AWS Certificate Manager Private Certificate Authority

User Guide

Version latest

AWS Certificate Manager Private Certificate Authority: User Guide

Table of Contents

What is ACM Private CA?

ACM 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

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

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. **ACM 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 ACM 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 ACM 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.

Regions

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 ACM Private CA.

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

Services integrated with AWS Certificate Manager 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 ACM 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 Securing Kubernetes with ACM Private CA (p. 219).

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

If you use the ACM 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

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

Supported algorithm

Private key algorithms	Signing algorithms
RSA_2048	SHA256WITHECDSA
RSA_4096	SHA384WITHECDSA
EC_prime256v1	SHA512WITHECDSA

Private key algorithms	Signing algorithms
EC_secp384r1	SHA256WITHRSA SHA384WITHRSA SHA512WITHRSA

This list applies only to certificates issued directly by ACM Private CA through its console, API, or command line. When AWS Certificate Manager issues certificates using a CA from ACM 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**
> In all cases, the specified signing algorithm family (RSA or ECDSA) must match the algorithm family of the CA's private key.

Quotas

ACM Private CA assigns quotas to your allowed number of certificates and certificate authorities. Request rates for API actions are also subject to quotas. ACM Private CA quotas are specific to an AWS account and Region.

ACM Private CA throttles API requests at different rates depending on the API operation. Throttling means that ACM 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, ACM Private CA returns a ThrottlingException error. ACM Private CA does not guarantee a minimum request rate for APIs.

To see what quotas can be adjusted, see the ACM Private CA quotas table in the *AWS General Reference Guide*.

You can view your current quotas and request quota increases using AWS Service Quotas.

To see an up-to-date list of your ACM 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.

ACM 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.

RFC compliance

ACM 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, ACM 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. ACM 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 ACM 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. ACM 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 (p. 231)) determines how many subordinate CAs may exist downstream from the imported CA certificate. ACM 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.

Not enforced

- Policy constraints. These constraints limit a CA's capacity to issue subordinate CA certificates.
- 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, ACM Private CA copies the SubjectPublicKeyInfo and SAN extensions from the provided CSR without performing validation.

Pricing

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 ACM Private CA API or ACM 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 ACM Private CA pricing information, see the ACM Pricing page on the AWS website. You can also use the AWS pricing calculator to estimate costs.

Security in AWS Certificate Manager 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 Certificate Manager 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 ACM Private CA. The following topics show you how to configure ACM 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 ACM Private CA resources.

Topics

Data protection in AWS Certificate Manager Private Certificate Authority

The AWS shared responsibility model applies to data protection in AWS Certificate Manager 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. This content includes 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 user accounts with 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 recommend TLS 1.2 or later.
- Set up API and user activity logging with AWS CloudTrail.
- 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 personal data that is stored in Amazon S3.
- If you require FIPS 140-2 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) 140-2.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form fields such as a **Name** field. This includes when you work with ACM 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 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 ACM 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 Security Requirements for Cryptographic Modules. The level of FIPS security compliance level varies by Region, as follows:

Region name	Region	FIPS PUB 140-2 Level 2 (or higher)	FIPS PUB 140-2 Level 3 (or higher)
US East (Ohio)	us-east-2		✓
US East (N. Virginia)	us-east-1		✓
US West (N. California)	us-west-1		✓
US West (Oregon)	us-west-2		✓
Africa (Cape Town)	af-south-1		✓
Asia Pacific (Hong Kong)	ap-east-1		✓
Asia Pacific (Mumbai)	ap-south-1		✓
Asia Pacific (Osaka)	ap-northeast-3	✓	
Asia Pacific (Seoul)	ap-northeast-2		✓
Asia Pacific (Singapore)	ap-southeast-1		✓
Asia Pacific (Sydney)	ap-southeast-2		✓
Asia Pacific (Tokyo)	ap-northeast-1		✓

Region name	Region	FIPS PUB 140-2 Level 2 (or higher)	FIPS PUB 140-2 Level 3 (or higher)
Canada (Central)	ca-central-1		✓
Europe (Frankfurt)	eu-central-1		✓
Europe (Ireland)	eu-west-1		✓
Europe (London)	eu-west-2		✓
Europe (Milan)	eu-south-1		✓
Europe (Paris)	eu-west-3		✓
Europe (Stockholm)	eu-north-1		✓
Middle East (Bahrain)	me-south-1		✓
South America (São Paulo)	sa-east-1		✓
AWS GovCloud (US-East)	us-gov-east-1		✓
AWS GovCloud (US-West)	us-gov-west-1		✓

Identity and Access Management for AWS Certificate Manager Private Certificate Authority

Access to ACM 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.

Authentication

You can access AWS as any of the following types of identities:

- **AWS account root user** – When you first create an AWS account, you begin with a single sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the best practice of using the root user only to create your first IAM user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.
- **IAM user** – An IAM user is an identity within your AWS account that has specific custom permissions (for example, permissions to create a directory in ACM Private CA). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

  In addition to a user name and password, you can also generate access keys for each user. You can use these keys when you access AWS services programmatically, either through one of the several SDKs or by using the AWS Command Line Interface (CLI). The SDK and CLI tools use the access keys to cryptographically sign your request. If you don't use AWS tools, you must sign the request yourself. ACM Private CA supports *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see Signature Version 4 signing process in the *AWS General Reference*.

- **IAM role** – An IAM role is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user in that it is an AWS identity with permissions policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session. IAM roles with temporary credentials are useful in the following situations:
  - **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. AWS assigns a role to a federated user when access is requested through an identity provider. For more information about federated users, see Federated users and roles in the *IAM User Guide*.
  - **AWS service access** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Understanding resources, ownership, and permissions policies

In ACM 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:region:account:certificate-authority/CA_ID
```

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.
- 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 ACM 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.

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 ACM Private CA API operation. You specify actions in a policy's `Action` element. The remaining columns provide the additional information.

ACM Private CA API operations and permissions

ACM Private CA API operations	Required permissions	Resources
CreateCertificateAuthority	`acm-pca:CreateCertificateAuthority`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`
CreateCertificateAuthorityAuditReport	`acm-pca:CreateCertificateAuthorityAuditReport`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`
CreatePermission	`acm-pca:CreatePermission`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`
DeleteCertificateAuthority	`acm-pca:DeleteCertificateAuthority`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`
DeletePermission	`acm-pca:DeletePermission`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`
DeletePolicy	`acm-pca:DeletePolicy`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`
DescribeCertificateAuthority	`acm-pca:DescribeCertificateAuthority`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`
DescribeCertificateAuthorityAuditReport	`acm-pca:DescribeCertificateAuthorityAuditReport`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`
GetCertificate	`acm-pca:GetCertificate`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`
GetCertificateAuthorityCertificate	`acm-pca:GetCertificateAuthorityCertificate`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`
GetCertificateAuthorityCsr	`acm-pca:GetCertificateAuthorityCsr`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`
GetPolicy	`acm-pca:GetPolicy`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`
ImportCertificateAuthorityCertificate	`acm-pca:ImportCertificateAuthorityCertificate`	`arn:aws:acm-pca:region:account:certificate-authority/CA_ID`

ACM Private CA API operations	Required permissions	Resources
IssueCertificate	acm-pca:IssueCertificate	arn:aws:acm-pca:*region*:*account*:certificate-authority/*CA_ID*
ListCertificateAuthorities	acm-pca:ListCertificateAuthorities	N/A
ListPermissions	acm-pca:ListPermissions	arn:aws:acm-pca:*region*:*account*:certificate-authority/*CA_ID*
ListTags	acm-pca:ListTags	N/A
PutPolicy	acm-pca:PutPolicy	arn:aws:acm-pca:*region*:*account*:certificate-authority/*CA_ID*
RevokeCertificate	acm-pca:RevokeCertificate	arn:aws:acm-pca:*region*:*account*:certificate-authority/*CA_ID*
TagCertificateAuthority	acm-pca:TagCertificateAuthority	arn:aws:acm-pca:*region*:*account*:certificate-authority/*CA_ID*
UntagCertificateAuthority	acm-pca:UntagCertificateAuthority	arn:aws:acm-pca:*region*:*account*:certificate-authority/*CA_ID*
UpdateCertificateAuthority	acm-pca:UpdateCertificateAuthority	arn:aws:acm-pca:*region*:*account*:certificate-authority/*CA_ID*

You can use IAM to create policies that apply permissions to IAM users, groups, and roles. These are called *identity-based policies*. IAM offers the following types of identity-based policies:

- AWS managed policies (p. 10) are policies available by default with ACM Private CA. These policies cover basic user roles.
- Customer managed policies (p. 13) are policies that you create and manage in your AWS account and which you can attach to multiple users, groups, and roles. You have more precise control when using customer managed policies than you have when using AWS managed policies.
- Inline policies (p. 14) are policies that you create and manage and which you embed directly into a single user, group, or role.
- Resource-based policies (p. 17) are used by ACM Private CA to enable cross-account access to private CAs.

AWS managed policies

ACM Private CA includes a set of predefined AWS managed policies for administrators, users, and auditors. Understanding these policies can help you implement Customer managed policies (p. 13).

- **FullAccess** – Unrestricted administrative control.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:*"
         ],
         "Resource":"*"
      }
   ]
}
```

- **ReadOnly** – Access limited to read-only API operations.

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":[
         "acm-pca:DescribeCertificateAuthority",
         "acm-pca:DescribeCertificateAuthorityAuditReport",
         "acm-pca:ListCertificateAuthorities",
         "acm-pca:GetCertificateAuthorityCsr",
         "acm-pca:GetCertificateAuthorityCertificate",
         "acm-pca:GetCertificate",
         "acm-pca:ListPermissions",
         "acm-pca:ListTags"
      ],
      "Resource":"*"
   }
}
```

- **PrivilegedUser** – 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.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:IssueCertificate"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*",
         "Condition":{
            "StringLike":{
               "acm-pca:TemplateArn":[
                  "arn:aws:acm-pca:::template/*CACertificate*/V*"
               ]
            }
         }
      },
      {
         "Effect":"Deny",
         "Action":[
            "acm-pca:IssueCertificate"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*",
         "Condition":{
            "StringNotLike":{
               "acm-pca:TemplateArn":[
                  "arn:aws:acm-pca:::template/*CACertificate*/V*"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:RevokeCertificate",
            "acm-pca:GetCertificate",
            "acm-pca:ListPermissions"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:ListCertificateAuthorities"
         ],
         "Resource":"*"
      }
   ]
}
```

- **User** – 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.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:IssueCertificate"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*",
         "Condition":{
            "StringLike":{
               "acm-pca:TemplateArn":[
                  "arn:aws:acm-pca:::template/EndEntityCertificate/V*"
               ]
            }
         }
      },
      {
         "Effect":"Deny",
         "Action":[
            "acm-pca:IssueCertificate"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*",
         "Condition":{
            "StringNotLike":{
               "acm-pca:TemplateArn":[
                  "arn:aws:acm-pca:::template/EndEntityCertificate/V*"
               ]
            }
         }
      },
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:RevokeCertificate",
            "acm-pca:GetCertificate",
            "acm-pca:ListPermissions"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:ListCertificateAuthorities"
         ],
         "Resource":"*"
      }
   ]
}
```

- **Auditor** – Access to read-only API operations and permission to generate a CA audit report.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:CreateCertificateAuthorityAuditReport",
            "acm-pca:DescribeCertificateAuthority",
            "acm-pca:DescribeCertificateAuthorityAuditReport",
            "acm-pca:GetCertificateAuthorityCsr",
            "acm-pca:GetCertificateAuthorityCertificate",
            "acm-pca:GetCertificate",
            "acm-pca:ListPermissions",
            "acm-pca:ListTags"
         ],
         "Resource":"arn:aws:acm-pca:*:*:certificate-authority/*"
      },
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:ListCertificateAuthorities"
         ],
         "Resource":"*"
      }
   ]
}
```

Customer managed policies

As a best practice, don't use your AWS account root user to interact with AWS, including ACM 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 ACM 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 ACM Private CA operations that you want. For more examples, see Inline policies (p. 14).

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.

```
{
   "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 ACM 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 ACM 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.

Topics

Listing private CAs

The following policy allows a user to list all of the private CAs in an account.

```
{
   "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.

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:GetCertificateAuthorityCertificate",
      "Resource":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID"
   }
}
```

Importing a private CA certificate

The following policy allows a user to import a private CA certificate.

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:ImportCertificateAuthorityCertificate",
      "Resource":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID"
   }
}
```

Deleting a private CA

The following policy allows a user to delete a specific private CA.

```
{
   "Version":"2012-10-17",
   "Statement":{
      "Effect":"Allow",
      "Action":"acm-pca:DeleteCertificateAuthority",
      "Resource":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID"
   }
}
```

Read-only access to ACM 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.

```
{
   "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 ACM Private CA

The following policy allows a user to perform any ACM Private CA action.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "acm-pca:*"
         ],
         "Resource":"*"
      }
   ]
}
```

Administrator access to all AWS resources

The following policy allows a user to perform any action on any AWS resource.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":"*",
         "Resource":"*"
      }
   ]
}
```

Cross-account access to private CAs

An ACM 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 ACM Private CA or AWS Certificate Manager resources.

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, ACM 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 ACM 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 (p. 17).

The Controlling access to a private CA (p. 83) 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. Using RAM to apply a resource-based policy, an ACM Private CA administrator can share access to a CA with a user in a different AWS account directly or through AWS Organizations. Alternatively, an ACM 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.

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 (p. 128)
- EndEntityClientAuthCertificate/V1 (p. 130)
- EndEntityServerAuthCertificate/V1 (p. 132)
- BlankEndEntityCertificate_APICSRPassthrough (p. 120)

The following examples contain resource-based policies and the commands to apply them. In addition to specifying the ARN of a CA, the administrator provides an AWS user account ID or an AWS Organizations ID that will be granted access to the CA. For readability, the JSON of each policy is provided as a file instead of inline.

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.

Example 1: Sharing access to a CA with a user in a different account

```
$ aws acm-pca put-policy \
   --region region \
   --resource-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID \
   --policy file:///[path]/policy1.json
```

The file `policy1.json` has the following content:

```
{
  "Version":"2012-10-17",
  "Statement":[
     {
        "Sid":"1",
        "Effect":"Allow",
        "Principal":{

           "AWS":"account"

        },
         "Action":[
            "acm-pca:DescribeCertificateAuthority",
            "acm-pca:GetCertificate",
            "acm-pca:GetCertificateAuthorityCertificate",
            "acm-pca:ListPermissions",
            "acm-pca:ListTags"

         ],

         "Resource":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID"
      },
      {
         "Sid":"1",
         "Effect":"Allow",
         "Principal":{
            "AWS":"account"
         },
         "Action":[
            "acm-pca:IssueCertificate"
         ],
         "Resource":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
         "Condition":{
            "StringEquals":{
               "acm-pca:TemplateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1"
            }
         }
      }
   ]
}
```

Example 2: Sharing access to a CA through AWS Organizations

```
$ aws acm-pca put-policy \
   --region region \
   --resource-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID
   --policy file:///[path]/policy2.json
```

The file `policy2.json` has the following content:

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"1",
         "Effect":"Allow",
         "Principal":"*",
         "Action":"acm-pca:IssueCertificate",
         "Resource":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
         "Condition":{
            "StringEquals":{
               "acm-pca:TemplateArn":"arn:aws:acm-pca:::template/EndEntityCertificate/V1",
               "aws:PrincipalOrgID":"o-1b2c3d4z5"
            }
         }
      },
      {
         "Sid":"1",
         "Effect":"Allow",
         "Principal":"*",
         "Action":[
            "acm-pca:DescribeCertificateAuthority",
            "acm-pca:GetCertificate",
            "acm-pca:GetCertificateAuthorityCertificate",
            "acm-pca:ListPermissions",
            "acm-pca:ListTags"
         ],
         "Resource":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
         "Condition":{
            "StringEquals":{
               "aws:PrincipalOrgID":"o-a1b2c3d4z5"
            }
         }
      }
   ]
}
```

Logging and monitoring for AWS Certificate Manager Private Certificate Authority

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Certificate Manager Private Certificate Authority and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs.

The following topics describe AWS cloud-monitoring tools available for use with ACM Private CA.

Topics

Supported 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. ACM 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

Using 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. 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:region:account:certificate-authority/CA_ID"
   ],
   "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:region:account:certificate-authority/CA_ID"
   ],
   "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:region:account:certificate-authority/CA_ID",
      "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:region:account:certificate-authority/CA_ID",
      "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:region:account:certificate-authority/CA_ID",
      "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":"region1",
   "resources":[
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID"
   ],
   "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:region:account:certificate-authority/CA_ID"
   ],
   "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:region:account:certificate-authority/CA_ID"
   ],
   "detail":{
      "result":"failure",
      "reason":"Failed to write CRL to S3. Internal failure."
   }
}
```

Failure 3 – ACM PCA 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":"1account",
   "time":"2019-11-07T23:01:25Z",
   "region":"region",
   "resources":[
      "arn:aws:acm-pca:region:account:certificate-authority/CA_ID"
   ],
   "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:region:account:certificate-authority/CA_ID",
      "audit_report_ID"
   ],
   "detail":{
      "result":"success"
   }
}
```

Failure

An audit report can fail when ACM PCA 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:region:account:certificate-authority/CA_ID",
      "audit_report_ID"
   ],
   "detail":{
      "result":"failure"
   }
}
```

Using CloudTrail

You can use AWS CloudTrail to record API calls that are made by AWS Certificate Manager Private Certificate Authority. For more information, see the following topics.

Topics

Creating a policy

The following CloudTrail example shows the results of a call to the PutPolicy operation.

```
{
   "eventVersion":"1.08",
   "userIdentity":{
      "type":"AssumedRole",
      "principalId":"account",
      "arn":"arn:aws:sts::account:assumed-role/role",
      "accountId":"account",
      "accessKeyId":"key_ID",
      "sessionContext":{
         "sessionIssuer":{
            "type":"Role",
            "principalId":"account",
            "arn":"arn:aws:iam::account:role/role",
            "accountId":"account",
            "userName":"name"
         },
         "webIdFederationData":{

         },
         "attributes":{
            "mfaAuthenticated":"false",
            "creationDate":"2021-02-26T21:01:38Z"
         }
      },
      "invokedBy":"agent"
   },
   "eventTime":"2021-02-26T21:25:36Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"PutPolicy",
   "awsRegion":"region",
   "sourceIPAddress":"xx.xx.xx.xx",
   "userAgent":"agent",
   "requestParameters":{
      "resourceArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
      "policy":"{\"Version\":\"2012-10-17\",\"Statement\":[{\"Sid\":\"01234567-89ab-
cdef-0123-456789abcdef4-external-principals\",\"Effect\":\"Allow\",\"Principal\":{\"AWS
\":\"account\"},\"Action\":\"acm-pca:IssueCertificate\",\"Resource\":\"arn:aws:acm-
pca:region:account:certificate-authority/CA_ID\",\"Condition\":{\"StringEquals\":{\"acm-
pca:TemplateArn\":\"arn:aws:acm-pca:::template/EndEntityCertificate/V1\"}}},{\"Sid
\":\"01234567-89ab-cdef-0123-456789abcdef-external-principals\",\"Effect\":\"Allow\",
\"Principal\":{\"AWS\":\"account\"},\"Action\":[\"acm-pca:DescribeCertificateAuthority
\",\"acm-pca:GetCertificate\",\"acm-pca:GetCertificateAuthorityCertificate\",
\"acm-pca:ListPermissions\",\"acm-pca:ListTags\"],\"Resource\":\"arn:aws:acm-
pca:region:account:certificate-authority/CA_ID\"}]}"
   },
   "responseElements":null,
   "requestID":"01234567-89ab-cdef-0123-456789abcdef",
   "eventID":"01234567-89ab-cdef-0123-456789abcdef",
   "readOnly":false,
   "eventType":"AwsApiCall",
   "managementEvent":true,
   "eventCategory":"Management",
   "recipientAccountId":"account"
}
```

Retrieving a policy

The following CloudTrail example shows the results of a call to the GetPolicy operation.

```
{
   "eventVersion":"1.08",
   "userIdentity":{
      "type":"AssumedRole",
      "principalId":"account",
      "arn":"arn:aws:sts::account:assumed-role/role",
      "accountId":"account",
      "accessKeyId":"key_ID",
      "sessionContext":{
         "sessionIssuer":{
            "type":"Role",
            "principalId":"account",
            "arn":"arn:aws:iam::account:role/role",
            "accountId":"account",
            "userName":"name"
         },
         "webIdFederationData":{

         },
         "attributes":{
            "mfaAuthenticated":"false",
            "creationDate":"2021-02-26T20:49:51Z"
         }
      }
   },
   "eventTime":"2021-02-26T21:19:14Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"GetPolicy",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "errorCode":"ResourceNotFoundException",
   "errorMessage":"Could not find policy for resource arn:aws:acm-
pca:region:account:certificate-authority/CA_ID.",
   "requestParameters":{
      "resourceArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "readOnly":true,
   "eventType":"AwsApiCall",
   "managementEvent":true,
   "eventCategory":"Management",
   "recipientAccountId":"account"
}
```

Deleting a policy

The following CloudTrail example shows the results of a call to the DeletePolicy operation.

```
{
   "eventVersion":"1.08",
   "userIdentity":{
      "type":"AssumedRole",
      "principalId":"account",
      "arn":"arn:aws:sts::account:assumed-role/role",
      "accountId":"account",
      "accessKeyId":"key_ID",
      "sessionContext":{
         "sessionIssuer":{
            "type":"Role",
            "principalId":"account",
            "arn":"arn:aws:iam::account:role/role",
            "accountId":"account",
            "userName":"name"
         },
         "webIdFederationData":{

         },
         "attributes":{
            "mfaAuthenticated":"false",
            "creationDate":"2021-02-26T21:23:17Z"
         }
      }
   },
   "eventTime":"2021-02-26T21:23:31Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"DeletePolicy",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "resourceArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "readOnly":false,
   "eventType":"AwsApiCall",
   "managementEvent":true,
   "eventCategory":"Management",
   "recipientAccountId":"account"
}
```

Creating a certificate authority

The following CloudTrail example shows the results of a call to the CreateCertificateAuthority operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T21:22:33Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"CreateCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityConfiguration":{
         "keyType":"RSA2048",
         "signingAlgorithm":"SHA256WITHRSA",
         "subject":{
            "country":"US",
            "organization":"Example Company",
            "organizationalUnit":"Corp",
            "state":"WA",
            "commonName":"www.example.com",
            "locality":"Seattle"
         }
      },
      "revocationConfiguration":{
         "crlConfiguration":{
            "enabled":true,
            "expirationInDays":3650,
            "customCname":"your-custom-name",
            s3BucketName:"your-bucket-name"
         }
      },
      "certificateAuthorityType":"SUBORDINATE",
      "idempotencyToken":"98256344"
   },
   "responseElements":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID"
   },
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

GenerateCRL

The following CloudTrail example shows the record for a GenerateCRL event.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "accountId": "account",
        "invokedBy": "acm-pca.amazonaws.com"
    },
    "eventTime": "2021-02-09T17:37:45Z",
    "eventSource": "acm-pca.amazonaws.com",
    "eventName": "GenerateCRL",
    "awsRegion": "region",
    "sourceIPAddress": "acm-pca.amazonaws.com",
    "userAgent": "acm-pca.amazonaws.com",
    "requestParameters": null,
    "responseElements": null,
    "eventID": "01234567-89ab-cdef-0123-456789abcdef",
    "readOnly": false,
    "resources": [
        {
            "type": "AWS::ACMPCA::CertificateAuthority",
            "ARN": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID"
        }
    ],
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "account"
}
```

GenerateOCSPResponse

The following CloudTrail example shows the record for a GenerateOCSPResponse event.

```
{
   "eventVersion":"1.08",
   "userIdentity":{
      "accountId":"account",
      "invokedBy":"acm-pca.amazonaws.com"
   },
   "eventTime":"2021-02-08T23:52:29Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"GenerateOCSPResponse",
   "awsRegion":"region",
   "sourceIPAddress":"acm-pca.amazonaws.com",
   "userAgent":"acm-pca.amazonaws.com",
   "eventID":"01234567-89ab-cdef-0123-456789abcdef",
   "readOnly":false,
   "resources":[
      {
         "type":"AWS::ACMPCA::Certificate",
         "ARN":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
      }
   ]
}
```

Creating an audit report

The following CloudTrail example shows the results of a call to the CreateCertificateAuthorityAuditReport operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T21:56:00Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"CreateCertificateAuthorityAuditReport",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID",
      s3BucketName:"bucket_name",
      "auditReportResponseFormat":"JSON"
   },
   "responseElements":{
      "auditReportId":"report_ID",
      s3Key:"audit-report/CA_ID/audit_report_ID.json"
   },
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Deleting a certificate authority

The following CloudTrail example shows the results of a call to the DeleteCertificateAuthority operation. In this example, the certificate authority cannot be deleted because it is in the `ACTIVE` state.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:01:11Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"DeleteCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "errorCode":"InvalidStateException",
   "errorMessage":"The certificate authority is not in a valid state for deletion.",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Restoring a certificate authority

The following CloudTrail example shows the results of a call to the RestoreCertificateAuthority operation. In this example, the certificate authority cannot be restored because it is not in the `DELETED` state.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:01:11Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"RestoreCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"xIP_address",
   "userAgent":"agent",
   "errorCode":"InvalidStateException",
   "errorMessage":"The certificate authority is not in a valid state for restoration.",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Describing a certificate authority

The following CloudTrail example shows the results of a call to the DescribeCertificateAuthority operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T21:58:18Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"DescribeCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Retrieving a certificate authority certificate

The following CloudTrail example shows the results of a call to the GetCertificateAuthorityCertificate operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:03:52Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"GetCertificateAuthorityCertificate",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Retrieving the certificate authority signing request

The following CloudTrail example shows the results of a call to the GetCertificateAuthorityCsr operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T21:40:33Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"GetCertificateAuthorityCsr",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Retrieving a certificate

The following CloudTrail example shows the results of a call to the GetCertificate operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:22:54Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"GetCertificate",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID",
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Importing a certificate authority certificate

The following CloudTrail example shows the results of a call to the ImportCertificateAuthorityCertificate operation.

```
{
   "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:region:account:certificate-
authority/CA_ID",
      "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"
}
```

Issuing a certificate

The following CloudTrail example shows the results of a call to the IssueCertificate operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:18:43Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"IssueCertificate",
   "awsRegion":"region",
   "sourceIPAddress":"xIP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID",
      "csr":{
         "hb":[
            45,
            45,
            ...10
         ],
         "offset":0,
         "isReadOnly":false,
         "bigEndian":true,
         "nativeByteOrder":false,
         "mark":-1,
         "position":1090,
         "limit":1090,
         "capacity":1090,
         "address":0
      },
      "signingAlgorithm":"SHA256WITHRSA",
      "validity":{
         "value":365,
         "type":"DAYS"
      },
      "idempotencyToken":"1234"
   },
   "responseElements":{
      "certificateArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/certificate_ID"
   },
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Listing certificate authorities

The following CloudTrail example shows the results of a call to the ListCertificateAuthorities operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:09:43Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"ListCertificateAuthorities",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "maxResults":10
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Listing tags

The following CloudTrail example shows the results of a call to the ListTags operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-02-02T00:21:56Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"ListTags",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID"
   },
   "responseElements":{
      "tags":[
         {
            "key":"Admin",
            "value":"Alice"
         },
         {
            "key":"User",
            "value":"Bob"
         }
      ]
   },
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Revoking a certificate

The following CloudTrail example shows the results of a call to the RevokeCertificate operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name
      "accessKeyId": "key_ID"
   },
   "eventTime": "2018-01-26T22:35:03Z",
   "eventSource": "acm-pca.amazonaws.com",
   "eventName": "RevokeCertificate",
   "awsRegion": "region",
   "sourceIPAddress": "IP_address",
   "userAgent": "agent",
   "requestParameters": {
      "certificateAuthorityArn": "arn:aws:acm-pca:region:account:certificate-
authority/CA_ID",
      "certificateSerial": "67:07:44:76:83:a9:b7:f4:05:56:27:ff:d5:5c:eb:cc",
      "revocationReason": "KEY_COMPROMISE"
   },
   "responseElements": null,
   "requestID": "request_ID",
   "eventID": "event_ID",
   "eventType": "AwsApiCall",
   "recipientAccountId": "account"
}
```

Tagging private certificate authorities

The following CloudTrail example shows the results of a call to the TagCertificateAuthority operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-02-02T00:18:48Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"TagCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID",
      "tags":[
         {
            "key":"Admin",
            "value":"Alice"
         }
      ]
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Removing tags from a private certificate authority

The following CloudTrail example shows the results of a call to the UntagCertificateAuthority operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-02-02T00:21:50Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"UntagCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID",
      "tags":[
         {
            "key":"Admin",
            "value":"Alice"
         }
      ]
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Updating a certificate authority

The following CloudTrail example shows the results of a call to the UpdateCertificateAuthority operation.

```
{
   "eventVersion":"1.05",
   "userIdentity":{
      "type":"IAMUser",
      "principalId":"account",
      "arn":"arn:aws:iam::account:user/name",
      "accountId":"account",
      "accessKeyId":"key_ID"
   },
   "eventTime":"2018-01-26T22:08:59Z",
   "eventSource":"acm-pca.amazonaws.com",
   "eventName":"UpdateCertificateAuthority",
   "awsRegion":"region",
   "sourceIPAddress":"IP_address",
   "userAgent":"agent",
   "requestParameters":{
      "certificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-
authority/CA_ID",

      "revocationConfiguration":{
         "crlConfiguration":{
            "enabled":true,
            "expirationInDays":3650,
            "customCname":"your-custom-name",
            s3BucketName:"your-bucket-name"
         }
      },
      "status":"DISABLED"
   },
   "responseElements":null,
   "requestID":"request_ID",
   "eventID":"event_ID",
   "eventType":"AwsApiCall",
   "recipientAccountId":"account"
}
```

Compliance validation for AWS Certificate Manager Private Certificate Authority

Third-party auditors assess the security and compliance of AWS Certificate Manager 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 ACM 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 ACM 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 — 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.

Using 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 (p. 41).

The audit report file has the following path and file name. The ARN for an Amazon S3 bucket is the value for `bucket-name`. `CA_ID` is the unique identifier of an issuing CA. `UUID` is the unique identifier of an audit report.

```
bucket-name/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,issuedAt,revo
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=#0a1b3c4
 Company,L=Seattle,ST=Washington,C=US",2020-03-02T21:43:57+0000,2020-04-07T22:43:57+0000,2020-03-02T22:
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=#0a1b3c4
 Company,L=Seattle,ST=Washington,C=US",2020-03-02T20:53:39+0000,2020-04-07T21:53:39+0000,2020-03-02T21:
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=#0a1b3c4d5e,2.5.4.
 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=#0a1b3c4d5e,2.5.4.
 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 ACM Private CA API on behalf of a customer to renew the certificate.

Preparing an Amazon S3 bucket for audit reports

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 an attached permissions policy. Authorized users and service principals require `Put` permission to allow ACM 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. For more information, see Adding a bucket policy using the Amazon S3 console.

Note
During the console procedure for creating an audit report, you can choose to let ACM Private CA create a new bucket and apply a default permissions policy. The default policy applies no `SourceArn` restriction on the CA and is therefore more permissive than the recommended policy. If you choose the default, you can always modify it later.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"acm-pca.amazonaws.com"
         },
         "Action":[
            "s3:PutObject",
            "s3:PutObjectAcl",
            "s3:GetBucketAcl",
            "s3:GetBucketLocation"
         ],
         "Resource":[
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
         ],
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"account",
               "aws:SourceArn":"arn:partition:acm-pca:region:account:certificate-
authority/CA_ID"
            }
         }
      }
   ]
}
```

Creating 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 ACM 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. 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**, ACM Private CA creates and attaches the default policy to your bucket. 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 Preparing an Amazon S3 bucket for audit reports (p. 40). For information about attaching a policy, see How Do I Add an S3 Bucket Policy?
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 Preparing an Amazon S3 bucket for audit reports (p. 40). For information about attaching a policy, see How Do I Add an S3 Bucket Policy?
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:region:account:certificate-authority/CA_ID
 \
--s3-bucket-name bucket_name \
--audit-report-response-format JSON
```

Retrieving 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. ACM 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
ACM 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 ACM 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 ACM 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 ACM 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 Certificate Manager Private Certificate Authority

As a managed service, ACM Private CA is protected by the AWS global network security procedures that are described in the Amazon Web Services: Overview of Security Processes whitepaper.

You use AWS published API calls to access ACM Private CA through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

Topics

ACM Private CA VPC endpoints (AWS PrivateLink)

You can create a private connection between your VPC and ACM Private CA by configuring an interface VPC endpoint. Interface endpoints are powered by AWS PrivateLink, a technology for privately accessing ACM Private CA API operations. AWS PrivateLink routes all network traffic between your VPC and ACM 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 ACM Private CA without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. The instances in your VPC don't need public IP addresses to communicate with the ACM Private CA API.

To use ACM 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 (AWS VPN) or AWS Direct Connect. For information about AWS VPN, see VPN Connections in the *Amazon VPC User Guide*. For information about AWS Direct Connect, see Creating a Connection in the *AWS Direct Connect User Guide*.

ACM 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 ACM Private CA VPC endpoints

Before you set up interface VPC endpoints for ACM Private CA, be aware of the following considerations:

- ACM 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 ACM 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 Certificate Manager does not support VPC endpoints.
- FIPS endpoints (and their Regions) do not support VPC endpoints.

Creating the VPC endpoints for ACM Private CA

You can create a VPC endpoint for the ACM 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*. ACM 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 ACM 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 ACM Private CA, such as `us-east-2` for the US East (Ohio) Region. For a list of ACM Private CA, see AWS Certificate Manager Private Certificate Authority Endpoints and Quotas.

For more information, see Interface VPC Endpoints (AWS PrivateLink) in the *Amazon VPC User Guide*.

Create a VPC endpoint policy for ACM Private CA

You can create a policy for Amazon VPC endpoints for ACM 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 User Guide*.

Example – VPC endpoint policy for ACM Private CA actions

When attached to an endpoint, the following policy grants access for all principals to the ACM 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:region:account:certificate-authority/CA_ID"
         ],
         "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:region:account:certificate-authority/CA_ID"
         ]
      }
   ]
}
```

Planning your ACM Private CA deployment

ACM 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 ACM 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 ACM Private CA. In this scenario, you then create a subordinate CA in ACM Private CA backed by a parent CA outside of ACM Private CA. For more information, see Installing a subordinate CA certificate signed by an external parent CA.

Topics

Setting up your AWS account and the AWS CLI

If you're not already an Amazon Web Services (AWS) customer, you must sign up to be able to use ACM Private CA. Your account automatically has access to all available services, but you are charged only for services that you use.

Note
ACM Private CA is not available in the AWS Free Tier.

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 and entering a verification code on the phone keypad.

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.

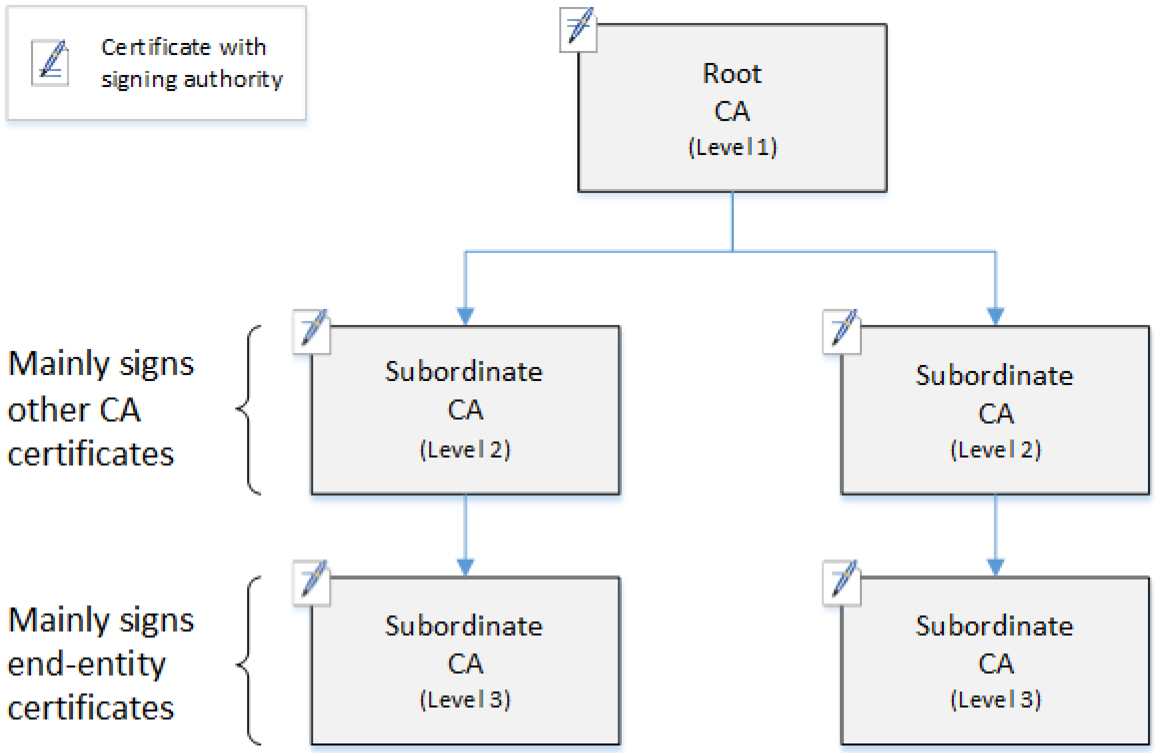
Designing a CA hierarchy

With ACM 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.

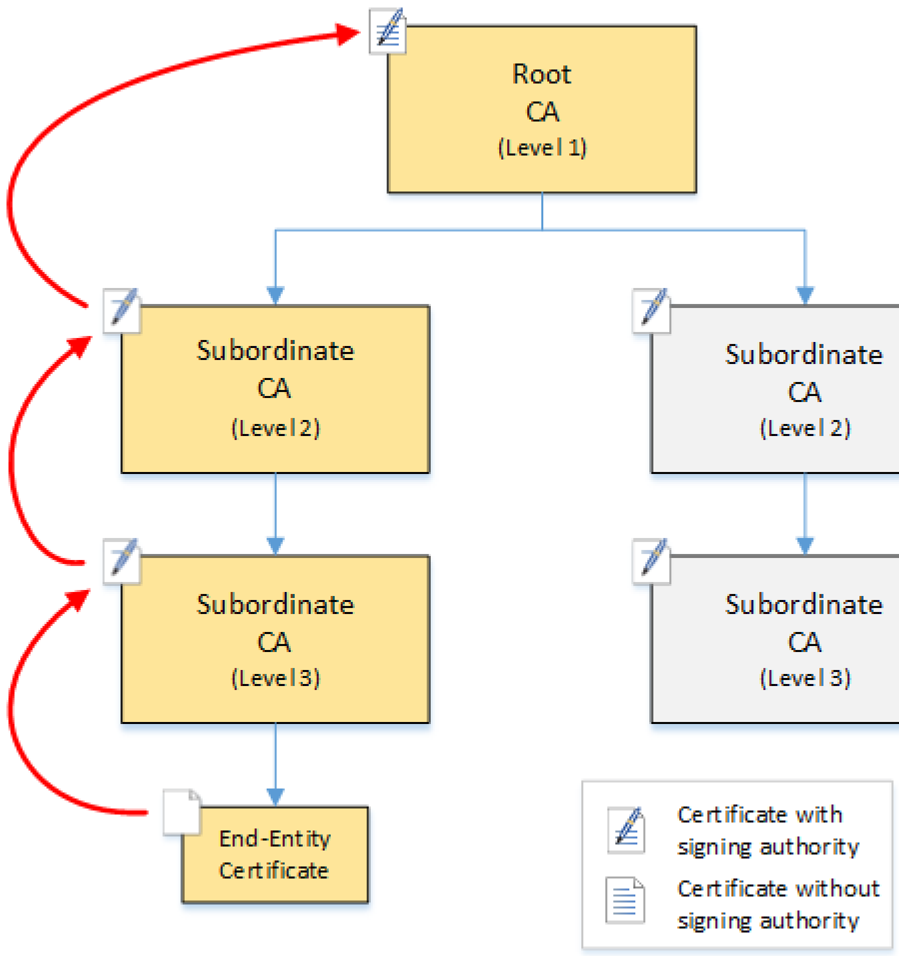
Topics

Validating 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.

Planning 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 (p. 231) 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. ACM 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**

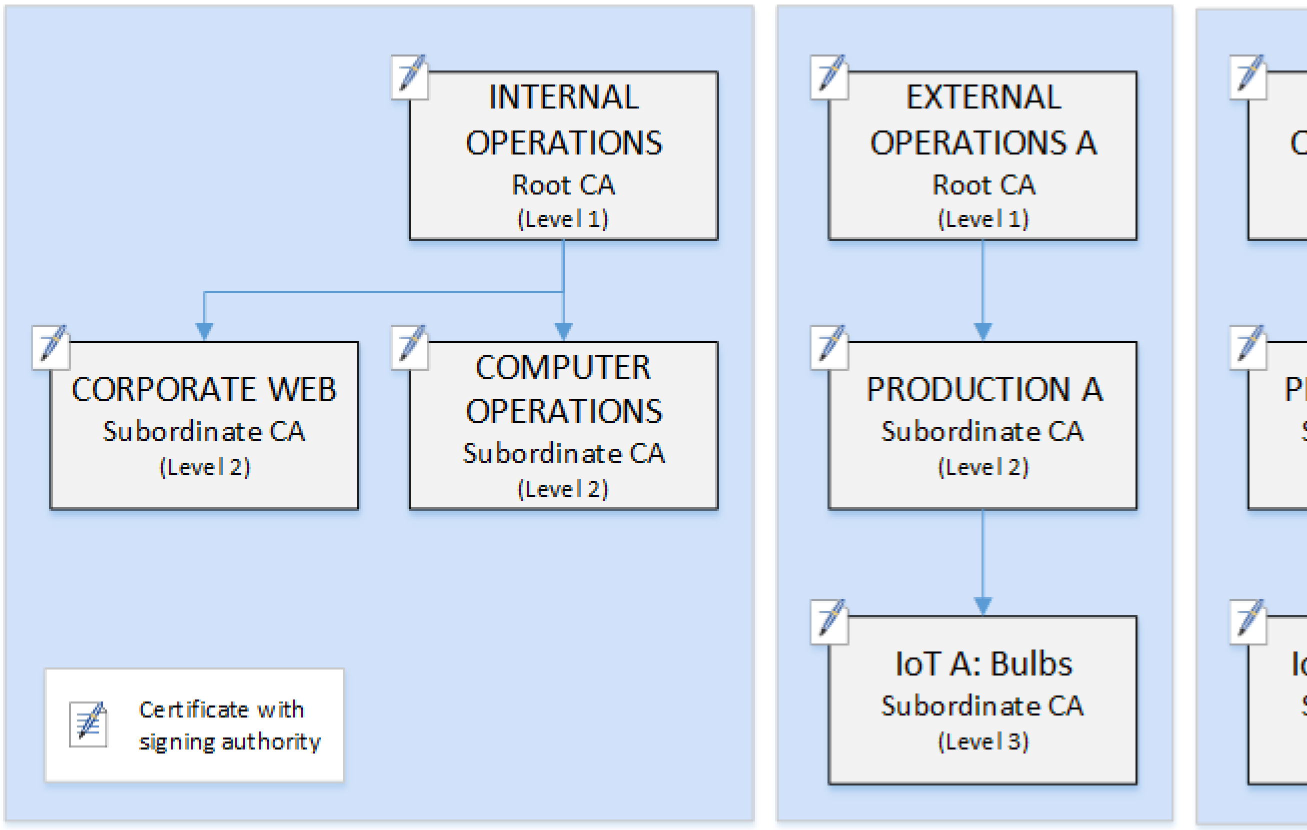
  This structure is commonly used for development and testing when a full chain of trust is not required. Used in production, it is atypical. Moreover, it violates the best practice of maintaining separate security policies for the root CA and the CAs that issue end-entity certificates.

  However, if you are already issuing certificates directly from a root CA, you can migrate to ACM 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.

Setting 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.

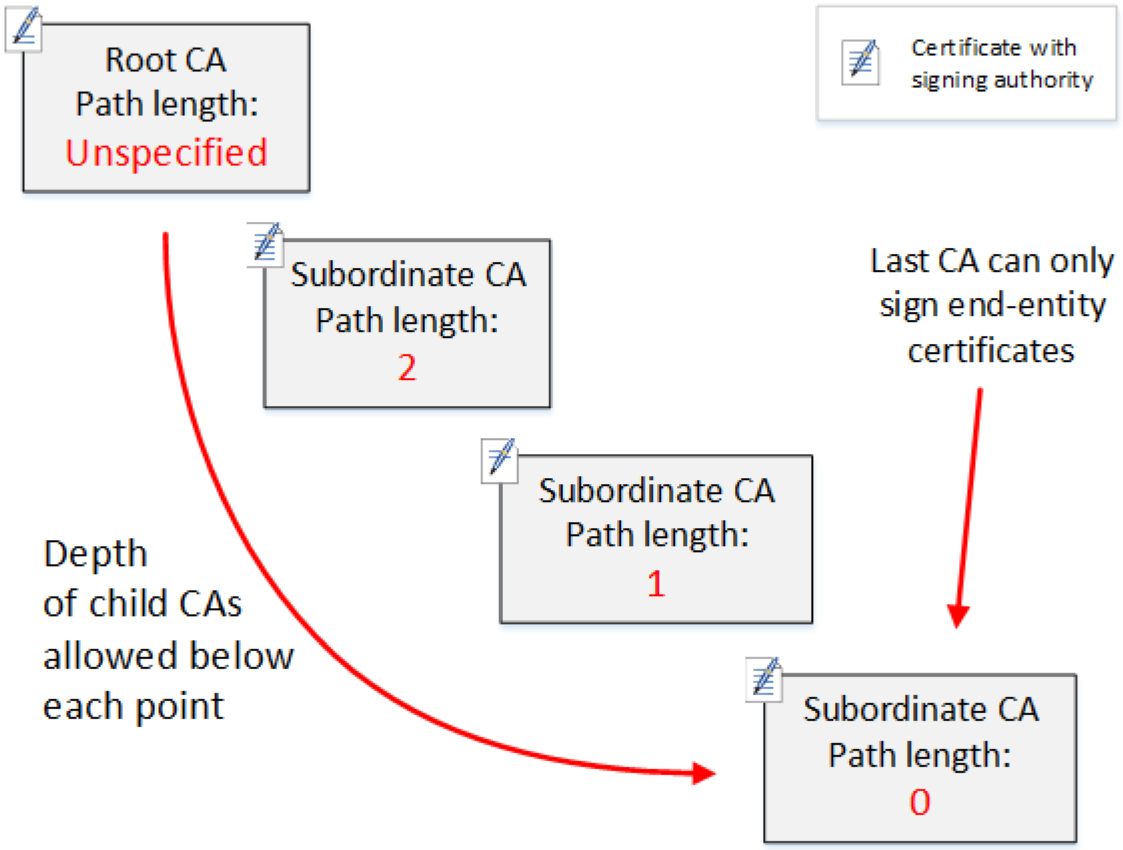
- `pathLenConstraint` – The maximum number of lower-level subordinate CAs that can exist in a valid chain of trust. The end-entity certificate is not counted because it is not a CA certificate.

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**
> ACM 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.


Root CA
Path length:
Unspecified
Certificate with signing authority
Subordinate CA
Path length:
2
Last CA can only sign end-entity certificates
Subordinate CA
Path length:
1
Depth of child CAs allowed below each point
Subordinate CA
Path length:
0

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.

Managing path length with templates

ACM 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 Understanding certificate templates (p. 113).

Automating 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 *AWS CloudFormation User Guide*.

Managing 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.

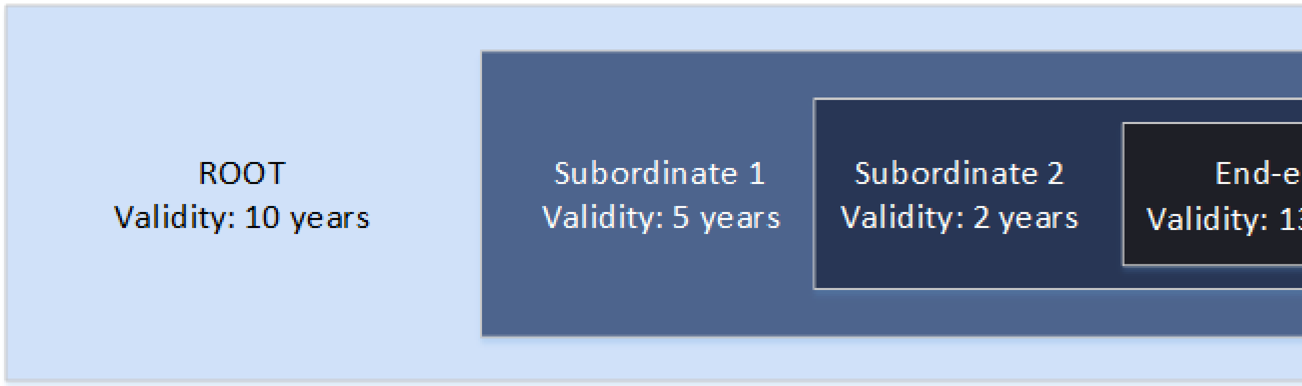
Choosing validity periods

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.)

ACM Private CA and AWS Certificate Manager assist with the configuration of certificate validity periods subject to the following constraints:

- A certificate managed by ACM 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 ACM 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 ACM 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 ACM 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 ACM Private CA default for root certificates is ten years.

Managing CA succession

You have two ways to manage CA succession: Replace the old CA, or reissue the CA with a new validity period.

Replacing 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.

Revoking 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 (p. 56). You should revoke a CA certificate only if you want to revoke all of its issued end-entity and subordinate CA certificates.

Setting up a certificate revocation method

As you plan your private PKI with ACM 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 ACM 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 Updating your private CA (p. 89).

- **Online Certificate Status Protocol (OCSP)**

  ACM 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 ACM Private CA console, the API, the CLI, or through AWS 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, ACM 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 ACM 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 ACM 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 CRL contains a list of revoked certificates. When you configure a CA to generate CRLs, ACM Private CA includes the *CRL Distribution Points* extension in each new certificate issued. This extension provides the URL for the CRL. The extension allows clients such as web browsers to query the CRL and determine whether an end-entity or subordinate CA certificate can be trusted.

Because a client must download CRLs and process them locally, their use is more memory-intensive than OCSP. CRLs may consume less network bandwidth because the list of CRLs is downloaded and cached, compared with OCSP which checks revocation status for each new connection attempt.

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, ACM Private CA makes further attempts every 15 minutes.

Topics

Planning a certificate revocation list (CRL)

Before you can configure a CRL as part of the CA creation process (p. 66), 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 4. Configure revocation (p. 68) and Configuring a Custom URL for ACM Private CA OCSP (p. 63).

Topics

- Access policies for CRLs in Amazon S3 (p. 59)
- Enabling the S3 Block Public Access (BPA) feature (p. 59)
- Encrypting Your CRLs (p. 61)

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. ACM 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 permissions policy. Authorized users and service principals require `Put` permission to allow ACM Private CA to place objects in the bucket, and `Get` permission to retrieve them. During the console procedure for creating (p. 66) a CA, you can choose to let ACM Private CA create a new bucket and apply a default permissions policy.

The default policy applies no `SourceArn` restriction on the CA. We recommend that you manually apply the less permissive policy shown below, which restricts access to both a specific AWS account and a specific private CA. For more information, see Adding a bucket policy using the Amazon S3 console.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"acm-pca.amazonaws.com"
         },
         "Action":[
            "s3:PutObject",
            "s3:PutObjectAcl",
            "s3:GetBucketAcl",
            "s3:GetBucketLocation"
         ],
         "Resource":[
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET/*",
            "arn:aws:s3:::DOC-EXAMPLE-BUCKET"
         ],
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"account",
               "aws:SourceArn":"arn:partition:acm-pca:region:account:certificate-
authority/CA_ID"
            }
         }
      }
   ]
}
```

If you choose to allow the default policy, you can always modify it later.

Enabling the S3 Block Public Access (BPA) feature

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 Amazon S3 with BPA

In S3, create a new bucket for your CRL, as usual, then enable BPA on it.

To configure an Amazon S3 bucket that blocks public access to your CRL

1. Create a new S3 bucket using the procedure in Creating a bucket. During the procedure, select the **Block *all* public access** option.

   For more information, see Blocking public access to your Amazon S3 storage.
2. When the bucket has been created, choose its name from the list, navigate to the **Permissions** tab, choose **Edit** in the **Object ownership** section, and select **Bucket owner preferred**.
3. Also on the **Permissions** tab, add an IAM policy to the bucket as described in Access policies for CRLs in Amazon S3 (p. 59).

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 Procedure for creating a CA (CLI) (p. 70).

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.

Encrypting Your CRLs

You can optionally configure encryption on the Amazon S3 bucket containing your CRLs. ACM Private CA supports two encryption modes for assets in Amazon 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
ACM 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 ACM 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 ACM 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 ACM 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
```

Configuring a Custom URL for ACM Private CA OCSP

Note
This topic is for customers who want to customize the public URL of the OCSP responder endpoint for branding or other purposes. If you plan to use the default configuration of ACM Private CA managed OCSP, you can skip this topic and follow the configuration instructions in Configure revocation (p. 68).

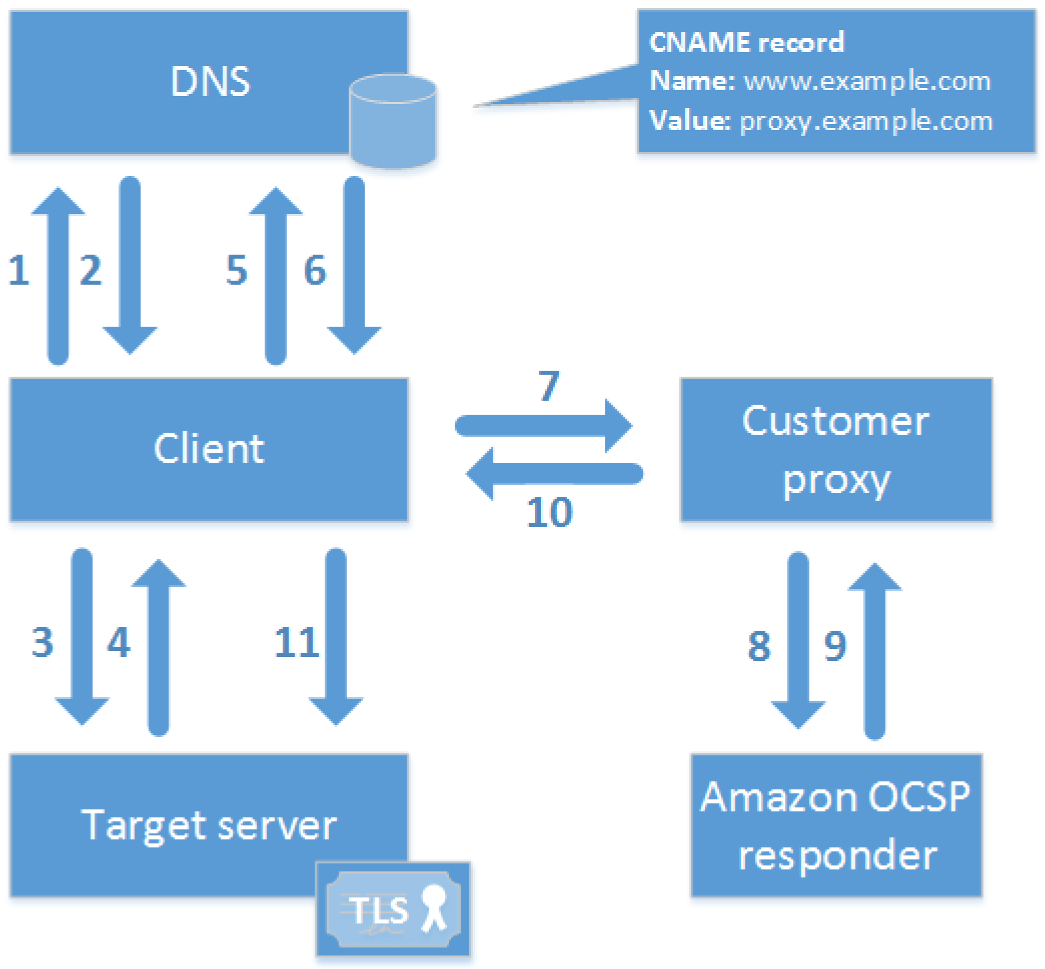
By default, when you enable OCSP for ACM Private CA, each certificate that you issue contains the URL for the AWS OCSP responder. This allows clients requesting a cryptographically secure connection to send OCSP validation queries directly to AWS. However, in some cases it might be preferable to state 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 (p. 68) and Planning a certificate revocation list (CRL) (p. 57).

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 the section called "Example 2: Create a CA with OCSP and a custom CNAME enabled" in Procedure for creating a CA (CLI) (p. 70).
- **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 the section called "Example 2: Create a CA with OCSP and a custom CNAME enabled" in Procedure for creating a CA (CLI) (p. 70).
- **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.
    - 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.

Planning for resilience

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. ACM Private CA is available in multiple Regions, which allows you to create redundant CAs in multiple Regions. The ACM 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 two redundant subordinate CAs that chain to the same root CA. 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.

Private CA administration

Using ACM Private CA, 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. ACM 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 ACM 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 ACM Private CA using the AWS Management Console, the AWS CLI, and the ACM 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 Certificate Manager Private Certificate Authority API Reference. For Java examples that show you how to use the API, see Using the ACM Private CA API (Java examples) (p. 146).

Topics

Creating a 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 Updating your private CA (p. 89).

For information about using a CA to sign end-entity certificates for your users, devices, and applications, see Issuing private end-entity certificates (p. 102).

> **Note**
> Your account is charged a monthly price for each private CA starting from the time that you create it.
> For the latest ACM Private CA pricing information, see the ACM Pricing page on the AWS website. You can also use the AWS pricing calculator to estimate costs.

Topics

Procedure for creating a CA (console)

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 ACM 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**.

1. Select CA type

On the **Select the certificate authority (CA) type** page, choose the type of private certificate authority that you want to create.

- Choosing **Root CA** 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 CA** creates a CA that must be signed by a parent CA above it in the hierarchy. Subordinate CAs are typically be used to create other subordinate CAs or to issue end-entity certificates to users, computers, and applications.

  Note
  ACM Private CA provides an automated signing process when your subordinate CA's parent CA is also hosted by ACM 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, ACM 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 Installing a subordinate CA certificate signed by an external parent CA (p. 82).

After choosing a CA type, choose **Next**.

2. Configure CA subject name

Under **Subject distinguished name options**, configure the subject name of your private CA. You must enter at least one of the following values:

- **Organization (O)**
- **Organization Unit (OU)**
- **Country name (C)**
- **State or province name**
- **Locality name**
- **Common Name (CN)**

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.

When done, choose **Next**.

3. Configure CA key algorithm

Under **Key algorithm options**, choose the key algorithm and the bit-size of the key. The default value is an RSA algorithm with a 2048-bit key length. If you expand the **Advanced** options, you can choose from the following algorithms:

- RSA 2048
- RSA 4096
- ECDSA P256
- ECDSA P384

Make a selection and then choose **Next**.

4. Configure revocation

On the **Configure revocation - *optional*** page, under **Configure revocation options** there are two methods for sharing revocation status with clients that use your certificates: configuring a certificate revocation list (CRL), or using Online Certificate Status Protocol (OCSP). These are displayed as follows:

- **CRL distribution**
- **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 (p. 222). Before completing this step, see Setting up a certificate revocation method (p. 56) 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 Updating your private CA (p. 89).

To configure a CRL

1. On the **Configure revocation - *optional*** page, choose **CRL distribution**.
2. To create a new Amazon S3 bucket for your CRL entries, choose **Yes** for the **Create a new S3 bucket** option and type a unique bucket name. (You do not need to include the path to the bucket.) Otherwise, choose **No** and choose an existing bucket from the list.

   If you choose **Yes**, ACM Private CA creates and attaches the required access policy (p. 59) to your bucket. If you choose **No**, you must attach a policy to your bucket before you can begin generating CRLs. Use one of the policy patterns described in Access policies for CRLs in Amazon S3 (p. 59). For information about attaching a policy, see How Do I Add an S3 Bucket Policy?

   > **Note**
   > When you are using the ACM 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 S3 bucket or account.
   > - You asked ACM Private CA to create an 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.

   - 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.
   - Type the number of days your CRL will remain valid. The default value is 7 days. For online CRLs, a validity period of 2-7 days is common. ACM Private CA tries to regenerate the CRL at the midpoint of the specified period.

To configure OCSP

1. On the **Configure revocation** ***- optional*** page, choose **OCSP**.
2. In the **Custom OCSP endpoint** ***- optional*** field, you can provide a non-default fully qualified domain name (FQDN) for your OCSP endpoint.

   When you supply an FQDN in this field, ACM 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 Configuring a Custom URL for ACM Private CA OCSP (p. 63).

   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://".

When you are done configuring revocation methods, choose **Next**.

5. Add tags

On the **Add tags** ***- optional*** page, 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 ACM Private CA tag parameters and for instructions on how to add tags to CAs after creation, see Adding tags to your private CA (p. 87).

Choose **Next**.

6. Configure CA permissions

Optionally delegate automatic renewal permissions to the ACM 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 ACM Private CA CreatePermission API or create-permission CLI command.

The default is to enable these permissions.

Choose **Next**.

7. Review and create

On the **Review and create** page, confirm that your configuration is correct. You can choose **Edit** next to each step to return to the step and change its settings. If you are creating your CA in the ap-northeast-3

Region, check the box to acknowledge that the key storage security standard is FIPS 140-2 Level 2 or higher. In all cases, check the box to acknowledge pricing information.

Note
For the latest ACM Private CA pricing information, see the ACM Pricing page on the AWS website. You can also use the AWS pricing calculator to estimate costs.

Finally, choose **Confirm and create**.

If you want to continue on to creating and installing a CA certificate, choose **Install CA certificate** in the **Success!** window. Otherwise choose **Cancel**, which takes you to a list of your **Private CAs**. You can finish setting up the CA by following the instructions at Creating and installing the CA certificate (p. 76).

Procedure for creating a CA (CLI)

Use the create-certificate-authority command to create a private CA. (If you want to modify an existing CA using the AWS CLI, see Updating a CA (CLI) (p. 94).) You must specify the CA configuration, the revocation configuration if you plan to use OCSP and/or a CRL, and the CA type. This information is contained in two files that you supply as arguments to the command. Optionally, you can also supply tags and 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 (p. 59).

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
- 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 Enabling the S3 Block Public Access (BPA) feature (p. 59).
- (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.

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) (p. 94).

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 same `ca_config.txt`:

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 ACM 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.json \
     --revocation-configuration file://revoke_config.json \
     --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:region:account:certificate-authority/CA_ID"
}
```

Use the following command to inspect the configuration of your CA.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:region:account:certificate-
authority/CA_ID" \
      --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 supply an FQDN in this field, ACM 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 Configuring a Custom URL for ACM Private CA OCSP (p. 63).

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:region:account:certificate-authority/CA_ID"
}
```

Use the following command to inspect the configuration of your CA.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:region:account:certificate-
authority/CA_ID" \
      --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":"DOC-EXAMPLE-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:region:account:certificate-authority/CA_ID"
}
```

Use the following command to inspect the configuration of your CA.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:region:account:certificate-
authority/CA_ID" \
      --output json
```

This description should contain the following section.

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "S3BucketName": "DOC-EXAMPLE-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":"DOC-EXAMPLE-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:region:account:certificate-authority/CA_ID"
}
```

Use the following command to inspect the configuration of your CA.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:region:account:certificate-
authority/CA_ID" \
      --output json
```

This description should contain the following section.

```
"RevocationConfiguration": {
   ...
   "CrlConfiguration": {
      "Enabled": true,
      "ExpirationInDays": 7,
      "CustomCname": "alternative.example.com",
      "S3BucketName": "DOC-EXAMPLE-BUCKET",
   ...
   }
}
```

Example 5: Create a CA with a custom subject name

In this example, we create a CA with a customized subject name. The defined name and associated object identifier (OID) is declared in the following configuration file.

File: ca_config.txt

```
{
  "KeyAlgorithm": "RSA_2048",
  "SigningAlgorithm": "SHA256WITHRSA",
  "Subject": {
    "CustomAttributes": [
      {
        "ObjectIdentifier": "1.3.6.1.4.1.37244.1.3",
        "Value": "ABCDABCD12341234"
      }
    ]
  }
}
```

Command

```
$ aws acm-pca create-certificate-authority \
      --certificate-authority-configuration fileb://ca_config.txt \
      --revocation-configuration file://revoke_config.txt \
      --certificate-authority-type ROOT
```

If successful, this command outputs the Amazon Resource Name (ARN) of the CA.

```
{
    "CertificateAuthorityArn":"arn:aws:acm-pca:region:account:certificate-authority/CA_ID"
}
```

View the CA, which contains a `CustomAttributes` section.

```
$ aws acm-pca describe-certificate-authority --certificate-authority-arn "arn:aws:acm-pca:region:account:certificate-authority/CA_ID"
```

```
{
    "CertificateAuthority": {
        "Status": "PENDING_CERTIFICATE",
        "LastStateChangeAt": 1234567891.234,
        "KeyStorageSecurityStandard": "FIPS_140_2_LEVEL_3_OR_HIGHER",
        "OwnerAccount": "account",
        "Arn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
        "CertificateAuthorityConfiguration": {
            "KeyAlgorithm": "RSA_2048",
            "SigningAlgorithm": "SHA256WITHRSA",
            "Subject": {
                "CustomAttributes": [
                    {
                        "ObjectIdentifier": "1.3.6.1.4.1.37244.1.3",
                        "Value": "ABCDABCD12341234"
                    }
                ]
            }
        },
        "Type": "ROOT",
        "RevocationConfiguration": {
            "CrlConfiguration": {
                "Enabled": false
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        },
        "CreatedAt": 1234567891.234
    }
}
```

Using AWS CloudFormation to create a CA

For information about creating a private CA using AWS CloudFormation, see ACM PCA Resource Type Reference in the *AWS CloudFormation User Guide*.

Creating and installing the CA certificate

Complete the following procedures to create and install your private CA certificate. Your CA will then be ready to use.

ACM Private CA supports three scenarios for installing a CA certificate:

- Installing a certificate for a root CA hosted by ACM Private CA
- Installing a subordinate CA certificate whose parent authority is hosted by ACM 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**.

If you are installing 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. Begin as required by the following conditions:

   - If you previously created a root CA and chose **Install CA certificate** from the **Success!** window, you were sent directly to the **Install root CA certificate** wizard.
   - If you postponed certificate installation, open the ACM Private CA console at https://console.aws.amazon.com/acm-pca/home. On the **List certificate authorities** page, choose a root CA with status **Pending Certificate** or **Active**. Then choose **Actions**, **Install CA Certificate** to open the **Install root CA certificate** wizard.
2. In the section **Specify the root CA certificate parameters**, specify the following certificate parameters:

   - **Validity value** and **Validity type** — Specifies the time until the CA certificate expires. The value is an integer; the type is **Years**, **Months**, or **Days**. The ACM 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.

   Choose **Next**.
3. On the **Review, generate, and install root CA certificate** page, confirm that the configuration is correct and choose **Confirm and install**. ACM Private CA exports a CSR for your CA and issues a self-signed root CA certificate using your CA and a root CA template (p. 113). ACM Private CA then imports the self-signed root CA certificate.

   The **List certificate authorities** page 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 list.

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:region:account:certificate-
authority/CA_ID \
     --output text \
     --endpoint service_endpoint \
     --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 ACM 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:region:account:certificate-authority/CA_ID
 \
 --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 ACM 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:region:account:certificate-authority/CA_ID \
 --output json
```

The status now appears as ACTIVE.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
        "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": "DOC-EXAMPLE-BUCKET1"
            },
            "OcspConfiguration": {
                "Enabled": false
            }
        }
    }
}
```

If you are installing a subordinate CA certificate hosted by ACM Private CA

You can use the AWS Management Console to create and install a certificate for your ACM Private CA hosted subordinate CA.

To create and install a certificate for your ACM Private CA hosted subordinate CA

1. Begin as required by the following conditions:

- If you previously created a subordinate CA and chose **Install CA certificate** from the **Success!** window, you were sent directly to the **Install subordinate CA certificate** wizard.
- If you postponed certificate installation, open the ACM Private CA console at https://console.aws.amazon.com/acm-pca/home. On the **List certificate authorities** page, choose a subordinate CA with status **Pending Certificate** or **Active**. Then choose **Actions**, **Install CA Certificate** to open the **Install subordinate CA certificate** wizard.

2. On the **Install subordinate CA certificate** page, choose **ACM Private CA** to install a certificate that is managed by ACM Private CA. Then choose **Next**.
3. On the **Configure CA certificate** page, under **Select parent ACM Private CA**, choose a CA from the **Parent private CA** list.
4. Under **Specify the subordinate CA certificate parameters**, specify the following certificate parameters:

   - **Validity value** and **Validity type** — Specifies the time until the CA certificate expires. The value is an integer; the type is **Years**, **Months**, or **Days**. The ACM Private CA default validity period for a root CA certificate is 10 years.

     Note
     The expiration date of the subordinate CA certificate cannot be later than the expiration date of the parent CA certificate.
   - **Signature algorithm** — This specifies the signing algorithm to use when the subordinate CA issues new certificates. Only algorithms compatible with the parent CA certificate are offered as options.
   - **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** — 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 Understanding certificate templates (p. 113).

   Choose **Next**.
5. On the **Review and generate** page, confirm that your configuration is correct and choose **Confirm and install**. ACM Private CA exports a CSR for your subordinate CA, issues a CA certificate from the parent CA and template you selected, and imports the CA certificate.

Installing a subordinate CA certificate signed by an external parent CA

After you create a subordinate private CA as described in Procedure for creating a CA (console) (p. 66) or Procedure for creating a CA (CLI) (p. 70), 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 externally signed CA are outside the scope of this guide.

When you have the subordinate CA and the external parent CA, you need to complete the following tasks:

1. Obtain a certificate signing request (CSR) from ACM Private CA.
2. Submit the CSR to your external trust services provider and obtain a signed CA certificate along with any chain certificates.
3. Import the CA certificate and chain into ACM Private CA to activate your subordinate CA.

For detailed procedures, see Signing private CA certificates with an external CA (p. 213).

Controlling access to a private CA

Any user with the necessary permissions on a private CA from ACM 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 (p. 17).

Authorized users, whether single-account or cross-account, can use ACM Private CA or AWS Certificate Manager resources when issuing certificates. Certificates that are issued from the ACM 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.

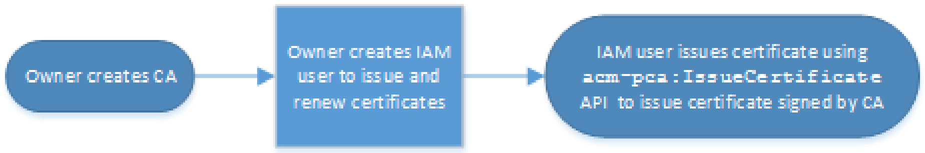
Topics

Create single-account permissions for an IAM user

When the CA administrator (that is, the owner of the CA) and the certificate issuer reside in a single AWS account, a best practice (p. 222) 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 ACM Private CA, along with example permissions, see Understanding resources, ownership, and permissions policies (p. 8).

Single-account case 1: Issuing an unmanaged certificate

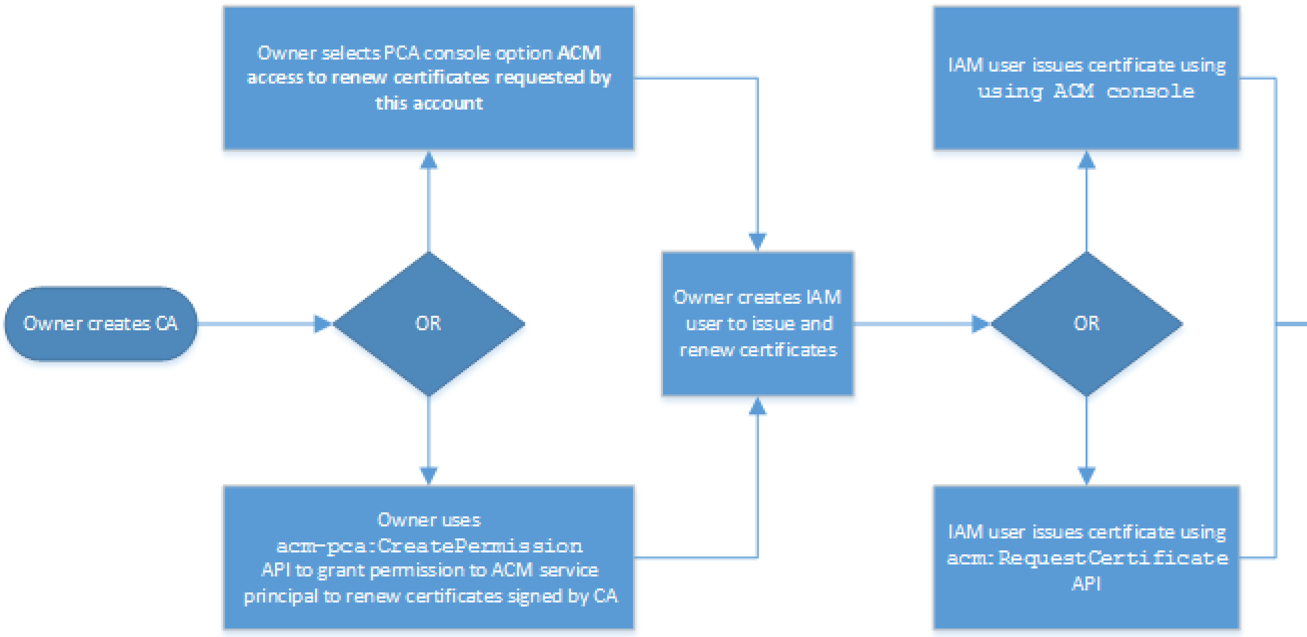
In this case, the account owner creates a private CA and then creates an IAM user with permission to issue certificates signed by the private CA. The IAM user issues a certificate by calling the ACM Private CA `IssueCertificate` API.


Owner creates CA
Owner creates IAM user to issue and renew certificates
IAM user issues certificate using acm-pca:IssueCertificate API to issue certificate signed by CA

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 ACM 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 (p. 69) 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.


Owner creates CA
OR
Owner selects PCA console option ACM access to renew certificates requested by this account
Owner uses acm-pca:CreatePermission API to grant permission to ACM service principal to renew certificates signed by CA
Owner creates IAM user to issue and renew certificates
OR
IAM user issues certificate using using ACM console
IAM user issues certificate using acm:RequestCertificate API

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 (p. 85).

Renewal permissions can be delegated during private CA creation (p. 66) or altered anytime after as long as the CA is in the `ACTIVE` state.

You can manage private CA permissions from the ACM Private CA Console, the AWS Command Line Interface (AWS CLI), or the ACM Private CA API:

To assign private CA permissions to ACM (console)

1. Sign in to your AWS account and open the ACM 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 ACM 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.

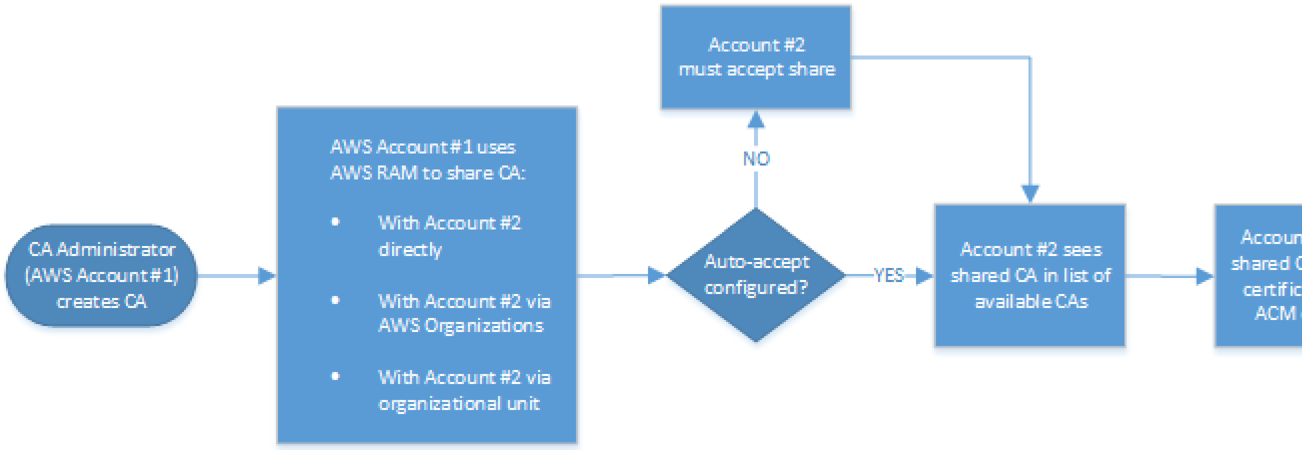
  Note
  You can easily open the RAM console by choosing a CA and then choosing **Actions**, **Manage resource shares**.

- 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.

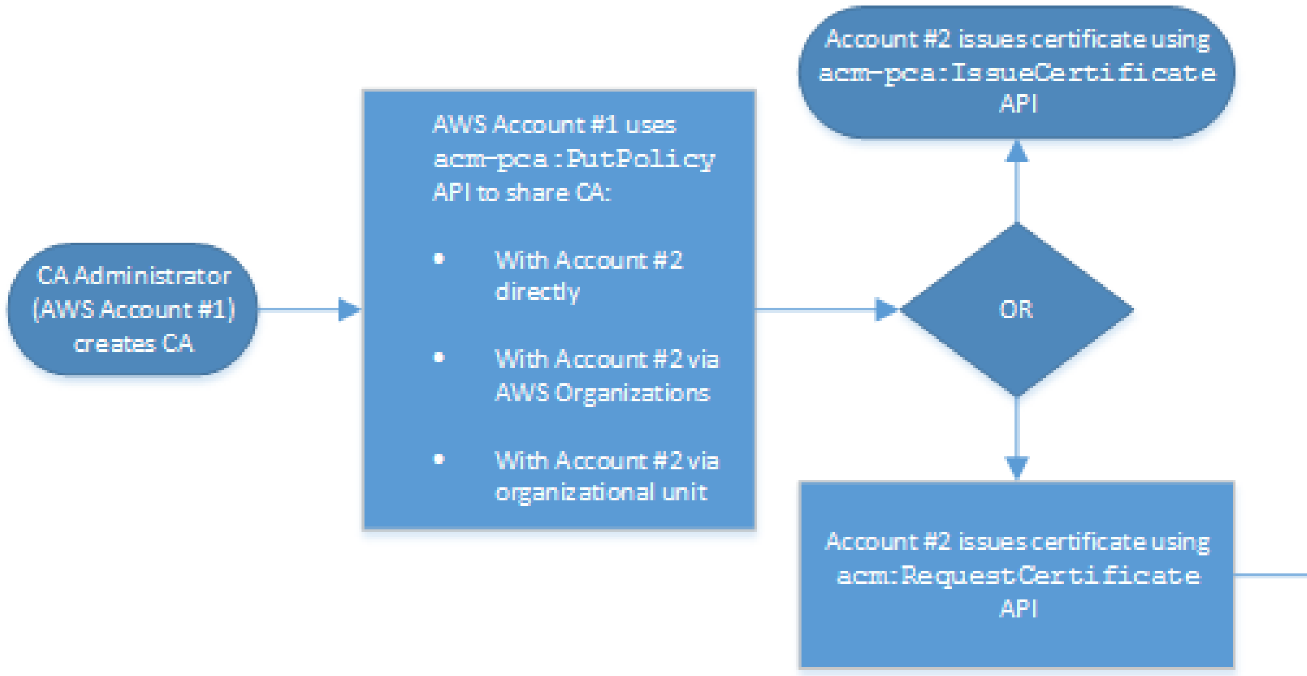

CA Administrator (AWS Account #1) creates CA
AWS Account #1 uses AWS RAM to share CA:
• With Account #2 directly
• With Account #2 via AWS Organizations
• With Account #2 via organizational unit
Auto-accept configured?
NO
Account #2 must accept share
YES
Account #2 sees shared CA in list of available CAs
Accoun
shared C
certific
ACM

After RAM shares a resource through AWS Organizations, the recipient principal must accept the share 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 allows it to make unattended certificate calls on ACM 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 ACM Private CA API. All of these operations can also be carried out using the corresponding AWS CLI commands.


CA Administrator (AWS Account #1) creates CA
AWS Account #1 uses acm-pca:PutPolicy API to share CA:
• With Account #2 directly
• With Account #2 via AWS Organizations
• With Account #2 via organizational unit
OR
Account #2 issues certificate using acm-pca:IssueCertificate API
Account #2 issues certificate using acm:RequestCertificate API

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 ACM 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.

Adding tags to 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 ACM 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 custom tags to 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. You can add tags to a CA during creation or anytime after. For more information, see Create a Private CA (p. 66).

Other AWS resources also support tagging. You can assign the same tag to different resources to indicate whether 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 for Linux Instances.

The following basic restrictions apply to ACM 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.
- ACM Private CA tags are not available for use in the AWS Management Console' Resource Groups and Tag Editor in the AWS Management Console.

You can tag a private CA from the ACM Private CA Console, the AWS Command Line Interface (AWS CLI), or the ACM Private CA API.

To tag a private CA (console)

1. Sign in to your AWS account and open the ACM 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
```

Updating your private CA

After creating a private CA, you can update its status or change its revocation configuration (p. 56). This topic provides details about CA status and the CA lifecycle, along with examples of console and CLI updates to CAs.

Updating CA status

The status of a CA that is managed by ACM 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.

ACM 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	Validate certs with OCSP	Generate CRLs	Generate audits	Certificates can be revoked	You are billed for the CA
`CREATING` – The CA is being created.	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	Yes
`ACTIVE`	Yes	Yes	Yes	Yes	Yes	Yes
`DISABLED` – You have manually disabled the CA.	No	Yes	Yes	Yes	Yes No	Yes
`EXPIRED` – The CA certificate has expired.**	No	No	No	No	Yes No	Yes
`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 seven to 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 Deleting your private CA (p. 99).					No

* To complete activation, you need to generate a CSR, get a signed CA certificate from a CA, and import the certificate into ACM 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 Creating and installing the CA certificate (p. 76).

** You cannot directly change the status of an expired CA. If you import a new certificate for the CA, ACM Private CA resets the status to `ACTIVE` unless it was set to `DISABLED` after 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 (p. 84).
- 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 ListCertificate Authorities` 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.

CA status and CA lifecycle

The following diagram illustrates the CA lifecycle as an interaction of management actions with CA status.

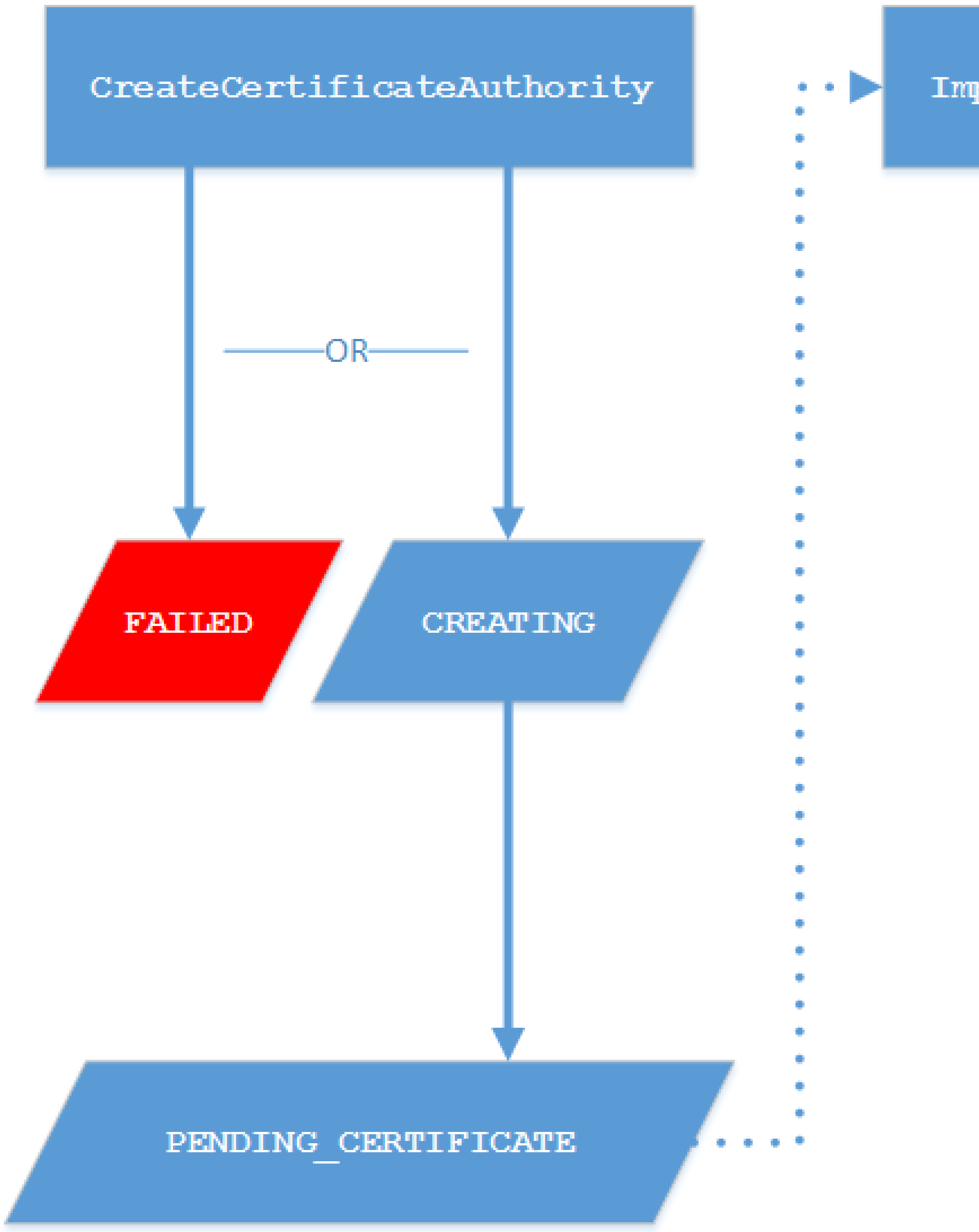

CreateCertificateAuthority
Im
OR
FAILED
CREATING
PENDING_CERTIFICATE

Diagram key

■	▰	→	⇢
Management action	CA status	Action results in a state change.	New state enables new action.

At the top of the diagram, management actions are applied through the ACM 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.

Updating a CA (console)

The following procedures show how to update existing CA configurations using the AWS Management Console.

To update CA status (console)

Complete the following steps to update the status of a CA.

1. Sign in

Sign in to your AWS account and open the ACM Private CA console at https://console.aws.amazon.com/acm-pca/home.

3. Select a CA

On the **Private certificate authorities** page, choose your private CA from the list.

4. Select an action

On the **Actions** menu, choose **Disable** to disable a private CA that is currently active or choose **Enable** to set the CA status to active.

Updating a CA's revocation configuration (console)

You can update the revocation configuration (p. 56) 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 values:

- 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 values:

- Whether the private CA generates a certificate revocation list (CRL)

- The number of days before a CRL expires. Note that ACM 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 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.

1. Sign in

Sign in to your AWS account and open the ACM Private CA console at https://console.aws.amazon.com/acm-pca/home.

3. Select a CA

On the **Private certificate authorities** page, choose your private CA from the list.

4. Select an action

On the **Actions** menu, choose **Update CA revocation**.

5. Update revocation

The **Update CA revocation configuration** page presents two options for sharing revocation status with clients that use your certificates: using Online Certificate Status Protocol (OCSP), or configuring a certificate revocation list (CRL), displayed as follows:

- **CRL distribution**
- **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 (p. 222). Before completing this step, see Setting up a certificate revocation method (p. 56) 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 **CRL distribution**.
2. To create a new Amazon S3 bucket for your CRL entries, choose **Yes** for the **Create a new S3 bucket** option and type a unique bucket name. (You do not need to include the path to the bucket.) Otherwise, choose **No** and choose an existing bucket from the list.

   If you choose **Yes**, ACM Private CA creates and attaches the required access policy (p. 59) to your bucket. If you choose **No**, you must attach a policy to your bucket before you can begin generating CRLs. Use one of the policy patterns described in Access policies for CRLs in Amazon S3 (p. 59). For information about attaching a policy, see How Do I Add an S3 Bucket Policy?

   > **Note**
   > When you are using the ACM 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 S3 bucket or account.
   > - You asked ACM Private CA to create an S3 bucket automatically.

In this situation, the console attempts, by default, to create a publicly accessible bucket, and 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.

   - 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.
   - Type the number of days your CRL will remain valid. The default value is 7 days. For online CRLs, a validity period of 2-7 days is common. ACM Private CA tries to regenerate the CRL at the midpoint of the specified period.

4. Choose **Update** when done.

To configure OCSP

1. On the **Certificate revocation** page, choose **OCSP**.
2. (Optional) In the **Custom OCSP name** field, provide a fully qualified domain name (FQDN) for your OCSP endpoint.

   When you supply an FQDN in this field, ACM 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 Configuring a Custom URL for ACM Private CA OCSP (p. 63).

   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 **Update** when done.

Updating a CA (CLI)

The following procedures show how to update the status and revocation configuration (p. 56) 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:region:account:certificate-
authority/CA_ID" \
      --output json
```

This results in output similar to the following.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
        "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": "DOC-EXAMPLE-BUCKET1"
   },
            "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:region:account:certificate-
authority/CA_ID \
      --status "ACTIVE"
```

Inspect the new status of the CA.

```
$ aws acm-pca describe-certificate-authority \
      --certificate-authority-arn "arn:aws:acm-pca:region:account:certificate-
authority/CA_ID" \
      --output json
```

The status now appears as `ACTIVE`.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
        "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": "DOC-EXAMPLE-BUCKET1"
            },
            "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:region:account:certificate-authority/CA_ID
 --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:region:account:certificate-authority/CA_ID",
        "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":"DOC-EXAMPLE-BUCKET"
   }
}
```

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:region:account:certificate-
authority/CA_ID \
      --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:region:account:certificate-authority/CA_ID
 --output json
```

The output confirms that CA is now configured to use a CRL.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
        "CreatedAt": "2021-03-08T14:36:26.449000-08:00",
        "LastStateChangeAt": "2021-03-08T14:50:52.224000-08:00",
        "Type": "ROOT",
        "Serial": "serial_numbner",
        "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": "DOC-EXAMPLE-BUCKET1",
            },
            "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:region:account:certificate-
authority/CA_ID \
      --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:region:account:certificate-authority/CA_ID
 --output json
```

The output confirms that CA is now configured to use OCSP.

```
{
    "CertificateAuthority": {
        "Arn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID",
        "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.

Deleting your private CA

You can delete a private CA 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.

ACM 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 (p. 39) 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 to 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 Restoring a private CA (p. 100).
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 (p. 4).

You can delete a private CA from the AWS Management Console or AWS CLI.

To delete a private CA (console)

1. Sign in to your AWS account and open the ACM 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 seven to 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
```

Restoring 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 period identifies the number of days, from 7 to 30, that the private CA remains restorable. At the end of that period, the private CA is permanently deleted. For more information, see Deleting your private CA (p. 99). 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 (p. 4).

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 ACM 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.

Restoring 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 days to 30. 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
```

Certificate administration

After you have created and activated a private certificate authority (CA) and configured access to it, you or your authorized users can perform the tasks discussed in this section. 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 Controlling access to a private CA (p. 83).

Topics

Issuing private end-entity certificates

With a private CA in place, you can request private end-entity certificates from either AWS Certificate Manager (ACM) or ACM Private CA. The capabilities of both services are compared in the following table.

Capability	ACM	ACM 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 though ACM
Console support	✓	Not supported
API support	✓	✓
CLI support	✓	✓

When ACM 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 (p. 128) template is applied by default. For more information about available certificate templates, see Understanding certificate templates (p. 113).

While ACM certificates are designed around public trust, ACM Private CA serves the needs of your private PKI. Consequently, you can configure certificates using the ACM 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 certificate using ACM 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 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*.

Issuing a standard certificate (AWS CLI)

You can use the ACM 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 ACM 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 (p. 39).

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 the required input files. This ensures that ACM Private CA parses the Base64-encoded 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, an end-entity certificate is issued by default.

```
$ aws acm-pca issue-certificate \
      --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-authority/CA_ID \
      --csr fileb://cert_1.csr \
      --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
ACM 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 Issuing a standard certificate (AWS CLI) (p. 103), 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:

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

Subject name	Object ID
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

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 fileb://api_passthrough_config.txt \
      --template-arn arn:aws:acm-pca:::template/BlankEndEntityCertificate_APIPassthrough/V1
 \
      --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID
```

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:region:account:certificate-
authority/CA_ID \
      --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. For more information see Example 5: Create a CA with a custom subject name (p. 75).

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 Issuing a standard certificate (AWS CLI) (p. 103).

The sample configuration file `api_passthrough_config.txt` contains the following code:

```
{
  "Extensions": {
    "CustomExtensions": [
      {
        "ObjectIdentifier": "2.5.29.30",
        "Value": "MBWgEzARgg8ucGVybW10dGVkLnRlc3Q=",
        "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 fileb://api_passthrough_config.txt \
      --template-arn arn:aws:acm-pca:::template/EndEntityCertificate_APIPassthrough/V1 \
      --certificate-authority-arn arn:aws:acm-pca:region:account:certificate-
authority/CA_ID
```

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:region:account:certificate-
authority/CA_ID \
      --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
```

Retrieving a private certificate

You can use the ACM Private CA API and AWS CLI to issue a private certificate. If you do, you can use the AWS CLI or ACM 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 Using audit reports with your private CA (p. 39).

```
$ 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:region:account:certificate-
authority/CA_ID | \
      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 ACM 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:region:account:certificate-authority/CA_ID
 \
     | jq -r '.Certificate'
```

This command outputs the CA certificate in the following standard format.

```
-----BEGIN CERTIFICATE-----
...Base64-encoded certificate...
-----END CERTIFICATE----
```

Listing 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 ACM Private CA audit reports, see Using audit reports with your private CA (p. 39). 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:region:account:certificate-authority/CA_ID
 \
     --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"
}
```

Revoking a private certificate

You can revoke an ACM Private CA certificate using the revoke-certificate AWS CLI command or the RevokeCertificate API action. A certificate may need to be revoked prior to 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.

ACM 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 ACM Private CA, see Setting up a certificate revocation method (p. 56)

Revoked certificates are always recorded in ACM Private CA audit reports.

Note
Cross-account (p. 16) certificate issuers cannot revoke the certificates that they issue. The CA owner must perform revocation.

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:region:account:certificate-authority/CA_ID
 \
     --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, ACM 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 Planning a certificate revocation list (CRL) (p. 57).

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 Using audit reports with your private CA (p. 39).

```
[
   {
      "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=Sales,O
 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.example
 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"
   }
]
```

Automating export of a renewed certificate

When you use ACM 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 be 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.

Understanding certificate templates

ACM 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 (p. 128)
> - EndEntityClientAuthCertificate/V1 (p. 130)
> - EndEntityServerAuthCertificate/V1 (p. 132)
> - BlankEndEntityCertificate_APICSRPassthrough (p. 120)
>
> For more information, see Resource-based policies (p. 17).

Topics

Template varieties

ACM 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 Template order of operations (p. 119).
- **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 Template order of operations (p. 119).
- **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 Template order of operations (p. 119).

The tables below list all of the template types supported by ACM Private CA with links to their definitions.

Note
For information about template ARNs in GovCloud regions, see AWS Certificate Manager Private Certificate Authority in the *AWS GovCloud (US) User Guide*.

Base templates

Template Name	Template ARN	Certificate Type
CodeSigningCertificate/V1 (p. 126)	`arn:aws:acm-pca:::template/CodeSigningCertificate/V1`	Code signing
EndEntityCertificate/V1 (p. 128)	`arn:aws:acm-pca:::template/EndEntityCertificate/V1`	End-entity
EndEntityClientAuthCertificate/V1 (p. 130)	`arn:aws:acm-pca:::template/EndEntityClientAuthCertificate/V1`	End-entity
EndEntityServerAuthCertificate/V1 (p. 132)	`arn:aws:acm-pca:::template/EndEntityServerAuthCertificate/V1`	End-entity
OCSPSigningCertificate/V1 (p. 134)	`arn:aws:acm-pca:::template/OCSPSigningCertificate/V1`	OCSP signing
RootCACertificate/V1 (p. 136)	`arn:aws:acm-pca:::template/RootCACertificate/V1`	CA
SubordinateCACertificate_PathLen0/V1 (p. 137)	`arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0/V1`	CA
SubordinateCACertificate_PathLen1/V1 (p. 139)	`arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1/V1`	CA
SubordinateCACertificate_PathLen2/V1 (p. 141)	`arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2/V1`	CA
SubordinateCACertificate_PathLen3/V1 (p. 143)	`arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3/V1`	CA

CSRPassthrough templates

Template Name	Template ARN	Certificate Type
BlankEndEntityCertificate_CSRPassthrough (p. 120)	`arn:aws:acm-pca:::template/BlankEndEntityCertificate_CSRPassthrough/V1`	End-entity

Template Name	Template ARN	Certificate Type
CodeSigningCertificate_CSRPassthrough/V1 (p. 128)	`arn:aws:acm-pca:::template/CodeSigningCertificate_CSRPassthrough/V1`	Code signing
EndEntityCertificate_CSRPassthrough/V1 (p. 130)	`arn:aws:acm-pca:::template/EndEntityCertificate_CSRPassthrough/V1`	End-entity
EndEntityClientAuthCertificate_CSRPassthrough/V1 (p. 132)	`arn:aws:acm-pca:::template/EndEntityClientAuthCertificate_CSRPassthrough/V1`	End-entity
EndEntityServerAuthCertificate_CSRPassthrough/V1 (p. 134)	`arn:aws:acm-pca:::template/EndEntityServerAuthCertificate_CSRPassthrough/V1`	End-entity
OCSPSigningCertificate_CSRPassthrough/V1 (p. 136)	`arn:aws:acm-pca:::template/OCSPSigningCertificate_CSRPassthrough/V1`	OCSP signing
SubordinateCACertificate_PathLen0_CSRPassthrough/V1 (p. 139)	`arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen0_CSRPassthrough/V1`	CA
BlankSubordinateCACertificate_PathLen0_CSRPassthrough (p. 121)	`arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen0_CSRPassthrough/V1`	CA
SubordinateCACertificate_PathLen1_CSRPassthrough/V1 (p. 141)	`arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen1_CSRPassthrough/V1`	CA
BlankSubordinateCACertificate_PathLen1_CSRPassthrough (p. 123)	`arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_CSRPassthrough/V1`	CA
BlankSubordinateCACertificate_PathLen1_APICSRPassthrough (p. 123)	`arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/V1`	CA
SubordinateCACertificate_PathLen2_CSRPassthrough/V1 (p. 143)	`arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen2_CSRPassthrough/V1`	CA
BlankSubordinateCACertificate_PathLen2_CSRPassthrough (p. 124)	`arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen2_CSRPassthrough/V1`	CA
SubordinateCACertificate_PathLen3_CSRPassthrough/V1 (p. 145)	`arn:aws:acm-pca:::template/SubordinateCACertificate_PathLen3_CSRPassthrough/V1`	CA
BlankSubordinateCACertificate_PathLen3_CSRPassthrough (p. 125)	`arn:aws:acm-pca:::template/BlankSubordinateCACertificate_PathLen3_CSRPassthrough/V1`	CA

APIPassthrough templates

Template Name	Template ARN	Certificate Type
BlankEndEntityCertificate_APIPassthrough (p. 119)	arn:aws:acm-pca:::template/ BlankEndEntityCertificate_APIPassthrough/ V1	End-entity
CodeSigningCertificate_APIPassthrough (p. 127)	arn:aws:acm-pca:::template/ CodeSigningCertificate_APIPassthrough/ V1	Code signing
EndEntityCertificate_APIPassthrough (p. 129)	arn:aws:acm-pca:::template/ EndEntityCertificate_APIPassthrough (p. 129)/ V1	End-entity
EndEntityClientAuthCertificate_APIPassthrough (p. 131)	arn:aws:acm-pca:::template/ EndEntityClientAuthCertificate_APIPassthrough/ V1	End-entity
EndEntityServerAuthCertificate_APIPassthrough (p. 133)	arn:aws:acm-pca:::template/ EndEntityServerAuthCertificate_APIPassthrough/ V1	End-entity
OCSPSigningCertificate_APIPassthrough (p. 135)	arn:aws:acm-pca:::template/ OCSPSigningCertificate_APIPassthrough/ V1	OCSP signing
RootCACertificate_APIPassthrough (p. 137)	arn:aws:acm-pca:::template/ RootCACertificate_APIPassthrough/ V1	CA
SubordinateCACertificate_PathLen0_APIPassthrough (p. 138)	arn:aws:acm-pca:::template/ SubordinateCACertificate_PathLen0_APIPassthrough/ V1	CA
BlankSubordinateCACertificate_PathLen0_APIPassthrough (p. 122)	arn:aws:acm-pca:::template/ BlankSubordinateCACertificate_PathLen0_APIPassthrough/ V1	CA
SubordinateCACertificate_PathLen1_APIPassthrough (p. 140)	arn:aws:acm-pca:::template/ SubordinateCACertificate_PathLen1_APIPassthrough/ V1	CA
BlankSubordinateCACertificate_PathLen1_APIPassthrough (p. 122)	arn:aws:acm-pca:::template/ BlankSubordinateCACertificate_PathLen1_APIPassthrough/ V1	CA
SubordinateCACertificate_PathLen2_APIPassthrough (p. 142)	arn:aws:acm-pca:::template/ SubordinateCACertificate_PathLen2_APIPassthrough/ V1	CA
BlankSubordinateCACertificate_PathLen2_APIPassthrough (p. 124)	arn:aws:acm-pca:::template/ BlankSubordinateCACertificate_PathLen2_APIPassthrough/ V1	CA
SubordinateCACertificate_PathLen3_APIPassthrough (p. 144)	arn:aws:acm-pca:::template/ SubordinateCACertificate_PathLen3_APIPassthrough/ V1	CA

Template Name	Template ARN	Certificate Type
BlankSubordinateCACertificate_PathLen3_APIPassthrough (p. 125)	arn:aws:acm-pca:::template/ BlankSubordinateCACertificate_PathLen3_APIPassthrough/ V1	CA

APICSRPassthrough templates

Template Name	Template ARN	Certificate Type
BlankEndEntityCertificate_APICSRPassthrough (p. 120)	arn:aws:acm-pca:::template/ BlankEndEntityCertificate_APICSRPassthrough/ V1	End-entity
CodeSigningCertificate_APICSRPassthrough (p. 127)	arn:aws:acm-pca:::template/ CodeSigningCertificate_APICSRPassthrough/ V1	Code signing
EndEntityCertificate_APICSRPassthrough (p. 129)	arn:aws:acm-pca:::template/ EndEntityCertificate_APICSRPassthrough/ V1	End-entity
EndEntityClientAuthCertificate_APICSRPassthrough (p. 131)	arn:aws:acm-pca:::template/ EndEntityClientAuthCertificate_APICSRPassthrough/ V1	End-entity
EndEntityServerAuthCertificate_APICSRPassthrough (p. 133)	arn:aws:acm-pca:::template/ EndEntityServerAuthCertificate_APICSRPassthrough/ V1	End-entity
OCSPSigningCertificate_APICSRPassthrough (p. 135)	arn:aws:acm-pca:::template/ OCSPSigningCertificate_APICSRPassthrough/ V1	OCSP signing
SubordinateCACertificate_PathLen0_APICSRPassthrough (p. 138)	arn:aws:acm-pca:::template/ SubordinateCACertificate_PathLen0_APICSRPassthrough/ V1	CA
BlankSubordinateCACertificate_PathLen0_APICSRPassthrough (p. 121)	arn:aws:acm-pca:::template/ BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/ V1	CA
SubordinateCACertificate_PathLen1_APICSRPassthrough (p. 140)	arn:aws:acm-pca:::template/ SubordinateCACertificate_PathLen1_APICSRPassthrough/ V1	CA
BlankSubordinateCACertificate_PathLen1_APICSRPassthrough (p. 123)	arn:aws:acm-pca:::template/ BlankSubordinateCACertificate_PathLen1_APICSRPassthrough/ V1	CA
SubordinateCACertificate_PathLen2_APICSRPassthrough (p. 142)	arn:aws:acm-pca:::template/ SubordinateCACertificate_PathLen2_APICSRPassthrough (p. 142 V1	CA
BlankSubordinateCACertificate_PathLen2_APICSRPassthrough (p. 124)	arn:aws:acm-pca:::template/ BlankSubordinateCACertificate_PathLen2_APICSRPassthrough/ V1	CA

Template Name	Template ARN	Certificate Type
SubordinateCACertificate_PathLen3_APICSRPassthrough (p. 144)	arn:aws:acm-pca:::template/ SubordinateCACertificate_PathLen3_APICSRPassthrough/ V1	CA
BlankSubordinateCACertificate_PathLen3_APICSRPassthrough (p. 126)	arn:aws:acm-pca:::template/ BlankSubordinateCACertificate_PathLen3_APICSRPassthrough/ V1	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 (p. 131) 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 (p. 131) 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.

Template definitions

The following sections provide configuration details about supported ACM 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 ACM 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.

Blank passthrough templates are useful for issuing 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, keyEnciphermet, 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_CSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition (p. 119).

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.

BlankEndEntityCertificate_APICSRPassthrough/V1 definition

For general information about blank templates, see BlankEndEntityCertificate_APIPassthrough/V1 definition (p. 119).

BlankEndEntityCertificate_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]
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 (p. 119).

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 (p. 119).

BlankSubordinateCACertificate_PathLen0_APICSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from API or CSR]

X509v3 Parameter	Value
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 (p. 119).

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 (p. 119).

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 (p. 119).

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 (p. 119).

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 (p. 119).

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 (p. 119).

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 (p. 119).

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`
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 (p. 119).

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 (p. 119).

BlankSubordinateCACertificate_PathLen3_CSRPassthrough/V1

X509v3 Parameter	Value
Subject alternative name	[Passthrough from CSR]
Subject	[Passthrough from CSR]

X509v3 Parameter	Value
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 (p. 119).

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 ACM 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 ACM 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]

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_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, ACM 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, ACM 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]

X509v3 Parameter	Value
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]
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, ACM 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`

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, 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, ACM 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]

X509v3 Parameter	Value
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]
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, ACM 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
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, ACM 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`
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, ACM 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, ACM 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
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, ACM 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, ACM 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 (p. 231)) 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`
Authority key identifier	[SKI from CA certificate]
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	[SKI from CA certificate]
Subject key identifier	[Derived from CSR]
Key usage	Critical, digital signature, keyCertSign, CRL sign
CRL distribution points*	N/A

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]

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 certificates 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]
Basic constraints	Critical, `CA:TRUE`, `pathlen: 0`

X509v3 Parameter	Value
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, ACM 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 ACM 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]
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 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_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]

X509v3 Parameter	Value
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]

* 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, ACM 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 ACM 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]
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]

X509v3 Parameter	Value
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]

* 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, ACM 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 ACM 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]
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]

X509v3 Parameter	Value
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]

* 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, ACM 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 ACM 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]
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.

Using the ACM Private CA API (Java examples)

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

In addition to the HTTP API, you can use the AWS SDKs and command line tools to interact with ACM 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 ACM PCA API.

The GetCertificateAuthorityCsr (p. 175), GetCertificate (p. 172), and DescribeCertificateAuthorityAuditReport (p. 170) 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.

Topics

- Create certificates with custom extensions (p. 202)

Create and activate a root CA programmatically

This Java sample shows how to activate a root CA using the following ACM 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 ACM 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:region:account:certificate-authority/CA_ID";

        // 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
- 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:region:account:certificate-authority/CA_ID
```

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:region:account:certificate-
authority/CA_ID");

      // 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:region:account:certificate-
authority/CA_ID");

      // 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 AWS 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 Restoring (p. 100).

For a Java example that shows you how to use the RestoreCertificateAuthority operation, see RestoreCertificateAuthority (p. 189).

```
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:region:account:certificate-
authority/CA_ID");

      // 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:region:account:certificate-
authority/CA_ID");

      // 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.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:region:account:certificate-
authority/CA_ID");

      // 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 AWS 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;
      }
   }
}
```

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` – ACM 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:region:account:certificate-
authority/CA_ID");

      // 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:region:account:certificate-
authority/CA_ID");

      // 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:region:account:certificate-
authority/CA_ID");

      // 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:region:account:certificate-
authority/CA_ID");

      // 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:region:account:certificate-
authority/CA_ID");

      // 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:region:account:certificate-authority/CA_ID");

        // 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 ACM 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:region:account:certificate-
authority/CA_ID");
```

```
      // 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 Understanding certificate templates (p. 113).

```
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:region:account:certificate-
authority/CA_ID");

      // 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:region:account:certificate-
authority/CA_ID");

      // 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:region:account:certificate-
authority/CA_ID");

      // 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 ACM 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:region:account:certificate-authority/CA_ID");

        // 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 Restoring (p. 100). See also the DeleteCertificateAuthority (p. 164) 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:region:account:certificate-
authority/CA_ID");

      // 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. ACM 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:region:account:certificate-
authority/CA_ID");

      // 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:region:account:certificate-
authority/CA_ID");
      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:region:account:certificate-
authority/CA_ID");
      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:region:account:certificate-
authority/CA_ID");

      // 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 Template varieties (p. 114). For more information about using custom attrributes, see Issuing private end-entity certificates (p. 102) and Procedure for creating a CA (CLI) (p. 70).

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

```
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 Template varieties (p. 114). For more information about using custom extensions, see Issuing private end-entity certificates (p. 102).

Topics

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);
   }
}
```

Signing private CA certificates with an external CA

If your private CA hierarchy's root of trust must be a CA outside of ACM 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. Use this externally obtained CA to sign a private subordinate CA certificate that ACM Private CA manages.

Note
Procedures for creating or obtaining an externally signed CA are outside the scope of this guide.

Using an external parent CA with ACM Private CA also allows you to enforce CA name constraints. Name constraints are defined in the internet public key infrastructure (PKI) standard RFC 5280. The constraints provide a way for CA administrators to restrict subject names in certificates. For more information, see the Name Constraints section of RFC 5280.

Topics

Step 1: Getting a certificate signing request (CSR) from ACM Private CA

If you have created a private subordinate CA that you want to sign with an external CA, you must retrieve a certificate signing request (CSR). Then save it to a file. You can do this using the AWS Management Console or the AWS CLI as discussed in the procedures that follow.

How to obtain a CSR (console): Case 1

Use this procedure if you followed the steps to create a private CA (p. 66) in ACM Private CA and left the **Success** dialog box open. These procedures assume that while creating the CA, you specified that it was a subordinate CA.

To obtain a CSR (console): Case 1

1. Immediately after ACM Private CA has successfully created your private CA, in the **Success!** window, choose **Install CA certificate**.
2. Choose **External private CA** and **Next**.
3. On the **Export CSR** page, the console returns the CSR. Choose **Export CSR to a file** and save it locally.
4. If you cannot immediately perform the offline steps to obtain a signed certificate from your external signing authority, choose **Cancel**. Once you possess a signed certificate and a certificate chain, you can use the How to obtain a CSR (console): Case 2 (p. 214) procedure to import them into ACM Private CA.

   Otherwise, if you are ready, choose **Next**.
5. Follow the instructions in Step 2: Signing the private CA certificate (p. 215).

How to obtain a CSR (console): Case 2

Use this procedure if you followed the steps to create a private CA (p. 66) in ACM Private CA and closed the **Success!** window by choosing **Cancel**.

To obtain a CSR (console): Case 2

1. Sign in to your AWS account and open the ACM 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. Choose **Actions**, **Install CA certificate**.
4. On the **Install subordinate CA certificate** page, choose **External private CA** and **Next**.
5. The ACM Private CA console returns the CSR. Choose **Export CSR to a file** and save it locally.
6. Choose **Next**.
7. Follow the instructions in Step 2: Signing the private CA certificate (p. 215).

Retrieving a CSR (AWS CLI)

Use this procedure to retrieve a CSR using the AWS Command Line Interface.

To retrieve a CSR (AWS 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:region:account:certificate-
authority/CA_ID \
     --output text
```

2. Follow the instructions in Step 2: Signing the private CA certificate (p. 215).

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
```

Step 2: Signing the private CA certificate

After you create your private CA (p. 66) using ACM Private CA and obtain a certificate signing request (CSR) (p. 213), you must take the CSR to your external X.509 infrastructure. Use an intermediate or root CA to create your private CA certificate and sign it. Signing affirms the identity of the private CA within your organization. When you have completed this process, follow the instructions in Step 3: Import your private CA certificate into ACM Private CA (p. 217).

Important

- Details of your on-premises X.509 infrastructure and the CA hierarchy within it are beyond the scope of this guide. For more information, see Creating and signing a private CA certificate (p. 225).
- The validity period of a private CA is determined by the validity period you specify when you create the private CA certificate. Set the **Not Before** and **Not After** fields. Aside from enforcing the defined period, ACM Private CA does not restrict the lifetime of a CA.
- If you must create a CA certificate that effectively never expires, set the special value `99991231235959Z` in the **Not After** field. We do not recommend this as a general practice.

The signed certificate is typically returned to you as a base64-encoded PEM file or string. This is shown by the following example. If the certificate is encoded in a different format, you must convert it to PEM. Various OpenSSL commands are available to perform format conversion.

```
-----BEGIN CERTIFICATE-----
```

```
MIIDRzCCA..................
..........9YFbLXtPgZooy2IgZ
------END CERTIFICATE------
```

You can use the OpenSSL `x509` command to view the contents of your signed PEM format certificate.

```
openssl x509 -in path_to_certificate_file -text -noout
```

This command returns a certificate similar to the following example.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4122 (0x101a)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=Washington, L=Seattle, O=Example Company, OU=Corp,
 CN=www.example.com/emailAddress=corp@www.example.com
        Validity
            Not Before: Mar 29 19:28:43 2018 GMT
            Not After : Mar 26 19:28:43 2028 GMT
        Subject: O=Example Company, OU=Corporate Office, CN=Example Company 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)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                3D:5B:CC:96:FA:A5:91:37:2A:C9:97:22:F8:AF:10:A7:4E:E1:A0:6A
            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, Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         7e:4b:7c:ca:31:b2:66:25:eb:99:26:91:e2:77:1b:7c:2c:a5:
         d5:e4:ab:c3:98:2a:a2:d7:d9:3b:4a:89:27:cd:94:5c:50:fb:
         59:00:9b:13:56:08:da:87:3c:50:e4:eb:f9:b3:35:92:f8:72:
         d9:11:f0:1e:f3:3b:2e:f5:42:12:de:46:ce:36:ab:f7:b9:2f:
         7e:dd:50:47:49:ad:a6:ee:f6:67:b3:c6:2f:6c:7a:fe:16:9c:
         47:41:81:18:cd:ff:4e:b9:66:8b:ed:04:7a:d0:ce:cb:f3:88:
         c8:89:20:68:6a:2f:6c:3d:60:56:cb:5e:d3:e0:66:8a:32:d8:
         88:19
```

Step 3: Import your private CA certificate into ACM Private CA

After you create your private subordinate CA (p. 66), obtain the certificate signing request (CSR) (p. 213), and have your external root or subordinate CA sign the CA certificate (p. 215), you must import the signed certificate into ACM Private CA. After signing and importing the certificate, you can use your private subordinate CA to issue and revoke trusted private TLS certificates. These enable trusted communication between users, applications, computers, and other devices internal to your organization. The certificates cannot be publicly trusted.

You must also obtain the certificate chain that contains the certificate of the root or subordinate CA used to sign your private CA certificate as well as any preceding certificates. To create the chain, concatenate your root certificate, if available, and any subordinate certificates that you might have into a single file. You can use the `cat` command (Linux) to do so. Each certificate must directly certify the one preceding, and the entire chain must be PEM-formatted. The following example contains three certificates, but your PKI infrastructure might have more or fewer.

```
-----BEGIN CERTIFICATE-----
Base64-encoded intermediate CA certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Base64-encoded intermediate CA certificate
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
Base64-encoded root or intermediate CA certificate
-----END CERTIFICATE-----
```

Note
ACM Private CA securely generates and stores your certificate's private key. You never import an externally generated secret key.

Importing the Private CA certificate (console)

You can import a private CA certificate using the AWS Management Console.

To import the CA certificate (console)

1. If your console is still open to the **Import signed CA certificate** page, skip to step 7. Otherwise, continue.
2. Sign in to your AWS account and open the ACM Private CA console at https://console.aws.amazon.com/acm-pca/home.
3. On the **Private certificate authorities** page, choose your subordinate CA from the list of CAs. In the details panel below the list, the **Detailed status** is **Action required**.
4. Choose **Actions**, **Install CA certificate**.
5. Choose **External private CA** and **Next**.
6. On the **Export a certificate signing request (CSR)** page, choose **Next**.
7. On the **Import signed CA certificate** page, provide the required information.
   - For **Certificate body**, copy your signed private CA certificate into the text box or import it from a file.
   - For **Certificate chain**, copy the certificate chain into the text box or import it from a file.

Then choose **Next**.

8. On the **Review and install** page, choose **Confirm and install** to import the certificate.

Importing the private CA certificate (AWS CLI)

Before beginning, make sure that you have your signed CA certificate and your certificate chain in PEM formatted files.

To import the CA certificate (AWS CLI)

Use the import-certificate-authority-certificate command to import the private CA certificate into ACM Private CA.

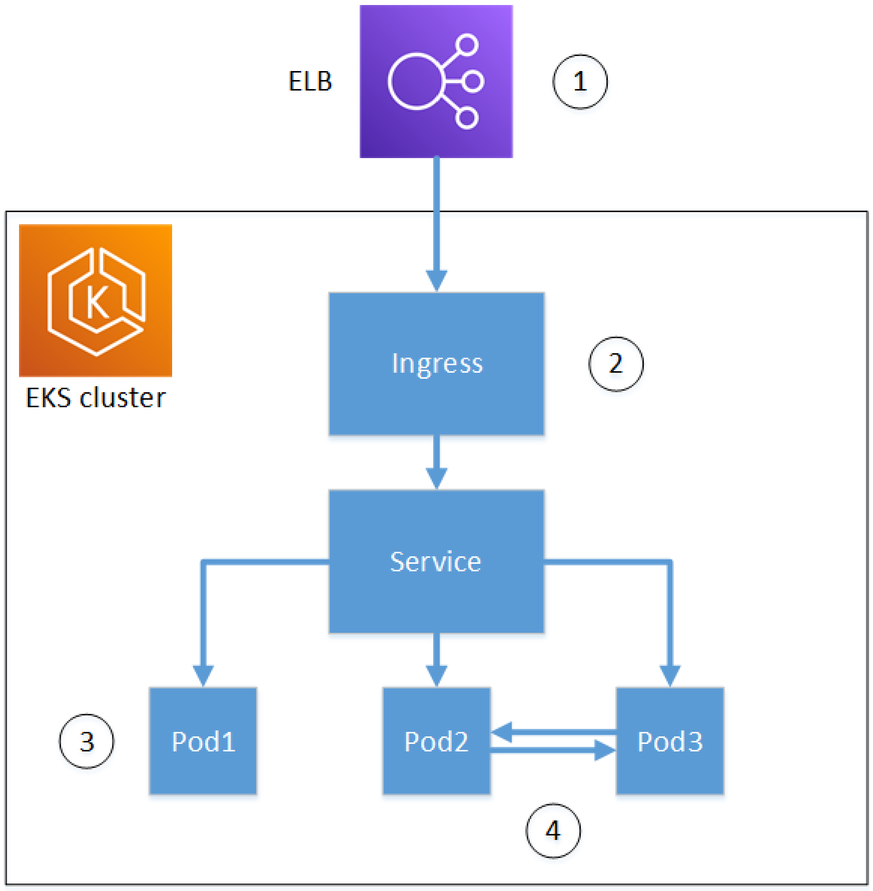
```
$ aws acm-pca import-certificate-authority-certificate \
--certificate-authority-arn arn:aws:acm-pca:region:account:\
certificate-authority/12345678-1234-1234-1234-123456789012 \
--certificate file://C:\example_ca_cert.pem \
--certificate-chain file://C:\example_ca_cert_chain.pem
```

Securing Kubernetes with ACM Private CA

Kubernetes containers and applications use digital certificates to provide secure authentication and encryption over TLS. A widely adopted solution for TLS certificate life-cycle management in Kubernetes is cert-manager, an add-on to Kubernetes that requests certificates, distributes them to Kubernetes containers, and automates certificate renewal.

ACM Private CA provides an open-source plug-in to cert-manager, aws-privateca-issuer, for cert-manager users who want to set up a CA without storing private keys in the cluster. Users with regulatory requirements for controlling access to and auditing their CA operations can use this solution to improve auditability and support compliance. You can use the AWS Private CA Issuer plugin with Amazon Elastic Kubernetes Service (Amazon EKS), a self-managed Kubernetes on AWS, or in on-premises Kubernetes.

The diagram below shows some of the options available for using TLS in an Amazon EKS cluster. This example cluster, containing various resources, is situated behind a load balancer. The numbers identify possible end-points for TLS-secured communications, including the external load balancer, the ingress controller, an individual pod within a service, and a pair of pods communicating securely with each other.



1. **Termination at the load balancer.**

Elastic Load Balancing (ELB) is an AWS Certificate Manager integrated service, which means that you can provision ACM with a private CA, sign a certificate with it, and install it using using the ELB console. This solution provides encrypted communication between a remote client and the load balancer. Data is passed unencrypted to the EKS cluster. Alternatively, you could provide a private certificate to a non-AWS load balancer to terminate TLS.

2. **Termination at the Kubernetes ingress controller.**

   The ingress controller resides inside the EKS cluster as a native load balancer and router. If you have installed both **cert-manager** and **aws-privateca-issuer**, and provisioned the cluster with a private CA, Kubernetes can install a signed TLS certificate on the controller, allowing it to serve as the cluster's end-point for external communications. Communications between the load balancer and the ingress controller are encrypted, and after ingress, data passes unencrypted to the cluster's resources.

3. **Termination at a pod.**

   Each pod is a group of one or more containers that shares storage and network resources. If you have installed both **cert-manager** and **aws-privateca-issuer**, and provisioned the cluster with a private CA, Kubernetes can install a signed TLS certificates on pods as needed. A TLS connection terminating at a pod is unavailable by default to other pods in the cluster.

4. **Secure communications between pods.**

   You can also provision multiple pods with certificates that 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 do not satisfy HIPAA or FIPS requirements.
   - Provisioning with certificates signed by a private CA. As in numbers 2 and 3 above, this requires installing both **cert-manager** and **aws-privateca-issuer**, and provision the cluster with a private CA. Kubernetes can then install signed TLS certificates on the pods as needed.

Cross-account use of the cert-manager

Administrators with cross-account access to a CA can use cert-manager to provision a Kubernetes cluster. For more information, seeCross-account access to private CAs (p. 16).

Note
Only certain ACM Private CA certificate templates can be used in cross-account scenarios. See the section called “Supported certificate templates ” (p. 220) for a list of available templates.

Supported certificate templates

The following table lists ACM Private CA templates that can be used with cert-manager to provision a Kubernetes cluster.

Templates supported for Kubernetes	Support for cross-account use
BlankEndEntityCertificate_CSRPassthrough/V1 definition (p. 120)	
CodeSigningCertificate/V1 definition (p. 126)	
EndEntityCertificate/V1 definition (p. 128)	✓
EndEntityClientAuthCertificate/V1 definition (p. 130)	✓

Templates supported for Kubernetes	Support for cross-account use
EndEntityServerAuthCertificate/V1 definition (p. 132)	✓
??? (p. 134)	

Example solutions

The following integration solutions show how to configure access to ACM Private CA on an Amazon EKS cluster.

- TLS-enabled Kubernetes clusters with ACM Private CA and Amazon EKS
- Setting up end-to-end TLS encryption on Amazon EKS with the new AWS Load Balancer Controller

ACM Private CA best practices

Best practices are recommendations that can help you use ACM Private CA effectively. The following best practices are based on real-world experience from current AWS Certificate Manager and ACM 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, ACM 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 for AWS Certificate Manager Private Certificate Authority (p. 7).

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.

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. ACM Private CA offers two standard options for managed revocation: Online Certificate Status Protocol (OCSP), and certificate revocation lists (CRLs). For more information, see Setting up a certificate revocation method (p. 56).

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 Using CloudTrail (p. 24) to see example trails for ACM 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.

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 Deleting your private CA (p. 99).

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

Block public access to your CRLs

ACM 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 Enabling the S3 Block Public Access (BPA) feature (p. 59).

Amazon EKS application best practices

When using ACM 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 ACM Private CA with Kubernetes, see Securing Kubernetes with ACM Private CA (p. 219).

Troubleshooting

Consult the following topics if you have problems using AWS Certificate Manager Private Certificate Authority.

Topics

Creating and signing a private CA certificate

After you create your private CA, you must retrieve the CSR and submit it to an intermediate or root CA in your X.509 infrastructure. Your CA uses the CSR to create your private CA certificate and then signs the certificate before returning it to you.

Unfortunately, it is not possible to provide specific advice on problems related to creating and signing your private CA certificate. The details of your X.509 infrastructure and the CA hierarchy within it are beyond the scope of this ACM Private CA documentation.

Latency in OCSP responses

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.

Configuring Amazon S3 to allow creation of a CRL bucket

Your private CA may fail to create a CRL bucket if Amazon S3 **Block public access (bucket settings)** are enforced on your account. Check your Amazon S3 settings if this occurs. For more information, see Using Amazon S3 Block Public Access.

Deleting a self-signed CA certificate

You cannot revoke a self-signed CA certificate. Instead, you must delete the CA.

Handling exceptions

An ACM Private CA command might fail for several reasons. For information on each exception and recommendations for resolving them, see the table below.

ACM Private CA Exceptions

Exception Returned by ACM Private CA	Description	Recommendation
`AccessDeniedException`	The permissions required to use the given command have not been delegated by a private CA to the calling account.	For information on delegating permissions in ACM Private CA, see Assign certificate renewal permissions to ACM (p. 84).
`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 (p. 100). 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 ACM Private CA cannot be verified or validated.	Confirm that your CSR was properly generated and configured.
`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.

Exception Returned by ACM Private CA	Description	Recommendation
`ThrottlingException`	A requested API action failed because it exceeded a quota.	Confirm that you are not issuing more calls than allowed by ACM 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.
`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 Updating your private CA (p. 89).

Terms and Concepts

The following terms and concepts can help you as you work with AWS Certificate Manager Private Certificate Authority (ACM Private CA).

Topics

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 (p. 228).

You can use ACM 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 (p. 230). Your private CA also requires a certificate before you can use it. For more information, see CA certificate (p. 229).

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 ACM 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 ACM Private CA.

The following example shows the typical fields contained in an ACM 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 "subscriber" 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

ACM 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.

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.

update-history-change	update-history-description	update-history-date
New region support (p. 233)	Endpoint added for Asia Pacific (Jakarta). For a complete list of ACM PCA endpoints, see AWS Certificate Manager Private Certificate Authority Endpoints and Quotas.	May 4, 2022
Support for Custom Attributes and Extensions (p. 233)	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 (p. 233)	See Setting up a certificate revocation method for revocation options including OCSP.	August 18, 2021
Support for S3 Block Public Access feature for CRLs (p. 233)	See Enabling the S3 Block Public Access feature.	May 27, 2021
New and updated Java implementation examples (p. 233)	See Using the ACM Private CA API (Java Examples).	September 9, 2020
New region support (p. 233)	Endpoints added for Africa (Cape Town) and Europe (Milan). For a complete list of ACM PCA endpoints, see AWS Certificate Manager Private Certificate Authority Endpoints and Quotas.	August 27, 2020
Cross-account private CA access supported (p. 233)	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 (p. 233)	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 (p. 233)	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. (p. 233)	For more information, see Creating an Audit Report for Your Private CA.	March 6, 2020
CloudFormation support (p. 233)	Support added for AWS CloudFormation. For more information, see ACMPCA Resource Type Reference in the AWS CloudFormation User Guide.	January 22, 2020
CloudWatch Events integration (p. 233)	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 (p. 233)	FIPS endpoints added for AWS GovCloud (US-East) and AWS GovCloud (US-West). For a complete list of ACM PCA endpoints, see AWS Certificate Manager Private Certificate Authority Endpoints and Quotas.	December 13, 2019
Tag-based permissions (p. 233)	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 (p. 233)	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 (p. 233)	New certificate templates added, including templates for code signing with AWS Signer. For more information, see Using Templates.	October 1, 2019

Planning your CA (p. 233)	New section added on planning your PKI using ACM PCA. For more information, see Planning Your ACM Private CA Deployment.	September 30, 2019
Added region support (p. 233)	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 (p. 233)	Support for creating and hosting root CAs removes need for an external parent.	June 20, 2019
Added region support (p. 233)	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 (p. 233)	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 (p. 233)	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 (p. 233)	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 (p. 233)	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 Certificate Manager Private Certificate Authority before June 2018.

Change	Description	Date
New guide	This release introduces AWS Certificate Manager Private Certificate Authority.	April 04, 2018