` actions generate entries in the CloudTrail log files.

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.
- Whether the request was made by a SCEP client device.

For more information, see the CloudTrail userIdentity element.

Connector for SCEP management events

Connector for SCEP integrates with CloudTrail to record API actions made by a user, a role, or an AWS service in Connector for SCEP. You can use CloudTrail to monitor Connector for SCEP API requests in real time and store logs in Amazon Simple Storage Service, Amazon CloudWatch Logs, and Amazon CloudWatch Events. Connector for SCEP supports logging the following actions as events in CloudTrail log files:

- CreateChallenge
- CreateConnector
- GetConnector
- GetChallengeMetadata
- GetChallengePassword
- DeleteConnector
- DeleteChallenge

Connector for SCEP data events in CloudTrail

Data events provide information about the resource operations performed on or in a resource for example, when your client sends a SCEP `GetCACaps` message to a connector endpoint. These are also known as data plane operations. Data events are often high-volume activities. By default, CloudTrail doesn't log any data events, and the CloudTrail **Event history** doesn't record them.

Additional charges apply for data events. For more information about CloudTrail pricing, see AWS CloudTrail Pricing.

You can log data events for the `AWS::PCAConnectorSCEP::Connector` resource type by using the CloudTrail console, AWS CLI, or CloudTrail API operations. For more information about how to log data events, see Logging data events with the AWS Management Console and Logging data events with the AWS Command Line Interface in the *AWS CloudTrail User Guide*.

The following table lists the Connector for SCEP resource type for which you can log data events. The **Data event type (console)** column shows the value to choose from the **Data event type** list on the CloudTrail console. The **resources.type value** column shows the `resources.type` value, which you would specify when configuring advanced event selectors using the AWS CLI or CloudTrail APIs. The **Data APIs logged to CloudTrail** column shows the API calls logged to CloudTrail for the resource type.

Data event type (console)	resources.type value	Data APIs logged to CloudTrail
Connector	`AWS::PCAConnectorSCEP::Connector`	• `PKIOperationGet` - Generated if an HTTP GET SCEP request containing a PKCSReq message is made to the dataplane endpoint of a connector, and the operation of that message is set to `PKIOperation` . • `PKIOperationPost` - Generated if an HTTP POST SCEP request containing a PKCSReq message is made to the dataplane endpoint of a connector, and the operation of that message is set to `PKIOperation` . • `GetCACaps` - Generated if a SCEP request containing a `GetCACaps` message is made to the dataplane endpoint of a connector. • `GetCACert` - Generated if a SCEP request containing a `GetCACert` message is made to the dataplane endpoint of a connector.

You can configure advanced event selectors to filter on the `eventName`, `readOnly`, and `resources.ARN` fields to log only those events that are important to you. The following example is the JSON view of a data event configuration that logs events for a specific function only. For more information about these fields, see AdvancedFieldSelector in the *AWS CloudTrail API Reference*.

```
[
  {
    "name": "connector-scep-events",
    "fieldSelectors": [
      {
        "field": "eventCategory",
        "equals": [
          "Data"
        ]
      },
      {
        "field": "resources.type",
        "equals": [
          "AWS::PCAConnectorSCEP::Connector"
        ]
      },
      {
        "field": "resources.ARN",
        "equals": [
          "arn:aws:pca-connector-scep:US West (N.
 California):111122223333:connector/11223344-1122-2233-3344-cae95a00d2a7"
        ]
      }
    ]
  }
]
```

Example 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.

Example 1: Management event, `CreateConnector`

The following example shows a CloudTrail log entry that demonstrates the `CreateConnector` action.


```
{
        "eventVersion": "1.09",
        "userIdentity": {
          "type": "AssumedRole",
          "principalId": "AABB1122CCDD4455HHJJ1:11cc33nn2a97724dc48a89071111111111",
          "arn": "arn:aws:sts::111122223333:assumed-role/Admin",
          "accountId": "111122223333",
          "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "AABB1122CCDD4455HHJJ1",
                  "arn": "arn:aws:iam::111122223333:role/Admin",
                  "accountId": "111122223333",
                  "userName": "my-user-name"
              },
              "attributes": {
                  "creationDate": "2024-08-16T17:46:41Z",
                  "mfaAuthenticated": "false"
              }
          }
        },
        "eventTime": "2024-08-16T17:48:07Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "CreateConnector",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "requestParameters": {
          "ClientToken": "11223344-2222-3333-4444-666555444555",
          "CertificateAuthorityArn": "arn:aws:acm-pca:us-
east-1:111122223333:certificate-authority/a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
        },
        "responseElements": {
          "ConnectorArn": "arn:aws:pca-connector-scep:us-east-1:111122223333:connector/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        },
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
```

```
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
        }
```

Example 2: Management event, `CreateChallenge`

The following example shows a CloudTrail log entry that demonstrates the `CreateChallenge` action.

```
{
        "eventVersion": "1.09",
        "userIdentity": {
          "type": "AssumedRole",
          "principalId": "AABB1122CCDD4455HHJJ1:11cc33nn2a97724dc48a89071111111111",
          "arn": "arn:aws:sts::111122223333:assumed-role/Admin",
          "accountId": "111122223333",
          "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "AABB1122CCDD4455HHJJ1",
                  "arn": "arn:aws:iam::111122223333:role/Admin",
                  "accountId": "111122223333",
                  "userName": "user-name"
              },
              "attributes": {
                  "creationDate": "2024-08-16T17:46:41Z",
                  "mfaAuthenticated": "false"
              }
          }
        },
        "eventTime": "2024-08-16T17:47:52Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "CreateChallenge",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "requestParameters": {
          "ConnectorArn": "arn:aws:pca-connector-scep:us-east-1:111122223333:connector/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
          "ClientToken": "11223344-2222-3333-4444-666555444555"
```

```
        },
        "responseElements": {
          "Challenge": {
              "Arn": "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/9cac40bc-acba-412e-9a24-f255ef2fe79a/a1b2c3d4-5678-90ab-
cdef-EXAMPLE22222",
              "ConnectorArn": "arn:aws:pca-connector-scep:us-
east-1:111122223333:connector/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
              "CreatedAt": 1723830472.942,
              "Password": "***",
              "UpdatedAt": 1723830472.942
          }
        },
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
        }
```

Example 3: Management event, `GetChallengePassword`

The following example shows a CloudTrail log entry that demonstrates the `GetChallengePassword` action.

```
{
        "eventVersion": "1.09",
        "userIdentity": {
          "type": "AssumedRole",
          "principalId": "AABB1122CCDD4455HHJJ1:11cc33nn2a97724dc48a89071111111111",
          "arn": "arn:aws:sts::111122223333:assumed-role/Admin",
          "accountId": "111122223333",
          "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
          "sessionContext": {
              "sessionIssuer": {
                  "type": "Role",
                  "principalId": "AABB1122CCDD4455HHJJ1",
                  "arn": "arn:aws:iam::111122223333:role/Admin",
                  "accountId": "905418114790",
                  "userName": "111122223333"
              },
```

```
            "attributes": {
                "creationDate": "2024-08-16T17:55:01Z",
                "mfaAuthenticated": "false"
            }
          },
          "invokedBy": "signin.amazonaws.com"
        },
        "eventTime": "2024-08-16T17:55:54Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "GetChallengePassword",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "requestParameters": {
          "ChallengeArn": "arn:aws:pca-connector-scep:us-east-1:111122223333:challenge/
a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
        },
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": true,
        "eventType": "AwsApiCall",
        "managementEvent": true,
        "recipientAccountId": "111122223333",
        "eventCategory": "Management"
        }
```


Example 4: Data event, `PkiOperationPost`

The following example shows a CloudTrail log entry that demonstrates a failed `PkiOperationPost` call. The log includes an error code and error message with an explanation of the failure.


```
{
        "eventVersion": "1.10",
        "userIdentity": {
          "type": "FederatedUser",
          "principalId": "111122223333",
          "accountId": "111122223333"
        },
        "eventTime": "2024-08-16T17:40:09Z",
        "eventSource": "pca-connector-scep.amazonaws.com",
        "eventName": "PkiOperationPost",
```

```
        "awsRegion": "us-east-1",
        "sourceIPAddress": "10.0.0.0",
        "userAgent": "Python/3.11.8 Darwin/22.6.0 exe/x86_64",
        "errorCode": "BadRequestException",
        "errorMessage": "The certificate authority is not in a valid state for issuing
 certificates (Service: AcmPca, Status Code: 400, Request ID: a1b2c3d4-5678-90ab-cdef-
EXAMPLE55555)",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "readOnly": false,
        "resources": [
          {
              "accountId": "111122223333",
              "type": "AWS::PCAConnectorSCEP::Connector",
              "ARN": "arn:aws:pca-connector-scep:us-east-1:111122223333:connector/
a1b2c3d4-5678-90ab-cdef-EXAMPLE33333"
          }
        ],
        "eventType": "AwsApiCall",
        "managementEvent": false,
        "recipientAccountId": "905418114790",
        "eventCategory": "Data",
        "tlsDetails": {
          "clientProvidedHostHeader": "111122223333-a1b2c3d4-5678-90ab-cdef-
EXAMPLE33333.enroll.pca-connector-scep.us-east-1.api.aws"
        }
        }
```


Troubleshoot AWS Private Certificate Authority Connector for SCEP issues

You might need to troubleshoot issues related to your Connector for SCEP implementation. This chapter provides detailed information about the HTTP and client errors sent by the service.

Topics

- Troubleshoot HTTP errors from Connector for SCEP
- Troubleshoot Connector for SCEP client errors

Troubleshoot HTTP errors from Connector for SCEP

When your client triggers a Connector for SCEP dataplane API action and it results in an error, Connector for SCEP sends a HTTP response code to the requesting client with information about the error.

In addition to the service responses provided directly to your clients, you can use the monitoring tools described in the Monitor Connector for SCEP section to view and debug errors resulting in an HTTP error.

The following are error messages returned by the service to SCEP clients, the potential causes, and the steps you can take to resolve the issues.

HTTP 400 Bad Request

An HTTP 400 response code means that Connector for SCEP can't process the request due to an apparent client error, such as missing or invalid data in the request. If the error results from a SCEP-protocol specific error, Connector for SCEP includes the SCEP response as a binary in the message. Connector for SCEP APIs can return 400 responses for any of the following reasons.

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
LimitExceededException	Certificate authority issuance limit exceeded.	The private certificate authority (CA) associated with the connector has exceeded its quota for the number of certificates it can issue.	A SCEP connector can only be connected to one private CA through its lifetime. If you have exhausted the limits of your private CA, either create a new connector or request a quota increase. For more informati	No

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
			on about private CA quotas, see AWS Private Certificate Authority quotas.	
ValidationException	The request must contain base64.	Connector for SCEP can't process the HTTP GET request because the body isn't valid Base64.	If possible, configure your clients to use HTTP POST messages instead of HTTP GET messages. If you must use HTTP GET, the messages must use the Base64 format. If your clients are incompatible with these requirements, contact AWS Support for assistance.	No
ValidationException	The certificate authority is not active.	The private CA associated with the connector is inactive.	Reactivate the private CA. For information, see Update a private CA in AWS Private Certificate Authority.	No

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
ValidationException	The certificate authority certificate validity must be at least one year from today.	The private CA associated with the general-purpose connector must have a validity period of one year from today.	Reissue the certificate with a validity period greater than one year from today. For information about managing certificates, see Manage the private CA lifecycle .	No
ValidationException	The certificate included in the request is expired.	The transient certificate generated by the client device on each transaction was expired on reception by the service.	It's most likely that your client devices don't have their time settings properly configured, and they're creating certificates with dates behind the real time. If you can't resolve this issue, contact AWS Support for assistance.	No

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
ValidationException	The request contains invalid Cryptographic Message Syntax.	The service was unable to decode the SCEP request message.	Check if your SCEP messages conform to the Cryptographic Message Syntax defined in SCEP RFC 8894. If you can't resolve this issue, contact AWS Support for assistance.	No
ValidationException	The connector is not active.	The connector's status is not **active**.	You can find a connector's status in the console or in the Status field in the API. A connector's status can be **creating**, **active**, **deleting**, or **failed**. If the status is **creating**, try your request later. If the status is **failed**, view the status reason to troubleshoot the issue, and then create a new connector.	No

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
ValidationException	There must be a valid certificate included in the request.	The transient certificate included in the request message from the client was either missing or invalid.	SCEP-compatible clients must provide a self-signed certificate to authenticate themselves. If your client is unable to provide the required self-signed certificate, contact AWS Support for assistance.	No
ValidationException	The request URI is invalid.	Connector for SCEP can't parse the request because the URI path or query of the request are invalid.	Administrators should verify the configuration settings of the client devices, which are typically managed through a Mobile Device Management (MDM) system. For more information, see Step 2: Copy connector details into your MDM system.	No

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
ValidationException	Exactly one host header is required in the request.	The client did not provide a valid HTTP Host header in the request, which is required for the request to be processed.	The HTTP host header is required to distinguish requests coming to different connectors. If your client is unable to provide the required HTTP host header, contact AWS Support for assistance.	No
ValidationException	The request could not be decoded. Please send a valid SCEP request.	The service couldn't decode and process the Cryptographic Message Syntax (CMS) request that your client sent.	If your clients are having trouble with our implementation of SCEP, note the request ID (`x-amzn-requestid`) from the response and contact AWS Support.	No

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
ValidationException	The response could not be encoded with values derived from the request. Please send a valid SCEP request.	The service wasn't able to encode the SCEP response.	This issue usually occurs when the service is unable to use the provided requestor certificate to properly encode the SCEP response message. This can happen, for example, if the requestor certificate has an Elliptic Curve Digital Signature Algorithm (ECDSA) key, which Connector for SCEP doesn't support. If you encounter this problem, first configure your MDM or SCEP client to use RSA. If you still can't resolve the issue, note the request ID (`x-amzn-requestid`) from the response and contact AWS Support for assistance.	No

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
ValidationException	Unsupported algorithm: <OID>	The request was either signed or encrypted by an unsupported cryptographic algorithm.	Our service doesn't support certain outdated and weak cryptographic algorithms. This information is communicated to clients through the `GetCACaps` request. However, some clients may not use this method to check the supported algorithms. If your clients appear to be incompatible with the cryptographic algorithms supported by our service, contact AWS Support for assistance.	No

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
ValidationException	Unsupported PkiOperation messageType.	The request message contained an invalid `PkiOperation` message type and could not be processed by the service.	Our service supports only a subset of the SCEP protocol message types defined in RFC 8894. Specifically, we recognize and process the following message types: CertRep, PKCSReq, GetCert, GetCRL, and CertPoll. We communicate the supported message types to clients through the GetCACaps method. Unfortunately, some clients may not be utilizing this method and could be non-compliant with our service's capabilities. If your clients appear to be incompatible with the SCEP message types supported by our	No

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
			service, contact AWS Support.	
BadRequestException	The challenge password is invalid.	The challenge password provided by the client was invalid for the contacted service endpoint and its associated connector. The challenge password is a required security measure defined in the SCEP protocol to ensure only authorized clients can access the service.	Make sure that your client is providing the correct challenge password in its request. You can find in the connector details in the console or through the GetChallengePassword API. For more information, see Step 2: Copy connector details into your MDM system.	Yes

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
BadRequestException	Exactly one challenge password is required in the certificate signing request.	The client provided either zero or multiple challenge passwords in its request.	Make sure that your client is providing one challenge password in its request. You can find challenge passwords in the connector's details in the console or through the GetChallengePassword API. For more information, see Step 2: Copy connector details into your MDM system.	Yes
BadRequestException	The connector does not have access to Azure.	Connector for Microsoft Intune authorizes client requests through Microsoft Intune. This requires that you grant permission for Connector for SCEP to access your Azure resources.	Configure the permissions detailed in Step 1: Grant AWS Private CA permission to use your Microsoft Entra ID Application.	Yes

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
BadRequestException	The Azure application does not have access to perform <action>.	Connector for Microsoft Intune authorizes client requests through Microsoft Intune. This requires that you grant permission for Connector for SCEP to access your Azure resources.	Configure the permissions detailed in Step 1: Grant AWS Private CA permission to use your Microsoft Entra ID Application.	Yes
BadRequestException	The Azure application was not found.	Connector for Microsoft Intune authorizes client requests through Microsoft Intune. This error indicates that you don't have an App Registration in your Microsoft Entra ID, or your connector 's Intune details are misconfigured.	Follow the guidance in the Configure Microsoft Intune for Connector for SCEP topic.	Yes

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
BadRequestException	Intune certificate signing request validation failed. Reason: <reason>	Connector for Microsoft Intune authorizes client requests through Microsoft Intune. This error message indicates that the Intune validation process has failed, and the corresponding Intune error code is provided.	Follow the guidance in the Configure Microsoft Intune for Connector for SCEP topic. If your problem persists, contact Microsoft Support.	Yes

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
BadRequestException	Unsupported PkiOperation messageType: <message type>.	The request message contained an invalid message type and could not be processed by the service.	Our service supports only a subset of the SCEP protocol message types defined in RFC 8894. Specifically, we recognize and process the following message types: CertRep, PKCSReq, GetCert, GetCRL, and CertPoll. We communicate the supported message types to clients through the GetCACaps method. Unfortunately, some clients may not be utilizing this method and could be non-compliant with our service's capabilities. If your clients appear to be incompatible with the SCEP message types supported by our	Yes

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
			service, contact AWS Support.	
BadRequestException	Key algorithm or length is not supported.	The service does not support the provided public key included in the certificate signing request.	Our service only supports standard RSA keys up to 16,384 bits, and ECDSA keys up to 521 bits. If your clients require the use of a currently unsupported algorithm, please contact AWS Support for assistance.	Yes

HTTP 401 Unauthorized

A 401 Unauthorized response status code indicates that the client request hasn't been completed because it lacks valid authentication credentials for the requested resource.

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
AccessDeniedException	The connector does not have access to the certificate authority.	Connector for SCEP doesn't have access to the connector's associated private CA.	Share your private CA with the Connector for SCEP using AWS	No

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
			Resource Access Manager.	
AccountDoesNotExistException	The AWS account does not exist.	The Connector for SCEP resource no longer exists.	The account owning the target resource has been deleted. If this was done by mistake, contact AWS Support within the 90-day post-closure period.	No

HTTP 404 Not Found

An HTTP 404 response code usually means that the resource you were looking for couldn't be found.

Response header (x-amzn-ErrorType	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
ResourceNotFoundException	The certificate authority does not exist.	The connector's associated private CA has been deleted.	There is a grace period during which a private Certificate Authority (CA) can be restored if it has been deleted by mistake. For more informati	No

Response header (x-amzn-ErrorType	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
			on, see Restore a private CA.	
ResourceNotFoundException	A connector with endpoint <URL> doesn't exist.	The client device has attempted to connect to a URL that doesn't belong to any existing connectors.	Make sure that your client is providing the correct endpoint for the connector. To view a connector's `Endpoint`, call the GetConnector API or view it in the connector's details page in the console.	No

HTTP 409 Conflict

An HTTP 409 Conflict response signals that a private CA associated with a connector has changed since the request was initiated.

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
ConflictException	The connector has changed since the request was initiated.	The private CA associated with the connector has been updated, triggering a rotation of the	Try your request again in a few minutes. If the problem doesn't resolve, contact AWS	No

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
		connector's internal certificate used for communication with client devices via SCEP. This certificate rotation may result in temporary issues during the update period, as the new certificate is being deployed. However, this error should be resolved automatically in a timely manner.	Support for assistance.	

HTTP 429 Too Many Requests

Connector for SCEP has account-level quotas, per Region. If you exceed the limit of requests to a connector, your requests will be denied with an HTTP 429 error. If you need to increase your quota, see AWS Private Certificate Authority endpoints and quotas.

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?

Response header (x-amzn-ErrorType)	Error message (x-amzn-ErrorMessage)	Root cause	Remediation	Includes SCEP response?
ThrottlingException	The request was denied due to request throttling.	Too many requests have been issued to this Connector, triggering some requests to be denied. This certificate rotation may result in temporary issues during the update period, as the new certificate is being deployed. However, this error should be resolved automatically in a timely manner.	If you exceed the limit of requests to a connector, your requests will be denied. If you need to increase your quota, see Connector for SCEP endpoints and quotas.	No

Troubleshoot Connector for SCEP client errors

Use the following guidance to troubleshoot client errors related to Connector for SCEP.

Message example	Root cause	Solution
ECDSA keys are not supported	The connector is connected to a private CA that uses an ECDSA key instead of RSA. While this service supports ECDSA keys, not all client	Consider using an RSA-encrypted private CA instead of ECDSA. If you create a private CA that uses RSA, you'll need to also create a new

Message example	Root cause	Solution
	devices may be compatible with this algorithm.	connector. A connector can only be tied to one private CA through its lifespan.
Encryption or signing certificate is not present	According to RFC 8894, a SCEP service returns intermediate CA certificates to the client. These certificates are used by the client to perform encryption and signature validation operations as part of the SCEP protocol. Connector for SCEP uses the same certificate for both encryption and signature validation purposes, which is a common approach. However, some clients may expect to have two separate certificates instead.	If you are unable to use compatible clients, contact AWS Support for assistance.

AWS Private CA service quotas

AWS Private CA assigns quotas to your allowed number of certificates and certificate authorities. Request rates for API actions are also subject to quotas. AWS Private CA quotas are specific to an AWS account and Region.

AWS Private CA throttles API requests at different rates depending on the API operation. Throttling means that AWS Private CA rejects an otherwise valid request because the request exceeds the operation's quota for the number of requests per second. When a request is throttled, AWS Private CA returns a ThrottlingException error. AWS Private CA does not guarantee a minimum request rate for APIs.

To see what quotas can be adjusted, see the AWS Private CA quotas table in the *AWS General Reference*.

You can view your current quotas and request quota increases using AWS Service Quotas.

To see an up-to-date list of your AWS Private CA quotas

1. Log into your AWS account.
2. Open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.
3. In the **Services** list, choose **AWS Certificate Manager Private Certificate Authority (ACM PCA)**. Each quota in the **Service quotas** list shows your currently applied quota value, the default quota value, and whether or not the quota is adjustable. Choose the name of a quota for more information about it.

To request a quota increase

1. In the **Service quotas** list, choose the radio button for an adjustable quota.
2. Choose the **Request quota increase** button.
3. Complete and submit the **Request quota increase** form.

AWS Private CA is integrated with AWS Certificate Manager. You can use the ACM console, AWS CLI, or ACM API to request private certificates from an existing private CA. These private PKI certificates, which are managed by ACM, are subject both to PCA quotas and to the quotas that ACM places on public and imported certificates. For more information about ACM requirements, see Request a Private Certificate and Quotas in the AWS Certificate Manager User Guide.

Document History

The following table describes significant changes to this documentation since January 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 that you send to us. To be notified about significant changes, use the link in the upper right corner to subscribe to the RSS feed.

Change	Description	Date
Documentation update	Updated the Secure Kubernetes with AWS Private Certificate Authority with a new getting started procedure, examples, and monitoring and troublesh ooting topics.	October 1, 2025
Dual-stack support	AWS Private Certificate Authority supports dual-stac k.	June 23, 2025
Child domain support for Connector for AD is now generally available	You can now set up Connector for AD with your child domain.	June 2, 2025
New managed policy: AWSPrivateCAConnec toForKubernetesPol icy	New managed policy introduced for use with AWS Private CA Connector for Kubernetes.	May 19, 2025
Updated AWSPrivat eCAPrivilegedUser and AWSPrivateCAUser managed policies	Replaced StringLike with ArnLike in AWSPrivat eCAUser and AWSPrivat eCAPrivilegedUser . Updated template ARN to include wild cards arn:aws:a cm-pca:::template	January 22, 2025

	to `arn:aws:acm-pca:*:*:template` .	
Connector for SCEP is now generally available	Connector for SCEP is now generally available.	September 16, 2024
New troubleshooting topic	Added a new topic that helps you to troubleshoot issues related to updating your Connector for Active Directory templates.	July 31, 2024
Added how to update Connector for AD templates	Added a procedure describing how to update a Connector for AD template, and how AWS Private CA propagates those updates.	July 31, 2024
Connector for SCEP for Omnissa Workspace ONE is now generally available	Connector for SCEP for Omnissa Workspace ONE is now generally available.	July 23, 2024
Added constraint for audit reports	AWS Private CA doesn't support the use of Amazon S3 Object Lock with buckets used for audit reports.	July 3, 2024
Now supports SM2 for China Region	AWS Private CA now supports the SM2 signing algorithm, for China Region only.	June 27, 2024
AWS Private CA now supports Connector for SCEP (Preview)	Use Connector for SCEP to link AWS Private CA to your SCEP-enabled clients and devices.	June 11, 2024
New connector troubleshooting guidance	Added new sections on troubleshooting connector and SPN creation failures.	April 4, 2024

Adding CDP extension for Matter	Adds support for the Certificate Revocation List Distribution Point (CDP) extension for Matter.	January 25, 2024
AWS Private CA API support for mDL	Added API support for creating certificates that conform to the ISO/IEC standard for mobile driving license (mDL).	January 16, 2024
AWS Private CA Connector for Active Directory	User guide, API, and CLI support for Connector for AD. For more information, see the Connector for AD documentation.	August 24, 2023
Changing security policy names to match new service name	Adoption of new names for AWS managed IAM policies that specify standard permissions on AWS Private CA. For more information, see AWS managed policies.	February 13, 2023
Adding change tracker for AWS managed policies	Documentation added to track changes to AWS managed IAM policies that specify standard permissions on AWS Private CA. For more information, see Updates to AWS managed policies for AWS Private CA.	November 11, 2022

API and CLI support for CAs that issue short-lived certificates	With the introduction of CA usage modes, a CA can be configured to issue either general-purpose or exclusively short-lived certificates. For more information, see Certificate authority modes.	October 24, 2022
Service rebranding and console update	The service is renamed to AWS Private Certificate Authority (AWS Private CA). The AWS Private CA console gets usability improvements including integrated help panels that link to complete documentation.	September 27, 2022
Matter-compliant certificate support	Three new certificate templates add support for Matter-compliant CA and end-entity certificates. For more information, see Understanding certificate templates.	July 20, 2022
New region support	Endpoint added for Asia Pacific (Jakarta). For a complete list of AWS Private CA endpoints, see ACM Private Certificate Authority Endpoints and Quotas.	May 4, 2022

Support for Custom Attributes and Extensions	Use the CustomAttribute object to configure customized CAs and certificates, and the CustomExtension object to configure customized certificates.	March 16, 2022
Support for Managed OCSP	See Setting up a certificate revocation method for revocation options including OCSP.	August 18, 2021
Support for S3 Block Public Access feature for CRLs	See Enabling the S3 Block Public Access feature.	May 27, 2021
New and updated Java implementation examples	See Using the ACM Private CA API (Java Examples).	September 9, 2020
New region support	Endpoints added for Africa (Cape Town) and Europe (Milan). For a complete list of AWS Private CA endpoints, see AWS Certificate Manager Private Certificate Authority Endpoints and Quotas.	August 27, 2020
Cross-account private CA access supported	AWS Certificate Manager users can be authorized to issue certificates using private CAs that they do not own. For more information, see Cross-Account Access to Private CAs.	August 17, 2020

VPC endpoints (PrivateLink) support	Added support for use of VPC endpoints (AWSPrivateLink) for enhanced network security. For more information, see ACM Private CA VPC Endpoints (AWS PrivateLink).	March 26, 2020
Dedicated security section added	Security documentation for AWS has been consolidated into a dedicated security section. For information about security, see Security in AWS Certificate Manager Private Certificate Authority.	March 26, 2020
Template ARN added to audit reports.	For more information, see Creating an Audit Report for Your Private CA.	March 6, 2020
CloudFormation support	Support added for CloudFormation. For more information, see ACMPCA Resource Type Reference in the CloudFormation User Guide.	January 22, 2020
CloudWatch Events integration	Integration with CloudWatch Events for asynchronous events, including CA creation, certificate issuance, and CRL creation. For more information, see Using CloudWatch Events.	December 23, 2019

FIPS endpoints	FIPS endpoints added for AWS GovCloud (US-East) and AWS GovCloud (US-West). For a complete list of AWS Private CA endpoints, see AWS Certificate Manager Private Certificate Authority Endpoints and Quotas.	December 13, 2019
Tag-based permissions	Tag-based permissions supported using the new APIs `TagResource` , `UntagResource` , and `ListTagsForResource` . For general information about tag-based controls, see Controlling Access to and for IAM Users and Roles Using IAM Resource Tags.	November 5, 2019
Name constraints enforcement	Added support for enforcing subject name constraints on imported CA certificates. For more information, see Enforcing Name Constraints on a Private CA.	October 28, 2019
New certificate templates	New certificate templates added, including templates for code signing with AWS Signer. For more information, see Using Templates.	October 1, 2019

Planning your CA	New section added on planning your PKI using AWS Private CA. For more information, see Planning Your ACM Private CA Deployment.	September 30, 2019
Added region support	Added region support for the AWS Asia Pacific (Hong Kong) Region. For a complete list of supported regions, see AWS Certificate Manager Private Certificate Authority Endpoints and Quotas.	July 24, 2019
Added complete private CA hierarchy support	Support for creating and hosting root CAs removes need for an external parent.	June 20, 2019
Added region support	Added region support for the AWS GovCloud (US-West and US-East) Regions. For a complete list of supported regions, see AWS Certificate Manager Private Certificate Authority Endpoints and Quotas.	May 8, 2019

Added region support	Added region support for the AWS Asia Pacific (Mumbai and Seoul), US West (N. California), and EU (Paris and Stockholm) Regions. For a complete list of supported regions, see AWS Certificate Manager Private Certificate Authority Endpoints and Quotas.	April 4, 2019
Testing certificate renewal workflow	Customers can now manually test the configuration of their ACM managed renewal workflow. For more information, see Testing ACM's Managed Renewal Configuration.	March 14, 2019
Added region support	Added region support for the AWS EU (London) Region. For a complete list of supported regions, see AWS Certificate Manager Private Certificate Authority Endpoints and Quotas.	August 1, 2018
Restore deleted CAs	Private CA restore allows customers to restore certificate authorities (CAs) for up to 30 days after they have been deleted. For more information, see Restoring Your Private CA.	June 20, 2018

Earlier Updates

The following table describes the documentation release history of AWS Private Certificate Authority before June 2018.

Change	Description	Date
New guide	This release introduces AWS Private Certificate Authority.	April 04, 2018