AWS Certificate Manager Private Certificate Authority

User Guide

Version latest

AWS Certificate Manager Private Certificate Authority: User Guide

Table of Contents

What Is ACM PCA?

AWS Certificate Manager Private Certificate Authority is a managed private CA service with which you can easily and securely manage your certificate authority infrastructure and your private certificates. ACM PCA provides a highly available private CA service without the investment and maintenance costs of operating your own certificate authority. ACM PCA extends ACM certificate management to private certificates, enabling you to manage public and private certificates in one console.

Private certificates identify resources within an organization. Examples include clients, servers, applications, services, devices, and users. In establishing a secure encrypted communications channel, each resource endpoint uses a certificate and cryptographic techniques to prove its identity to another endpoint. 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. For more information, see Concepts (p. 2).

Both public and private certificates help customers identify resources on networks and secure communication between these resources. Public certificates identify resources on the public internet whereas private certificates do so for private networks. One key difference is that applications and browsers trust public certificates by default whereas an administrator must explicitly configure applications to trust private certificates. Public CAs, the entities that issue public certificates, must follow strict rules, provide operational visibility, and meet security standards imposed by the browser and operating system vendors. Private CAs are managed by private organizations, and private CA administrators can make their own rules for issuing private certificates. These include practices for issuing certificates and what information a certificate can include.

To get started using ACM PCA, you must have an intermediate or root CA available for your organization. This might be an on–premises CA, or one that is in the cloud, or one that is commercially available. Create your private CA and then use your organization's CA to create and sign the private CA certificate. After the certificate is signed, import it back into ACM PCA For more information about the steps you must follow to create a private CA, see Getting Started (p. 16).

Once you have a private CA, you can do the following:

- Issue private certificates (p. 26)
- Create a certificate revocation list (CRL) (p. 24)
- Revoke issued certificates (p. 28)
- Retrieve issued or revoked certificates (p. 29)
- Create audit reports that contain issuance and revocation information (p. 35)
- Change the status of your private CA (p. 30)
- Add tags to your private CA (p. 37)
- Delete your private CA (p. 32)
- Restore your private CA (p. 33)

The primary reason to create a private CA is, of course, to issue private certificates. You can use ACM PCA to do so, or you can use AWS Certificate Manager (ACM). ACM PCA is tightly integrated with ACM. Each service provides unique benefits.

The AWS Certificate Manager service:

- Provides a console for you to use to request and manage private certificates.

- Manages the private keys that are associated with your certificates.
- Renews private certificates that ACM manages.
- Enables you to export your private certificates for use anywhere.
- Enables you to deploy your certificates with other integrated AWS services.

ACM PCA allows you to:

- Create a certificate with any subject name to identify anything you want.
- Use any supported private key algorithm and key length.
- Use any supported signing algorithm.
- Avoid constraints imposed on public certificates and CAs.

Topics

Concepts

This topic introduces basic terms and concepts related to AWS Certificate Manager Private Certificate Authority (ACM PCA).

Topics

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 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 key pairs, see Asymmetric Key Cryptography (p. 5). For information about certificate signing, see Certificate Signature (p. 4).

You can use ACM PCA 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. 4). Your private CA also requires a certificate before you can use it. For more information, see Certificate Authority Certificate (p. 3).

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

When you use ACM PCA to create a private CA, ACM PCA creates a certificate signing request (CSR). You must take the CSR to your on-premises public key infrastructure (p. 6) (PKI) to create a signed CA certificate. Import the CA certificate into ACM PCA. Your private CA will be subordinate to the CA that you used to sign it.

```
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
```

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 hash function such as SHA256 over the variable-sized certificate content to produce an irreversible fixed-size data string. The fixed data is called a hash. The CA then encrypts the hash value with its private key and concatenates the encrypted hash with the certificate.

To validate a signed certificate, a client application uses the CA public key to decrypt the signature. The client then uses the same signing algorithm that the CA used to compute a hash over the rest of the certificate. Note that the signing algorithm used by the CA is listed in the certificate. If the computed hash value is the same as the decrypted hash value, the certificate has not been tampered with.

Private Certificate

ACM PCA certificates are private SSL/TLS certificates that you can use within your organization. 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
```

Asymmetric Key Cryptography

Asymmetric cryptography uses different but mathematically related keys to encrypt and decrypt content. One of the keys is public and is made available in an X.509 v3 certificate. The other key is private and is stored securely. The X.509 certificate binds the identity of a user, computer, or other resource (the certificate subject) to the public key.

Private CA certificates and certificates issued by a private CA are X.509 SSL/TLS certificates. They bind the identity of a user, service, application, computer, or other device with the public key that's embedded in the certificate. The associated private key is securely stored in AWS CloudHSM.

Domain Name System

The Domain Name System (DNS) is a hierarchical distributed naming system for computers and other resources connected to the internet or a private network. DNS is primarily used to translate textual domain names, such as `aws.amazon.com`, into numerical IP (internet protocol) addresses of the form `111.222.333.444`.

Domain Names

A domain name is a text string such as `www.example.com` that can be translated by the Domain Name System (DNS) into an IP address. Computer networks, including the internet, use IP addresses rather than text names. A domain name consists of distinct labels separated by periods.

TLD

The rightmost label is called the top-level domain (TLD). Common examples include `.com`, `.net`, and `.edu`. Also, the TLD for entities registered in some countries is an abbreviation of the country name and is called a country code. Examples include `.uk` for the United Kingdom, `.ru` for Russia, and `.fr` for France. When country codes are used, a second-level hierarchy for the TLD is often introduced to identify the type of the registered entity. For example, the `.co.uk` TLD identifies commercial enterprises in the United Kingdom.

Apex domain

The apex domain name includes and expands on the top-level domain. For domain names that include a country code, the apex domain includes the code and the labels, if any, that identify the type of the registered entity. The apex domain does not include subdomains (see the following paragraph). In `www.example.com`, the name of the apex domain is `example.com`. In `www.example.co.uk`, the name of the apex domain is `example.co.uk`. Other names that are often used instead of apex include base, bare, root, root apex, or zone apex.

Subdomain

Subdomain names precede the apex domain name and are separated from it and from each other by a period. The most common subdomain name is `www`, but any name is possible. Also, subdomain names can have multiple levels. For example, in `jake.dog.animals.example.com`, the subdomains are `jake`, `dog`, and `animals` in that order.

FQDN

A fully qualified domain name (FQDN) is the complete DNS name for a computer, website, or other resource connected to a network or to the internet. For example `aws.amazon.com` is the FQDN for Amazon Web Services. An FQDN includes all domains up to the top-level domain. For example,

`[subdomain1].[subdomain2]...[subdomainn].[apex domain].[top-level domain]` represents the general format of an FQDN.

PQDN

A domain name that is not fully qualified is called a partially qualified domain name (PQDN) and is ambiguous. A name such as `[subdomain1.subdomain2.]` is a PQDN because the root domain cannot be determined.

Registration

The right to use a domain name is delegated by domain name registrars. Registrars are typically accredited by the Internet Corporation for Assigned Names and Numbers (ICANN). In addition, other organizations called registries maintain the TLD databases. When you request a domain name, the registrar sends your information to the appropriate TLD registry. The registry assigns a domain name, updates the TLD database, and publishes your information to WHOIS. Typically, domain names must be purchased.

Fully Qualified Domain Name (FQDN)

See Domain Names (p. 5).

Public Key Infrastructure

A public key infrastructure (PKI) is a comprehensive system that enables the creation, issuance, management, distribution, use, storage, and revocation of digital certificates. A PKI consists of people, hardware, software, policies, documents, and procedures. In ACM PCA, the purpose of a PKI is to manage private certificates. ACM PCA implements the following:

Public Key Certificates

Your private CA issues certificates to affirm the identities of users, services, applications, computers, and other devices within your organization. For more information, see Private Certificate (p. 4).

Certificate Repository

Your private CA stores all of the certificates it has issued. You can, of course, delete a certificate from the ACM PCA repository.

Certificate Revocation

ACM PCA supports certificate revocation. Certificates that can no longer be trusted must be revoked by the certificate authority. A certificate can be revoked before the end of its validity period for any number of reasons. Common reasons include key compromise, end of operations, a change in affiliation, or the withdrawal of privilege. When you create a private CA, you specify whether you want the CA to support certificate revocation. If the CA supports revocation, you can request an audit report to review certificates that you have revoked.

Key Storage

Your CA private keys are locked away securely in a hardware security module (HSM) that is owned and managed by Amazon.

Root 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.

Secure Sockets Layer (SSL)

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are cryptographic protocols that provide communication security over a computer network. TLS is the successor of SSL. They both use X.509 certificates to authenticate the server. Both protocols negotiate a symmetric key to encrypt data that flows between the client and server.

Secure HTTPS

HTTPS stands for HTTP over SSL/TLS, a secure form of HTTP that is supported by all major browsers and servers. All HTTP requests and responses are encrypted before being sent across a network. HTTPS combines the HTTP protocol with symmetric, asymmetric, and X.509 certificate-based cryptographic techniques. HTTPS works by inserting a cryptographic security layer below the HTTP application layer and above the TCP transport layer in the Open Systems Interconnection (OSI) model. The security layer uses the Secure Sockets Layer (SSL) protocol or the Transport Layer Security (TLS) protocol.

SSL 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. An SSL/TLS certificate is signed by a certificate authority (CA) and contains the name of the server, the validity period, the public key, the signature algorithm, and more.

Transport Layer Security (TLS)

See Secure Sockets Layer (SSL) (p. 7).

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.

X.500 Distinguished Name

X.500 distinguished names (DN) are used to identify users, computers, applications, services, servers, and other devices in the X.509 public key certificates that ACM PCA creates. This includes private certificates and private CA certificates. Common fields include the following:

- **organizationName (O)** – Name of the organization that issued or is the subject of a certificate
- **organiztionUnit (OU)** – Department or division within an organization
- **country (C)** – Two letter country code
- **stateName (S)** – Name of a state or province such as Washington
- **localityName (L)** – Locality name such as Seattle
- **commonName (CN)** – Common name of the certificate subject or issuer

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 PCA.

Note
ACM is currently available in some regions that ACM PCA is not.

Services Integrated with AWS Certificate Manager Private Certificate Authority

If you use ACM to request a private certificate, you can associate the certificate with any of the services that are integrated with ACM with the exception of AWS CloudFormation. For more information, see Integrated Services. If you use the ACM PCA API or AWS CLI to issue a certificate or if you export a private certificate from ACM, you can install the certificate anywhere you want.

Limits

This section specifies limits that affect AWS Certificate Manager Private Certificate Authority.

Topics

Limits on Certificates

The following ACM PCA certificate limits apply to each region and each account. To request higher limits, create a case at the AWS Support Center. New AWS accounts might start with limits that are lower than those that are described here.

Item	Default Limit
Number of private certificate authorities (CAs)	10
Number of private certificates per private CA	50,000

Note
A private CA that has been deleted will count towards your certificate limit until the end of its restoration period. For more information, see Delete Your Private CA.

ACM PCA is integrated with ACM. You can use the ACM console, AWS CLI, or ACM API to request private certificates from an existing private certificate authority (CA). The certificates are managed by ACM and have the same restrictions as public certificates issued by ACM. For a list of the restrictions, see Request a Private Certificate. You can also issue private certificates with the ACM PCA API or AWS CLI. For more information, see Issue a Private Certificate (p. 26). Regardless of which method you use, you can create 10 private CAs and 50,000 private certificates for each. ACM places limits on public and imported certificates. For more information, see ACM Limits.

API Rate Limits

The following limits apply to the ACM PCA API for each region and account. ACM PCA throttles API requests at different limits depending on the API operation. Throttling means that ACM PCA rejects an otherwise valid request because the request exceeds the operation's limit for the number of requests per second. When a request is throttled, ACM PCA returns a `ThrottlingException` error. The following table lists each API operation and the limit at which ACM PCA throttles requests for that operation.

Requests per second limit for each ACM PCA API operation

API operation	Requests per second
`CreateCertificateAuthority`	1
`CreateCertificateAuthorityAuditReport`	1
`DeleteCertificateAuthority`	10
`DescribeCertificateAuthority`	20
`DescribeCertificateAuthorityAuditReport`	20
`GetCertificate`	20
`GetCertificateAuthorityCertificate`	20
`GetCertificateAuthorityCsr`	10
`ImportCertificateAuthorityCertificate`	10
`IssueCertificate`	5
`ListCertificateAuthorities`	20
`ListTags`	20
`RestoreCertificateAuthority`	20
`RevokeCertificate`	20
`TagCertificateAuthority`	10
`UntagCertificateAuthority`	10
`UpdateCertificateAuthority`	10

Note
At this time, ACM PCA does not support individual API rate limit increases per customer.

Private Certificate Authority Security

The private keys for your private CAs are stored in AWS managed hardware security modules (HSMs). These adhere to FIPS 140-2 Level 3 security standards. You can control access to your CAs by setting AWS Identity and Access Management (IAM) policies on users, groups, and roles in your organization.

Your private CA can automatically create certificate revocation lists (CRLs). Client applications can query a CRL to determine whether a certificate has been revoked. You can generate audit reports that list the

certificates that your private CA has issued and revoked. You can turn on CloudTrail to log AWS and ACM PCA access.

For more information about AWS and ACM PCA features that you can use to help secure your private CA, see the following topics:

- Create a Certificate Revocation List (CRL) (p. 24)
- Create an Audit Report for Your Private CA (p. 35)
- Create an IAM Administrator (p. 12)
- Authentication and Access (p. 82)

Best Practices

Best practices are recommendations that can help you use AWS Certificate Manager Private Certificate Authority (ACM PCA) more effectively. The following best practices are based on real-world experience from current ACM customers.

Topics

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. 39) to see example trails for ACM PCA operations.

Update the CA Private Key

It is a best practice to periodically update the private key for your private CA. Doing so gives an attacker less time to attempt to crack you key and less time to use the key if it has already been compromised. You cannot update a key by importing a new CA certificate. You must replace the private CA with a new CA. We recommend that you follow the process outlined in Delete Your Private CA (p. 32).

Delete a CA

You can permanently delete a private CA. You might want to do so if you no longer need the CA or if you want to replace it with a CA that has a newer private key. To safely delete a CA, we recommend that you follow the process outlined in Delete Your Private CA (p. 32).

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 PCA 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 at which the certificate was issued. This metric applies only to the IssueCertificate operation.
`Success`	Specifies whether a certificate was successfully issued. This metric applies only to the **IssueCertificate** operation.
`Failure`	Indicates that 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

Pricing

You are charged for private certificates whose private key you can access. This includes certificates that you export from ACM and certificates that you create from the ACM PCA API or ACM PCA CLI. You are not charged for a private certificate after it has been deleted (p. 32). However, if you restore (p. 33) 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 PCA pricing information, see the ACM Pricing page on the AWS website.

Setting Up

Before you can use AWS Certificate Manager Private Certificate Authority, you must have a certificate authority (CA) available to sign your private CA certificate. This can be a commercial CA or one that you set up and run on premises. Best practices also include creating an IAM administrative group and one or more administrators within that group to run your private CAs. You can optionally install the AWS CLI if you want to interact with ACM PCA and ACM on the command line.

Topics

Install the AWS Command Line Interface (Optional)

If you have not installed the AWS CLI but want to use it, follow the directions at AWS Command Line Interface. If you are using Windows, you can download and run a 64-bit or 32-bit Windows installer. If you are using Linux or macOS, you can install the AWS CLI using pip.

If you already have the AWS CLI installed, check the version number by typing `aws --version` in a command window or on the command line. Compare the version number to the most recent available on GitHub. If your version is old, update the CLI.

On the command line, type `aws configure`. You'll need your access key ID and secret access key to complete the following steps. For more information, see Access Keys.

- Type your access key ID when prompted.
- Type your secret access key.
- Choose your default region. See Regions (p. 8) for a list of those available.
- Accept `json` as the default output format.

Type `aws acm-pca` on the command line followed by the ACM PCA command that you want to run. For example, if you want to list all of your private certificate authorities, type the following command.

```
aws acm-pca list-certificate-authorities --max-results 10
```

Create an IAM Administrator

As a best practice, don't use your AWS account root user to interact with AWS, including ACM PCA. 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 policy that you can assign to users. Users can perform only the ACM PCA actions that the policy allows.

Topics

Sign Up for AWS

If you're not already an Amazon Web Services (AWS) customer, you must sign up to be able to use ACM PCA. Your account is automatically signed up for all available services, but you are charged for only the services that you use. Also, if you are a new AWS customer, some services are available for free during a limited period. For more information, see AWS Free Tier.

> **Note**
> ACM PCA is not available in the free tier.

To sign up for an AWS account

1. Go to https://aws.amazon.com/ and choose **Sign Up**.
2. Follow the on-screen instructions.

> **Note**
> Part of the sign-up procedure includes receiving an automated telephone call and entering the supplied PIN on the telephone keypad. You must also supply a credit card number.

Create an Administrator

You can use the IAM console to create users and groups with the permissions they need to work with ACM PCA.

To create an IAM user for yourself and add the user to an Administrators group

1. Use your AWS account email address and password to sign in as the *AWS account root user* to the IAM console at https://console.aws.amazon.com/iam/.

   > **Note**
   > We strongly recommend that you adhere to the best practice of using the **`Administrator`** IAM user below and securely lock away the root user credentials. Sign in as the root user only to perform a few account and service management tasks.
2. In the navigation pane of the console, choose **Users**, and then choose **Add user**.
3. For **User name**, type **`Administrator`**.
4. Select the check box next to **AWS Management Console access**, select **Custom password**, and then type the new user's password in the text box. You can optionally select **Require password reset** to force the user to create a new password the next time the user signs in.
5. Choose **Next: Permissions**.
6. On the **Set permissions** page, choose **Add user to group**.
7. Choose **Create group**.
8. In the **Create group** dialog box, for **Group name** type **`Administrators`**.
9. For **Filter policies**, select the check box for **AWS managed - job function**.
10. In the policy list, select the check box for **AdministratorAccess**. Then choose **Create group**.
11. Back in the list of groups, select the check box for your new group. Choose **Refresh** if necessary to see the group in the list.
12. Choose **Next: Tags** to add metadata to the user by attaching tags as key-value pairs.

13. Choose **Next: Review** to see the list of group memberships to be added to the new user. When you are ready to proceed, choose **Create user**.

You can use this same process to create more groups and users, and to give your users access to your AWS account resources. To learn about using policies to restrict users' permissions to specific AWS resources, go to Access Management and Example Policies.

You can create multiple administrators in your account and add each to the Administrators group. To sign in to the AWS Management Console, each user needs an AWS account ID or alias. To get these, see Your AWS Account ID and Its Alias in the *IAM User Guide*.

Create a Customer Managed Policy

The following sample customer managed policy allows a user to create a CA audit report. This is a sample only. You can choose any ACM PCA operations you want. For more examples, see Inline Policies (p. 84).

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 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 assign the policy to any IAM user to limit what ACM PCA actions the user can perform.

Set Up a Certificate Authority

You must have a certificate authority (CA) available to sign your private CA certificate. This can be a commercial or an on-premises CA. It can be a root or an intermediate CA. You can use OpenSSL to create a test certificate authority. We recommend, however, that you use a production CA rather than a test CA. You use the certificate authority to sign your private CA certificate. Signing the certificate is part of the process that you must follow to create your CA and get it ready to use:

1. Step 1: Create a Private Certificate Authority (p. 16)
2. Step 2: Get a Certificate Signing Request (CSR) (p. 19)

3. Step 3: Sign Your Private CA Certificate (p. 21)
4. Step 4: Import Your Private CA Certificate into ACM PCA (p. 22)

Getting Started

To work with ACM PCA, you can use the AWS Management Console, the AWS CLI, and the API. The following topics show you how to use the console and the CLI. To learn more about the API, see AWS Certificate Manager Private Certificate Authority API Reference. For Java examples that show you how to use the API, see Using the ACM PCA API (p. 50).

Topics

Step 1: Create a Private Certificate Authority

You can use a private certificate authority (CA) to issue and revoke private certificates within an organization. You can issue certificates for users, applications, computers, servers, and other devices. The certificates are SSL/TLS public key certificates based on the ISO X.509 standard.

- Creating a CA (Console)
- Creating a CA (AWS CLI)

To create a CA (console)

1. Sign in to your AWS account and open the ACM PCA console at **`https://console.aws.amazon.com/acm-pca/home`**. The introductory page will appear if your console opens to a region in which you do not have a CA. Choose **Get started** beneath **Private certificate authority**. Choose **Get started** again. If the console opens to a region in which you already have one or more CAs, the introductory page will not be shown. Choose **Private CAs** and then choose **Create CA**.
2. Choose the type of the private certificate authority that you want to create. Currently, you must choose **Subordinate**. Choose **Next**.

   Your private subordinate CA certificate must be authenticated and then signed by an intermediate or root CA. That root or intermediate CA must be higher in your organization's CA hierarchy than the CA that you are creating. For more information, see Step 3: Sign Your Private CA Certificate (p. 21). Subordinate CAs are typically used to issue end-entity certificates to users, computers, and applications. This enables you to protect your root by removing it from the network and perhaps physically isolating it when you are not using it to commission new subordinate CAs.
3. Configure the subject name of your private CA. Use the X.500 distinguished name format. You must enter at least one of the following values and then choose **Next**. For more information about each of the values that make up a subject distinguished name, see X.500 Distinguished Name (p. 7).

   - Organization
   - Organization Unit
   - Country name
   - State or province name
   - Locality name

- Common Name

4. Choose the private 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 one of the following combinations. Make a selection and then choose **Next**.

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

5. Configure a certificate revocation list (CRL) if you want ACM PCA to maintain one for the certificates revoked by your private CA. Clients such as web browsers query CRLs to determine whether a certificate can be trusted. For more information, see Create a Certificate Revocation List (CRL) (p. 24). If you want to create a CRL, do the following:

   1. Choose **Enable 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. Otherwise, choose **No** and select an existing bucket from the list.

      If you choose **Yes**, ACM PCA creates the necessary bucket policy for you. If you choose **No**, make sure the following policy is attached to your bucket. For more information, see Adding a Bucket Policy.

```
{
  "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:::your-bucket-name/*",
          "arn:aws:s3:::your-bucket-name"
      ]
    }
  ]
}
```

   3. Expand **Advanced** if you want to specify more about your CRL.
      - Add a **Custom CRL Name** to hide the name of your S3 bucket from public view.
      - Type the number of days your CRL will remain valid. For online CRLs, a validity period of two (2) to seven (7) days is common. ACM PCA tries to regenerate the CRL at the midpoint of the specified period.

6. Choose **Next**. If everything looks correct, choose **Confirm and create**.
7. If you want to continue getting and importing a certificate signing request (CSR), leave the **Success** dialog box open and follow the instructions at Step 2: Get a Certificate Signing Request (CSR) (p. 19). Otherwise choose **You can also finish this later**.

To create a CA (AWS CLI)

Use the `create-certificate-authority` command to create a private 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
- 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 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

You can modify the following example files to use with this command.

```
C:\ca_config.txt

{
  "KeyAlgorithm": "RSA_2048",
  "SigningAlgorithm": "SHA256WITHRSA",
  "Subject":
    {"Country": "US",
     "Organization": "Example Corp",
     "OrganizationalUnit": "Sales",
     "State": "WA",
     "Locality": "Seattle",
     "CommonName": "www.example.com"}
}
```

```
C:\revoke_config.txt

{
  "CrlConfiguration":
    {"Enabled": true,
     "ExpirationInDays": 7,
     "CustomCname": "some_name.crl",
     "S3BucketName": "your-bucket-name"}
}
```

```
aws acm-pca create-certificate-authority \
 --certificate-authority-configuration file://C:\ca_config.txt \
 --revocation-configuration file://C:\revoke_config.txt \
 --certificate-authority-type "SUBORDINATE" \
 --idempotency-token 98256344
```

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

```
{
  "CertificateAuthorityArn": "arn:aws:acm-pca:region:account:
        certificate-authority/12345678-1234-1234-123456789012"
}
```

Step 2: Get a Certificate Signing Request (CSR)

After you create (p. 16) a private certificate authority (CA), you must retrieve the CA certificate signing request (CSR). You can do so using the AWS Management Console or the AWS CLI as discussed in the procedures that follow. Save the CSR to a file. If you want to look at the CSR, use the following OpenSSL command.

```
openssl req -in path_to_CSR_file -text -noout
```

The preceding command generates output similar to the following. Notice that the **CA** extension is `TRUE`. This indicates that the CSR is for a certificate authority.

```
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
```

Topics

Retrieving a CSR (Console): Method 1

Use this procedure if you followed the steps to create a private CA (p. 16) and left the **Success** dialog box open.

To retrieve a CSR (console): method 1

1. Immediately after ACM PCA has successfully created your private CA, choose **Get started**. The ACM PCA console returns the CSR. You can return to this step later.
2. Choose **Export CSR to a file** and save it locally.
3. Choose **Next**.
4. Follow the instructions in Step 3: Sign Your Private CA Certificate (p. 21).

Retrieving a CSR (Console): Method 2

Use this procedure if you followed the steps to create a private CA (p. 16) and closed the **Success** dialog box.

To retrieve a CSR (console): method 2

1. When you are ready to continue, open the AWS Certificate Manager console and choose **Private CAs** in the left navigation pane.
2. Select your private CA from the list.
3. From the **Actions** menu, choose **Import CA certificate**. The ACM PCA console returns the CSR.
4. Choose **Export CSR to a file** and save it locally.
5. Choose **Next**.
6. Follow the instructions in Step 3: Sign Your Private CA Certificate (p. 21).

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/12345678-1234-1234-1234-123456789012 \
--output text
```

2. Follow the instructions in Step 3: Sign Your Private CA Certificate (p. 21).

Step 3: Sign Your Private CA Certificate

After you create (p. 16) your private CA and retrieve (p. 19) the certificate signing request (CSR), you must take the CSR to your 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 4: Import Your Private CA Certificate into ACM PCA (p. 22).

Important

- The details of your X.509 infrastructure and the CA hierarchy within it are beyond the scope of this documentation. For more information, see Troubleshoot Creating and Signing a Private CA Certificate (p. 81).
- 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 PCA does not restrict the lifetime of a CA.
- If you must create a CA certificate that does not effectively expire, 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 outputs 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 4: Import Your Private CA Certificate into ACM PCA

After you create (p. 16) your private CA, retrieve (p. 19) the certificate signing request (CSR), and sign (p. 21) the CA certificate, you must import the certificate into ACM PCA. After signing and importing the certificate, you can use your private CA to issue and revoke trusted private SSL/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 retrieve the certificate chain that contains the certificate of the intermediate or root CA used to sign your private CA certificate and 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. The following example contains three certificates, but your PKI infrastructure might have more or fewer.

Note
Your chain must be PEM formatted.

```
-----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-----
```

Topics

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 a signed certificate authority (CA) certificate** page, skip to step 7. Otherwise, continue.
2. Sign in to your AWS account and open the ACM PCA console at https://console.aws.amazon.com/acm-pca/home.
3. Choose **Private CAs**.
4. Select your private CA from the list.
5. On the **Actions** menu, choose **Import CA certificate**.
6. Choose **Next**.
7. For **Certificate body**, copy your signed private CA certificate into the textbox or import it from a file.
8. For **Certificate chain**, copy the certificate chain into the textbox or import it from a file.
9. Choose **Next**.
10. Choose **Confirm and Import** to import the private CA 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 PCA.

```
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
```

Using Your Private Certificate Authority (CA)

After you have created a private certificate authority (CA), you can use it to perform the following tasks:

Topics

Create a Certificate Revocation List (CRL)

ACM PCA can create a certificate revocation list (CRL) for you when you create a new private CA or update an existing CA. The CRL is written to an Amazon S3 bucket that you specify. Note that ACM PCA does not support Amazon S3 server-side encryption (SSE) for CRLs and audit reports. Each CRL is a DER encoded file. You can download the file and use OpenSSL to view it.

```
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
```

Topics

Creating the CRL When You Create a CA

You can use the integrated ACM console to configure the parameters for a CRL when you create a private CA. For more information, see step 5 in Creating a CA (Console). You can also use the AWS CLI, but you must specify a file that contains your revocation configuration. For more information, see Creating a CA (AWS CLI). You can also use the CreateCertificateAuthority API.

Updating a CRL Configuration (Console)

You can use the ACM console to update the CRL configuration for an existing private CA.

To update a CRL configuration (console)

1. Sign in to your AWS account and open the ACM PCA console at https://console.aws.amazon.com/acm-pca/home. Choose **Private CAs**.
2. Choose your CA from the list.
3. On the **Actions** menu, choose **Update CA revocation**.
4. Choose **Enable CRL distribution**
5. To create a new S3 bucket for your CRL entries, choose **Yes** for the **Create a new S3 bucket** option and type a unique bucket name. Otherwise, choose **No** and choose an existing bucket from the list.
6. Choose **Advanced** if you want to specify more about your CRL.
7. Add an optional **Custom CRL Name** to hide the name of your S3 bucket from public view.
8. Type the number of days your CRL will remain valid. For online CRLs, a validity period of 7 days is common. ACM PCA tries to regenerate the CRL at 1/2 of the specified period.

9. Choose **Update**.

Updating a CRL Configuration (AWS CLI)

You can use the AWS CLI to update the CRL configuration for an existing private CA.

To update a CRL configuration (AWS CLI)

Use the update-certificate-authority command to update a CRL configuration. You can modify the following example file to use this command.

```
C:\revoke_config.txt

{
  "CrlConfiguration":
    {"Enabled": true,
     "ExpirationInDays": 7,
     "CustomCname": "some_name.crl",
     "S3BucketName": "your-bucket-name"}
}
```

```
aws acm-pca update-certificate-authority \
--certificate-authority-arn arn:aws:acm-pca:region:account:\
certificate-authority/12345678-1234-1234-1234-1232456789012 \
--revocation-configuration file://C:\revoke_config.txt \
--status "ACTIVE"
```

Issue a Private Certificate

You can request a private certificate by using ACM or by using the ACM PCA API or AWS CLI. For more information about using ACM, see Request a Private Certificate. ACM has many benefits:

- You can use the AWS Management Console to request certificates.
- You to use the console to deploy your private certificates with integrated services.
- ACM stores the CA private keys in an AWS hardware security module (HSM) and uses AWS KMS to manage the keys.
- ACM can renew private certificates that it manages.

You can use the ACM PCA API the AWS CLI to issue a certificate. Private certificates that you issue by using the ACM PCA API or AWS CLI are not subject to the same restrictions as private certificates that issued by ACM. You can use ACM PCA to do the following:

- Create a certificate with any subject name.
- Use any of the supported private key algorithms and key lengths.
- Use any of the signing algorithms that are currently supported.
- Specify any validity period for your private CA and private certificates.
- Import your private certificates into ACM and IAM.

Topics

Creating a Certificate Signing Request (CSR)

Before you can issue a private certificate, you must have a certificate signing request (CSR). If you are using the ACM PCA API and AWS CLI, you can use the following OpenSSL command to create one. The command asks for a passphrase for the private key and for details about your organization.

```
openssl req -new -newkey rsa:2048 -days 365 -keyout my_private_key.pem -out my_csr.csr
```

```
Generating a 2048 bit RSA private key
.....................+++
.............+++
writing new private key to 'private/my_private_key.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:Washington
Locality Name (eg, city) [Default City]:Seattle
Organization Name (eg, company) [Default Company Ltd]:Example
Organizational Unit Name (eg, section) []:Corporate Offices
Common Name (eg, your name or your server's hostname) []:Example Company
Email Address []:corp@www.example.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Issuing a Certificate (AWS CLI)

You can use the issue-certificate command to request a private certificate. This command requires the Amazon Resource Name (ARN) of the private CA that you want to use to issue the certificate. It also requires the CSR for the certificate that you want to issue. You can also use the IssueCertificate operation.

> **Note**
> If you use the ACM PCA API and AWS CLI to issue a private certificate, you cannot use the ACM console, CLI, or API to view or export it. You already have the private key because you needed one to create the CSR (see the previous discussion). You can use the get-certificate command to retrieve the certificate details. You can also create an audit report (p. 35) to make sure that your certificate was issued.

```
aws acm-pca issue-certificate \
--certificate-authority-arn arn:aws:acm-pca:region:account:\
certificate-authority/12345678-1234-1234-1234-123456789012 \
--csr file://C:\cert_1.csr \
--signing-algorithm "SHA256WITHRSA" \
--validity Value=365,Type="DAYS" \
--idempotency-token 1234
```

This command outputs the ARN of the issued certificate.

```
{
  "CertificateArn": "arn:aws:acm-pca:region:account:\
    certificate-authority/12345678-1234-1234-1234-123456789012/
certificate/a2b95975ec6e1cd85a21c2104c5be129"
}
```

Revoke a Private Certificate

You can use theAWS CLI or API to revoke an issued certificate. The certificate will be included in the certificate revocation list (CRL), if it exists, of the private issuing CA. Revoked certificates are always recorded in the audit report.

Topics

Revoked Certificates in a CRL

The following example shows a revoked certificate in a certificate revocation list (CRL). A CRL is typically updated approximately 30 minutes after a certificate is revoked. For more information about creating and configuring CRLs, see Create a Certificate Revocation List (CRL) (p. 24).

```
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 Create an Audit Report for Your Private CA (p. 35).

```
[{
  "awsAccountId": "123456789012",
  "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/e8cbd2bedb122329f97706bcfec990f8",
  "serial": "e8:cb:d2:be:db:12:23:29:f9:77:06:bc:fe:c9:90:f8",
  "subject":
 "1.2.840.113549.1.9.1=#161173616c6573406578616d706c652e636f6d,CN=www.example1.com,OU=Sales,O=Example
 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": "123456789012",
  "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/2bae9a75d71b42b4e41e36f8b4b488fc",
  "serial": "2b:ae:9a:75:d7:1b:42:b4:e4:1e:36:f8:b4:b4:88:fc",
  "subject":
 "1.2.840.113549.1.9.1=#161970726f64407777772e70616c6f75736573616c65732e636f6d,CN=www.example3.com.com,
 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"
}]
```

Using the AWS CLI to Revoke a Certificate

Use the `revoke-certificate` command to revoke a private certificate. You can also use the `RevokeCertificate` function. 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/12345678-1234-1234-1234-123456789012 \
--certificate-serial 67:07:44:76:83:a9:b7:f4:05:56:27:ff:d5:5c:eb:cc \
--revocation-reason "KEY_COMPROMISE"
```

Retrieve a Private Certificate

If you used the ACM PCA API and AWS CLI to issue a private certificate, you can use the AWS CLI or API to retrieve it. 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.

Use the get-certificate command to retrieve a private certificate. You can also use the GetCertificate function. Use the `--output text` option to output the certificate without <CR><LF> pairs.

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 Create an Audit Report for Your Private CA (p. 35).

```
aws acm-pca get-certificate \
--certificate-authority-arn arn:aws:acm-pca:region:account:\
certificate-authority/12345678-1234-1234-1234-123456789012 \
--certificate-arn arn:aws:acm-pca:region:account:\
certificate-authority/12345678-1234-1234-1234-123456789012/\
certificate/6707447683a9b7f4055627ffd55cebcc \
--output text
```

This command outputs the base64 encoded PEM format certificate and the certificate chain.

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

Retrieve a Certificate Authority (CA) Certificate

You can use the ACM PCA 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. Use the `--output text` option to output the certificate without <CR><LF> pairs.

```
aws acm-pca get-certificate-authority-certificate \
--certificate-authority-arn arn:aws:acm-pca:region:account:\
certificate-authority/12345678-1234-1234-1234-123456789012 \
--output text
```

This command outputs the base64 encoded PEM format certificate and the certificate chain.

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

Update Your Private CA

You can update the revocation configuration of a private CA by changing any of the following values:

- A value that specifies whether the private CA generates a certificate revocation list (CRL).
- The number of days before a CRL expires. Note that ACM PCA begins trying to regenerate the CRL at 1/2 the number of days you specify.
- The name of the 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 consequences. For example, disabling CRL generation, changing the validity period, or changing the S3 bucket after you have placed your private CA is in production could 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.

ACM PCA updates the status of your private CA at certain times. You can also change the status, but you can only change it from `ACTIVE` to `DISABLED` and back. For example, if you want to delete (p. 32) your CA, you must disable it 30 days before deletion. At various times, ACM PCA can set the following values:

- `CREATING` – ACM PCA is trying to create your private CA.
- `PENDING_CERTIFICATE` – Your private CA has been created but you must import the signed CA certificate.
- `ACTIVE` – Your private CA has been created and is functioning normally.
- `EXPIRED` – The CA certificate for your private CA has expired.
- `FAILED` – Your private CA is not functioning properly and has been placed in a failed state.

If the CA certificate for your private CA expires, ACM PCA sets the status to `EXPIRED`. You can set the status to `DISABLED` at this point, but you cannot set it to `ENABLED`. You might want to set the status to `DISABLED` if you want to delete the CA in 30 days. If you import a new CA certificate for your private CA, ACM PCA resets the status to `ACTIVE` unless you set it to `DISABLED` after the CA certificate expired. Only you can set the status to `DISABLED`.

You can update a certificate from the AWS Management Console or AWS CLI.

- Updating certificate revocation (console) (p. 31)
- Updating certificate status (console) (p. 31)
- Updating a private CA (AWS CLI) (p. 32)

To update certificate revocation (console)

1. Sign in to your AWS account and open the ACM PCA console at https://console.aws.amazon.com/acm-pca/home.
2. Choose **Private CAs**.
3. Choose your private CA from the list.
4. On the **Actions** menu, choose **Update CA revocation**.
5. Select **Enable CRL distribution** to generate certificate revocation lists (CRLs).
6. 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.
7. For **Custom CRL Name**, type an alias to hide your S3 bucket name from public view.
8. For **Valid for**, type a validity period in days.
9. Choose **Update**.

To update certificate status (console)

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

2. Choose **Private CAs**.
3. Choose your private CA from the list.
4. On the **Actions** menu, choose **Disable** to disable a private CA that's currently active or choose **Enable** to set a CA active.

To update your private CA (AWS CLI)

Use the update-certificate-authority command. You can use a file similar to the following to specify the CRL configuration.

```
{
  "CrlConfiguration":
    {"Enabled": true,
     "ExpirationInDays": 7,
     "CustomCname": "https://www.somename.crl",
     "S3BucketName": "your-crl-bucket-name"}
}
```

The following command uses the preceding file to configure revocation and sets 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/12345678-1234-1234-1234-1232456789012 \
--revocation-configuration file://C:\revoke_config.txt \
--status "ACTIVE"
```

Delete 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. This is necessary because you cannot update the key by importing an updated CA certificate that is associated with a new key. If you want to replace a CA, we recommend that you follow this process:

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 PCA does not check that all of the issued certificates have expired before it processes a delete request. You can generate an audit report (p. 35) 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`, `DISABLED`, or `FAILED` state. In order to delete a CA in the `ACTIVE` state, you must first disable it, or 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, the CA is restorable and its status is set to `DELETED`.

Because a private CA that is deleted in the `CREATING` or `FAILED` state is not given a restoration period, it cannot be restored later. For more information, see Restore Your Private CA (p. 33). 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. 11).

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

- Deleting a private CA (console) (p. 33)
- Deleting a private CA (AWS CLI) (p. 33)

To delete a private CA (console)

1. Sign in to your AWS account and open the ACM PCA console at https://console.aws.amazon.com/acm-pca/home.
2. Choose **Private CAs**.
3. Choose your private CA from the list.
4. If your CA is in the `ACTIVE` state, you must disable it. On the **Actions** menu, choose **Disable**.
5. On the **Actions** menu, choose **Delete**.
6. If your CA is in the `DISABLED` or `PENDING_CERTIFICATE` state, specify a restoration period from 7 to 30 days, during which the private CA can be restored. Then choose **Delete**.

   Note
   If your private CA is not in one of these states, it can not be restored later.
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 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/12345678-1234-1234-1234-123456789012 \
--permanent-deletion-time-in-days 16
```

Restore Your 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 Delete Your Private CA (p. 32). 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. 11).

Topics

- Restoring a Private CA (Console) (p. 34)

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 PCA console at https://console.aws.amazon.com/acm-pca/home.
2. Choose **Private CAs**.
3. Choose your private CA from the list.
4. You can restore a private CA if its current status is `DELETED`. On the **Actions** menu, choose **Restore**.
5. In the dialog box, choose **Restore** again.
6. If successful, the status of the private CA is set to its pre-deletion state. Choose **Enable** on the **Actions** menu 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 (p. 22) 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/12345678-1234-1234-1234-123456789012 \
--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/12345678-1234-1234-1234-123456789012
```

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/12345678-1234-1234-1234-123456789012 \
--status ACTIVE
```

Create an Audit Report for 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. The file is named in the following manner.

```
bucket-name/audit-report/CA-ARN/file-ARN.[json|csv]
```

You can generate a new report every 30 minutes and download it from your bucket. The following example shows a JSON-formatted report.

```
[{
  "awsAccountId": "123456789012",
  "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/e8cbd2bedb122329f97706bcfec990f8",
  "serial": "e8:cb:d2:be:db:12:23:29:f9:77:06:bc:fe:c9:90:f8",
  "subject":
 "1.2.840.113549.1.9.1=#161173616c6573406578616d706c652e636f6d,CN=www.example1.com,OU=Sales,O=Example
 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": "123456789012",
  "certificateArn": "arn:aws:acm-pca:region:account:certificate-authority/CA_ID/
certificate/2bae9a75d71b42b4e41e36f8b4b488fc",
  "serial": "2b:ae:9a:75:d7:1b:42:b4:e4:1e:36:f8:b4:b4:88:fc",
  "subject":
 "1.2.840.113549.1.9.1=#161970726f64407777772e70616c6f75736573616c65732e636f6d,CN=www.example3.com.com,
 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"
}]
```

You can create a report from the console or the AWS CLI.

- Creating a report (console) (p. 35)
- Creating a report (AWS CLI) (p. 36)

To create an audit report (console)

1. Sign in to your AWS account and open the ACM PCA console at https://console.aws.amazon.com/acm-pca/home.
2. Choose **Private CAs**.

3. Choose your private CA from the list.
4. On the **Actions** menu, choose **Generate audit report**.
5. 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 PCA creates and attaches the necessary policy to your bucket. If you choose **No**, you must attach the following policy to your bucket before you can generate an audit report. For instructions, see How Do I Add an S3 Bucket Policy?.

```
{
    "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:::bucket-name/*",
                "arn:aws:s3:::bucket-name"
            ]
        }
    ]
}
```

6. For **Output format**, choose **JSON** for JavaScript Object Notation or **CSV** for comma-separated values.
7. Choose **Generate audit report**.

To create an audit report (AWS CLI)

Use the `create-certificate-authority-audit-report` command to create the audit report. You must attach the following policy to the bucket.

```
{
    "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:::bucket-name/*",
                "arn:aws:s3:::bucket-name"
            ]
        }
    ]
```

```
}
```

```
aws acm-pca create-certificate-authority-audit-report \
--certificate-authority-arn arn:aws:acm-pca:region:account:\
certificate-authority/12345678-1234-1234-1234-123456789012 \
--s3-bucket-name >your-bucket-name \
--audit-report-response-format JSON
```

Add Tags to your Private Certificate Authority

A *tag* is a label that you can assign to a private CA. Each tag consists of a *key* and a *value*. You can use the ACM PCA console, AWS Command Line Interface (AWS CLI), or the PCA API to add, view, or remove tags for private CAs.

You can create custom tags that suit your needs. For example, you could tag private CAs with an `Environment=Prod` or `Environment=Beta` tag to identify which environment the CA is intended for.

Other AWS resources also support tagging. You can, therefore, 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.

The following basic restrictions apply to ACM PCA tags:

- The maximum number of tags per private CA is 50.
- The maximum length of a tag key is 127 characters.
- The maximum length of a tag value is 255 characters.
- Tag keys and values are case sensitive.
- The `aws:` prefix is reserved for AWS use; you cannot add, edit, or delete tags whose key begins with `aws:`. Tags that begin with `aws:` do not count against your tags-per-resource limit.
- If you plan to use your tagging schema across multiple services and resources, remember that other services may have other restrictions for allowed characters. Refer to the documentation for that service.
- ACM PCA tags are not available for use in the AWS Management Console's Resource Groups and Tag Editor.

You can tag a private CA from the AWS Management Console or AWS CLI:

- Tagging a private CA (console) (p. 37)
- Tagging a private CA (AWS CLI) (p. 38)

To tag a private CA (console)

1. Sign in to your AWS account and open the ACM PCA console at https://console.aws.amazon.com/acm-pca/home.
2. Choose **Private CAs**.
3. Choose your private CA from the list.
4. Choose the **Tags** tab.
5. Choose **Edit**.
6. Type a key and value pair.
7. Choose **Add Tag**.

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/12345678-1234-1234-1234-123456789012 \
--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/123455678-1234-1234-1234-123456789012 \
--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:region:account:\
certificate-authority/12345678-1234-1234-1234-123456789012 \
--tags Key=Purpose,Value=Website
```

Using CloudTrail

You can use 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 Certificate Authority

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


```
{
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  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/ac5a7c2e-19c8-4258-b74e-351c2b791fe1"
  },
  requestID: "332b0c69-8779-4625-bdef-2e95a4a18265",
  eventID: "cd27c874-ae6e-4585-9c1b-5abf5537ec39",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Creating an Audit Report

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

```
{
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:region:account:certificate-
authority/ac5a7c2e-19c8-4258-b74e-351c2b791fe1",
    s3BucketName: "your-bucket-name",
    auditReportResponseFormat: "JSON"
  },
  responseElements: {
    auditReportId: "2a7d28e7-a835-40a6-b19f-371186c62346",
    s3Key: "audit-report/ac5a7c2e-19c8-4258-b74e-351c2b791fe1/2a7d28e7-a835-40a6-
b19f-371186c62346.json"
  },
  requestID: "3b56d220-1660-4941-8160-b54dcc70592d",
  eventID: "ea95f673-e7be-411d-bb54-ca1ab844baaf",
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  errorCode: "InvalidStateException",
  errorMessage: "The certificate authority is not in a valid state for deletion.",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:region:account:certificate-
authority/09517d62-4f11-4bf8-a2c9-9e863792b675"
  },
  responseElements: null,
  requestID: "dae3e14f-62f6-42f3-acf4-630c47a09ee4",
  eventID: "c40abfac-53f7-420a-9b55-c3f2f2139de8",
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  errorCode: "InvalidStateException",
```

```
  errorMessage: "The certificate authority is not in a valid state for restoration.",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:region:account:certificate-
authority/09517d62-4f11-4bf8-a2c9-9e863792b675"
  },
  responseElements: null,
  requestID: "dae3e14f-62f6-42f3-acf4-630c47a09ee4",
  eventID: "c40abfac-53f7-420a-9b55-c3f2f2139de8",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Describing a Certificate Authority

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

```
{
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:region1:account:certificate-
authority/ac5a7c2e-19c8-4258-b74e-351c2b791fe1"
  },
  responseElements: null,
  requestID: "289d6bd9-cdca-43d1-a446-62e64a7006a4",
  eventID: "ff5f55b7-06de-4cb4-8fa1-5d29d3948f7b",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Retrieving a Certificate Authority Certificate

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

```
{
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:region:account:certificate-
authority/ac5a7c2e-19c8-4258-b74e-351c2b791fe1"
  },
  responseElements: null,
  requestID: "94cee046-bf52-4a69-b95c-eae662818410",
  eventID: "7dd83274-8c5f-4b9a-b9b6-371b53771ce9",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Retrieving the Certificate Authority Signing Request

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

```
{
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:region:account:certificate-
authority/ac5a7c2e-19c8-4258-b74e-351c2b791fe1"
  },
  responseElements: null,
  requestID: "7ce9f3bc-b459-436b-bac1-61e75fca3c6e",
  eventID: "93115f0b-d528-447a-9b22-87f868dbfd8e",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Retrieving a Certificate

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

```
{
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:region:account:certificate-
authority/ac5a7c2e-19c8-4258-b74e-351c2b791fe1",
    certificateArn: "arn:aws:acm-pca:region:account:certificate-
authority/ac5a7c2e-19c8-4258-b74e-351c2b791fe1/
certificate/6707447683a9b7f4055627ffd55cebcc"
  },
  responseElements: null,
  requestID: "ec0681a9-6202-496e-985a-adba939311e4",
  eventID: "89be1a2a-3340-4d74-a633-5582a2fba1d3",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Importing a Certificate Authority Certificate

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

```
{
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:region:account:certificate-
authority/ac5a7c2e-19c8-4258-b74e-351c2b791fe1",
    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: "36bbba0c-2d08-4995-99fc-964926103841",
  eventID: "17a38b26-49c1-41d2-8d15-f9362eeaaaa0",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Issuing a Certificate

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

```
{
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:region:account:certificate-
authority/ac5a7c2e-19c8-4258-b74e-351c2b791fe1",
    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/ac5a7c2e-19c8-4258-b74e-351c2b791fe1/
certificate/6707447683a9b7f4055627ffd55cebcc"
  },
  requestID: "0a5808dd-fc92-46f8-ba07-090ef8e9bcb4",
  eventID: "2816c6f0-184c-4d8b-b4ca-e54ab8dd6f2c",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Listing Certificate Authorities

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

```
{
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    maxResults: 10
  },
  responseElements: null,
  requestID: "047fb10f-b000-4915-b339-6b6ee8d9c5d6",
  eventID: "dbf32c79-77d2-4a3d-863b-8bf964e65fda",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Listing Tags

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

```
{
  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: "us-east-1",
```

```
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:us-east-1:account:certificate-
authority/ac5a7c2e-19c8-4258-b74e-351c2b791fe1"
  },
  responseElements: {
    tags: [{
      key: "Admin",
      value: "Alice"
    },
    {
      key: "User",
      value: "Bob"
    }]
  },
  requestID: "72819d8d-c6bc-4921-a944-95bb899ed911",
  eventID: "a349328f-e3e0-48ee-abc9-00526768080a",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Revoking a Certificate

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

```
{
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:region:account:certificate-
authority/ac5a7c2e-19c8-4258-b74e-351c2b791fe1",
    certificateSerial: "67:07:44:76:83:a9:b7:f4:05:56:27:ff:d5:5c:eb:cc",
    revocationReason: "KEY_COMPROMISE"
  },
  responseElements: null,
  requestID: "fb43ebee-a065-4b08-af60-29fd611bfa09",
  eventID: "06b7d2bf-3b4d-46ee-b617-6531ba2aa290",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Tagging Private Certificate Authorities

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

```
{
```

```
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:us-east-1:account:certificate-
authority/ac5a7c2e-19c8-4258-b74e-351c2b791fe1",
    tags: [{
      key: "Admin",
      value: "Alice"
    }]
  },
  responseElements: null,
  requestID: "816df59d-5022-47af-aa58-173e5c73c20a",
  eventID: "8c99648b-3d77-483f-b56b-5aaa85cb6cde",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Updating a Certificate Authority

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

```
{
  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: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:region:account:certificate-
authority/09517d62-4f11-4bf8-a2c9-9e863792b675",
    revocationConfiguration: {
      crlConfiguration: {
        enabled: true,
        expirationInDays: 3650,
        customCname: "your-custom-name",
        s3BucketName: "your-bucket-name"
      }
    },
    status: "DISABLED"
  },
  responseElements: null,
```

```
  requestID: "24f849f9-9966-4f13-8ff6-3e2e84b327fc",
  eventID: "16c78ea0-e3d7-4817-9bb0-0b997789678f",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Removing Tags from a Private Certificate Authority

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

```
{
  eventVersion: "1.05",
  userIdentity: {
    type: "IAMUser",
    principalId: "account",
    arn: "arn:aws:iam::account:user/namee",
    accountId: "account",
    accessKeyId: "Key_ID"
  },
  eventTime: "2018-02-02T00:21:50Z",
  eventSource: "acm-pca.amazonaws.com",
  eventName: "UntagCertificateAuthority",
  awsRegion: "us-east-1",
  sourceIPAddress: "xx.xx.xx.xx",
  userAgent: "aws-cli/1.14.28 Python/2.7.9 Windows/8 botocore/1.8.32",
  requestParameters: {
    certificateAuthorityArn: "arn:aws:acm-pca:us-east-1:account:certificate-
authority/ac5a7c2e-19c8-4258-b74e-351c2b791fe1",
    tags: [{
      key: "Admin",
      value: "Alice"
    }]
  },
  responseElements: null,
  requestID: "b9b385b4-a721-4018-be15-d679ce5c2d6e",
  eventID: "e32c054d-bc19-40a7-bef9-194ed6848a3b",
  eventType: "AwsApiCall",
  recipientAccountId: "account"
}
```

Using the ACM PCA API

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 PCA. This is recommended over the HTTP API. For ore information, see Tools for Amazon Web Services. The following topics show you how to use the AWS SDK for Java to program the PCA API.

The GetCertificateAuthorityCsr (p. 62), GetCertificate (p. 59), and DescribeCertificateAuthorityCsr (p. 57) commands support waiters, which allow you to control the progression of your code based on the presence or state of certain resources. For more information, see the respective pages below, as well as Waiters in the AWS SDK for Java in the AWS blog.

Topics

CreateCertificateAuthority

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

The function 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 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, and X.500 subject information. The CRL configuration specifies the CRL expiration period in days (the validity period of the CRL), the Amazon S3 bucket that will contain the CRL, and 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.CreateCertificateAuthorityRequest;
import com.amazonaws.services.acmpca.model.CreateCertificateAuthorityResult;
import com.amazonaws.services.acmpca.model.ASN1Subject;
import com.amazonaws.services.acmpca.model.KeyAlgorithm;
import com.amazonaws.services.acmpca.model.SigningAlgorithm;
import com.amazonaws.services.acmpca.model.CertificateAuthorityConfiguration;
import com.amazonaws.services.acmpca.model.CertificateAuthorityType;
import com.amazonaws.services.acmpca.model.CrlConfiguration;

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


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 your credentials from file.", e);
      }

      // Define the endpoint for your sample.
      String endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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 the CA subject.
      ASN1Subject subject = new ASN1Subject();
      subject.setOrganization("Example Company");
      subject.setOrganizationalUnit("Corporate");
      subject.setCountry("US");
      subject.setState("Washington");
      subject.setLocality("Seattle");
      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);

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

        // 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/12345678-1234-1234-1234-123456789012
```

CreateCertificateAuthorityAuditReport

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

The function creates an audit report that lists every time a certificate is issued or revoked. The report is saved in the 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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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();

      req.setCertificateAuthorityArn("arn:aws:acm-pca:region:account:" +
            "certificate-authority/12345678-1234-1234-1234-123456789012");

      // 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
```

DeleteCertificateAuthority

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

This operation deletes the private certificate authority (CA) that you created by 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 which the private CA can be restored to `disabled` status. By default, if you do not set the `PermanentDeletionTimeInDays` parameter, the restoration period is 30 days. After this period expires, the private CA is permanently deleted and cannot be restored. For more information, see Restore a CA (p. 33).

For a Java example that shows you how to use the RestoreCertificateAuthority operation, see RestoreCertificateAuthority (p. 72).

```
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;


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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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 ARN.
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:account:" +
            "certificate-authority/11111111-2222-3333-4444-555555555555");

   // 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;
```

```
        }
    }
}
```

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 PCA 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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
```

```
      String endpointRegion = "region";
      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.
      DescribeCertificateAuthorityRequest req = new DescribeCertificateAuthorityRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:account:"+
          "certificate-authority/12345678-1234-1234-1234-123456789012");

      // 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 function.

The function lists information about a specific audit report that you created by calling the CreateCertificateAuthorityAuditReport function. 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;

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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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/12345678-1234-1234-1234-123456789012");

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

      // Create waiter to wait on successful creation of the audit repor filet.
      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 function.

The function retrieves a certificate from your private CA. The ARN of the certificate is returned when you call the IssueCertificate function. You must specify both the ARN of your private CA and the ARN of the issued certificate when calling the **GetCertificate** function. You can retrieve the certificate if it is in the `ISSUED` state. You can call the CreateCertificateAuthorityAuditReport function 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;

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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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/12345678-1234-1234-1234-123456789012" +
            "/certificate/793f0d5b6a04125e2c9cfb52373598fe");

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

      // 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 function.

This function retrieves the certificate and certificate chain for your private certificate authority (CA). Both the certificate and the chain are base64 PEM-encoded. 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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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,
      GetCertificateAuthorityCertificateRequest req =
```

```
            new GetCertificateAuthorityCertificateRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:account:" +
            "certificate-authority/12345678-1234-1234-1234-123456789012");

      // 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 function.

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

```
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;

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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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 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(PrivateCAException e) {
      //Unexpected service exception.
      }

      // Create the request object and set the CA ARN.
      GetCertificateAuthorityCsrRequest req = new GetCertificateAuthorityCsrRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:account: " +
        "certificate-authority/12345678-1234-1234-1234-123456789012");

      // 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-----
```

ImportCertificateAuthorityCertificate

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

This function imports your signed private CA certificate into ACM PCA. Before you can call this function, you must create the private certificate authority by calling the CreateCertificateAuthority function. You must then generate a certificate signing request (CSR) by calling the GetCertificateAuthorityCsr function. 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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
   String endpointRegion = "region";
   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-----" +
         "base64-encoded certificate" +
         "-----END CERTIFICATE-----";
   ByteBuffer certByteBuffer = stringToByteBuffer(strCertificate);
   req.setCertificate(certByteBuffer);

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

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

   // 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 function.

This function uses your private certificate authority (CA) to issue a private certificate. This action returns the Amazon Resource Name (ARN) of the certificate. You can retrieve the certificate by calling the GetCertificate and specifying the ARN.

```
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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
    String endpointRegion = "region";
    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-----" +
    "    base64-encoded certificate     " +
    "------END CERTIFICATE REQUEST------";
    ByteBuffer csrByteBuffer = stringToByteBuffer(strCSR);
    req.setCsr(csrByteBuffer);

    // 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/12345678-1234-1234-1234-123456789012/
certificate/2669d5cacb539c0830998c23babab8dc
```

ListCertificateAuthorities

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

This function lists the private certificate authorities (CAs) that you created by using the CreateCertificateAuthority 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.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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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());
   }
}
```

Your output should be similar to the following if you have any certificate authorities to list.

```
[{
 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
  }
 }
}]
```

ListTags

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

This function 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 function to add one or more tags to your CA. Call the UntagCertificateAuthority function 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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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.
      ListTagsRequest req = new ListTagsRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:account:" +
          "certificate-authority/12345678-1234-1234-1234-123456789012");

      // 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);
   }
}
```

Your output should be similar to the following if you have any tags to list.

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

RestoreCertificateAuthority

The following Java sample shows how to use the RestoreCertificateAuthority function. A private CA can be restored at any time during its restoration period. Currently, this period can last 7 to 30 days from the date of deletion and can be defined when you delete the CA. For more information, see Restore a CA (p. 33). See also the DeleteCertificateAuthority (p. 54) 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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      EndpointConfiguration endpoint =
            new AwsClientBuilder.EndpointConfiguration(endpointProtocol, endpointRegion);

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

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

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

      // 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 function.

This function revokes a certificate that you issued by calling the IssueCertificate function. If you enable a certificate revocation list (CRL) when you create or update your private CA, information about the revoked certificates will be included in the CRL. ACM PCA writes the CRL to an 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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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/12345678-1234-1234-1234-123456789012");

      // 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 function.

This function 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. You specify the private CA on input by its Amazon Resource Name (ARN). You specify the tag by using a key-value pair. You can apply a tag to just one private CA if you want to identify a specific characteristic of that CA, or you can apply the same tag to multiple private CAs if you want to filter for a common relationship among those CAs. To remove one or more tags, use the UntagCertificateAuthority function. Call the ListTags function 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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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 ARN of the certificate.
      TagCertificateAuthorityRequest req = new TagCertificateAuthorityRequest();
      req.setCertificateAuthorityArn("arn:aws:acm-pca:region:account:" +
          "certificate-authority/12345678-1234-1234-1234-123456789012");
      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 function.

This function 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 function, the tag will be 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 function. Call the ListTags function 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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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);
      tags.add(tag);

      // Create a request object and specify the ARN of the certificate.
      UntagCertificateAuthorityRequest req = new UntagCertificateAuthorityRequest();
      req.withCertificateAuthorityArn("arn:aws:acm-pca:region:account:" +
            "certificate-authority/12345678-1234-1234-1234-123456789012");
      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 function.

The function 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 endpointProtocol = "https://acm-pca.region.amazonaws.com/";
      String endpointRegion = "region";
      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/12345678-1234-1234-1234-123456789012");

      // 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);

      // 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;
      }
   }
}
```

Troubleshooting

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

Topics

Troubleshoot 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 documentation.

Authentication and Access

Access to ACM PCA 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.

Topics

Managing Access to Your Private Certificate Authority.

Every AWS resource belongs to an AWS account, and permissions to create or access the resources are defined in permissions policies in that account. An account administrator can attach permissions policies to IAM identities (users, groups, and roles). Some services (including ACM PCA) also support attaching permissions policies to resources.

> **Note**
> An *account administrator* (or administrator user) is a user with administrator permissions. For more information, see Creating an Admin User and Group in the *IAM User Guide*.

When managing permissions, you decide who gets the permissions, the resources they get permissions for, and the specific actions they are allowed to perform.

Topics

ACM PCA Resources and Operations

In ACM PCA, the primary resource that you work with is a *private certificate authority (PCA)*. Every private CA you own or control is identified by an Amazon Resource Name (ARN), which has the following form.

```
arn:aws:acm:AWS_Region:AWS_Account:certificate-
authority/12345678-1234-1234-1234-123456789012
```

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 PCA). 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 PCA 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. It is similar to an *IAM user*, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys that can be used to access AWS services and resources. IAM roles with temporary credentials are useful in the following situations:

  - **Federated user access** – Instead of creating an IAM user, you can use existing user 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** – You can use an IAM role in your account to grant an AWS service permissions to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an Amazon S3 bucket on your behalf and then load data from that bucket into an Amazon Redshift cluster. 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 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 Resource Ownership

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 will be able to create the CA. However, the account to which the role belongs will own the private CA.

Managing Access to Private CAs

A *permissions policy* describes who has access to what. This section explains the available options for creating permissions policies.

> **Note**
> This documentation discusses using IAM in the context of ACM PCA. 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.

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:

- Customer Managed Policies (p. 84) 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. 84) are policies that you create and manage and which you embed directly into a single user, group, or role.

Customer Managed Policies

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. For more information, see Customer Managed Policies .

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 PCA 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:AWS_Region:AWS_Account:certificate-
authority/12345678-1234-1234-1234-123456789012"
  }
}
```

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:AWS_Region:AWS_Account:certificate/12345678-1234-1234-1234-123456789012"
  }
}
```

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:AWS_Region:AWS_Account:certificate/12345678-1234-1234-1234-123456789012"
  }
}
```

Read-Only Access to ACM PCA

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 PCA

The following policy allows a user to perform any ACM PCA 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": "*"
  }]
}
```

ACM PCA API Permissions: Actions and Resources Reference

When you are setting up access control and writing permissions policies that you can attach to an IAM identity (identity-based policies), you can use the following table as a reference. The first column in the table lists each ACM PCA API operation. You specify actions in a policy's `Action` element. The remaining columns provide the additional information.

You can use the IAM policy elements in your ACM PCA policies to express conditions. For a complete list, see Available Keys in the *IAM User Guide*.

Note
To specify an action, use the `acm-pca:` prefix followed by the API operation name (for example, `acm-pca:IssueCertificate`).

ACM PCA API Operations and Permissions

ACM PCA API Operations	Required Permissions (API Actions)	Resources
CreateCertificateAuthority	acm-pca:CreateCertificateAuthority	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate-authority/*certificate_ID*
CreateCertificateAuthorityAuditReport	acm-pca:CreateCertificateAuthorityAuditReport	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate-authority/*certificate_ID*
DeleteCertificateAuthority	acm-pca:DeleteCertificateAuthority	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate-authority/*certificate_ID*
DescribeCertificateAuthority	acm-pca:DescribeCertificateAuthority	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate-authority/*certificate_ID*
DescribeCertificateAuthorityAuditReport	acm-pca:DescribeCertificateAuthorityAuditReport	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate-authority/*certificate_ID*
GetCertificate	acm-pca:GetCertificate	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate-authority/*certificate_ID*
GetCertificateAuthorityCertificate	acm-pca:GetCertificateAuthorityCertificate	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate-authority/*certificate_ID*
GetCertificateAuthorityCsr	acm-pca:GetCertificateAuthorityCsr	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate-authority/*certificate_ID*
ImportCertificateAuthorityCertificate	acm-pca:ImportCertificateAuthorityCertificate	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate-authority/*certificate_ID*
IssueCertificate	acm-pca:IssueCertificate	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate-authority/*certificate_ID*
ListCertificateAuthorities	acm-pca:ListCertificateAuthorities	N/A
ListTags	acm-pca:ListTags	N/A
RevokeCertificate	acm-pca:RevokeCertificate	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate-authority/*certificate_ID*

ACM PCA API Operations	Required Permissions (API Actions)	Resources
TagCertificateAuthority	acm-pca:TagCertificateAuthority	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate_authority/*certificate_ID*
UntagCertificateAuthority	acm-pca:UntagCertificateAuthority	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate_authority/*certificate_ID*
UpdateCertificateAuthority	acm-pca:UpdateCertificateAuthority	arn:aws:acm-pca:*AWS_Region*:*AWS_Account*:certificate_authority/*certificate_ID*

Document History

Latest documentation update: June 20, 2018

The following table describes the documentation release history of ACM Private CA after May 2018.

Change	Description	Date
New content	Private CA restore allows customers to restore certificate authorities (CAs) for up to 30 days after they have been deleted. For more information, see Restore Your Private CA (p. 33).	June 20, 2018

The following table describes the documentation release history of AWS Certificate Manager Private Certificate Authority before June 2018.

Latest documentation update: April 04, 2018

Change	Description	Date
New guide	This release introduces ACM PCA.	April 04, 2018