aws

Developer Guide

AWS Signer

AWS Signer: Developer Guide

Table of Contents

What is AWS Signer?

AWS Signer is a fully managed code-signing service to ensure the trust and integrity of your code. Organizations validate code against a digital signature to confirm that the code is unaltered and from a trusted publisher. With AWS Signer, your security administrators have a single place to define your signing environment, including what AWS Identity and Access Management (IAM) role can sign code and in what Regions. AWS Signer manages the code-signing certificate's public and private keys, and enables central management of the code-signing lifecycle. Integration with AWS CloudTrail helps you track who is generating code signatures and to meet your compliance requirements.

For information about AWS services that Signer supports, see the section called “Interoperation with other AWS services”.

Topics

For information about the AWS Signer API, see the AWS Signer API Reference.

Interoperation with other AWS services

AWS Signer is integrated or used with the following AWS services.

AWS Lambda

With AWS Signer, you can digitally sign packages intended for Lambda deployment in your organization, ensuring that only trusted code runs in your Lambda functions. AWS Signer defines a trusted publisher in a signing profile. Authorized developers use the profile to generate certified code packages. AWS Lambda verifies signatures and package integrity when code is deployed.

To sign your code packages before deploying them to AWS Lambda, you can use the AWS Signer console, the Signer CLI the AWS Serverless Application Model (AWS SAM) CLI, or one of the AWS SDKs.

Amazon FreeRTOS and **AWS IoT Device Management**

You can sign code that you create for IoT devices supported by Amazon FreeRTOS and AWS IoT device management. Code signing for AWS IoT is integrated with AWS Certificate Manager (ACM). To sign code, you import a third-party code-signing certificate into ACM that is used to sign updates in FreeRTOS and AWS IoT Device Management.

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 AWS Signer for the initial manufacture of these devices and subsequent over-the-air updates. You can use AWS Signer through the Amazon FreeRTOS console to sign your code images before you deploy them to a microcontroller.

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 AWS Signer to authenticate each device in your IoT environment. You can use AWS Signer through the AWS IoT Device Management console to sign your code images before you deploy them to a microcontroller.

You can sign your firmware images before deploying them to a microcontroller using the FreeRTOS console. To sign your code images before deploying them in an over-the-air (OTA) update, you can use the AWS IoT Device Management console, the AWS CLI, or one of the AWS SDKs.

Amazon Elastic Container Registry (Amazon ECR)

With AWS Signer and the Notation CLI from the Notary Project, you can sign container images stored in a container registry such as Amazon Elastic Container Registry (Amazon ECR). The signatures are stored in the registry alongside the images, where they are available for verifying image authenticity and integrity.

For more information, see the Amazon Elastic Container Registry User Guide.

Amazon Elastic Kubernetes Service (Amazon EKS)

Amazon EKS and self-managed Kubernetes customers on Amazon EC2 can verify the ownership and integrity of signed images at the time of deployment. For more information, see the *Amazon EKS User Guide*.

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 made to AWS Signer. 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.

Supported Regions

Visit AWS Signer endpoints and quotas to see an up-to-date list of supported Regions.

Quotas for Signer

AWS Signer sets per-second quotas on the allowed rate at which you can call API actions. Each API's quota is specific to an AWS account and Region. If the number of requests for an API exceeds its quota, AWS Signer rejects an otherwise valid request, returning a ThrottlingException error. AWS Signer does not offer a minimum request rate for APIs.

To view your quotas and see which ones can be adjusted, see the AWS Signer quotas table in the *AWS General Reference Guide*.

You can also view and adjust quotas using the Service Quotas console.

To see an up-to-date list of your AWS Signer quotas

1. Log in to your AWS account.
2. Open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.
3. In the **AWS services** list, enter **signer** into the search box, and choose **AWS Signer**. Each quota in the **Service quotas** list shows your currently applied quota value, the default quota value, and whether the quota is adjustable. Choose the name of a quota for more information about it.

To request a quota increase

1. In the **Service quotas** list, choose the radio button for an adjustable quota.
2. Choose the **Request quota increase** button.
3. Complete and submit the **Request quota increase** form.

Pricing for Signer

There is no additional charge to use AWS Signer with AWS IoT Device Management, AWS Lambda, Amazon ECR, Amazon EKS, or third-party container services. Refer to the pricing for the related services for other charges that you may incur. For example, if you use Signer with Lambda, you pay for the storage of signed and unsigned objects (such as your Lambda zip-file archives) in Amazon S3.

Get started with AWS Signer

Before you can begin signing code and binaries with AWS Signer, you need to set up an AWS account, create administrative and root users, apply security policies using AWS Identity and Access Management (IAM), and create a signing profile that contains the configuration for your signing tasks.

Topics

Set up to use Signer

Access to AWS Signer 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.

Sign up for an AWS account

If you do not have an AWS account, complete the following steps to create one.

To sign up for an AWS account

1. Open https://portal.aws.amazon.com/billing/signup.
2. Follow the online instructions.

   Part of the sign-up procedure involves receiving a phone call or text message and entering a verification code on the phone keypad.

   When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, assign administrative access to a user, and use only the root user to perform tasks that require root user access.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to https://aws.amazon.com/ and choosing **My Account**.

Create a user with administrative access

After you sign up for an AWS account, secure your AWS account root user, enable AWS IAM Identity Center, and create an administrative user so that you don't use the root user for everyday tasks.

Secure your AWS account root user

1. Sign in to the AWS Management Console as the account owner by choosing **Root user** and entering your AWS account email address. On the next page, enter your password.

   For help signing in by using root user, see Signing in as the root user in the *AWS Sign-In User Guide*.

2. Turn on multi-factor authentication (MFA) for your root user.

   For instructions, see Enable a virtual MFA device for your AWS account root user (console) in the *IAM User Guide*.

Create a user with administrative access

1. Enable IAM Identity Center.

   For instructions, see Enabling AWS IAM Identity Center in the *AWS IAM Identity Center User Guide*.

2. In IAM Identity Center, grant administrative access to a user.

   For a tutorial about using the IAM Identity Center directory as your identity source, see Configure user access with the default IAM Identity Center directory in the *AWS IAM Identity Center User Guide*.

Sign in as the user with administrative access

- To sign in with your IAM Identity Center user, use the sign-in URL that was sent to your email address when you created the IAM Identity Center user.

For help signing in using an IAM Identity Center user, see Signing in to the AWS access portal in the *AWS Sign-In User Guide*.

Assign access to additional users

1. In IAM Identity Center, create a permission set that follows the best practice of applying least-privilege permissions.

   For instructions, see Create a permission set in the *AWS IAM Identity Center User Guide*.
2. Assign users to a group, and then assign single sign-on access to the group.

   For instructions, see Add groups in the *AWS IAM Identity Center User Guide*.

Create a Signer signing profile

Before you can perform signing jobs, you must create a *signing profile*. A signing profile is unique AWS Signer resource that you can use to perform signing jobs. Signing profiles enable you to sign and verify code artifacts, such as container images and AWS Lambda deployment bundles. Each signing profile designates the signing platform to sign for, a platform ID, and other platform-specific information.

You can create, list, and cancel signing profiles using the Signer console, AWS CLI, or API. Signer manages the code signing certificate and keys associated for only AWS Lambda and Container images workflows. For Internet of Things (IoT) workflows, you can import your own code signing certificate into AWS Certificate Manager.

Console

This section describes the procedures and options for creating a signing profile from the AWS console.

To create a signing profile

1. Log into the AWS Signer console.
2. Choose **Create signing profile**.
3. On the **Create signing profile** page, provide a unique **Profile name** for your signing profile. Valid characters include uppercase A-Z, lowercase a-z, numbers 0-9, and underscore (_).

4. For **Signing platform**, choose one of the listed platforms.

API name	Display name
`AWSLambda-SHA384-ECDSA`	AWS Lambda
`Notation-OCI-SHA384-ECDSA`	Notation for container registries

5. Specify the **Signature validity period** in months, days, or years. The default value is 135 months (11 years and 6 months).
6. In the **Tags - optional** section, you can create a **Tag key** and a **Tag value**, then save it with the **Add tag** button. When you assign tags to your signing profile, you can use tag-based resource policies to manage access to the profile.

   You can assign up to 50 tags to a profile.
7. Choose **Create profile**.

CLI

This section describes the procedures and options for creating and managing signing profiles using the AWS CLI. A signing profile is a template that defines the following settings for associated signing jobs:

- The *signing platform* that designates the file type to be signed. The following platforms are available in the AWS CLI.

API name	Display name
`AWSIoTDeviceManagement-SHA256-ECDSA`	AWS IoT Device Management SHA256-ECDSA
`AmazonFreeRTOS-Default`	Amazon FreeRTOS SHA256-ECDSA
`AmazonFreeRTOS-TI-CC3220SF`	Amazon FreeRTOS SHA1-RSA CC3220SF-Format

API name	Display name
AWSLambda-SHA384-ECDSA	AWS Lambda
Notation-OCI-SHA384-ECDSA	Notation for container registries

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

- The signature format.
- The signature algorithms.
- The validity period of signatures. By default, signature validity is set to 135 months (11 years and 3 months), which is the maximum validity supported. The signature validity period is only applicable for AWSLambda-SHA384-ECDSA and Notation-OCI-SHA384-ECDSA signing platforms.

After you create the signing profile, you can delegate control of it using AWS Identity and Access Management (IAM). For more information about managing user permissions in AWS Signer, see Identity and Access Management for AWS Signer.

Signing profiles can be created, inspected, listed, and canceled as shown in the following examples.

- **put-signing-profile**

  This command creates and saves an AWS Signer signing profile.

  Signatures generated using this platform will expire after the time specified by `--signature-validity-period`. This value may be specified using DAYS, MONTHS, or YEARS. If no validity period is specified, the default value is 135 months.

  In this example, the specified signing platform is AWSLambda-SHA384-ECDSA.

```
$ aws signer put-signing-profile \
     --profile-name my_lambda_signing_profile \
     --platform-id AWSLambda-SHA384-ECDSA \
```

```
    --signature-validity-period value=10, type='MONTHS'
```

- **get-signing-profile**

  This command retrieves a signing profile for inspection.

```
$ aws signer get-signing-profile --profile-name my_lambda_signing_profile
```

- **list-signing-profiles**

  This command lists the signing profiles that you own or control.

```
$ aws signer list-signing-profiles
```

- **cancel-signing-profile**

  This command deletes a signing profile.

```
$ aws signer cancel-signing-profile \
    --profile-name my_lambda_signing_profile \
    --profile-version profile_version \
    --reason "e2e notation testing" \
    --effective-time 1111111111
```

API

Signing profiles can be created, inspected, listed, and deleted using the following Signer API actions.

- PutSigningProfile
- CancelSigningProfile

- GetSigningProfile
- ListSigningProfiles

Set up cross-account signing for Signer

> **Note**
>
> Cross-account signing is only available for AWS Lambda and container registries signing platforms, which are referred to as platformId in the AWS CLI and API.

Cross-account signing enables accounts other than the signing profile's owning account to sign code artifacts, and optionally revoke signatures generated by the shared signing profile. For example, an organization's security administrator can create a signing profile, and then grant a group of developers the permission to sign code artifacts using the shared signing profile. The developers could also revoke the signatures generated by the signing profile. This enables accounts other than the owning account to use signing profiles in an organization.

The following procedure illustrates how a security administrator can enable cross-account signing using the AWS CLI. To begin, you'll create a signing profile. Then, you'll grant developer accounts access to the profile for code signing.

To set up cross-account signing using the CLI

The following example uses the AWS Lambda platform, but if you want to use container registry platform, you could instead use "`Notation-OCI-SHA384-ECDSA`" platform as the value for the `platform-id`. The example commands in this procedure are pre-populated with values for things like profile names, IDs, and descriptions. Change those as appropriate for your application.

1. The following command creates a signing profile for the AWS Lambda platform type, with a profile name of `profile_for_application_ABC`.

   ```
   aws signer put-signing-profile --platform-id "AWSLambda-SHA384-ECDSA" --profile-
   name profile_for_application_ABC
   ```

   Signer will respond with a signing profile version Amazon Resource Name (ARN) such as:

   ```
   arn:aws:signer:region:111122223333:/signing-
   profiles/profile_for_application_ABC/resource-identifierE1WG1ZNPRXT0D4
   ```

2. Now that you've created a signing profile, you can now grant the developers' accounts access to use the profile for signing. You do that by using the `add-profile-permission`

command. The following example grants permission only for the `signer:StartSigningJob` action that's used with the AWS Lambda workflow. If it were a container image signing platform, you'd set the `--action` value to `signer:SignPayload`. You might want to grant permissions for other actions, such as `signer:GetSigningProfile` or `signer:RevokeSignature`, by making additional calls to `add-profile-permission`.

The following command grants permission to another account. Replace *555555555555* with the principal wish to grant cross-account access. The principal can be an IAM role or another AWS account ID.

```
aws signer add-profile-permission \
 --profile-name profile_for_application_ABC \
 --action signer:StartSigningJob \
 --principal 555555555555 \
 --statement-id OptionalStatementId
```

Note

The signatures generated when using cross-account signing are embedded with the signing profile ARN of the owner account. The owner account is the account that created the signing profile. For verifying signed Lambda .zip archives, you must configure your Lambda code signing configuration to use the signing profile version ARN of the owner account. For verifying signed container images, you must configure the Notation trust policy to use the signing profile ARN of the owner account.

Code signing workflows in Signer

The signing procedures for AWS services by Signer are service-specific workflows. The following topics describe workflows for signing IoT binaries, Lambda zip files, and container images.

> **Important**
>
> Steps common to all workflows have been covered in Get started with AWS Signer. Workflow steps in the following topics are not interchangeable.

Topics

Sign Internet of Things (IoT) objects

This section describes procedures for signing binary objects intended for deployment on Internet of Things (IoT) devices. Before you begin, make sure you have completed the prerequisites listed in Get started with AWS Signer.

Topics

Obtain and import a code-signing certificate

Before you can use AWS Signer with AWS IoT Device Management or Amazon FreeRTOS, 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
```

Important

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. Encouraging your users to trust arbitrary certificates with no reputational backing is a poor security practice.

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.

Create and populate an Amazon S3 source bucket for your unsigned object files

This topic discusses how to prepare an Amazon S3 bucket and add your unsigned object files to it.

To create a bucket, sign into the AWS Management Console at https://console.aws.amazon.com/console/home and follow the procedure in Create your first S3 bucket.

While you are configuring the bucket, note the following requirements:

- Accept the default security option **Block *all* public access**.
- Set **Bucket Versioning** to **Enable**.

After you create the bucket, you can add objects to it as described in the Upload an object to your bucket topic.

Create an Amazon S3 destination bucket for your signed object files

This topic discusses how to prepare an Amazon S3 destination bucket where AWS Signer can deposit your signed object files.

To create a bucket, sign into the AWS Management Console at https://console.aws.amazon.com/console/home and follow the procedure in Create your first S3 bucket.

While you are configuring the bucket, note the following requirement.

- Accept the default security option **Block *all* public access**.

Create a signing job for IoT in AWS Signer

To start a signing job, you need to specify the following:

- The source S3 bucket of the IoT binary to be signed
- A signing profile
- The destination S3 bucket for the signed file

A signing job has a status of `InProgress` while it is being processed, and after completion, the status changes to `Succeeded`. If Signer is unable to generate a signature, the signing job updates to `Failed`. Signing fails for a zip file if the file is empty, already has a signature, or is malformed.

To perform a signing job (CLI)

Use the following CLI commands to run and manage signing jobs.

- **start-signing-job**

To get the status of a particular signing job, use the following action or command:

- **describe-signing-job**

For a list of all available signing jobs, including those in the Failed state, use the following action or command:

- **list-signing-jobs**

To perform a signing job (API)

Use the following API actions to run and manage signing jobs.

- `StartSigningJob`
- `DescribeSigningJob`
- `ListSigningJobs`

For more information about configurations and parameters related to signing jobs, see `SigningJob` in the *AWS Signer API Reference.*

Sign AWS Lambda code

This section describes procedures for signing code intended for deployment on AWS Lambda. Before you begin, make sure you have completed the prerequisites listed in Get started with AWS Signer.

Topics

- Create and populate an Amazon S3 source bucket for your unsigned object files
- Create an Amazon S3 destination bucket for your signed object files
- Create a signing job for Lambda in AWS Signer

Create and populate an Amazon S3 source bucket for your unsigned object files

This topic discusses how to prepare an Amazon S3 bucket and add your unsigned object files it.

To create a bucket, sign into the AWS Management Console at https://console.aws.amazon.com/console/home and follow the procedure in Create your first S3 bucket.

While you are configuring the bucket, note the following requirements:

- Accept the default security option **Block *all* public access**.
- Set **Bucket Versioning** to **Enable**.

After you create the bucket, you can add objects to it as described in Upload an object to your bucket topic.

Create an Amazon S3 destination bucket for your signed object files

This topic discusses how to prepare an Amazon S3 destination bucket where AWS Signer can deposit your signed object files.

To create a bucket, sign into the AWS Management Console at https://console.aws.amazon.com/console/home and follow the procedure in Create your first S3 bucket.

While you are configuring the bucket, note the following requirements:

- Accept the default security option **Block *all* public access**.

Create a signing job for Lambda in AWS Signer

To start a signing job, you need to specify the following:

- The source S3 bucket of the IoT code or Lambda zip file to be signed
- A signing profile
- The destination S3 bucket for the signed file

A signing job has a status of `InProgress` while it is being processed, and after completion, the status changes to `Succeeded`. If Signer is unable to generate a signature, the signing job updates to `Failed`. Signing fails for a zip file if the file is empty, already has a signature, or is malformed.

To perform a signing job (console)

1. Log into the AWS Signer console.
2. Choose **Start signing jobs**.
3. From the list of profiles, choose a signing profile to perform code signing for your Lambda application.

4. Do either of the following:

   - For **Code asset source location**, enter the URL for the Amazon S3 bucket that contains your code.
   - Choose **Browse**, and locate the S3 bucket that contains your code.

   > **Note**
   >
   > Be sure your file is in zip format. The AWS Signer console does not accept other file formats.

5. Do one of the following:

   - In the **Signature Destination path with Prefix**, enter the URL for the S3 bucket where you store your signed code.
   - Choose **Browse** and locate the S3 bucket that stores your signed code.

6. Choose **Start**.

   AWS Signer updates the **Manage signing jobs** page with your new profile, and displays the following information:

   - **Job ID** – The generated ID number
   - **Profile name** – The name of the profile
   - **Signing status** – The signing status of the job
   - **Revocation status** – The status of the revocation if any

7. If you receive a **Failed** under **Signing status**, return to the list of the signing jobs, and choose **Failed** to see the details of the signing job.

The **Signing job details** page lists the following information:

- **Job ID** – The identifier of the signing job
- **Signing profile used** – The signing profile used for the job
- **Version of signing profiles used** – The version of the signing profile used for the job
- **Requested by** – Identity of the requester of the job
- **Signing platform** – The signing platform used for the job (Lambda only)

- **Signing status** – The status of the job as either **Successful** or **Failed**
- **Status reason** – Explanation for the failure if the signing job failed
- **Started at** – The time and date that the signing job started
- **Completed at** – The time and date that the job ended

The **Code assets details** displays additional information:

- **Code asset source bucket** – The S3 source bucket of the code file used
- **Code asset source key** – The name of the code file used for signing code
- **Code asset source version** – The version of the code file

To perform a signing job (AWS CLI)

Use the following command to start a signing job:

- **start-signing-job**

To get the status of a particular signing job, use the following command:

- **describe-signing-job**

For a list of all available signing jobs, including those in the Failed state, use the following command:

- **list-signing-jobs**

To perform a signing job (API)

Following API actions can be used to run and track signing jobs.

- `StartSigningJob`
- `DescribeSigningJob`
- `ListSigningJobs`

For more information about configurations and parameters related to signing jobs, see `SigningJob` in the *AWS Signer API Reference.*

Sign container images in Signer

This section describes procedures for signing container images stored in an Open Container Initiative (OCI) compliant container registry. Before you begin, make sure you have completed the prerequisites listed in Get started with AWS Signer.

> **Note**
>
> If you're coming here from the Amazon ECR image signing documentation, be aware that you must fulfill all of the requirements related to Amazon ECR before beginning these AWS Signer procedures. For more information, see Signing an image in the *Amazon Elastic Container Registry User Guide*.

Topics

Prerequisites for signing container images

Before you begin signing, you need to set up an environment that bridges AWS Signer with Amazon ECR. Complete the following steps.

To prepare your signing environment

1. **Prepare the AWS CLI**

   Install and configure the latest version of the AWS CLI. For more information, see Installing or updating the latest version of the AWS CLI in the *AWS Command Line Interface User Guide*.

2. **Prepare Amazon ECR**

Have an existing container image stored in an Amazon ECR private repository to sign. For more information, see Pushing an image in the *Amazon Elastic Container Registry User Guide*.

3. **Download the container-signing tools**

   Two software packages need to be installed in your local environment for you to sign images:

   - The open source supply chain security program Notation, developed by the Notary Project
   - The AWS Signer plugin for Notation. You can either use our plugin binary, or our open source library.

   Plugin binary

   The AWS Signer installer installs both the Notation client and the AWS Signer plugin for Notation. Separate binaries are available to install only the AWS Signer plugin.

   The installer includes the following.

   - Notation binary and third party license.
   - AWS Signer plugin binary and third party license.
   - Notation license.
   - Trust store set up with AWS Signer's Notation signing root certificate.
   - GovCloud trust store and root certificate, for use in the AWS GovCloud (US) Region.
   - A configurable trust policy. For information about configuring the trust policy, see Locally verify an image after signing.

   The following table provides the installer and related files for each supported operating system and architecture. You can download our latest CHANGELOG to see the versions of the Notation CLI and plugin included in each installer release.

Notation binary and AWS Signer Plugin installer files

Platform	Architecture	Installer for Notation and AWS Signer plugin	AWS Signer plugin binary only	Signature file
RPM-based Linux (e.g., Amazon Linux)	x86_64	aws-signer-notation-cli_amd64.rpm	notation-aws-signer-plugin.zip	aws-signer-notation-cli_amd64.rpm.sig (installer) notation-aws-signer-plugin.sig (plugin)
	arm64	aws-signer-notation-cli_arm64.rpm	notation-aws-signer-plugin.zip	aws-signer-notation-cli_arm64.rpm.sig (installer) notation-aws-signer-plugin.sig (plugin)
Debian-based Linux	x86_64	aws-signer-notation-cli_amd64.deb	notation-aws-signer-plugin.zip	aws-signer-notation-cli_amd64.deb.sig (installer) notation-aws-signer-plugin.sig (plugin)

Platform	Architecture	Installer for Notation and AWS Signer plugin	AWS Signer plugin binary only	Signature file
	arm64	aws-signer-notation-cli_arm64.deb	notation-aws-signer-plugin.zip	aws-signer-notation-cli_arm64.deb.sig notation-aws-signer-plugin.sig (plugin)
macOS	x86_64	aws-signer-notation-cli_amd64.pkg	notation-aws-signer-plugin.zip	Included in the files.
	arm64	aws-signer-notation-cli_arm64.pkg	notation-aws-signer-plugin.zip	Included in the files.
Microsoft Windows	x86_64	aws-signer-notation-cli.msi	notation-aws-signer-plugin.zip	Validate in Explorer

Open source library

The open source Signer plugin is available as a library for use with your toolchain to generate and verify container artifacts signatures. Access the source code and instructions to build the Signer plugin in the Signer Notation plugin Github repository.

4. **(Optional) Verify signed packages.**

For instructions to complete this step, select the tab for your platform.

Linux

1. Download the public key.

```
$ wget  https://d2hvyiie56hcat.cloudfront.net/linux/public.key
```

2. Import the public key into your keyring. If you're using the unzip the AWS Signer plugin, first unzip downloaded file and then run command against the binary file within the zip file.

```
$  gpg --import public.key
gpg: key A3B52DA65461CF90: public key "AWS Signer Notation" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

   Make a note of the key value, as you need it in the next step. In the preceding example, the key value is A3B52DA65461CF90.

3. Verify the fingerprint by running the following command, replacing **key-value** with the value from the preceding step:

```
$  gpg --fingerprint key-value
pub   rsa3072 2023-04-24 [SC]
    E84A F8A2 A9B5 2F1F 4435  AE71 A3B5 2DA6 5461 CF90
uid           [ unknown] AWS Signer Notation
```

   The fingerprint string should be E84A F8A2 A9B5 2F1F 4435 AE71 A3B5 2DA6 5461 CF90.

   If the fingerprint string doesn't match, don't run the installer. Contact Amazon Web Services.

   After you have verified the fingerprint, you can use it to verify the signature of the AWS Signer Notation package.

4. Download the package signature file using **wget**. To determine the correct signature file, see the preceding table.

```
$ wget signature-file-link
```

5. To verify the signature, run gpg --verify:

```
$  gpg --verify sig-filename downloaded-filename
gpg: Signature made Mon May 22 16:16:34 2023 PDT
```

```
gpg:                using RSA key A3B52DA65461CF90
gpg: Good signature from "AWS Signer Notation" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: E84A F8A2 A9B5 2F1F 4435  AE71 A3B5 2DA6 5461 CF90
```

If the output includes the phrase *BAD signature*, check whether you performed the procedure correctly. If you continue to get this response, contact Amazon Web Services and avoid using the downloaded file.

Note the warning about *trust*. A key is trusted only if you or someone who you trust has signed it. This doesn't mean that the signature is invalid, only that you have not verified the public key.

Windows

To verify the signature of the MSI installer or the plugin EXE file, using Windows PowerShell run the following command:

```
C:\> Get-AuthenticodeSignature  filename
```

You can also verify the signature by right-clicking on the file in an Explorer window, choosing **Properties**, and then choosing **Digital Signatures**.

You should see a result similar to the following:

```
SignerCertificate                         Status            Path
-----------------                         ------            ----
[40-character hexamecimal number]         Valid             downloaded-file
```

MacOS

1. To verify the signature of the PKG installer, run the following command. This example uses the amd64 package, but the signature of the arm64 package can be verified similarly.

   ```
   $ pkgutil --check-signature aws-signer-notation-cli_amd64.pkg
   ```

   You should see a result similar to the following:

```
Package "aws-signer-notation-cli_amd64.pkg":
    Status: signed by a developer certificate issued by Apple for distribution
    Notarization: trusted by the Apple notary service
    Signed with a trusted timestamp on: 2023-05-19 15:17:15 +0000
    Certificate Chain:
     1. Developer ID Installer: AMZN Mobile LLC (94KV3E626L)
         Expires: 2027-06-28 22:57:06 +0000
         SHA256 Fingerprint:
              49 68 39 4A BA 83 3B F0 CC 5E 98 3B E7 C1 72 AC 85 97 65 18 B9
 4C
              BA 34 62 BF E9 23 76 98 C5 DA

 ------------------------------------------------------------------------
    2. Developer ID Certification Authority
         Expires: 2031-09-17 00:00:00 +0000
         SHA256 Fingerprint:
              F1 6C D3 C5 4C 7F 83 CE A4 BF 1A 3E 6A 08 19 C8 AA A8 E4 A1 52
 8F
              D1 44 71 5F 35 06 43 D2 DF 3A

 ------------------------------------------------------------------------
    3. Apple Root CA
         Expires: 2035-02-09 21:40:36 +0000
         SHA256 Fingerprint:
              B0 B1 73 0E CB C7 FF 45 05 14 2C 49 F1 29 5E 6E DA 6B CA ED 7E
 2C
              68 C5 BE 91 B5 A1 10 01 F0 24
```

2. To verify the signature of the AWS Signer Notation plugin executable, run the following command.

```
$ codesign -dv --verbose=4 ./notation-com.amazonaws.signer.notation.plugin
```

You should see a result similar to the following.

```
Executable=/path/to/notation-com.amazonaws.signer.notation.plugin
Identifier=notation-com.amazonaws.signer.notation.plugin_darwin_arm64
Format=Mach-O thin (arm64)
CodeDirectory v=20500 size=74278 flags=0x10000(runtime) hashes=2314+2
 location=embedded
VersionPlatform=1
```

```
VersionMin=720896
VersionSDK=720896
Hash type=sha256 size=32
CandidateCDHash sha256=e4000dbdf4e6243be9d290b1520d95bf9027a5e4
CandidateCDHashFull
 sha256=e4000dbdf4e6243be9d290b1520d95bf9027a5e42b699a354fc39ac0f498477f
Hash choices=sha256
CMSDigest=e4000dbdf4e6243be9d290b1520d95bf9027a5e42b699a354fc39ac0f498477f
CMSDigestType=2
Executable Segment base=0
Executable Segment limit=3571712
Executable Segment flags=0x1
Page size=4096Launch Constraints:None
CDHash=e4000dbdf4e6243be9d290b1520d95bf9027a5e4
Signature size=9070
Authority=Developer ID Application: AMZN Mobile LLC (94KV3E626L)
Authority=Developer ID Certification Authority
Authority=Apple Root CATimestamp=May 19, 2023 at 7:51:07 AM
Info.plist=not bound
TeamIdentifier=94KV3E626L
Runtime Version=11.0.0
Sealed Resources=none
Internal requirements count=1 size=220
```

5. **Install the packages**

   For instructions to complete this step, select the tab for your platform.

   Linux (RPM)

   If you downloaded an RPM package on a Linux server, change to the directory containing the package and enter the following:

   ```
   $ sudo rpm -U  filename
   ```

   Linux (DEB)

   If you downloaded a DEB package on a Linux server, change to the directory containing the package and enter the following:

   ```
   $ sudo dpkg -i -E  filename
   ```

Windows

Install the package with the following command.

```
C:\> msiexec /i filename
```

This command also works from within PowerShell. For more information, see Microsoft Standard Installer command-line options in the Microsoft Windows documentation.

MacOS

If you downloaded a PKG package on a macOS server, change to the directory containing the package and enter the following:

```
$ sudo installer -pkg  filename -target /
```

6. **Verify the package installation**

After downloading and installing the package, to verify the installation was successful, do the following.

a. Verify that the Notation directory structure for your operating system was created.

b. Use the following command to display the Notation client version.

```
notation version
```

c. Use the following command to list the installed plugins for the Notation client and verify that you see the `com.amazonaws.signer.notation.plugin` plugin.

```
notation plugin ls
```

Required AWS Identity and Access Management permissions to sign and verify a container image

To sign and verify an image present in Amazon Elastic Container Registry, you need an AWS Identity and Access Management policy that allows Notation to interact with Amazon ECR and Signer.

The following is an example of a user managed policy that allows Notation to interact with Amazon ECR and Signer:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ManageRepositoryContents",
            "Effect": "Allow",
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:CompleteLayerUpload",
                "ecr:DescribeRepositories",
                "ecr:UploadLayerPart",
                "ecr:InitiateLayerUpload",
                "ecr:BatchCheckLayerAvailability",
                "ecr:PutImage"
            ],
            "Resource": "arn:aws:ecr:us-east-1:111122223333:repository/my-repo"
        },
        {
            "Sid": "GetAuthorizationToken",
            "Effect": "Allow",
            "Action": [
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SignAndRevocationCheck",
            "Effect": "Allow",
            "Action": [
                "signer:PutSigningProfile",
                "signer:SignPayload",
                "signer:GetRevocationStatus"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

Sign an image

The procedures on this page show you how create a signing profile, install a helper program, and sign a container image.

Step 1: Create a AWS Signer Notation signing profile

Create an AWS Signer Notation signing profile. If using the AWS Command Line Interface, API, AWS CloudFormation, or AWS SDKs set the platform ID to `Notation-OCI-SHA384-ECDSA`. In the console, for **signing platform** choose **Notation for container registries**. For more information on creating a signing profile, see Create a Signer signing profile.

Step 2: Install a helper program

Notation requires you to include a helper program in the client's host path in order to interact with the credential store. You can use either the Amazon Elastic Container Registry Docker credential helper or the Docker credential helper to manage your credentials. We recommend using the Amazon ECR Docker credential helper, as it includes a credentials store and handles authentication for you. The Amazon ECR Docker Credential Helper not only stores and uses credentials when signing and verifying images in Amazon ECR, but also eliminates the need to use the Notation CLI notation login command or write custom logic to refresh authentication tokens and provide transparent access to your Amazon ECR repositories.

Amazon ECR Docker credential helper

Download the Amazon Elastic Container Registry Docker credential helper. Configure `config.json` for use with Amazon ECR.

Docker credential helper

The following procedure explains how to install and configure the Docker credential helper.

To use the Docker credential helper

1. First set up a credentials store. Notation relies on a credentials store for secure storage and retrieval of credentials from Amazon ECR. Most operating systems come with a default credentials store, such as osxkeychain for macOS, or wincred for Windows. If you have

the Docker CLI installed on the same host where Notation is installed, Notation uses the credentials store configuration that you set up for the Docker CLI.

Alternatively, you can install a third-party credentials store such as pass. You can pass these credentials to Notation as environment variables. For more information about environment variables, see Configure environment variables to authenticate to an OCI-compliant registry in the Notary Project user guide.

2. Download the Docker credential helper. Set the `credsStore` option in `config.json` to the suffix of the program that you want to use.
3. Manually configure Notation client authentication. Because the Notation CLI doesn't support standard AWS authentication methods, you must manually configure Notation client authentication so that Amazon ECR knows who's requesting to sign (push signature) or verify (pull signature) an image. You can accomplish this with the Notation CLI `notation login` command, which authenticates to an Amazon ECR registry and provides an authorization token that's valid for 12 hours. Or, if you're using the AWS Command Line Interface, you can use the `get-login-password` command which retrieves the token, decodes it, and converts into a `notation login` command for you.

   The following command allows Notation to get credentials for authenticating with Amazon ECR:

   ```
   aws ecr get-login-password --region us-west-1 | notation login --username AWS --
   password-stdin 111122223333.dkr.ecr.us-west-1.amazonaws.com
   ```

Step 3: Sign the image using the Notation CLI

Use the Notation CLI to sign the image, specifying the image using the repository name and the SHA256 digest. This creates the signature and pushes it to the same Amazon ECR private repository that the image being signed is in.

> **Note**
>
> You can specify the AWS Region and credentials profile that the Notation plugin uses for interactions with AWS Signer either by setting values for the `AWS_DEFAULT_REGION` and `AWS_PROFILE` environment variables or by providing the arguments **--plugin-config aws-region=${*Region*}** and **--plugin-config aws-profile=${*profile-name*}**

In the following example, we're signing an image in the `curl` repository with SHA digest `sha256:ca78e5f730f9a789ef8c63bb55275ac12dfb9e8099e6EXAMPLE`.

```
notation
 sign 111122223333.dkr.ecr.Region.amazonaws.com/
curl@sha256:ca78e5f730f9a789ef8c63bb55275ac12dfb9e8099e6EXAMPLE --plugin
 "com.amazonaws.signer.notation.plugin" --id "arn:aws:signer:Region:111122223333:/
signing-profiles/ecrSigningProfileName"
```

Step 4: Verify image

After you have signed your container image, you can verify the signature locally or during an Amazon EKS deployment and further manage the signature with Amazon ECR.

- Locally verify an image after signing
- Verify an image during in Amazon EKS or Kubernetes clusters
- Manage your signature in your Amazon ECR repository in the *Amazon Elastic Container Registry User Guide*.

Locally verify an image after signing

After you sign a container image using AWS Signer and Notation, you or an authorized member of your team can verify the origin and integrity of the image by cryptographic means.

Complete the following steps to verify that an image is valid with Notation.

To verify an image

1. A trust store is required for verification. If you used the installer for the AWS Signer plugin and Notation, a trust store for both AWS commercial and AWS GovCloud (US) Regions was set up automatically and provisioned with a root certificate. For more information, see Prerequisites for signing container images.
2. Set up a trust policy that includes the trust store for your partition.

   The following example includes trust stores for both the AWS commercial and AWS GovCloud (US) Region. You can choose to include one or both in your trust policy depending on where you are verifying your signed images. To verify images signed in AWS commercial Regions, set `signingAuthority` to `aws-signer-ts`. To verify images signed in AWS GovCloud (US) Region, set `signingAuthority` to `aws-us-gov-signer-ts`.

> **Important**
>
> Signatures are isolated to AWS partitions. Calls to GetRevocationStatus with a cross-partition signature will return a validation exception error.

```
{
   "version":"1.0",
   "trustPolicies":[
      {
         "name":"aws-signer-tp",
         "registryScopes":[
            "*"
         ],
         "signatureVerification":{
            "level":"strict"
         },
         "trustStores":[
            "signingAuthority:aws-signer-ts",
            "signingAuthority:aws-us-gov-signer-ts"
         ],
         "trustedIdentities":[
            "arn:aws:signer:Region:111122223333:/signing-
profiles/ecr_signing_profile",
            "arn:aws:signer:Region:111122223333:/signing-
profiles/ecr_signing_profile2"
         ]
      }
   ]
}
```

3. Import the policy into Notation.

```
$ notation policy import mypolicy.json
```

Output:

```
Existing trust policy configuration found, do you
want to overwrite it? [y/N] y
```

```
Trust policy configuration imported successfully.
```

4. Verify the signature, specifying the signature using the repository name and the SHA digest.

> **Note**
>
> You can specify the AWS Region and credentials profile that the Notation plugin uses to interact with AWS Signer by assigning a value to the AWS_PROFILE environment variable, or by passing the **--plugin-config aws-profile=${*profile-name*}** argument to the Notation **verify** command.

```
$ notation verify 111122223333.dkr.ecr.region.amazonaws.com/curl@SHA256_digest
```

Output:

```
Successfully verified signature for 111122223333.dkr.ecr.us-
west-2.amazonaws.com/curl@SHA256_digest
```

Verify an image during in Amazon EKS or Kubernetes clusters

For AWS Signer customers wishing to verify signed container images at the time of deployment, there are various open-source solutions such as the following.

- Deis Labs Gatekeeper and Ratify – Use Gatekeeper as the admission controller and Ratify configured with an AWS Signer plug-in as a web hook for validating signatures.
- Kyverno – A Kubernetes policy engine configured with a AWS Signer plugin for validating signatures.

> **Note**
>
> Before verifying container-image signatures, customers must configure the Notation trust store and trust policy as required by their selected admission controller.

Revoke signatures generated by Signer

Revocation of a signature becomes necessary when the signing certificate is compromised in some way, for example, if the secret key is publicly disclosed. Revoking the signature of an AWS Lambda deployment package invalidates it, causing it to fail Lambda signature checks in all Regions of the same partition. Revoking the signature of a container image causes validation to fail if you attempt to deploy the image.

> **Note**
>
> Revocation is an irreversible action and is recommended only for critical scenarios. Revocation checks are valid for six months beyond the expiry of a signature. Expired signatures will fail on expiry checks instead.

You can revoke individual signatures either by using the `RevokeSignature` API or by selecting a signing job in the AWS Signer console.

You can revoke a signing profile by using the `RevokeSigningProfile` API or by selecting and revoking a signing profile in the AWS Signer console. Once revoked, a signing profile can no longer be used for creating new signing jobs.

Revocation for a signing profile requires an effective start time in the past. The start time cannot be in the future. The effective start time can be changed to an earlier date and time by repeating the revocation, but cannot be revised to a later date and time.

Monitor Signer

Monitoring is an important part of maintaining the reliability, availability, and performance of Signer and your other AWS solutions. AWS provides the following monitoring tools to watch Signer, report when something is wrong, and take automatic actions when appropriate:

- *AWS CloudTrail* captures API calls and related events made by or on behalf of your AWS account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called AWS, the source IP address from which the calls were made, and when the calls occurred. For more information, see the AWS CloudTrail User Guide.

Amazon EventBridge is a serverless event bus service that makes it easy to connect your applications with data from a variety of sources. EventBridge delivers a stream of real-time data from your own applications, Software-as-a-Service (SaaS) applications, and AWS services and routes that data to targets such as Lambda. This enables you to monitor events that happen in services, and build event-driven architectures. For more information, see the Amazon EventBridge User Guide.

Automation with CloudWatch Events

You can automate your use of AWS Signer by tracking and responding to system events that are managed by Amazon CloudWatch Events. Events resulting from job-completion state changes and from application availability issues are delivered to CloudWatch Events in near-real time. You can define simple rules to indicate which events are of interest to you, and to specify actions to take when an event matches a rule. Examples of actions you can trigger include:

- Invoking an AWS Lambda function
- Invoking the Amazon EC2 `RunInstance` API action
- Relaying the event to Amazon Kinesis Data Streams
- Activating an AWS Step Functions state machine

AWS Signer reports to CloudWatch Events whenever the state of a signing job changes. Customers using a single account for both the signing profile and signing job will see only a single event. Customers using separate accounts for the signing profile and signing jobs will see the same event sent to each account.

The following JSON shows an example of the "Signer Job Status Change" event that AWS Signer reports.

```
{
   "version":"0",
   "id":"event_ID",
   "detail-type":"Signer Job Status Change",
   "source":"aws.signer",
   "account":"account_ID",
   "time":"2018-04-26T20:01:47Z",
   "region":"region",
   "resources":[
      "arn:aws:signer:us-east-1:account_ID:/signing-jobs/job_ID"
   ],
   "detail":{
      "certificate_arn":"arn:aws:acm:region:account_ID:certificate/certificate_ID",
      "job_id":"job_ID",
      "destination":{
         "bucketName":"S3_bucket_name",
         "key":"S3_key_ID"
      },
      "source":{
         "bucketName":"S3_bucket_name",
         "key":"code",
         "version":"version_ID"
      },
      "platform":"Platform",
      "status":"Succeeded"
   }
}
```

For more information, see the Amazon CloudWatch Events User Guide.

Security in AWS Signer

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The shared responsibility model describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the AWS Compliance Programs. To learn about the compliance programs that apply to AWS Signer, see AWS Services in Scope by Compliance Program.
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using AWS Signer. The following topics show you how to configure AWS Signer to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS Signer resources.

Topics

Identity and Access Management for AWS Signer

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 AWS Signer resources:

- **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 for a single IAM entity and which can be embedded directly into a single user, group, or role.

For more information, see:

- Customer managed policies for Signer
- Inline policies for Signer
- Use Signer actions in IAM
- Managed policies and inline policies in the IAM documentation.

Customer managed policies for Signer

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.

To manage policies in the AWS Management Console

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.
- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

Inline policies for Signer

Inline policies are standalone identity-based policies that an administrator creates and embeds directly into a single principal (user, group, or role). Administrators can create and manage policies using the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the IAM API.

To manage policies in the AWS Management Console

To provide access, add permissions to your users, groups, or roles:

- Users and groups in AWS IAM Identity Center:

  Create a permission set. Follow the instructions in Create a permission set in the *AWS IAM Identity Center User Guide*.
- Users managed in IAM through an identity provider:

  Create a role for identity federation. Follow the instructions in Create a role for a third-party identity provider (federation) in the *IAM User Guide*.
- IAM users:
  - Create a role that your user can assume. Follow the instructions in Create a role for an IAM user in the *IAM User Guide*.
  - (Not recommended) Attach a policy directly to a user or add a user to a user group. Follow the instructions in Adding permissions to a user (console) in the *IAM User Guide*.

Examples

- Limit Access for Signing to All Signing Profiles Within an Account
- Limit Access for Signing to a Specific Signing Profile
- Limit Access for Signing to a Specific Signing Profile Version
- Allow Full Access

Limit Access for Signing to All Signing Profiles Within an Account

The following policies allow a principal to discover every `SigningProfile` within an account and to use any of them to submit, describe, and list signing jobs.

Policy for Lambda

JSON

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

Policy for containers

JSON

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "signer:GetSigningProfile",
            "signer:ListSigningProfiles",
            "signer:SignPayload",
            "signer:GetRevocationStatus",
            "signer:DescribeSigningJob",
            "signer:ListSigningJobs"
         ],
         "Resource":"*"
      }
   ]
}
```

Limit Access for Signing to a Specific Signing Profile

The following policies allow a principal to call `GetSigningProfile` and `StartSigningJob` only on profile `MySigningProfile`.

Policy for Lambda

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "signer:GetSigningProfile",
                "signer:StartSigningJob"
            ],
            "Resource": "arn:aws:signer:us-east-1:444455556666:/signing-
profiles/MySigningProfile"
        },
        {
            "Effect": "Allow",
            "Action": [
                "signer:ListSigningJobs",
                "signer:ListSigningProfiles",
                "signer:DescribeSigningJob"
            ],
            "Resource": "*"
        }
    ]
}
```

Policy for containers

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "signer:GetSigningProfile",
                "signer:SignPayload"
            ],
            "Resource": "arn:aws:signer:us-east-1:444455556666:/signing-
profiles/MySigningProfile"
        },
        {
            "Effect": "Allow",
            "Action": [
                "signer:ListSigningJobs",
                "signer:ListSigningProfiles",
                "signer:DescribeSigningJob"
            ],
            "Resource": "*"
        }
    ]
}
```

Limit Access for Signing to a Specific Signing Profile Version

The following policy allows a principal to call `GetSigningProfile` and `StartSigningJob` only on version `abcde12345` of profile `MySigningProfile`.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "signer:GetSigningProfile",
                "signer:SignPayload"
            ],
            "Resource": "arn:aws:signer:us-east-1:444455556666:/signing-
profiles/MySigningProfile",
            "Condition": {
                "StringEquals": {
```

```
                    "signer:ProfileVersion": "version"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "signer:ListSigningJobs",
                "signer:ListSigningProfiles",
                "signer:DescribeSigningJob"
            ],
            "Resource": "*"
        }
    ]
}
```

Allow Full Access

The following policy allows a principal to perform any AWS Signer action.

JSON

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

Use Signer actions in IAM

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 Signer 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 IAM JSON policy element reference in the *IAM User Guide*.

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

AWS Signer API Operations and Permissions

API Operation	Required Permissions (API Actions)
AddProfilePermission	signer:AddProfilePermission
CancelSigningProfile	signer:CancelSigningProfile
DescribeSigningJob	signer:DescribeSigningJob
GetRevocationStatus	signer:GetRevocationStatus
GetSigningPlatform	signer:GetSigningPlatform
GetSigningProfile	signer:GetSigningProfile
ListProfilePermissions	signer:ListProfilePermissions
ListSigningJobs	signer:ListSigningJobs
ListSigningPlatforms	signer:ListSigningPlatforms
ListSigningProfiles	signer:ListSigningProfiles
ListTagsForResource	signer:ListTagsForResource
PutSigningProfile	signer:PutSigningProfile
RemoveProfilePermission	signer:RemoveProfilePermission
RevokeSignature	signer:RevokeSignature

API Operation	Required Permissions (API Actions)
RevokeSigningProfile	signer:RevokeSigningProfile
SignPayload	signer:SignPayload
StartSigningJob	signer:StartSigningJob
TagResource	signer:TagResource
UntagResource	signer:UntagResource

For the actions `StartSigningJob`, `GetSigningProfile`, `CancelSigningProfile`,`RevokeSigningProfile`, and `SignPayload`, use the `signer:ProfileVersion` condition key to limit what version of a signing profile a principal has access to.

AWS Signer API Condition Keys

Condition Key	Description	APIs
signer:ProfileVersion	Limit access to a specific version of a Signing Profile	StartSigningJob GetSigningProfile CancelSigningProfile RevokeSigningProfile SignPayload

Code examples

The following code examples show how to use Signer with the Java AWS software development kit (SDK).

Users need programmatic access if they want to interact with AWS outside of the AWS Management Console. The way to grant programmatic access depends on the type of user that's accessing AWS.

To grant users programmatic access, choose one of the following options.

Which user needs programmatic access?	To	By
IAM	(Recommended) Use console credentials as temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Login for AWS local development in the *AWS Command Line Interface User Guide*. • For AWS SDKs, see Login for AWS local development in the *AWS SDKs and Tools Reference Guide*.
Workforce identity (Users managed in IAM Identity Center)	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Configuring the AWS CLI to use AWS IAM Identity Center in the *AWS Command Line Interface User Guide*.

Which user needs programmatic access?	To	By
		• For AWS SDKs, tools, and AWS APIs, see IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*.
IAM	Use temporary credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions in Using temporary credentials with AWS resources in the *IAM User Guide*.
IAM	(Not recommended) Use long-term credentials to sign programmatic requests to the AWS CLI, AWS SDKs, or AWS APIs.	Following the instructions for the interface that you want to use. • For the AWS CLI, see Authenticating using IAM user credentials in the *AWS Command Line Interface User Guide*. • For AWS SDKs and tools, see Authenticate using long-term credentials in the *AWS SDKs and Tools Reference Guide*. • For AWS APIs, see Managing access keys for IAM users in the *IAM User Guide*.

Actions

The following code examples demonstrate how to perform individual Signer actions with the AWS Java SDK.

The following examples include only the most commonly used actions. For a complete list, see the AWS Signer API Reference.

Topics

AddProfilePermission

The following Java example shows how to use the `AddProfilePermission` 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.AddProfilePermissionRequest;
```

```
import com.amazonaws.services.signer.model.AddProfilePermissionResult;

public class AddProfilePermission {

    public static void main(String[] s) {

        String credentialsProfile = "default";
        String signingProfileName = "MyProfile";
        String signingProfileVersion = "SeFHjuJAjV";
        String principal = "account";

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

        // Add the first permission to the profile - no revisionId required.
        // Applies to all versions of the profile
        AddProfilePermissionResult result = client.addProfilePermission(new
 AddProfilePermissionRequest()
                .withProfileName(signingProfileName)
                .withStatementId("statement1")
                .withPrincipal(principal)
                .withAction("signer:StartSigningJob"));

        // Add the second permission to the profile - revisionId required.
        // Optionally specify a profile version to lock the permission to a specific
 profile version
        client.addProfilePermission(new AddProfilePermissionRequest()
                .withProfileName(signingProfileName)
                .withProfileVersion(signingProfileVersion)
                .withStatementId("statement2")
                .withPrincipal(principal)
                .withAction("signer:GetSigningProfile")
                .withRevisionId(result.getRevisionId()));
    }
}
```

CancelSigningProfile

The following Java 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 how to program a CancelSigningProfile operation .
*/
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("region")
            .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
            .build();

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

DescribeSigningJob

The following Java 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
* AWS Signer 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 file 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("jobID");

      // 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());

   }
}
```

GetRevocationStatus

The following Java example shows how to use the GetRevocationStatus 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.GetRevocationStatusRequest;
import com.amazonaws.services.signer.model.GetRevocationStatusResult;
```

```
import java.time.Instant;
import java.util.Arrays;
import java.util.Date;

public class GetRevocationStatus {

    public static void main(String[] s) {

        String credentialsProfile = "default";
        Date signatureTimestamp = Date.from(Instant.now());
        String platformId = "Notation-OCI-SHA384-ECDSA";
        String certificateHash =
 "136eb997783e8d18a073e5977238765c39f1ca9bc919cf7ccab4430e5e5c39b756f21aa8c1687e536365f5916a474
                                +
 "326c4931465816650759563436d1705657bad8ac49d370d6ea64404716e92fa2d65dcdf5bf5caa99743a8bf60594e
        String jobArn = "arn:aws:signer:region:account:/signing-jobs/jobID";
        String profileVersionArn = "arn:aws:signer:region:account:/signing-
profiles/MyProfile/version";

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

        // Get the revocation status
        GetRevocationStatusResult response = client.getRevocationStatus(new
 GetRevocationStatusRequest()
                .withSignatureTimestamp(signatureTimestamp)
                .withPlatformId(platformId)
                .withCertificateHashes(Arrays.asList(certificateHash))
                .withJobArn(jobArn)
                .withProfileVersionArn(profileVersionArn));

        // Print revoked resources
        System.out.println(response.getRevokedEntities());
    }
}
```

GetSigningPlatform

The following Java 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) {

        String credentialsProfile = "default";
        String codeSigningPlatformId = "aws-signer-platform-id";

        // Create a client.
        AWSsigner client = AWSsignerClient.builder()
                .withRegion("region")
                .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 Java 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("region")
        .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());
    }
}
```

ListProfilePermissions

The following Java example shows how to use the [ListProfilePermissions] 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.ListProfilePermissionsRequest;
import com.amazonaws.services.signer.model.ListProfilePermissionsResult;
import com.amazonaws.services.signer.model.Permission;

public class ListProfilePermissions {

    public static void main(String[] s) {
```

```
        String credentialsProfile = "default";
        String signingProfileName = "MyProfile";

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

        // List the permissions for a profile
        ListProfilePermissionsResult result = client.listProfilePermissions(new
 ListProfilePermissionsRequest()
                .withProfileName(signingProfileName));

        // Iterate through the permissions
        for (Permission permission: result.getPermissions()) {
            System.out.println("StatementId: " + permission.getStatementId());
            System.out.println("Principal: " + permission.getPrincipal());
            System.out.println("Action: " + permission.getAction());
            System.out.println("ProfileVersion: " + permission.getProfileVersion());
        }
        System.out.println("RevisionId: " + result.getRevisionId());
    }
}
```

ListSigningJobs

The following Java 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
* AWS Signer 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 file 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 Java 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("region")
                .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 Java 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("region")
                .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
                .build();

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

            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);
    }
}
```

ListTagsForResource

The following Java example shows how to use the [ListTagsForResource] 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.ListTagsForResourceRequest;
import com.amazonaws.services.signer.model.ListTagsForResourceResult;

import java.util.Map;

public class ListTagsForResource {

    public static void main(String[] s) {

        String credentialsProfile = "default";
        String signingProfileArn = "arn:aws:signer:region:account:/signing-
profiles/MyProfile";

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

        // List the tags for a signing profile
        ListTagsForResourceResult result = client.listTagsForResource(
                new ListTagsForResourceRequest().withResourceArn(signingProfileArn));

        // Iterate through the tags
        for (Map.Entry<String, String> tag: result.getTags().entrySet()) {
            System.out.println("Key: " + tag.getKey());
            System.out.println("Value: " + tag.getValue());
        }
    }
}
```

PutSigningProfile

Code signing for AWS IoT

The following Java examples show how to use the PutSigningProfile operation to create a new signing profile. Code signing profiles can 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) {

        String credentialsProfile = "default";
        String codeSigningProfileName = "MyProfile";
        String codeSigningCertificateArn =
 "arn:aws:acm:region:123456789:certificate/certID";

        // Create a client.
        AWSSigner client = AWSSignerClient.builder()
            .withRegion("region")
            .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
            .build();

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

Code signing for AWS Lambda

The next example shows how to use the PutSigningProfileProfile operation to create a new signing profile for AWS Lambda. Code signing profiles can 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;

public class PutSigningProfile {

    public static void main(String[] s) {

        String credentialsProfile = "default";
        String signingProfileName = "MyProfile";
        String signingPlatformId = "AWSLambda-SHA384-ECDSA";

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

        // Create a code signing profile.
        client.putSigningProfile(new PutSigningProfileRequest()
                .withProfileName(signingProfileName)
                .withPlatformId(signingPlatformId));
    }
}
```

RemoveProfilePermission

The following Java example shows how to use the RemoveProfilePermission 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.ListProfilePermissionsRequest;
```

```
import com.amazonaws.services.signer.model.ListProfilePermissionsResult;
import com.amazonaws.services.signer.model.RemoveProfilePermissionRequest;

public class RemoveProfilePermission {

    public static void main(String[] s) {

        String credentialsProfile = "default";
        String signingProfileName = "MyProfile";

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

        // Get the latest revisionId for the profile
        ListProfilePermissionsResult result = client.listProfilePermissions(new
 ListProfilePermissionsRequest()
                .withProfileName(signingProfileName));

        // Remove a specific permission from the profile
        client.removeProfilePermission(new RemoveProfilePermissionRequest()
                .withProfileName(signingProfileName)
                .withStatementId("statement1")
                .withRevisionId(result.getRevisionId()));
    }
}
```

RevokeSignature

The following Java example shows how to use the RevokeSignature 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.RevokeSignatureRequest;

public class RevokeSignature {

    public static void main(String[] s) {
```

```
        String credentialsProfile = "default";
        String signingJobId = "jobID";
        String revokeReason = "Reason for revocation";

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

        // Revoke a signing job
        client.revokeSignature(new RevokeSignatureRequest()
                .withJobId(signingJobId)
                .withReason(revokeReason));
    }
}
```

RevokeSigningProfile

The following Java example shows how to use the RevokeSigningProfile 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.RevokeSigningProfileRequest;

import java.time.Instant;
import java.util.Date;

public class RevokeSigningProfile {

    public static void main(String[] s) {

        String credentialsProfile = "default";
        String signingProfileName = "MyProfile";
        String signingProfileVersion = "version";
        String revokeReason = "Reason for revocation";

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

```
                .withRegion("region")
                .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
                .build();

        // Revoke a signing profile
        client.revokeSigningProfile(new RevokeSigningProfileRequest()
                .withProfileName(signingProfileName)
                .withProfileVersion(signingProfileVersion)
                .withReason(revokeReason)
                .withEffectiveTime(Date.from(Instant.now())));
    }
}
```

SignPayload

The following Java example shows how to use the SignPayload 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.SignPayloadRequest;

import java.nio.ByteBuffer;

public class SignPayload{

    public static void main(String[] s) {

        String credentialsProfile = "default";
        String signingProfileName = "MyProfile";
        String payloadFormat = "application/vnd.cncf.notary.payload.v1+json";
        String payload = "{\n" +
            "  \"targetArtifact\": {\n" +
            "    \"mediaType\": \"application/vnd.docker.distribution.manifest.v2+json
\",\n" +
            "    \"digest\":
 \"sha256:73c803930ea3ba1e54bc25c2bdc53edd0284c62ed651fe7b00369da519a3c333\",\n" +
            "    \"size\": 16724,\n" +
            "    \"annotations\": {\n" +
            "        \"io.wabbit-networks.buildId\": \"123\"" +
            "    }\n" +
```

```
            "  }\n" +
            "}";

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

        // Sign a payload
        client.signPayload(new SignPayloadRequest()
                .withProfileName(signingProfileName)
                .withPayloadFormat(payloadFormat)
                .withPayload(ByteBuffer.wrap(payload.getBytes())));
    }
}
```

StartSigningJob

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

```
package com.amazonaws.samples;

import com.amazonaws.services.signer.AWSSigner;
import com.amazonaws.services.signer.AWSSignerClient;
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
* AWS Signer 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 AWS Signer can use for
*                            your signed code
*                          - Optional Amazon S3 bucket prefix
*
*/

public class StartSigningJob {

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

      // Define variables.
      String bucketSrc = "amzn-s3-demo-source-bucket";
      String key = "Code-Image-File";
      String objectVersion = "Source-S3-File-Version";
      String bucketDest = "amzn-s3-demo-destination-bucket";
      S3Source s3src = new S3Source()
          .withBucketName(bucketSrc)
          .withKey(key)
          .withVersion(objectVersion);
      Source src = new Source().withS3(s3src);
      S3Destination s3Dest = new S3Destination().withBucketName(bucketDest);
      Destination dest = new Destination().withS3(s3Dest);
      String signingProfileName = "MyProfile";

      // 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)
            .withProfileName(signingProfileName);

      // 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());

   }
}
```

TagResource

The following Java example shows how to use the TagResource 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.TagResourceRequest;

import java.util.Collections;

public class TagResource {

    public static void main(String[] s) {

        String credentialsProfile = "default";
        String signingProfileArn = "arn:aws:signer:region:account:/signing-
profiles/MyProfile";
        String tagKey = "Key";
        String tagValue = "Value";

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

        // Add a tag to a signing profile
        client.tagResource(new TagResourceRequest()
```

```
                .withResourceArn(signingProfileArn)
                .withTags(Collections.singletonMap(tagKey, tagValue)));
    }
}
```

UntagResource

The following Java example shows how to use the UntagResource 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.UntagResourceRequest;

import java.util.Collections;

public class UntagResource {

    public static void main(String[] s) {

        String credentialsProfile = "default";
        String signingProfileArn = "arn:aws:signer:region:account:/signing-
profiles/MyProfile";
        String tagKey = "Key";

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

        // Remove a tag from a signing profile
        client.untagResource(new UntagResourceRequest()
                .withResourceArn(signingProfileArn)
                .withTagKeys(Collections.singletonList(tagKey)));
    }
}
```

Document History for Developer Guide

Latest documentation update: November 19, 2018

The following table describes the documentation release history of AWS Signer.

Change	Description	Date
Signer open source library now available	Adds a link to the Signer open source library. You now have the option of using the Signer binary or the open source library.	July 24, 2024
Added caveat about image verification in AWS GovCloud (US) Region	To verify images signed using a signing profile created in an AWS GovCloud (US) Region, you must set `signingAuthority` to `aws-us-gov-signer-ts` in your trust policy.	March 13, 2024
Added support for container image signing.	Introduced code signing for container images stored in an Open Container Initiative (OCI) compliant container registry such as Amazon ECR.	June 22, 2023
Added configuration steps using the console.	Introduced AWS Code Signer Console for Lambda applications.	May 8, 2020
Added new content	Integrated AWS Signer with AWS IoT Device Management.	November 8, 2018
Launched AWS Signer	This release introduces AWS Signer.	December 20, 2017

AWS Glossary

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