AWS Signer

Developer Guide

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 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 meet your compliance requirements.

- **Code signing for AWS Lambda**

  With Code Signing for AWS Lambda, you can ensure that only trusted code runs in your Lambda functions. You can use AWS Signer to create digitally signed packages for Lambda deployment. 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.
- **Code signing for AWS IoT**

  With Code Signing for AWS IoT, 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). In order to sign code, you import a third-party code-signing certificate with ACM that is used to sign updates in FreeRTOS and AWS IoT Device Management.

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

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

Integrated services

AWS Signer is integrated with the following services.

AWS Lambda

AWS Lambda provides the ability to deploy functions across organizations, but can also provide broad privileges, including the ability to reach inside VPCs to access databases and mission critical applications. Organizations consequently need to protect themselves from unauthorized code being deployed on their networks.

AWS Signer provides a mechanism to ensure that only signed and trusted AWS Lambda functions are deployed by an organization. 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. AWS Signer profiles can be managed through the Signer console, API, and CLI.

Amazon FreeRTOS

Amazon FreeRTOS is a microcontroller operating system based on the FreeRTOS kernel. It includes libraries for connectivity and security. You can build and deploy your embedded applications on top of Amazon FreeRTOS. To ensure the security of deployments to these microcontrollers, Amazon

FreeRTOS uses 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.

AWS IoT Device Management

With AWS IoT Device Management you can manage Internet-connected devices and establish secure, bidirectional communication between them. To do so, AWS IoT Device Management uses 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.

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.

CloudWatch Events

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

Supported Regions

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

Quotas

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 guarantee 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 into your AWS account.
2. Open the Service Quotas console at https://console.aws.amazon.com/servicequotas/.
3. In the **AWS services** list, type "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 or not the quota is adjustable. Choose the name of a quota for more information about it.

To request a quota increase

1. In the **Service quotas** list, choose the radio button for an adjustable quota.

2. Choose the **Request quota increase** button.
3. Complete and submit the **Request quota increase** form.

Pricing for AWS Signer

There is no additional charge to use AWS Signer with AWS IoT Device Management or AWS Lambda. You pay for the storage of signed and unsigned objects in Amazon S3 based on your usage history. There are no minimum fees and no required up-front commitments.

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

Security of integrated services

AWS Signer is used in combination with other AWS services and relies on the security architectures of those services. For example, the AWS Lambda and AWS IoT services that rely on code signing have extensive security requirements. Similarly, AWS Certificate Manager issues signing certificates within a framework of monitoring and managed key storage. AWS Signer customers should be familiar with the following security documentation:

- Security in AWS Certificate Manager in the *AWS Certificate Manager User Guide*
- Security in AWS IoT in the *AWS IoT Developer Guide*
- Security in AWS Lambda in the *AWS Lambda Developer Guide*.

Authentication and access control

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.

Authentication

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

- **AWS account root user**

  When you create an AWS account, you begin with one sign-in identity that has complete access to all AWS services and resources in the account. This identity is called the AWS account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks. Safeguard your root user credentials and use them to perform the tasks that only the root user can perform. For the complete list of tasks that require you to sign in as the root user, see Tasks that require root user credentials in the *AWS General Reference*.

- **IAM users and groups**

  An *IAM user* is an identity within your AWS account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see Managing access keys for IAM users in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

  An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

  Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see When to create an IAM user (instead of a role) in the *IAM User Guide*.

- **IAM role**

  An IAM role is an IAM identity that you can create in your account that has specific permissions. An IAM role is similar to an IAM user in that it is an AWS identity with permissions policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have standard long-term credentials such as a password or access keys associated with it. Instead, when you assume a role, it provides you with temporary security credentials for your role session. IAM roles with temporary credentials are useful in the following situations:

  - **Federated user access** – Instead of creating an IAM user, you can use existing identities from AWS Directory Service, your enterprise user directory, a web identity provider, or the IAM Identity Center identity store. These identities are known as *federated identities*. To assign permissions to federated identities, you can create a role and define permissions for the role. When an external identity authenticates, the identity is associated with the role and is granted the permissions that are defined by it. If you use IAM Identity Center, you configure a permission set. IAM Identity Center correlates the permission set to a role in IAM to control what your identities can access after they authenticate. For more information about identity federation, see Creating a role for a third-party Identity Provider in the *IAM User Guide*. For more information about IAM Identity Center, see What is IAM Identity Center? in the *AWS IAM Identity Center (successor to AWS Single Sign-On) User Guide*.
  - **AWS service access** – A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Creating a role to delegate permissions to an AWS service in the *IAM User Guide*.
  - **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making AWS CLI or AWS API requests. This is preferable to storing access keys within the EC2 instance. To assign an AWS role to an EC2 instance

and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see Using an IAM role to grant permissions to applications running on Amazon EC2 instances in the *IAM User Guide*.

Access control

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

Overview of managing access to Signer resources

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

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

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

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

Customer managed policies 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. For more information about using the console to administer customer managed policies, see the following topics in the IAM User Guide.

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

- Deleting Customer Managed Policies

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

Inline policies for Signer

Inline policies are policies that an administrator creates and manages and embeds directly into a single principal (user, group, or role). The following policy examples show how to grant permissions to perform AWS Signer actions. For more information about attaching inline policies, see Working with Inline Policies in the IAM User Guide. You can use the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the IAM API to create and embed inline policies.

Topics

Limit Access for Signing to All Signing Profiles Within an Account

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

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

Limit Access for Signing to a Specific Signing Profile

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

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "signer:GetSigningProfile",
            "signer:StartSigningJob"
```

```
         ],
         "Resource":"arn:aws:signer:us-west-2:123456789012:/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`.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "signer:GetSigningProfile",
            "signer:StartSigningJob"
         ],
         "Resource":"arn:aws:signer:us-west-2:123456789012:/signing-profiles/
MySigningProfile",
         "Condition":{
            "StringEquals":{
               "signer:ProfileVersion":"abcde12345"
            }
         }
      },
      {
         "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.

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

```
    ]
}
```

AWS Signer API permissions

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 Available Keys in the *IAM User Guide*.

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

AWS Signer API Operations and Permissions

API Operation	Required Permissions (API Actions)
AddProfilePermission	signer:AddProfilePermission
CancelSigningProfile	signer:CancelSigningProfile
DescribeSigningJob	signer:DescribeSigningJob
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
RevokeSigningProfile	signer:RevokeSigningProfile
StartSigningJob	signer:StartSigningJob
TagResource	signer:TagResource
UntagResource	signer:UntagResource

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

When applied to policies providing access to the actions `AddProfilePermission` or `RemoveProfilePermission`, the `signer:ProfileVersion` restricts the principal's add or remove permissions to a specific profile version.

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`
		`AddProfilePermission`
		`RemoveProfilePermission`

Getting started

You can use AWS Signer to sign IoT code through the Amazon FreeRTOS or AWS IoT Device Management consoles when you create an over-the-air (OTA) job, or by using the Signer API. You can use AWS Signer to sign AWS Lambda deployment packages using the Signer console or API.

Topics

(For IoT only) 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.

Note
This requirement applies only to IoT signing. It is not necessary for Lambda signing.

```
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.

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.

Define an IAM policy

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

To manually create an IAM policy:

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

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

```
    {
      "Effect": "Allow",
      "Action": [
        "signer:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

In order to use the StartSigningJob (p. 34) API operation, you must specify an Amazon S3 bucket to save the signing job. In order to do so, attach the following policy to the designated user.

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

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

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.

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**.

Signing platforms in AWS Signer

A signing platform in AWS Signer is a predefined set of instructions that specifies signature format and signing (hash and encryption) algorithms. AWS Signer uses these instructions to sign a zip file for AWS Lambda or a file in Amazon FreeRTOS or AWS IoT Device Management. Although you cannot edit a signing platform, you can modify some of the platform's implementation by including hash or encryption algorithm overrides in a signing profile (p. 14).

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

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

Signing profiles in AWS Signer (API/CLI)

A signing profile is a code-signing template that can be used to predefine the signature specifications for a signing job. A signing profile includes a signing platform to designate the file type to be signed (a binary file for IoT or a zip file for AWS Lambda), the signature format, and the signature algorithms. Once 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 Overview of managing access to Signer resources (p. 6).

For code signing for IoT, the signing profile also specifies the AWS Certificate Manager certificate ARN to generate signatures. The signing profile includes any hash or encryption algorithm overrides applied to the IoT signing platform.

For code signing for Lambda, the signing profile also specifies the validity period of signatures. By default, signature validity is set to 135 months (11 years and 3 months), which is the maximum validity allowed. The ARN of the signing profile version is used in AWS Lambda to designate a trusted source for validating signed zip files. If code signing is enabled for Lambda functions, only zip files signed by specified signing profile versions will pass signature validation checks.

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

Use the following action or command to create a signing profile:

- **API:** `PutSigningProfile`

- **AWS CLI:** **put-signing-profile**

Use the following action or command to cancel a signing profile:

- **API:** `CancelSigningProfile`
- **AWS CLI:** **cancel-signing-profile**

A canceled or revoked profile remains visible in customer accounts until all signatures generated by that profile have expired, plus an additional six months. After that time, the profile is automatically deleted.

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

- **API:** `GetSigningProfile`
- **AWS CLI:** **get-signing-profile**

For a list of all available signing profiles, including those in the `CANCELED` state, use the following action or command:

- **API:** `ListSigningProfiles`
- **AWS CLI:** **list-signing-profiles**

For more information about the configurations and parameters related to signing profiles, see SigningPlatform in the *AWS Signer API Reference Guide* or the AWS Command Line Interface.

Signing jobs in AWS Signer (API/CLI)

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 once completed 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.

Use the following action or command to start a signing job:

- **API:** `StartSigningJob`
- **AWS CLI:** **start-signing-job**

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

- **API:** `DescribeSigningJob`
- **AWS CLI:** **describe-signing-job**

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

- **API:** `ListSigningJobs`

- **AWS CLI: list-signing-jobs**

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

Revoking AWS Signer resources (AWS Lambda)

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. 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 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.

Automating AWS Signer 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.

Using the AWS Signer Console

You can create signing profiles for Lambda applications using the AWS Signer Console instead of the CLI or SDK.

> **Note**
> You cannot create signing profiles for other signing platforms using the console.

To create a signing profile (console)

1. Log into the AWS Signer console.
2. Choose **Create Signing Profile**.
3. Enter a unique name for your signing profile. Valid characters include uppercase A-Z, lowercase a-z, numbers 0-9, and underscore (_).
4. Specify the **Signature Validity Period** in months, days, or years. The default value is 135 months (11 years and 6 months).
5. Next, assign a **Tag key** and a **Tag value**. When you assign tags to your signing profile, you can manage access using tag-based resource policies.

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

The console displays a message that you have successfully created a signing profile and displays the following information:

- **Profile name** - Your profile name
- **Profile version** - The version of the created profile
- **Platform** - The signing platform, in this case, AWS Lambda.
- **Status** - **Active**
- **Profile ARN** - The ARN associated with the profile
- **Versioned profile ARN** - The profile ARN plus the profile version
- **Signature validity period** - The length of time that AWS Signer signs code with this profile

You can also choose **Cancel profile** to cancel the profile, or choose **Revoke Profile**.

If you choose **Cancel profile**, AWS Signer displays a confirmation prompt. Once you cancel the profile, you cannot use it again.

A canceled profile remains in the `CANCELED` state for two years, and is then automatically deleted.

If you choose **Revoke Profile**, select a date and the reason for the revocation. You cannot undo this action.

In the **Tags** section, you can manage the tags assigned to the profile.

You can begin signing code by choosing **Start Signing Job**.

Using the AWS Signer Console for Signing Jobs

Before you begin signing Lambda code, you must be sure you include permissions in your IAM policy for Amazon S3. See Define an IAM Policy" (p. 12) for an example.

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 has the `*.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 storese 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 requestor 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

Managing Signing Profiles with AWS Signer Console

You can manage your signing profiles from the **Manage signing profiles**page by choosing the profile name from the **Profile name** column.

After selecting a profile name, the AWS Signer console displays the signing profile details. You can perform the following actions:

- **Start signing job** – Start a new signing job using this profile
- **Cancel profile** – The profile can no longer be used to generate new signatures. Existing signatures generated by the profile remain unaffected.
- **Revoke profile** – The profile can no longer be used to generate signatures. Existing signatures generated after the effective start time of revocation become invalid.
- **Manage tags** – Add or remove tags from the signing profile

Using the AWS Signer API

You can use the AWS Signer API to interact with the service programmatically. For more information, see the AWS Signer API Reference. The following topics show you how to use Java to program the SDK.

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 = "123456789012";
```

```
        // Create a client.
        final AWSSigner client = AWSSignerClient.builder()
                .withRegion("us-west-2")
                .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("us-west-2")
            .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 in Linux.
      AWSCredentials credentials = null;
      try {
          credentials = new ProfileCredentialsProvider().getCredentials();
      }
      catch (Exception ex) {
          throw new AmazonClientException("Cannot load your credentials from file.", ex);
      }

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

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

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

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

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

   }
}
```

GetSigningPlatform

The following 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("us-west-2")
                .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
                .build();

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

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

GetSigningProfile

The following 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("us-west-2")
        .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
        .build();

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

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

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("us-west-2")
                .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 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("us-west-2")
                .withCredentials(new ProfileCredentialsProvider(credentialsProfile))
                .build();

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

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

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

ListSigningProfiles

The following 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("us-west-2")
                .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:us-west-2:123456789012:/signing-
profiles/MyProfile";

        // Create a client.
        final AWSSigner client = AWSSignerClient.builder()
                .withRegion("us-west-2")
                .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:us-
west-2:123456789:certificate/6e7e9e0c-0d2a-
        4835-b2cc-2326a16c86f0";

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

        // creating a code signing profile.
        client.putSigningProfile(new PutSigningProfileRequest()
            .withProfileName(codeSigningProfileName)
            .withSigningMaterial(new SigningMaterial()
                .withCertificateArn(codeSigningCertificateArn))
        .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("us-west-2")
                .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 = "02d65707-dfc3-4e9b-b8d7-ddf94e130d93";
        String revokeReason = "Reason for revocation";

        // Create a client.
        final AWSSigner client = AWSSignerClient.builder()
                .withRegion("us-west-2")
                .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 = "SeFHjuJAjV";
        String revokeReason = "Reason for revocation";

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

        // Revoke a signing profile
        client.revokeSigningProfile(new RevokeSigningProfileRequest()
                .withProfileName(signingProfileName)
                .withProfileVersion(signingProfileVersion)
```

```
                .withReason(revokeReason)
                .withEffectiveTime(Date.from(Instant.now())));
    }
}
```

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.

In order to use the `StartSigningJob` operation, make sure that the designated user's IAM policy includes Amazon S3 permissions. See `Define an IAM Policy` (p. 12) for an example.

```
package com.amazonaws.samples;

import com.amazonaws.services.signer.AWSSigner;
import com.amazonaws.services.signer.AWSSignerClient;
import com.amazonaws.services.signer.model.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 = "Source-Bucket-Name";
```

```
String key = "Code-Image-File";
String objectVersion = "Source-S3-File-Version";
String bucketDest = "Destination-Bucket-Name";
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:us-west-2:123456789012:/signing-
profiles/MyProfile";
        String tagKey = "Key";
        String tagValue = "Value";

        // Create a client.
        final AWSSigner client = AWSSignerClient.builder()
                .withRegion("us-west-2")
                .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:us-west-2:123456789012:/signing-
profiles/MyProfile";
        String tagKey = "Key";

        // Create a client.
        final AWSSigner client = AWSSignerClient.builder()
                .withRegion("us-west-2")
                .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.

update-history-change	update-history-description	update-history-date
Added configuration steps using the console. (p. 38)	Introduced AWS Code Signer Console for Lambda applications.	May 8, 2020
Added new content (p. 38)	Integrated AWS Signer with AWS IoT Device Management.	November 8, 2018
Launched AWS Signer (p. 38)	This release introduces AWS Signer.	December 20, 2017

AWS glossary

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