Code Signing for AWS IoT

Developer Guide

Code Signing for AWS IoT: Developer Guide

Table of Contents

What Is Code Signing for AWS IoT?

With code signing for AWS IoT, you can sign code that you create for any IoT device that is supported by Amazon Web Services (AWS). Code signing is available through Amazon FreeRTOS and AWS IoT Device Management, and integrated with AWS Certificate Manager (ACM). In order to sign code, you import a third-party code signing certificate with ACM that is used to sign updates in Amazon FreeRTOS and AWS IoT Device Management.

You can use code signing through the Amazon FreeRTOS console to sign your firmware images before deploying them to a microcontroller. You can access the Amazon FreeRTOS console here.

You can use code signing through the AWS IoT Device Management console to sign your code images before deploying them using an over-the-air (OTA) update job. You can access the AWS IoT Device Management console here.

You can also use one of the AWS SDKs or command line tools to interact with all code signing operations.

For general information about code signing for AWS IoT, see the following topics. For information about the code signing API, see the Code Signing for AWS IoT API Reference.

Topics

Integrated Services

Code signing is integrated with the following services.

Amazon FreeRTOS

Amazon FreeRTOS is a microcontroller operating system based on the FreeRTOS kernel. It includes libraries for connectivity and security. You can build and deploy your embedded applications on top of Amazon FreeRTOS. To ensure the security of deployments to these microcontrollers, Amazon FreeRTOS uses code signing for the initial manufacture of these devices and subsequent over-the-air updates. You can use code signing through the Amazon FreeRTOS console to sign your code images before you deploy them to a microcontroller.

AWS IoT Device Management

With AWS IoT Device Management you can manage Internet-connected devices and establish secure, bidirectional communication between them. To do so, AWS IoT Device Management uses code signing to authenticate each device in your IoT environment. You can use code signing through the AWS IoT Device Management console to sign your code images before you deploy them to a microcontroller.

AWS Certificate Manager (ACM)

ACM handles the complexity of creating and managing or importing SSL/TLS certificates. You use ACM to create an ACM certificate or import a third-party certificate that you use for signing. You must have a certificate to sign code. For more information about certificates, see AWS Certificate Manager User Guide.

CloudTrail

You can use AWS CloudTrail to record API calls that are made to code signing. CloudTrail is an AWS service that simplifies governance, compliance, and risk auditing by providing visibility into actions made in your AWS account. For more information, see the AWS CloudTrail User Guide.

CloudWatch Events

You can use CloudWatch Events to monitor code signing objects. CloudWatch Events is an AWS service that monitors the statuses of AWS resources in real time, making it easy to automate service work flows and notifications. For more information, see the Amazon CloudWatch Events User Guide.

Supported Regions

Code signing for AWS IoT is supported in the same regions as Amazon FreeRTOS and AWS IoT Device Management. Visit AWS Regions and Endpoints in the *AWS General Reference* or the AWS Region Table to see the regional support for both services.

Limits

You can make 5 calls per second to the StartSigningJob API operation. You can make 25 calls per second to any other code signing for AWS IoT API operations. These limits apply to each AWS region and each AWS account.

Pricing for Code Signing for AWS IoT

There is no additional charge for the code signing feature for Amazon FreeRTOS or AWS IoT Device Management. You pay for the storage of signed and unsigned objects in Amazon S3 based on your usage history. There are no minimum fees and no required upfront commitments.

Getting Started

You can use code signing through the Amazon FreeRTOS or AWS IoT Device Management consoles when you create an over-the-air (OTA) job, or by using the code signing API.

Topics

Obtain and Import a Code Signing Certificate

Before you can use code signing for AWS IoT, you must have or obtain a code signing certificate. Code signing certificates typically contain a `Digital Signature` value in the `Key Usage` extension and a `Code Signing` value in the `Extended Key Usage` extension.

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 4111 (0x100f)
    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: Nov 14 17:32:30 2017 GMT
            Not After : Nov 14 17:32:30 2018 GMT
        Subject: C=US, ST=Washington, L=Seattle, O=Example Company, OU=corp,
 CN=www.example.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:ac:96:8f:64:1a:4d:5c:cc:e4:50:a9:19:f3:c1:
                    03:8f:1a:db:f5:15:18:65:fb:6e:3f:84:ae:02:9e:
                    a2:e1:62:40:05:10:b6:35:59:63:c7:b3:17:4a:e1:
                    12:9f:29:42:e4:2b:bb:83:db:b1:cd:42:83:0a:9f:
                    70:ca:81:6a:9b:58:1d:4e:a0:69:04:bc:0b:f4:7e:
                    34:fc:af:79:f1:31:6c:7e:a5:eb:b1:85:9e:5e:ef:
                    df:34:7c:aa:13:01:f5:cc:ee:a1:9c:d9:4d:17:e8:
                    c8:8b:d0:77:2e:80:3f:7e:41:ea:84:2f:11:22:59:
                    bd:fa:90:eb:26:ec:e7:b2:0e:9d:ce:b5:8a:a0:b9:
                    17:4c:8b:3a:b5:28:61:eb:d3:a6:ed:db:5c:26:e6:
                    7d:af:33:b6:9f:f0:9d:fb:fc:10:e0:52:cb:60:5c:
                    08:c3:33:4a:b4:8a:4e:3a:54:4e:43:3d:b9:f2:5e:
                    4e:89:95:c2:a5:df:88:a2:24:71:d3:ee:b3:ef:0b:
                    18:1d:55:54:16:ff:9b:95:6e:ae:71:d3:f2:d1:7e:
                    f2:8b:67:34:f8:11:fe:ab:8f:6b:88:c3:b9:8e:1d:
                    07:bc:62:27:45:7e:0c:a0:7b:ef:bf:26:f8:50:df:
                    ac:d8:8f:a5:ed:fe:9f:ee:20:dc:a6:33:3e:94:25:
```

```
                ce:67
            Exponent: 65537 (0x10001)
    X509v3 extensions:
        X509v3 Basic Constraints:
            CA:FALSE
        X509v3 Subject Key Identifier:
            22:93:86:26:D3:1B:32:1C:79:1B:5C:E4:EB:2A:6A:DB:77:87:D7:FB
        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 Key Usage:
            Digital Signature
        X509v3 Extended Key Usage:
            Code Signing
Signature Algorithm: sha256WithRSAEncryption
     38:41:ba:c3:f0:88:97:3e:a1:0f:e3:d4:55:d6:d0:a2:4e:ac:
     da:83:67:27:49:23:88:9b:20:e1:e1:b7:55:78:3c:5a:9b:7a:
     75:ee:3a:0f:ed:20:4e:23:31:29:ac:07:91:61:f1:86:75:08:
     fa:f5:3c:4a:7b:79:3c:39:a5:45:97:10:5c:f4:a0:04:af:e8:
     5b:ca:d1:a5:ce:14:dc:14:c6:54:b1:ba:6a:2c:52:2c:2f:07:
     52:8a:a7:00:97:c7:ee:65:bb:df:36:7f:53:d0:7d:a4:6e:ba:
     bb:d2:d4:b5:25:bb:b1:0d:bd:91:10:28:e1:34:df:79:01:78:
     45:4e
```

We recommend that you purchase a code signing certificate from a company with a good reputation for security. Do not use a self-signed certificate for any purpose other than testing.

After you have obtained the certificate, you must import it into AWS Certificate Manager (ACM). ACM returns an Amazon Resource Name (ARN) for the certificate. You must use the ARN when you call the StartSigningJob action. For more information about importing, see Importing Certificates in the *AWS Certificate Manager User Guide*.

Add Your Source Files to an Amazon S3 Bucket

The following procedure discusses how to add your source code to an Amazon S3 bucket. For more information, see Create a Bucket in the *Amazon S3 Getting Started Guide*.

To add source code to an Amazon S3 bucket

1. Sign into the AWS Management Console at https://console.aws.amazon.com/console/home.
2. Choose the **S3** service.
3. Choose **Create Bucket**.
4. For **Bucket name**, enter a unique DNS-compliant name.
5. Choose the Region where you want your bucket to reside. The source bucket must be in the same Region as the one where you start your signing job. Also, this must be one of the Supported Regions (p. 2).
6. Choose **Next**.
7. Choose **Versioning**. Your S3 bucket must be version enabled.
8. Choose **Enable versioning** and then choose **Save**.
9. Choose **Next** and then choose **Next** again on the **Set permissions** page.
10. Review your choices. Choose **Create bucket** if you are satisfied with your input. Choose **Previous** to start over.
11. Choose the S3 bucket that you just created and then choose **Upload** to add your code image to the bucket. For more information, see Add an Object to a Bucket in the *Amazon S3 Getting Started Guide*.

Create a Destination Amazon S3 Bucket

The following procedure explains how to create an Amazon S3 bucket to which code signing for AWS IoT can write your signed code. For more information, see Create a Bucket in the *Amazon S3 Getting Started Guide*.

To create a destination Amazon S3 bucket for your code signing code

1. Sign into the AWS Management Console at https://console.aws.amazon.com/console/home.
2. Choose the **S3** service.
3. Choose **Create Bucket**.
4. For **Bucket name**, enter a unique DNS-compliant name.
5. Choose the region where you want your bucket to reside. The destination bucket must be in the same region as the one where you start your signing job. Also, this must be one of the Supported Regions (p. 2).
6. Choose **Next**.
7. On the **Set properties** page, choose **Next**.
8. On the **Set permissions** page, choose **Next**.
9. Review your choices on the **Review** page. Choose **Create bucket** if you are satisfied with your input. Choose **Previous** to start over.

Define an IAM Policy

To allow user access to code signing commands, you can attach a policy to an IAM group that grants permission to sign code. For example, you can manually create the following policy or edit it to create a more restrictive policy. For more information, see Overview of IAM Policies.

To manually create an IAM policy:

1. Sign in to the IAM console at https://console.aws.amazon.com/iam/.
2. In the left navigation pane, choose **Policies**.
3. Choose **Create Policy**.
4. Choose the **JSON** tab.
5. Select the existing text and press **Delete**.
6. Copy and paste the following. This policy allows the user to which it is attached to access all operations available in the code signing API. You can edit the policy to make it more restrictive. When you're done, choose **Review Policy**.

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

In order to use the StartSigningJob (p. 15) API operation, you must specify an Amazon S3 bucket to which to save the signing job. In order to do so, attach the following policy to the designated user.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "signer:StartSigningJob",
        "s3:GetObjectVersion",
        "s3:PutObject"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

7. Enter a policy name and description. Then choose **Create Policy**.
8. After you create the policy, choose **Users** in the navigation pane of the IAM console.
   a. Choose the name of a user.
   b. Make sure that the **Permissions** tab is active. Choose **Add permissions**.
   c. Choose **Attach existing policies directly**.
   d. Select the check box for the policy that you created in the preceding step. Choose **Next: Review**.
   e. If everything looks correct, choose **Add permissions**.

Signing Platforms in Code Signing for AWS IoT

A signing platform in code signing for AWS IoT is a predefined set of instructions that specifies hash and encryption algorithms. Code signing uses these instructions to sign a file in Amazon FreeRTOS or AWS IoT Device Management. Although users cannot edit a signing platform, they can modify the platform's implementation by including hash or encryption algorithm overrides in a signing profile (p. 6).

To see all available signing platforms, use the ListSigningPlatforms operation. For information about a particular signing platform, use the GetSigningPlatform operation.

For more information about the configurations and parameters that are contained in signing platforms, see SigningPlatform in the *Code Signing for AWS IoT API Reference*.

Signing Profiles in Code Signing for AWS IoT

A signing profile is a code signing template that can be used to carry out a predefined signing job. A signing profile designates the signing material (a file) to be signed with a particular signing platform (p. 6), as well as any hash or encryption algorithm overrides to be applied to that signing platform. Once created, administrators can use AWS Identity and Access Management (IAM) to delegate control over signing profiles. Doing so ensures that only approved users have access to particular code signing, Amazon FreeRTOS, AWS IoT Device Management, and AWS Certificate Manager resources. For more information about managing user permissions in code signing, see the section called “Customer Managed Policies” (p. 19).

In order to start a signing job with the StartSigningJob operation, you must designate a signing profile.

Use the PutSigningProfile operation to create a signing profile, and the CancelSigningProfile operation to cancel a signing profile. Canceled profiles remain in the `CANCELED` state for two years after the `CancelSigningProfile` operation is issued, after which time they are deleted. To find the status of a particular signing profile, use the GetSigningProfile operation.

For a list of all available signing profiles, including those in the `CANCELED` state, use the ListSigningProfiles operation.

For more information about the configurations and parameters related to signing profiles, see SigningPlatform in the *Code Signing for AWS IoT API Reference Guide*.

Using the Code Signing for AWS IoT API

You can use the code signing API to interact with the service programmatically. For more information, see the code signing Developer Guide. The following topics show you how to use Java to program the SDK.

Topics

CancelSigningProfile

The following example shows how to use the CancelSigningProfile operation.

```
package com.examples;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.signer.AWSSigner;
import com.amazonaws.services.signer.AWSSignerClient;
import com.amazonaws.services.signer.model.CancelSigningProfileRequest;

/**
* This examples demonstrates creating a code signing profile and using it to start a
 signing job.
*/
public class CancelSigningProfile {

    public static void main(String[] s) {

        final String credentialsProfile = "default";
        final String codeSigningProfileName = "MyProfile";


        // Create a client.
        final AWSSigner client = AWSSignerClient.builder()
            .withRegion("us-west-2")
            .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
            .build();
```

```
        // cancel a signing profile
        client.cancelSigningProfile(new
 CancelSigningProfileRequest().withProfileName(codeSigningProfileName));
    }
}
```

DescribeSigningJob

The following example shows you how to use the DescribeSigningJob operation. Call the StartSigningJob operation before calling `DescribeSigningJob`. `StartSigningJob` returns a `jobId` value that you use when you call `DescribeSigningJob`.

```
package com.amazonaws.samples;

import com.amazonaws.services.signer.AWSSigner;
import com.amazonaws.services.signer.AWSSignerClient;
import com.amazonaws.services.signer.model.DescribeSigningJobRequest;
import com.amazonaws.services.signer.model.DescribeSigningJobResult;

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

import com.amazonaws.services.signer.model.ResourceNotFoundException;
import com.amazonaws.services.signer.model.AccessDeniedException;
import com.amazonaws.services.signer.model.InternalServiceErrorException;
import com.amazonaws.AmazonClientException;

/**
* This sample demonstrates how to use the DescribeSigningJob operation in the
* code signing service.
*
* Input Parameters:
*
*   jobId  - String that contains the ID of the job that was returned by the
*            StartSigningJob operation.
*
*/


public class DescribeSigningJob {

   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 in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider().getCredentials();
      }
      catch (Exception ex) {
          throw new AmazonClientException("Cannot load your credentials from file.", ex);
      }

      // Specify the endpoint and region.
      EndpointConfiguration endpoint =
            new EndpointConfiguration("https://endpoint","region");

      // Create a client.
      AWSSigner client = AWSSignerClient.builder()
            .withEndpointConfiguration(endpoint)
```

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

      // Create a request object.
      DescribeSigningJobRequest req = new DescribeSigningJobRequest()
            .withJobId("cc9067a9-9258-489a-abae-1c3408191071");

      // Create a result object.
      DescribeSigningJobResult result = null;
      try {
         result = client.describeSigningJob(req);
      }
      catch (ResourceNotFoundException ex)
      {
         throw ex;
      }
      catch (AccessDeniedException ex)
      {
         throw ex;
      }
      catch (InternalServiceErrorException ex)
      {
         throw ex;
      }

      // Display the information for your signing job.
      System.out.println(result.toString());

   }
}
```

GetSigningPlatform

The following example shows how to use the GetSigningPlatform operation.

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.signer.AWSsigner;
import com.amazonaws.services.signer.AWSsignerClient;
import com.amazonaws.services.signer.model.GetSigningPlatformRequest;
import com.amazonaws.services.signer.model.GetSigningPlatformResult;

public class GetSigningPlatform {


    public static void main(String[] s) {

        final String credentialsProfile = "default";
        final String codeSigningPlatformId = "AmazonFreeRTOS";

        // Create a client.
        final AWSsigner client = AWSsignerClient.builder()
                .withRegion("us-west-2")
                .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
                .build();

        GetSigningPlatformResult result = client.getSigningPlatform(
                new GetSigningPlatformRequest().withPlatformId(codeSigningPlatformId));

        System.out.println("Display Name : " + result.getDisplayName());
        System.out.println("Platform Id : " + result.getPlatformId());
        System.out.println("Signing Configuration : " + result.getSigningConfiguration());
    }
```

```
}
```

GetSigningProfile

The following example shows how to use the GetSigningProfile operation.

```
package com.examples;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.signer.AWSSigner;
import com.amazonaws.services.signer.AWSSignerClient;
import com.amazonaws.services.signer.model.GetSigningProfileRequest;
import com.amazonaws.services.signer.model.GetSigningProfileResult;

/**
* This examples demonstrates retreiving a signing profile's information.
*/
public class GetSigningProfile {

    public static void main(String[] s) {

        final String credentialsProfile = "default";
        final String codeSigningProfileName = "MyProfile";

    // Create a client.
    final AWSSigner client = AWSSignerClient.builder()
        .withRegion("us-west-2")
        .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
        .build();

    // Get a signing profile.
    GetSigningProfileResult getSigningProfileResult = client.getSigningProfile(new
        GetSigningProfileRequest().withProfileName(codeSigningProfileName));

    System.out.println("Profile Name : " + getSigningProfileResult.getProfileName());
    System.out.println("Certificate Arn : " +
 getSigningProfileResult.getSigningMaterial().getCertificateArn());
    System.out.println("Platform : " + getSigningProfileResult.getPlatform());
    }
}
```

ListSigningJobs

The following example shows how to use the ListSigningJobs operations. This operation lists all of the signing jobs that you have performed in your account. Call the StartSigningJob operation before you call `ListSigningJobs`. You can also call DescribeSigningJob and specify a `jobId` to see information about a specific signing job created by calling `StartSigningJob`.

```
package com.amazonaws.samples;

import com.amazonaws.services.signer.AWSSigner;
import com.amazonaws.services.signer.AWSSignerClient;
import com.amazonaws.services.signer.model.ListSigningJobsRequest;
import com.amazonaws.services.signer.model.ListSigningJobsResult;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.AWSStaticCredentialsProvider;
```

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

import com.amazonaws.services.signer.model.ValidationException;
import com.amazonaws.services.signer.model.AccessDeniedException;
import com.amazonaws.services.signer.model.ThrottlingException;
import com.amazonaws.services.signer.model.InternalServiceErrorException;
import com.amazonaws.AmazonClientException;

/**
* This sample demonstrates how to use the ListSigningJobs operation in the
* code signing for AWS IoT service.
*
* Input Parameters:
*
* status      - String that specifies the status that you want to use for filtering.
*               This can be:
*                  - InProgress
*                  - Failed
*                  - Succeeded
* platform    - String that contains the name of the microcontroller platform that
*               you want to use for filtering.
* requestedBy - IAM  principal that requested the signing job.
* maxResults  - Use this parameter when paginating results to specify the maximum
*               number of items to return in the response. If additional items exist
*               beyond the number you specify, the nextToken element is sent in the
*               response. Use the nextToken value in a subsequent request to retrieve
*               additional items.
* nextToken   - Use this parameter only when paginating results and only in a
*               subsequent request after you receive a response with truncated results.
*               Set it to the value of nextToken from the response you
*               just received.
*
*/

public class ListSigningJobs {

   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 in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider().getCredentials();
      }
      catch (Exception ex) {
          throw new AmazonClientException("Cannot load your credentials from file.", ex);
      }

      // Specify the endpoint and region.
      EndpointConfiguration endpoint =
            new EndpointConfiguration("https://endpoint","region");

      // Create a client.
      AWSSigner client = AWSSignerClient.builder()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();

      // Create a request object.
      ListSigningJobsRequest req = new ListSigningJobsRequest()
            .withStatus("Succeeded")
            .withPlatform("Platform")
            .withMaxResults(10);

      // Create a result object.
```

```
      ListSigningJobsResult result = null;
      try {
         result = client.listSigningJobs(req);
      }
      catch (ValidationException ex)
      {
         throw ex;
      }
      catch (AccessDeniedException ex)
      {
         throw ex;
      }
      catch (ThrottlingException ex)
      {
         throw ex;
      }
      catch (InternalServiceErrorException ex)
      {
         throw ex;
      }

      // Display the information for your signing job.
      System.out.println(result.toString());

   }
}
```

ListSigningPlatforms

The following example shows how to use the ListSigningPlatforms operation.

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.signer.AWSsigner;
import com.amazonaws.services.signer.AWSsignerClient;
import com.amazonaws.services.signer.model.ListSigningPlatformsRequest;
import com.amazonaws.services.signer.model.ListSigningPlatformsResult;
import com.amazonaws.services.signer.model.SigningPlatform;

public class ListSigningPlatforms {

    public static void main(String[] s) {

        final String credentialsProfile = "default";

        // Create a client.
        final AWSsigner client = AWSsignerClient.builder()
                .withRegion("us-west-2")
                .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
                .build();

        ListSigningPlatformsResult result;
        String nextToken = null;
        do {
            result = client.listSigningPlatforms(new
 ListSigningPlatformsRequest().withNextToken(null));

            for (SigningPlatform platform : result.getPlatforms()) {
                System.out.println("Display Name : " + platform.getDisplayName());
                System.out.println("Platform Id : " + platform.getPlatformId());
                System.out.println("Signing Configuration : " +
 platform.getSigningConfiguration());
            }
```

```
            nextToken = result.getNextToken();
        } while (nextToken != null);
    }
}
```

ListSigningProfiles

The following example shows how to use the ListSigningProfiles operation.

```
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.signer.AWSsigner;
import com.amazonaws.services.signer.AWSsignerClient;
import com.amazonaws.services.signer.model.ListSigningProfilesRequest;
import com.amazonaws.services.signer.model.ListSigningProfilesResult;
import com.amazonaws.services.signer.model.SigningProfile;

public class ListSigningProfilesTest {

    public static void main(String[] s) {

        final String credentialsProfile = "default";

        // Create a client.
        final AWSsigner client = AWSsignerClient.builder()
                .withRegion("us-west-2")
                .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
                .build();

        ListSigningProfilesResult result;
        String nextToken = null;
        do {
            result = client.listSigningProfiles(new
 ListSigningProfilesRequest().withNextToken(null));

            for (SigningProfile profile : result.getProfiles()) {
                System.out.println("Profile Name : " + profile.getProfileName());
                System.out.println("Cert Arn : " +
 profile.getSigningMaterial().getCertificateArn());
                System.out.println("Profile Status : " + profile.getStatus());
                System.out.println("Platform Id : " + profile.getPlatformId());
            }

            nextToken = result.getNextToken();
        } while (nextToken != null);
    }
}
```

PutSigningProfile

The following example shows how to use the PutSigningProfileProfile operation to create a new signing profile. Code signing profiles can then be used in the StartSigningJob operation.

```
package com.examples;

import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.services.signer.AWSSigner;
import com.amazonaws.services.signer.AWSSignerClient;
```

```
import com.amazonaws.services.signer.model.PutSigningProfileRequest;
import com.amazonaws.services.signer.model.SigningMaterial;

public class PutSigningProfile {

    public static void main(String[] s) {

        final String credentialsProfile = "default";
        final String codeSigningProfileName = "MyProfile";
        final String codeSigningCertificateArn = "arn:aws:acm:us-
west-2:123456789:certificate/6e7e9e0c-0d2a-
        4835-b2cc-2326a16c86f0";

        // Create a client.
        final AWSSigner client = AWSSignerClient.builder()
            .withRegion("us-west-2")
            .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
            .build();

        // creating a code signing profile.
        client.putSigningProfile(new PutSigningProfileRequest()
            .withProfileName(codeSigningProfileName)
            .withSigningMaterial(new SigningMaterial()
                .withCertificateArn(codeSigningCertificateArn))
        .withPlatform(platoformArn));
    }
}
```

StartSigningJob

The following example shows how to use the StartSigningJob operation. You must call `StartSigningJob` before you call any other code signing API operation. `StartSigningJob` returns a `jobId` value that you can use when calling DescribeSigningJob operation.

In order to use the `StartSigningJob` operation, make sure that the designated user's IAM policy includes Amazon S3 permissions. See Define an IAM Policy" (p. 5) for an example.

```
package com.amazonaws.samples;

import com.amazonaws.services.signer.AWSSigner;
import com.amazonaws.services.signer.AWSSignerClient;
import com.amazonaws.services.signer.model.SigningMaterial;
import com.amazonaws.services.signer.model.Source;
import com.amazonaws.services.signer.model.S3Source;
import com.amazonaws.services.signer.model.Destination;
import com.amazonaws.services.signer.model.S3Destination;
import com.amazonaws.services.signer.model.StartSigningJobRequest;
import com.amazonaws.services.signer.model.StartSigningJobResult;

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

import com.amazonaws.services.signer.model.ValidationException;
import com.amazonaws.services.signer.model.ResourceNotFoundException;
import com.amazonaws.services.signer.model.AccessDeniedException;
import com.amazonaws.services.signer.model.ThrottlingException;
import com.amazonaws.services.signer.model.InternalServiceErrorException;
import com.amazonaws.AmazonClientException;
```

```
/**
* This sample demonstrates how to use the StartSigningJob operation in the
* code signing service.
*
* Input Parameters:
*
* source             - Structure that contains the following:
*                          - Name of the Amazon S3 bucket to which you copied your
*                            code image
*                          - Name of the file that contains your code image
*                          - Amazon S3 version number of your file
* destination        - Structure that contains the following:
*                          - Name of the Amazon S3 bucket that code signing can use for
*                            your signed code
*                          - Optional Amazon S3 bucket prefix
* signingmaterial    - Amazon Resource Name (ARN) of the certificate to use for signing
* signingparameters  - Map of custom key-value pairs that you want to use for signing
* platform           - The microcontroller platform
*
*/

public class StartSigningJob {

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

      // Define variables.
      String certArn =
          "arn:aws:acm:region:account:certificate/12345678-1234-1234-1234-123456789012";
      String bucketSrc = "Source-Bucket-Name";
      String key = "Code-Image-File";
      String bucketDest = "Destination-Bucket-Name";
      SigningMaterial material = new SigningMaterial().withCertificateArn(certArn);
      S3Source s3src = new S3Source()
          .withBucketName(bucketSrc)
          .withKey(key)
          .withVersion("W.OIrIFmjIFeuNXOaBJzPee66.wRg4GR");
      Source src = new Source().withS3(s3src);
      S3Destination s3Dest = new S3Destination().withBucketName(bucketDest);
      Destination dest = new Destination().withS3(s3Dest);
      String platform = "Platform";

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

      // Specify the endpoint and region.
      EndpointConfiguration endpoint =
            new EndpointConfiguration("https://endpoint","region");


      // Create a client.
      AWSSigner client = AWSSignerClient.builder()
            .withEndpointConfiguration(endpoint)
            .withCredentials(new AWSStaticCredentialsProvider(credentials))
            .build();


      // Create a request object.
      StartSigningJobRequest req = new StartSigningJobRequest()
            .withSource(src)
```

```
            .withDestination(dest)
            .withSigningMaterial(material)
            .withPlatform(platform);

      // Create a result object.
      StartSigningJobResult result = null;
      try {
         result = client.startSigningJob(req);
      }
      catch (ValidationException ex)
      {
         throw ex;
      }
      catch (ResourceNotFoundException ex)
      {
         throw ex;
      }
      catch (AccessDeniedException ex)
      {
         throw ex;
      }
      catch (ThrottlingException ex)
      {
         throw ex;
      }
      catch (InternalServiceErrorException ex)
      {
         throw ex;
      }

      // Display the job ID.
      System.out.println("Job ID: " + result.getJobId());

   }
}
```

Authentication and Access Control

Access to code signing for AWS IoT requires credentials that AWS can use to authenticate your requests. The credentials must have permissions to access AWS resources. The following sections provide details on how you can use AWS Identity and Access Management (IAM) to help secure your resources by controlling who can access them.

Authentication

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

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

Access Control

You can have valid credentials to authenticate your requests. But unless you have permissions, you cannot create or access code signing resources. For example, you must have permission to start a signing job, describe a signing job, and list all signing jobs. The following topics discuss how to manage permissions. We recommend that you read the overview first.

Overview of Managing Access to Your ACM Resources

An AWS account owner or an authorized administrator can attach permissions policies to IAM identities (users, groups, and roles) that were created in the account. When managing permissions, an account owner or administrator decides who gets the permissions and what specific actions are allowed.

A *permissions policy* describes who has access to what. Administrators can use IAM to create policies that apply permissions to IAM users, groups, and roles. The following types of *identity-based policies* can grant permission for code signing actions:

- **Customer-managed policies** – Policies that an administrator creates and manages in an AWS account and which can be attached to multiple users, groups, and roles.
- **Inline policies** – Policies that an administrator creates and manages and which can be embedded directly into a single user, group, or role.

For complete IAM documentation, see the IAM User Guide. For information about IAM policy syntax and descriptions, see AWS IAM Policy Reference.

Customer Managed Policies

Customer managed policies are standalone identity-based policies that an administrator creates and can attach to multiple users, groups, or roles in your AWS account. Administrators can manage and create policies using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the IAM API. For more information about using the console to administer customer managed policies, see the following topics in the IAM User Guide.

- Attaching Managed Policies
- Detaching Managed Policies
- Creating Customer Managed Policies
- Editing Customer Managed Policies
- Setting the Default Version of Customer Managed Policies
- Deleting Versions of Customer Managed Policies
- Deleting Customer Managed Policies

For more information about using the API, see Working with Managed Policies Using the AWS CLI or the IAM API .

Inline Policies

Inline policies are policies that an administrator creates and manages and embeds directly into a single principal (user, group, or role). The following policy examples show how to grant permissions to perform ACM actions. For more information about attaching 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

Start a Signing Job

The following policy allows a principal to start a code signing job. For more information, see StartSigningJob.

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

Describe a Signing Job.

The following policy allows a principal to describe a code signing job. For more information, see DescribeSigningJob.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "signer:DescribeSigningJob",
    "Resource": "*"
```

```
  }
}
```

List Signing Jobs

The following policy allows a principal to list information about all code signing jobs. For more information, see ListSigningJobs.

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

Full Access

The following policy allows a principal to perform any code signing action.

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

Code Signing for AWS IoT API Permissions: Actions Reference

Administrators who set up access control and write permissions policies that they attach to an IAM identity (identity-based policies) can use the following table as a reference. The first column in the table lists each AWS Certificate Manager (ACM) API operation. You specify actions in a policy's `Action` element. You can use the IAM policy elements in your ACM policies to express conditions. For a complete list, see Available Keys in the *IAM User Guide*.

> **Note**
> To specify an action, use the `signer` prefix followed by the API operation name (for example, `signer:StartSigningJob`).

ACM API Operations and Permissions

API Operation	Required Permissions (API Actions)
CancelSigningProfile	`signer:CancelSigningProfile`
DescribeSigningJob	`signer:DescribeSigningJob`
GetSigningPlatform	`signer:GetSigningPlatform`
GetSigningProfile	`signer:GetSigningProfile`

API Operation	Required Permissions (API Actions)
ListSigningJob	`signer:ListSigningJob`
ListSigningPlatforms	`signer:ListSigningPlatforms`
ListSigningProfiles	`signer:ListSigningProfiles`
PutSigningProfile	`signer:PutSigningProfile`
StartSigningJob	`signer:StartSigningJob`

Document History for Developer Guide

Latest documentation update: November 19, 2018

The following table describes the documentation release history of code signing for AWS IoT.

update-history-change	update-history-description	update-history-date
Added new content (p. 23)	Integrated code signing for AWS IoT with AWS IoT Device Management.	November 8, 2018
Launched code signing for AWS IoT (p. 23)	This release introduces code signing for AWS IoT.	December 20, 2017

AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the *AWS General Reference*.