# AWS Key Management Service

## Developer Guide

# AWS Key Management Service: Developer Guide

# Table of Contents

# What is AWS Key Management Service?

AWS Key Management Service (AWS KMS) is a managed service that makes it easy for you to create and control the encryption keys used to encrypt your data. AWS KMS is integrated with other AWS services including Amazon Elastic Block Store (Amazon EBS), Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon Elastic Transcoder, Amazon WorkMail, Amazon Relational Database Service (Amazon RDS), and others to make it simple to encrypt your data with encryption keys that you manage. AWS KMS is also integrated with AWS CloudTrail to provide you with key usage logs to help meet your auditing, regulatory and compliance needs.

AWS KMS lets you create master keys that can never be exported from the service and which can be used to encrypt and decrypt data based on policies you define.

You can perform the following management actions on master keys by using AWS KMS:

- Create, describe, and list master keys
- Enable and disable master keys
- Set and retrieve master key usage policies (access control)
- Create, delete, list, and update *aliases*, which are friendly names that point to your master keys
- Delete master keys to complete the key lifecycle

With AWS KMS you can also perform the following cryptographic functions using master keys:

- Encrypt, decrypt, and re-encrypt data
- Generate data encryption keys that you can export from the service in plaintext or encrypted under a master key that doesn't leave the service
- Generate random numbers suitable for cryptographic applications

By using AWS KMS, you gain more control over access to data you encrypt. You can use the key management and cryptographic features directly in your applications or through AWS services that are integrated with AWS KMS. Whether you are writing applications for AWS or using AWS services, AWS KMS enables you to maintain control over who can use your master keys and gain access to your encrypted data.

AWS KMS is integrated with AWS CloudTrail, a service that delivers log files to an Amazon S3 bucket that you designate. By using CloudTrail you can monitor and investigate how and when your master keys have been used and by whom.

For a more detailed introduction to AWS KMS, see AWS KMS Concepts (p. 2).

To learn more about how AWS KMS uses cryptography and secures master keys, see the AWS Key Management Service Cryptographic Details whitepaper.

AWS KMS Pricing

As with other AWS products, there are no contracts or minimum commitments for using AWS KMS. For more information about AWS KMS pricing, see AWS Key Management Service Pricing.

# AWS Key Management Service Concepts

Learn the basic terms and concepts in AWS Key Management Service (AWS KMS) and how they work together to help protect your data.

Topics



## Customer Master Keys

The primary resources in AWS KMS are *customer master keys* (CMKs). CMKs are either customer-managed or AWS-managed. You can use either type of CMK to protect up to 4 kibibytes (KiB) of data directly. Typically you use CMKs to protect *data encryption keys* (or *data keys*) which are then used to encrypt or decrypt larger amounts of data outside of the service. CMKs never leave AWS KMS unencrypted, but data keys can. AWS KMS does not store, manage, or track your data keys.

There is one *AWS-managed CMK* for each service that is integrated with AWS KMS (p. 78). When you create an encrypted resource in these services, you can choose to protect that resource under the AWS-managed CMK for that service. This CMK is unique to your AWS account and the AWS region in which it is used, and it protects the data keys used by the AWS services to protect your data. Many of these services allow you to specify a customer-managed CMK instead of the AWS-managed CMK, which is useful when you need more granular control over the CMK. For example, when you use a customer-managed CMK with these services you can request that AWS KMS rotate the CMK's key material every year. You cannot do this with an AWS-managed CMK.

## Data Keys

You use data keys to encrypt large data objects within your own application outside AWS KMS. When you make a GenerateDataKey API request, AWS KMS returns a plaintext copy of the data key and a ciphertext that contains the data key encrypted under the specified CMK. You use the plaintext data

key in your application to encrypt data, and you typically store the encrypted data key alongside your encrypted data. Security best practices dictate that you should remove the plaintext data key from memory as soon as practical after use. To decrypt data in your application, you pass the encrypted data key with a Decrypt API request. AWS KMS uses your CMK to decrypt the data key into plaintext, and then returns it to you. You use the plaintext data key to decrypt your data and then remove the plaintext data key from memory as soon as practical after use.

> **Note**
> You can use AWS KMS to generate, encrypt, and decrypt data keys, but AWS KMS does not store, manage, or track your data keys. You must do this inside your application.

# Envelope Encryption

AWS KMS uses *envelope encryption* to protect data. Envelope encryption is the practice of encrypting plaintext data with a unique data key, and then encrypting the data key with a *key encryption key* (KEK). You might choose to encrypt the KEK with another KEK, and so on, but eventually you must have a *master key*. The master key is an unencrypted (plaintext) key with which you can decrypt one or more other keys.

Envelope encryption offers several benefits:

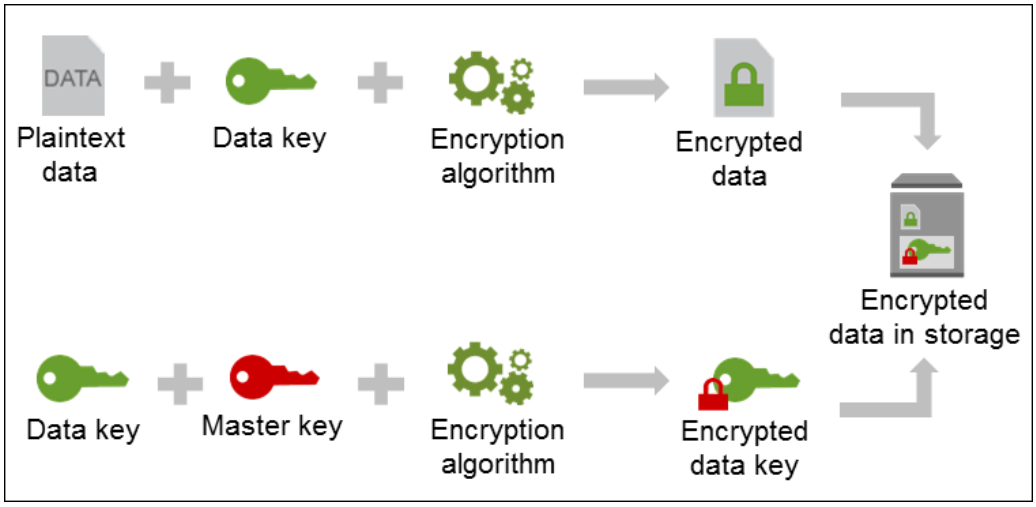
- **Protecting data keys**

  When you encrypt a data key, you don't have to worry about where to store the encrypted data key, because the security of that data key is inherently protected by encryption. You can safely store the encrypted data key alongside the encrypted data.
- **Encrypting the same data under multiple master keys**

  Encryption operations can be time consuming, particularly when the data being encrypted are large objects. Instead of reencrypting raw data multiple times with different keys, you can reencrypt only the data keys that protect the raw data.
- **Combining the strengths of multiple algorithms**

  In general, symmetric key algorithms are faster and produce smaller ciphertexts than public key algorithms, but public key algorithms provide inherent separation of roles and easier key management. You might want to combine the strengths of each.

The following image provides an overview of envelope encryption. In this scenario, the data key is encrypted with a single KEK, which is the master key. In AWS KMS, your CMK represents the master key.



# Encryption Context

All AWS KMS cryptographic operations accept an optional set of key–value pairs that can contain additional contextual information about the data. This set of key–value pairs is called *encryption context.* When encryption context is used with an encryption operation, the encryption context for the corresponding decryption operation must match for the decryption to succeed. Encryption context is not secret. Encryption context is logged, and you can use it for auditing and when controlling access to AWS KMS API operation. For more information, see Encryption Context (p. 153).

# Key Policies

When you create a CMK, you choose who can manage or use that CMK. These permissions are contained in a document called the *key policy*. You can use the key policy to add, remove, or modify permissions at any time for a customer-managed CMK, but you cannot edit the key policy for an AWS-managed CMK. For more information, see Authentication and Access Control for AWS KMS (p. 10).

# Grants

A *grant* is another mechanism for providing permissions, an alternative to the key policy. You can use grants to give long-term access that allows AWS principals to use your customer-managed CMKs. For more information, see Using Grants (p. 43).

# Grant Tokens

When you create a grant, the permissions specified in the grant might not take effect immediately due to eventual consistency. If you need to mitigate the potential delay, use the *grant token* that you receive in the response to your CreateGrant API request. You can pass the grant token with some AWS KMS API requests to make the permissions in the grant take effect immediately. The following AWS KMS API operations accept grant tokens:

- CreateGrant
- Decrypt
- DescribeKey
- Encrypt
- GenerateDataKey
- GenerateDataKeyWithoutPlaintext
- ReEncrypt
- RetireGrant

A grant token is not a secret. The grant token contains information about who the grant is for and therefore who can use it to cause the grant's permissions to take effect more quickly.

# Auditing CMK Usage

You can use AWS CloudTrail to audit key usage. CloudTrail creates log files that contain a history of AWS API calls and related events for your account. These log files include all AWS KMS API requests made with the AWS Management Console, AWS SDKs, and command line tools, as well as those made through integrated AWS services. You can use these log files to get information about when the CMK was used, the operation that was requested, the identity of the requestor, the IP address that the request came from, and so on. For more information, see Logging AWS KMS API Calls (p. 114) and the AWS CloudTrail User Guide.

# Key Management Infrastructure

A common practice in cryptography is to encrypt and decrypt with a publicly available and peer-reviewed algorithm such as AES (Advanced Encryption Standard) and a secret key. One of the main problems with cryptography is that it's very hard to keep a key secret. This is typically the job of a key management infrastructure (KMI). AWS KMS operates the KMI for you. AWS KMS creates and securely stores your master keys, called CMKs. For more information about how AWS KMS operates, see the AWS Key Management Service Cryptographic Details whitepaper.

# Getting Started

You can use the IAM section of the AWS Management Console to create, view, edit, enable, disable, and delete customer master keys (CMKs) in AWS Key Management Service (AWS KMS). For more information, see the following topics.



## Creating Keys

You can use the IAM section of the AWS Management Console to create a customer master key (CMK).

**To create a new CMK**

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Choose **Create Key**.
4. Type an alias for the CMK.

   **Note**
   The alias is a display name that you can use to easily identify the CMK. We recommend that you choose an alias that indicates the type of data you will protect or the application you will use with the CMK. The alias must be between 1 and 32 characters (inclusive) and may contain alphanumeric characters, hyphens (-), forward slashes (/), and underscores (_). An alias cannot begin with **aws**. Aliases that begin with **aws** are reserved by Amazon Web Services to represent AWS-managed CMKs in your account.
5. Type a description for the CMK.

**Note**
The description can be up to 256 characters and should describe what the CMK will be used to encrypt.

6. Choose **Next Step**.
7. Select which IAM users and roles can administer the CMK.

   **Note**
   The AWS account (root user) has full permissions by default. As a result, any IAM users and roles whose attached policies specify the appropriate permissions can also administer the CMK.

   Choose **Next Step**.
8. Select which IAM users and roles can use the CMK to encrypt and decrypt data with the AWS KMS API.

   **Note**
   The AWS account (root user) has full permissions by default. As a result, any IAM users and roles whose attached policies specify the appropriate permissions can also use the CMK.
9. (Optional) At the bottom of the page, you can also identify other AWS accounts that can use this CMK to encrypt and decrypt data. Choose **Add an External Account** and then type the ID of the account that can use this CMK. Repeat as necessary to add more than one external account.

   **Note**
   Administrators of the external accounts must also allow access to the CMK by creating IAM policies for their users. For more information, see Allowing External AWS Accounts to Access a CMK (p. 26).
10. Choose **Next Step**.
11. Choose **Finish** to create the CMK.

# Viewing Keys

You can use the IAM section of the AWS Management Console to view customer master keys (CMKs), including CMKs managed by you and those managed by AWS.

**To view your CMKs**

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).

The console shows all the CMKs in your AWS account in the chosen region, including customer-managed and AWS-managed CMKs. The page displays the alias, key ID, status, and creation date for each CMK. AWS-managed CMKs, denoted by the orange AWS icon before the alias, are permanently enabled for use by services that use AWS KMS (p. 78). You cannot disable, edit, or delete the AWS-managed CMKs.

# Editing Keys

You can use the IAM section of the AWS Management Console to change permissions and to specify rotation for customer master keys (CMKs). You start by viewing the key details page for the CMK.

### To view the key details page for a CMK

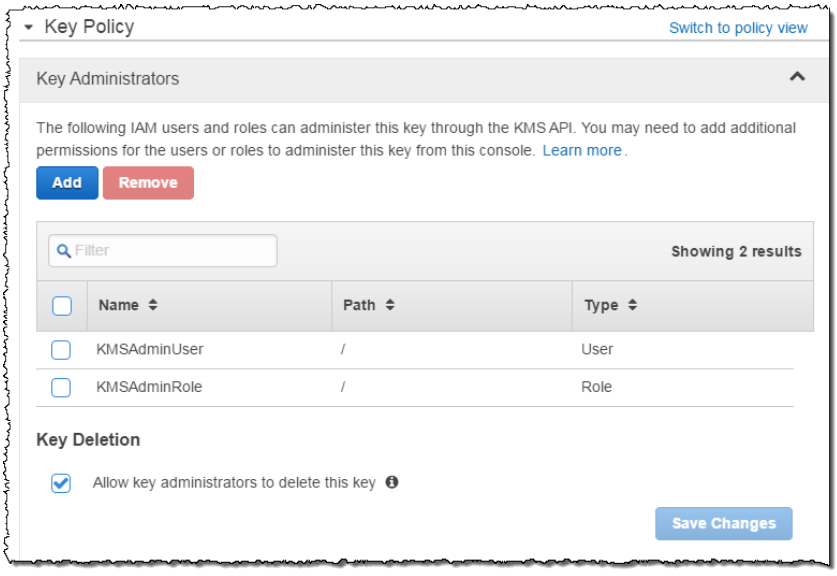
1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Choose the alias of the CMK whose details you want to see.

> **Note**
> AWS-managed CMKs, denoted by the orange AWS icon, are managed by AWS. You can view certain metadata for these CMKs, but you cannot modify them.

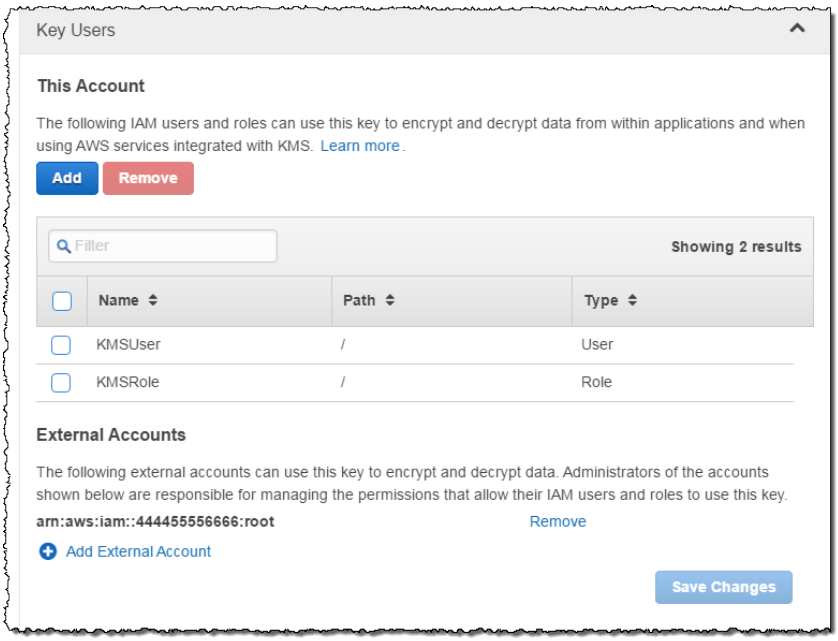
On the key details page, you can view metadata about the CMK, and you can edit the CMK in the following ways:

**Modify the description**
Use the **Description** field in the **Summary** section of the page. When you are finished, choose **Save Changes**.


**Add and remove key administrators, and allow or disallow key administrators to delete the CMK**
Use the controls in the **Key Administrators** area in the **Key Policy** section of the page.




**Add and remove key users, and allow and disallow external AWS accounts to use the CMK**
Use the controls in the **Key Users** area in the **Key Policy** section of the page.




**Enable or disable rotation**

Use the controls in the **Key Rotation** section of the page.


# Enabling and Disabling Keys

You can use the IAM section of the AWS Management Console to enable and disable customer master keys (CMKs). When you create a CMK, it is enabled by default. If you disable a CMK, it cannot be used to encrypt or decrypt data. Note that AWS-managed CMKs are permanently enabled for use by services that use AWS KMS (p. 78). You cannot disable them.

You can also delete CMKs. For more information, see Deleting Customer Master Keys (p. 63).

### To enable or disable a CMK

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Select the check box next to the name of the CMK(s) that you want to enable or disable.

   **Note**
   AWS-managed CMKs, denoted by the orange AWS icon, are managed by AWS. You can view certain metadata for these CMKs, but you cannot disable them.
4. To enable a CMK, choose **Key Actions**, **Enable**. To disable a CMK, choose **Key Actions**, **Disable**.

# Authentication and Access Control for AWS KMS

Access to AWS KMS requires credentials that AWS can use to authenticate your requests. The credentials must have permissions to access AWS resources, such as AWS KMS customer master keys (CMKs). The following sections provide details about how you can use AWS Identity and Access Management (IAM) and AWS KMS to help secure your resources by controlling who can access them.

**Topics**



## Authentication

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

- **AWS account root user** – When you sign up for AWS, you provide an email address and password for your AWS account. These are your *root credentials* and they provide complete access to all of your AWS resources.

  **Important**
  For security reasons, we recommend that you use the root credentials only to create an *administrator user*, which is an *IAM user* with full permissions to your AWS account. Then, you can use this administrator user to create other IAM users and roles with limited permissions. For more information, see Create Individual IAM Users (IAM Best Practices) and Creating An Admin User and Group in the *IAM User Guide*.

- **IAM user** – An IAM user is an identity within your AWS account that has specific permissions (for example, to use a KMS CMK). You can use an IAM user name and password to sign in to secure AWS webpages like the AWS Management Console, AWS Discussion Forums, or the AWS Support Center.

In addition to a user name and password, you can also create access keys for each user to enable the user to access AWS services programmatically, through one of the AWS SDKs or the command line tools. The SDKs and command line tools use the access keys to cryptographically sign API requests. If you don't use the AWS tools, you must sign API requests yourself. AWS KMS supports *Signature Version 4*, an AWS protocol for authenticating API requests. For more information about authenticating API requests, see Signature Version 4 Signing Process in the *AWS General Reference*.

- **IAM role** – An IAM role is another IAM identity you can create in your account that has specific permissions. It is similar to an IAM user, but it is not associated with a specific person. An IAM role enables you to obtain temporary access keys to access AWS services and resources programmatically. IAM roles are useful in the following situations:

  - **Federated user access** – Instead of creating an IAM user, you can use preexisting user identities from AWS Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. Federated users use IAM roles through an identity provider. For more information about federated users, see Federated Users and Roles in the *IAM User Guide*.

  - **Cross-account access** – You can use an IAM role in your AWS account to allow another AWS account permissions to access your account's resources. For an example, see Tutorial: Delegate Access Across AWS Accounts Using IAM Roles in the *IAM User Guide*.

  - **AWS service access** – You can use an IAM role in your account to allow an AWS service permissions to access your account's resources. For example, you can create a role that allows Amazon Redshift to access an S3 bucket on your behalf and then load data stored in the S3 bucket into an Amazon Redshift cluster. For more information, see Creating a Role to Delegate Permissions to an AWS Service in the *IAM User Guide*.

  - **Applications running on EC2 instances** – Instead of storing access keys on an EC2 instance for use by applications that run on the instance and make AWS API requests, you can use an IAM role to provide temporary access keys for these applications. To assign an IAM role to an EC2 instance, you create an *instance profile* and then attach it when you launch the instance. An instance profile contains the role and enables applications running on the EC2 instance to get temporary access keys. For more information, see Using Roles for Applications on Amazon EC2 in the *IAM User Guide*.

# Access Control

You can have valid credentials to authenticate your requests, but you also need permissions to make AWS KMS API requests to create, manage, or use AWS KMS resources. For example, you must have permissions to create a KMS CMK, to manage the CMK, to use the CMK for cryptographic operations (such as encryption and decryption), and so on.

The following pages describe how to manage permissions for AWS KMS. We recommend that you read the overview first.

- Overview of Managing Access (p. 12)
- Using Key Policies (p. 14)
- Using IAM Policies (p. 27)

# Overview of Managing Access to Your AWS KMS Resources

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

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

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

Topics



## AWS KMS Resources and Operations

To manage permissions, you should understand some basic information about resources and operations. In AWS KMS, the primary resource type is a *customer master key (CMK)*. AWS KMS also supports another resource type you can use with CMKs: an *alias*. An alias is a friendly name that points to a CMK. Some AWS KMS operations allow you to specify a CMK by its alias.

These resource types have unique Amazon Resource Names (ARNs) associated with them, as shown in the following list.

- **Customer master key (CMK)**

  ARN format:

  `arn:aws:kms:`*`AWS region`*`:`*`AWS account ID`*`:key/`*`CMK key ID`*

  Example ARN:

  `arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`

- **Alias**

  ARN format:

  `arn:aws:kms:`*`AWS region`*`:`*`AWS account ID`*`:alias/`*`alias name`*

  Example ARN:

```
arn:aws:kms:us-west-2:111122223333:alias/example-alias
```

AWS KMS provides a set of API operations to work with your AWS KMS resources. For a list of available operations and the resources affected by each operation, see AWS KMS API Permissions Reference (p. 31).

# Managing Access to AWS KMS CMKs

The primary way to manage access to your AWS KMS CMKs is with *policies*. Policies are documents that describe who has access to what. Policies attached to an IAM identity are called *identity-based policies* (or *IAM polices*), and policies attached to other kinds of resources are called *resource-based policies*. In AWS KMS, you must attach resource-based policies to your customer master keys (CMKs). These are called *key policies*. All KMS CMKs have a key policy.

You can control access to your KMS CMKs in these ways:

- **Use the key policy** – You must use the key policy to control access to a CMK. You can use the key policy alone to control access, which means the full scope of access to the CMK is defined in a single document (the key policy).

- **Use IAM policies in combination with the key policy** – You can use IAM policies in combination with the key policy to control access to a CMK. Controlling access this way enables you to manage all of the permissions for your IAM identities in IAM.

- **Use grants in combination with the key policy** – You can use grants in combination with the key policy to allow access to a CMK. Controlling access this way enables you to allow access to the CMK in the key policy, and to allow users to delegate their access to others.

For most AWS services, IAM policies are the only way to control access to the service's resources. Some services offer resource-based policies or other access control mechanisms to complement IAM policies, but these are generally optional and you can control access to the resources in these services with only IAM policies. This is not the case for AWS KMS, however. To allow access to a KMS CMK, you must use the key policy, either alone or in combination with IAM polices or grants. IAM policies by themselves are not sufficient to allow access to a CMK, though you can use them in combination with a CMK's key policy.

For more information about using key policies, see Using Key Policies (p. 14).

For more information about using IAM policies, see Using IAM Policies (p. 27).

For more information about using grants, see Using Grants (p. 43).

# Specifying Permissions in a Policy

AWS KMS provides a set of API operations. To control access to these API operations, AWS KMS provides a set of *actions* that you can specify in a policy. For more information, see AWS KMS API Permissions Reference (p. 31).

A policy is a document that describes a set of permissions. The following are the basic elements of a policy.

- **Resource** – In an IAM policy, you use an Amazon Resource Name (ARN) to specify the resource that the policy applies to. For more information, see AWS KMS Resources and

Operations (p. 12). In a key policy, you use "*" for the resource, which effectively means "this CMK." A key policy applies only to the CMK it is attached to.

- **Action** – You use actions to specify the API operations you want to allow or deny. For example, the `kms:Encrypt` action corresponds to the AWS KMS Encrypt API operation.

- **Effect** – You use the effect to specify whether to allow or deny the permissions. If you don't explicitly allow access to a resource, access is implicitly denied. You can also explicitly deny access to a resource, which you might do to make sure that a user cannot access it, even when a different policy allows access.

- **Principal** – In an IAM policy, you don't specify a principal. Instead, the identity (the IAM user, group, or role) that the policy is attached to is the implicit principal. In a key policy, you must specify the principal (the identity) that the permissions apply to. You can specify AWS accounts (root), IAM users, IAM roles, and some AWS services as principals in a key policy. IAM groups are not valid principals in a key policy.

For more information, see Using Key Policies (p. 14) and Using IAM Policies (p. 27).

## Specifying Conditions in a Policy

You can use another policy element called the *condition* to specify the circumstances in which a policy takes effect. For example, you might want a policy statement to take effect only after a specific date. Or, you might want a policy statement to control access based on whether a specific value exists in the API request.

To specify conditions, you use predefined *condition keys*. Some condition keys apply generally to AWS, and some are specific to AWS KMS. For more information, see Using Policy Conditions (p. 35).

# Using Key Policies in AWS KMS

Key policies are the primary way to control access to customer master keys (CMKs) in AWS KMS. They are not the only way to control access, but you cannot control access to CMKs without them. For more information, see Managing Access to AWS KMS CMKs (p. 13).

**Topics**



## Overview of Key Policies

A key policy is a document that uses JSON (JavaScript Object Notation) to specify permissions. You can work with these JSON documents directly, or you can use the AWS Management Console to work with them using a graphical interface called the *default view*. For more information about the console's default view for key policies, see Default Key Policy (p. 15) and Modifying a Key Policy (p. 22).

Key policy documents share a common JSON syntax with other permissions policies in AWS, and have the following basic structure:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "statement identifier",
    "Effect": "effect",
    "Principal": "principal",
    "Action": "action",
    "Resource": "resource",
    "Condition": {"condition operator": {"condition context key": "context
 key value"}}
  }]
}
```

A key policy document must have a `Version` element. We recommend setting the version to `2012-10-17` (the latest version). In addition, a key policy document must have one or more statements, and each statement consists of up to six elements:

- **Sid** – (Optional) The Sid is a statement identifier, an arbitrary string you can use to identify the statement.
- **Effect** – (Required) The effect specifies whether to allow or deny the permissions in the policy statement. The Effect must be Allow or Deny. If you don't explicitly allow access to a CMK, access is implicitly denied. You can also explicitly deny access to a CMK, which you might do to make sure that a user cannot access it, even when a different policy allows access.
- **Principal** – (Required) The principal is the identity that the permissions in the policy statement apply to. You can specify AWS accounts (root), IAM users, IAM roles, and some AWS services as principals in a key policy. IAM groups are not valid principals.
- **Action** – (Required) Actions specify the API operations to allow or deny. For example, the `kms:Encrypt` action corresponds to the AWS KMS Encrypt API operation. You can list more than one action in a policy statement. For more information, see AWS KMS API Permissions Reference (p. 31).
- **Resource** – (Required) In a key policy, you use `"*"` for the resource, which means "this CMK." A key policy applies only to the CMK it is attached to.
- **Condition** – (Optional) Conditions specify requirements that must be met for a key policy to take effect. With conditions, AWS can evaluate the context of an API request to determine whether or not the policy statement applies. For more information, see Using Policy Conditions (p. 35).

For more information about AWS policy syntax, see AWS IAM Policy Reference in the *IAM User Guide*.

# Default Key Policy

When you create a CMK programmatically—that is, with the AWS KMS API (including through the AWS SDKs and command line tools)—you have the option of providing the key policy for the new CMK. If you don't provide one, AWS KMS creates one for you. This default key policy has one statement that gives the AWS account (root user) that owns the CMK full access to the CMK. For more information about this policy statement, see Allows Access to the AWS Account and Enables IAM Policies (p. 16).

When you create a CMK with the AWS Management Console (p. 6), you can use the console's graphical interface to choose the IAM users, IAM roles, and AWS accounts that are given access to the CMK. The users, roles, and accounts that you choose are added to a key policy that the console creates for you. With the console, you can use the default view to view or modify this key policy, or you

can work with the key policy document directly. The default key policy created by the console allows the following permissions, each of which is explained in the corresponding section.

Permissions

- Allows Access to the AWS Account and Enables IAM Policies (p. 16)
- Allows Key Administrators to Administer the CMK (p. 16)
- Allows Key Users to Use the CMK (p. 18)

## Allows Access to the AWS Account and Enables IAM Policies

The default key policy gives the AWS account (root user) that owns the CMK full access to the CMK, which accomplishes two things:

- Reduces the risk of the CMK becoming unmanageable.
- Enables IAM policies to allow access to the CMK.

You cannot delete your AWS account's root user, so allowing access to this user reduces the risk of the CMK becoming unmanageable. Consider this scenario:

1. A CMK's key policy allows only one IAM user, Alice, to manage the CMK. This key policy does not allow access to the root user.
2. Someone deletes IAM user Alice.

In this scenario, the CMK is now unmanageable, and you must contact AWS Support to regain access to the CMK. The root user does not have access to a CMK unless the key policy explicitly allows it. This is different from most other resources in AWS.

IAM policies by themselves are not sufficient to allow access to a CMK, though you can use them in combination with a CMK's key policy if the key policy enables it. Giving the AWS account full access to the CMK implicitly enables you to use IAM policies to give IAM users and roles in the account access to the CMK. It does not by itself give any IAM users or roles access to the CMK, but it enables you to use IAM policies to do so. For more information, see Managing Access to AWS KMS CMKs (p. 13).

The following example shows the policy statement that gives the AWS account full access to the CMK.

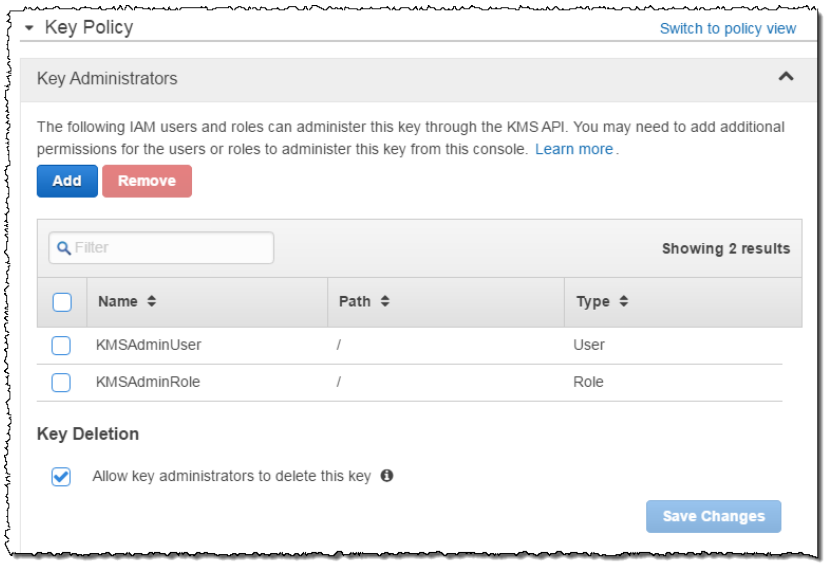
```
{
  "Sid": "Enable IAM User Permissions",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:root"},
  "Action": "kms:*",
  "Resource": "*"
}
```

## Allows Key Administrators to Administer the CMK

The default key policy created by the console allows you to choose IAM users and roles in the account and make them *key administrators*. Key administrators have permissions to manage the CMK, but do not have permissions to use the CMK to encrypt and decrypt data.

> **Caution**
> Even though key administrators do not have permissions to use the CMK to encrypt and decrypt data, they do have permission to modify the key policy, which means they can give themselves these permissions.

You can add IAM users and roles to the list of key administrators when you create the CMK, and you can edit the list with the console's default view for key policies, as shown in the following image. The default view for key policies is on the CMK details page.




When you use the console's default view to modify the list of key administrators, the console modifies the `Principal` element in a particular statement in the key policy document called the key administrators statement. The following example shows the key administrators statement.

```
{
  "Sid": "Allow access for Key Administrators",
  "Effect": "Allow",
  "Principal": {"AWS": [
    "arn:aws:iam::111122223333:user/KMSAdminUser",
    "arn:aws:iam::111122223333:role/KMSAdminRole"
  ]},
  "Action": [
    "kms:Create*",
    "kms:Describe*",
    "kms:Enable*",
    "kms:List*",
    "kms:Put*",
    "kms:Update*",
    "kms:Revoke*",
    "kms:Disable*",
    "kms:Get*",
    "kms:Delete*",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "*"
}
```

The key administrators statement allows the following permissions:

- **kms:Create*** – Allows key administrators to create aliases and grants (p. 43) for the CMK.
- **kms:Describe*** – Allows key administrators to retrieve information about the CMK including its identifiers, creation date, state, and more. This permission is necessary to view the key details page in the AWS Management Console.

- **kms:Enable*** – Allows key administrators to set the CMK's state to enabled and to specify annual rotation of the CMK's key material (p. 50).
- **kms:List*** – Allows key administrators to retrieve lists of the aliases, grants, and key policies for this CMK. This permission is necessary to view the list of CMKs in the AWS Management Console.
- **kms:Put*** – Allows key administrators to modify the key policy for this CMK.
- **kms:Update*** – Allows key administrators to change the target of an alias to this CMK, and to modify the CMK's description.
- **kms:Revoke*** – Allows key administrators to revoke the permissions for this CMK that are allowed by a grant (p. 43).
- **kms:Disable*** – Allows key administrators to set the CMK's state to disabled and to disable annual rotation of the CMK's key material (p. 50).
- **kms:Get*** – Allows key administrators to retrieve the key policy for this CMK and to determine whether the CMK's key material is rotated annually.
- **kms:Delete*** – Allows key administrators to delete an alias that points to this CMK. Note that this permission does not allow key administrators to delete the CMK (p. 63).
- **kms:ScheduleKeyDeletion** – Allows key administrators to delete this CMK (p. 63).
- **kms:CancelKeyDeletion** – Allows key administrators to cancel the pending deletion of this CMK.

Many of these permissions contain the wildcard character (*), which means that if AWS KMS adds new API operations in the future, key administrators will automatically be allowed to perform all new API operations that begin with Create, Describe, Enable, List, Put, Update, Revoke, Disable, Get, or Delete.

## Allows Key Users to Use the CMK

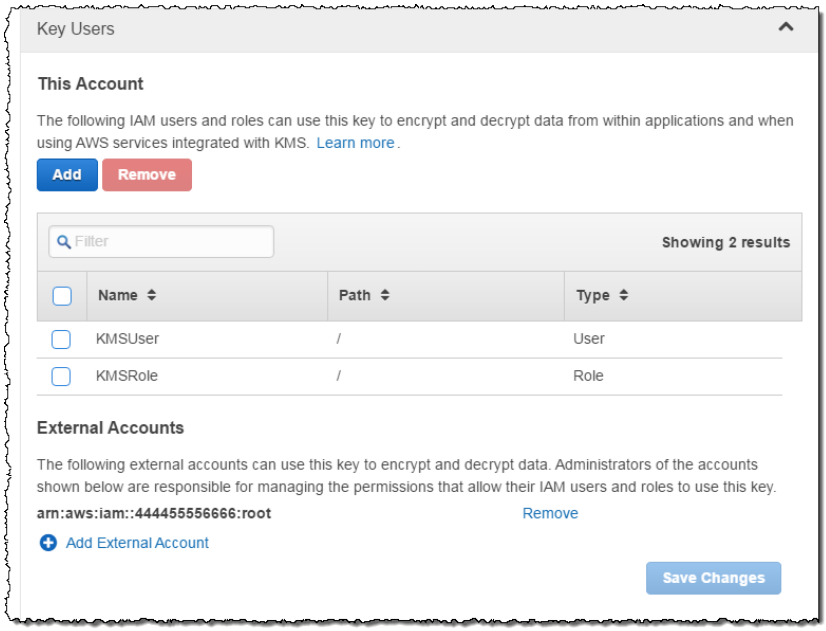
The default key policy created by the console allows you to choose IAM users and roles in the account, and external AWS accounts, and make them *key users*. Key users have permissions to use the CMK directly for encryption and decryption, and to delegate a subset of their own permissions to some of the AWS services that are integrated with AWS KMS (p. 78). Key users can implicitly give these services permissions to use the CMK in specific and limited ways. This implicit delegation is done using grants (p. 43). For example, key users can do the following things:

- Use this CMK with Amazon Elastic Block Store (Amazon EBS) and Amazon Elastic Compute Cloud (Amazon EC2) to attach an encrypted EBS volume to an EC2 instance. The key user implicitly gives Amazon EC2 permission to use the CMK to attach the encrypted volume to the instance. For more information, see How Amazon Elastic Block Store (Amazon EBS) Uses AWS KMS (p. 85).
- Use this CMK with Amazon Redshift to launch an encrypted cluster. The key user implicitly gives Amazon Redshift permission to use the CMK to launch the encrypted cluster and create encrypted snapshots. For more information, see How Amazon Redshift Uses AWS KMS (p. 95).
- Use this CMK with other AWS services integrated with AWS KMS (p. 78), specifically the services that use grants, to create, manage, or use encrypted resources with those services.

The default key policy gives key users permissions to allow these integrated services to use the CMK, but users also need permission to use the integrated services themselves. For details about giving users access to an AWS service that integrates with AWS KMS, consult the documentation for the integrated service.

The default key policy gives key users permissions to use a CMK with *all* of the integrated services that use grants, or none of them. You cannot use the default key policy to allow key users to use a CMK with some of these integrated services but not others. However, you can create a custom key policy to do this. For more information, see the kms:ViaService (p. 42) condition key.

You can add IAM users, IAM roles, and external AWS accounts to the list of key users when you create the CMK, and you can edit the list with the console's default view for key policies, as shown in the following image. The default view for key policies is on the CMK details page.




When you use the console's default view to modify the list of key users, the console modifies the `Principal` element in two statements in the key policy document. These statements are the key users statements. The following examples show the key users statements.

```
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {"AWS": [
    "arn:aws:iam::111122223333:user/KMSUser",
    "arn:aws:iam::111122223333:role/KMSRole",
    "arn:aws:iam::444455556666:root"
  ]},
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
  ],
  "Resource": "*"
}
```

```
{
  "Sid": "Allow attachment of persistent resources",
  "Effect": "Allow",
  "Principal": {"AWS": [
    "arn:aws:iam::111122223333:user/KMSUser",
    "arn:aws:iam::111122223333:role/KMSRole",
    "arn:aws:iam::444455556666:root"
  ]},
  "Action": [
    "kms:CreateGrant",
```

```
    "kms:ListGrants",
    "kms:RevokeGrant"
  ],
  "Resource": "*",
  "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
}
```

The first of these two statements allows key users to use the CMK directly, and includes the following permissions:

- **kms:Encrypt** – Allows key users to successfully request that AWS KMS encrypt data with the CMK.
- **kms:Decrypt** – Allows key users to successfully request that AWS KMS decrypt data with the CMK.
- **kms:ReEncrypt*** – Allows key users to successfully request that AWS KMS re-encrypt decrypt data that was originally encrypted with this CMK, or to use this CMK to re-encrypt previously encrypted data. The ReEncrypt API operation requires access to two CMKs, the original one for decryption and a different one for subsequent encryption. To accomplish this, you can allow the `kms:ReEncrypt*` permission for both CMKs (note the wildcard character "*" in the permission), or you can allow the `kms:ReEncryptFrom` permission on the CMK for decryption and the `kms:ReEncryptTo` permission on the CMK for encryption.
- **kms:GenerateDataKey*** – Allows key users to successfully request data encryption keys to use for client-side encryption. Key users can choose to receive two copies of the data encryption key—one in plaintext form and one that is encrypted with the CMK—or to receive only the encrypted form of the data key.
- **kms:DescribeKey** – Allows key users to retrieve information about this CMK including its identifiers, creation date, state, and more.

The second of these two statements allows key users to use grants to delegate a subset of their own permissions to some of the AWS services that are integrated with AWS KMS (p. 78), specifically the services that use grants. This policy statement uses a condition element to allow these permissions only when the key user is delegating permissions to an integrated AWS service. For more information about using conditions in a key policy, see Using Policy Conditions (p. 35).

# Example Key Policy

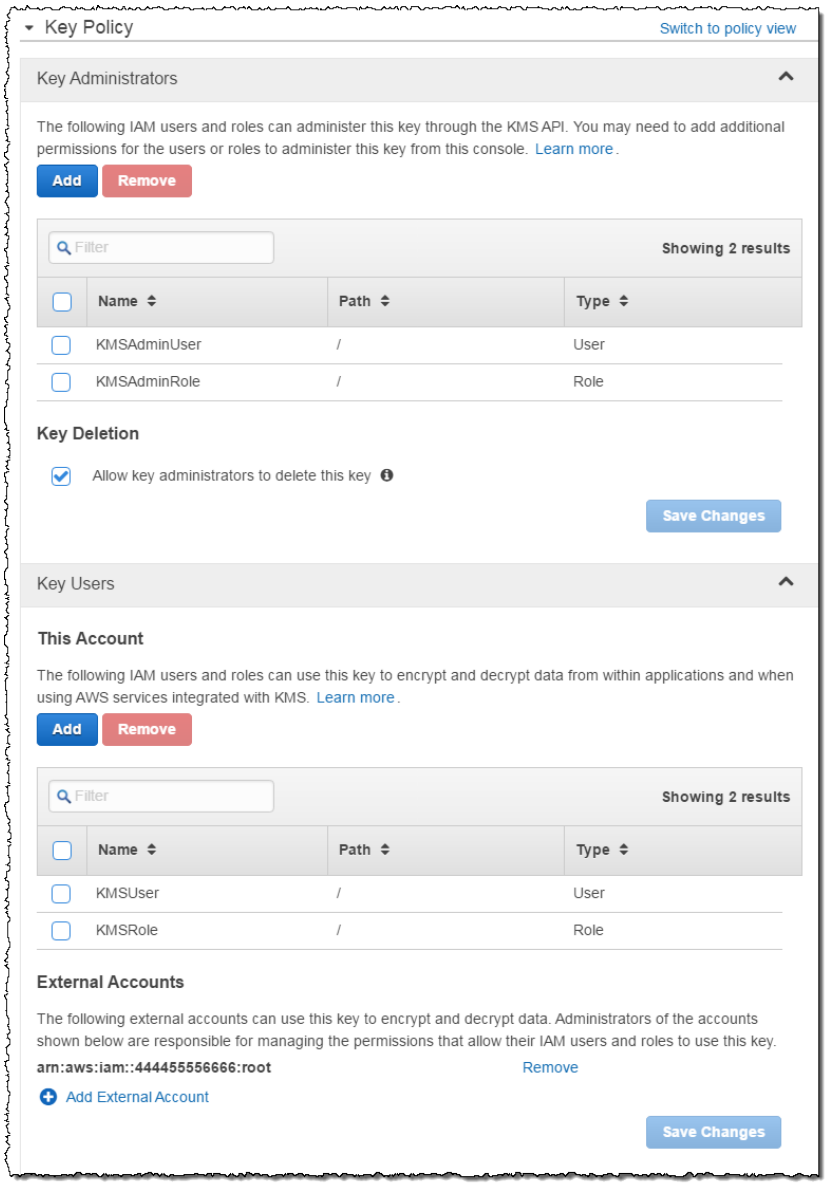
The following example shows a complete key policy. This key policy combines the example policy statements from the preceding default key policy (p. 15) section into a single key policy that accomplishes the following:

- Allows the AWS account (root user) 111122223333 full access to the CMK, and thus enables IAM policies in the account to allow access to the CMK.
- Allows IAM user KMSAdminUser and IAM role KMSAdminRole to administer the CMK.
- Allows IAM user KMSUser, IAM role KMSRole, and AWS account 444455556666 to use the CMK.

```
{
  "Version": "2012-10-17",
  "Id": "key-consolepolicy-2",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:root"},
      "Action": "kms:*",
      "Resource": "*"
    },
```

```
    {
      "Sid": "Allow access for Key Administrators",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:user/KMSAdminUser",
        "arn:aws:iam::111122223333:role/KMSAdminRole"
      ]},
      "Action": [
        "kms:Create*",
        "kms:Describe*",
        "kms:Enable*",
        "kms:List*",
        "kms:Put*",
        "kms:Update*",
        "kms:Revoke*",
        "kms:Disable*",
        "kms:Get*",
        "kms:Delete*",
        "kms:ScheduleKeyDeletion",
        "kms:CancelKeyDeletion"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:user/KMSUser",
        "arn:aws:iam::111122223333:role/KMSRole",
        "arn:aws:iam::444455556666:root"
      ]},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {"AWS": [
        "arn:aws:iam::111122223333:user/KMSUser",
        "arn:aws:iam::111122223333:role/KMSRole",
        "arn:aws:iam::444455556666:root"
      ]},
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": "*",
      "Condition": {"Bool": {"kms:GrantIsForAWSResource": "true"}}
    }
  ]
}
```

The following image shows an example of what this key policy looks like when viewed with the console's default view for key policies.




# Modifying a Key Policy

To change the permissions for a customer master key (CMK) in AWS KMS you modify the CMK's key policy (p. 14). You can add or remove IAM users, IAM roles, and AWS accounts (root users) in the key policy, and change the actions that are allowed or denied for those principals. For more information about the ways to specify principals and permissions in a key policy, see Using Key Policies (p. 14).

You cannot add IAM groups to a key policy, though you can add multiple IAM users. For more information, see Allowing Multiple IAM Users to Access a CMK (p. 25).

When you add external AWS accounts to a key policy, you must also use IAM policies in the external accounts to give permissions to IAM users, groups, or roles in those accounts. For more information, see Allowing External AWS Accounts to Access a CMK (p. 26).

Topics



# How to Modify a Key Policy

You can modify a key policy in three different ways, each of which is explained in the following sections.

Ways to modify a key policy



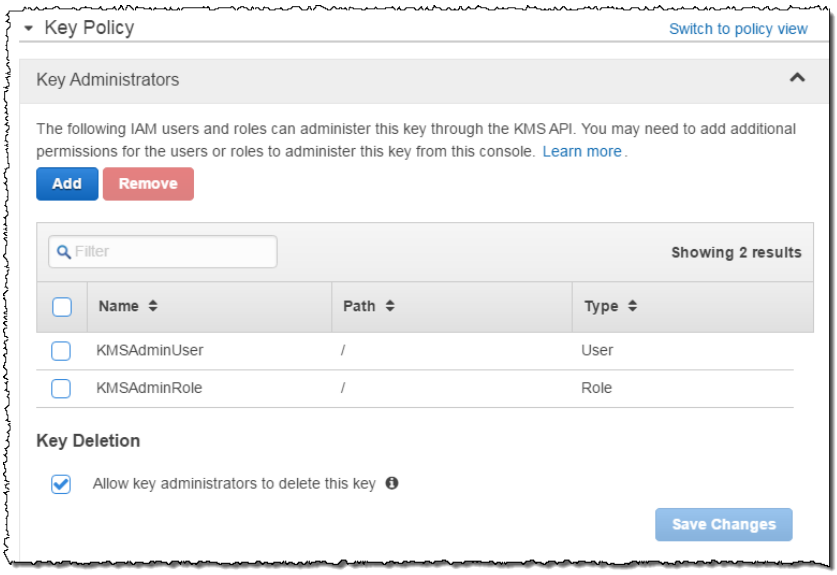
## Using the AWS Management Console's Default View

You can use the console to modify a key policy with a graphical interface called the *default view*.

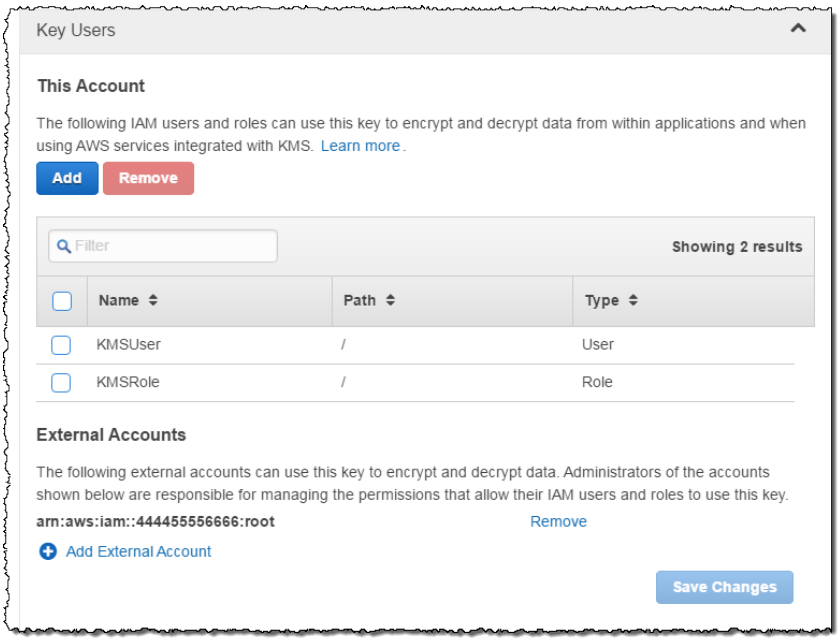
> **Note**
> If the following steps don't match what you see in the console, it means that this key policy was not created by the console, or the key policy has been modified in a way that the console's default view does not support. In that case, follow the steps at Using the AWS Management Console's Policy View (p. 24) or Using the AWS KMS API (p. 25).

### To modify a key policy (console default view)

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Choose the alias of the CMK whose key policy you want to modify.
4. Decide what to modify.
   - To add or remove key administrators (p. 16), and to allow or disallow key administrators to delete the CMK (p. 63), use the controls in the **Key Administrators** area in the **Key Policy** section of the page.




- To add or remove key users (p. 18), and to allow or disallow external AWS accounts to use the CMK, use the controls in the **Key Users** area in the **Key Policy** section of the page.




## Using the AWS Management Console's Policy View

You can use the console to modify a key policy document with the console's *policy view*.

### To modify a key policy document (console policy view)

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Choose the alias of the CMK whose key policy document you want to edit.

4. On the **Key Policy** line, choose **Switch to policy view**.


5. Edit the key policy document, and then choose **Save Changes**.

### Using the AWS KMS API

You can use the AWS KMS API to modify a key policy document. The following steps use the AWS KMS HTTP API. You can perform the same operations with the AWS SDKs or AWS command line tools, which is often easier than using the HTTP API. For the operations and syntax to use for other SDKs and tools, consult the reference documentation for that particular SDK or tool. For sample code that uses the AWS SDK for Java, see Working with Key Policies (p. 143).

**To modify a key policy document (API)**

1. Use GetKeyPolicy to retrieve the existing key policy document, and then save the key policy document to a file.
2. Open the key policy document in your preferred text editor, edit the key policy document, and then save the file.
3. Use PutKeyPolicy to apply the updated key policy document to the CMK.

## Allowing Multiple IAM Users to Access a CMK

IAM groups are not valid principals in a key policy. To allow multiple IAM users to access a CMK, do one of the following:

- Add each IAM user to the key policy. This approach requires that you update the key policy each time the list of authorized users changes.
- Ensure that the key policy includes the statement that enables IAM policies to allow access to the CMK (p. 16). Then create an IAM policy that allows access to the CMK, and then attach that policy to an IAM group that contains the authorized IAM users. Using this approach, you don't need to modify any policies when the list of authorized users changes. Instead, you only need to add or remove those users from the appropriate IAM group.

For more information about how AWS KMS key policies and IAM policies work together, see Understanding Policy Evaluation (p. 44).

# Allowing External AWS Accounts to Access a CMK

You can allow IAM users or roles in one AWS account to access a CMK in another account. For example, suppose that users or roles in account 111122223333 need to use a CMK in account 444455556666. To allow this, you must do two things:

1. Modify the key policy for the CMK in account 444455556666.
2. Add an IAM policy (or modify an existing one) for the users or roles in account 111122223333.

Neither step by itself is sufficient to give access to a CMK across accounts—you must do both.

## Modifying the CMK's Key Policy to Allow External Accounts

To allow IAM users or roles in one AWS account to use a CMK in a different account, you first add the external account (root user) to the CMK's key policy. Note that you don't add the individual IAM users or roles to the key policy, only the external account that owns them.

Decide what permissions you want to give to the external account:

- To add the external account to a key policy as a *key user*, you can use the AWS Management Console's default view for the key policy. For more information, see Using the AWS Management Console's Default View (p. 23).

  You can also modify the key policy document directly using the console's policy view or the AWS KMS API, as described in Using the AWS Management Console's Policy View (p. 24) and Using the AWS KMS API (p. 25).
- To add the external account to a key policy as a *key administrator* or give custom permissions, you must modify the key policy document directly using the console's policy view or the AWS KMS API. For more information, see Using the AWS Management Console's Policy View (p. 24) or Using the AWS KMS API (p. 25).

For an example of the JSON syntax to use when you add an external account to the `Principal` element of a key policy document, refer to the policy statement in the default key policy that allows key users to use the CMK (p. 18).

## Adding or modifying an IAM Policy to Allow Access to a CMK in Another AWS Account

After you add the external account to the CMK's key policy, you then add an IAM policy (or modify an existing one) to the users or roles in the external account. In this scenario, users or roles in account 111122223333 need to use a CMK that is in account 444455556666. To allow this, you create an IAM policy in account 111122223333 that allows access to the CMK in account 444455556666, and then attach the policy to the users or roles in account 111122223333. The following example shows a policy that allows access to a CMK in account 444455556666.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowUseOfCMKInAccount444455556666",
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
```

```
      ],
      "Resource": "arn:aws:kms:us-
west-2:444455556666:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d"
    },
    {
      "Sid": "AllowUseofCMKToCreateEncryptedResourcesInAccount444455556666",
      "Effect": "Allow",
      "Action": "kms:CreateGrant",
      "Resource": "arn:aws:kms:us-
west-2:444455556666:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d",
      "Condition": {
        "Bool": {
          "kms:GrantIsForAWSResource": true
        }
      }
    }
  ]
}
```

This policy allows users and roles in account 111122223333 to use the CMK in account 444455556666 directly for encryption and decryption, and to delegate a subset of their own permissions to some of the AWS services that are integrated with AWS KMS (p. 78), specifically the services that use grants. Note the following details about this policy:

- The policy allows the use of a specific CMK in account 444455556666, identified by the CMK's Amazon Resource Name (ARN) (p. 12) in the `Resource` element of the policy statements. When you give access to CMKs with an IAM policy, always list the specific CMK ARNs in the policy's `Resource` element. Otherwise, you might inadvertently give access to more CMKs than you intend.
- IAM policies do not contain the `Principal` element, which differs from KMS key policies. In IAM policies, the principal is implied by the identity to which the policy is attached.
- The policy gives key users permissions to allow integrated services to use the CMK, but these users also need permission to use the integrated services themselves. For details about giving users access to an AWS service that integrates with AWS KMS, consult the documentation for the integrated service. Also, note that for users or roles in account 111122223333, the CMK in account 444455556666 will not appear in the AWS Management Console to select when creating encrypted resources, even when the users or roles have a policy like this attached. The console does not show CMKs in other accounts.

For more information about working with IAM policies, see Using IAM Policies (p. 27).

# Using IAM Policies with AWS KMS

You can use IAM policies in combination with key policies (p. 14) to control access to your customer master keys (CMKs) in AWS KMS.

> **Note**
> This section discusses using IAM in the context of AWS KMS. It doesn't provide detailed information about the IAM service. For complete IAM documentation, see the IAM User Guide.

Policies attached to IAM identities (that is, users, groups, and roles) are called *identity-based policies* (or *IAM policies*), and policies attached to resources outside of IAM are called *resource-based policies*. In AWS KMS, you must attach resource-based policies to your CMKs. These are called *key policies*. All KMS CMKs have a key policy, and you must use it to control access to a CMK. IAM policies by themselves are not sufficient to allow access to a CMK, though you can use them in combination with a CMK's key policy. To do so, ensure that CMK's key policy includes the policy statement that enables IAM policies (p. 16).

Topics



# Overview of IAM Policies

You can use IAM policies in the following ways:

- **Attach a permissions policy to a user or a group** – You can attach a policy that allows an IAM user or group of users to, for example, create new CMKs.
- **Attach a permissions policy to a role for federation or cross-account permissions** – You can attach an IAM policy to an IAM role to enable identity federation, allow cross-account permissions, or give permissions to applications running on EC2 instances. For more information about the various use cases for IAM roles, see IAM Roles in the *IAM User Guide*.

The following example shows an IAM policy with AWS KMS permissions. This policy allows the IAM identities to which it is attached to retrieve a list of all CMKs and aliases.

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

This policy doesn't specify the `Principal` element because in IAM policies you don't specify the principal who gets the permissions. When you attach this policy to an IAM user, that user is the implicit principal. When you attach this policy to an IAM role, the *assumed role user* gets the permissions.

For a table showing all of the AWS KMS API actions and the resources that they apply to, see the AWS KMS API Permissions Reference (p. 31).

# Permissions Required to Use the AWS KMS Console

To work with the AWS KMS console, users must have a minimum set of permissions that allow them to work with the AWS KMS resources in their AWS account. In addition to these AWS KMS permissions, users must also have permissions to list IAM users and roles. If you create an IAM policy that is more restrictive than the minimum required permissions, the AWS KMS console won't function as intended for users with that IAM policy.

For the minimum permissions required to allow a user read-only access to the AWS KMS console, see Allow a User Read-Only Access to All CMKs through the AWS KMS Console (p. 29).

To allow users to work with the AWS KMS console to create and manage CMKs, attach the **AWSKeyManagementServicePowerUser** managed policy to the user, as described in the following section.

You don't need to allow minimum console permissions for users that are working with the AWS KMS API through the AWS SDKs or command line tools, though you do need to grant these users permission to use the API. For more information, see AWS KMS API Permissions Reference (p. 31).

# AWS Managed (Predefined) Policies for AWS KMS

AWS addresses many common use cases by providing standalone IAM policies that are created and managed by AWS. These are called *AWS managed policies*. AWS managed policies provide the necessary permissions for common use cases so you don't have to investigate which permissions are needed. For more information, see AWS Managed Policies in the *IAM User Guide*.

AWS provides one AWS managed policy for AWS KMS called AWSKeyManagementServicePowerUser. This policy allows the following permissions:

- Allows users to list all CMKs and aliases.
- Allows users to retrieve information about each CMK, including its identifiers, creation date, rotation status, key policy, and more.
- Allows users to create CMKs that they can administer or use. When users create a CMK, they can set permissions in the CMK's key policy (p. 14). This means users can create CMKs with any permissions they want, including allowing themselves to administer or use the CMK. The **AWSKeyManagementServicePowerUser** policy does not allow users to administer or use any other CMKs, only the ones they create.

# Customer Managed Policy Examples

In this section, you can find example IAM policies that allow permissions for various AWS KMS actions.

> **Important**
> Some of the permissions in the following policies are allowed only when the CMK's key policy also allows them. For more information, see AWS KMS API Permissions Reference (p. 31).

Examples

- Allow a User Read-Only Access to All CMKs through the AWS KMS Console (p. 29)
- Allow a User to Encrypt and Decrypt with Any CMK in a Specific AWS Account (p. 30)
- Allow a User to Encrypt and Decrypt with Any CMK in a Specific AWS Account and Region (p. 30)
- Allow a User to Encrypt and Decrypt with Specific CMKs (p. 30)
- Prevent a User from Disabling or Deleting Any CMKs (p. 31)

## Allow a User Read-Only Access to All CMKs through the AWS KMS Console

The following policy allows users read-only access to the AWS KMS console. That is, users can use the console to view all CMKs, but they cannot make changes to any CMKs or create new ones.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:ListKeys",
      "kms:ListAliases",
      "kms:DescribeKey",
```

```
      "kms:ListKeyPolicies",
      "kms:GetKeyPolicy",
      "kms:GetKeyRotationStatus",
      "iam:ListUsers",
      "iam:ListRoles"
    ],
    "Resource": "*"
  }
}
```

## Allow a User to Encrypt and Decrypt with Any CMK in a Specific AWS Account

The following policy allows a user to successfully request that AWS KMS encrypt and decrypt data with any CMK in AWS account 111122223333.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt"
    ],
    "Resource": [
      "arn:aws:kms:*:111122223333:key/*"
    ]
  }
}
```

## Allow a User to Encrypt and Decrypt with Any CMK in a Specific AWS Account and Region

The following policy allows a user to successfully request that AWS KMS encrypt and decrypt data with any CMK in AWS account 111122223333 in the US West (Oregon) region.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt"
    ],
    "Resource": [
      "arn:aws:kms:us-west-2:111122223333:key/*"
    ]
  }
}
```

## Allow a User to Encrypt and Decrypt with Specific CMKs

The following policy allows a user to successfully request that AWS KMS encrypt and decrypt data with the two CMKs specified in the policy's `Resource` element.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt"
    ],
    "Resource": [
      "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"
    ]
  }
}
```

## Prevent a User from Disabling or Deleting Any CMKs

The following policy prevents a user from disabling or deleting any CMKs, even when another IAM policy or a key policy allows these permissions. A policy that explicitly denies permissions overrides all other policies, even those that explicitly allow the same permissions. For more information, see Determining Whether a Request is Allowed or Denied in the *IAM User Guide*.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": [
      "kms:DisableKey",
      "kms:ScheduleKeyDeletion"
    ],
    "Resource": "*"
  }
}
```

# AWS KMS API Permissions: Actions and Resources Reference

When you are setting up access control (p. 11) with key policies (p. 14) and IAM policies (p. 27), you can use the following table as a reference. The first column in the table lists each AWS KMS API operation and the corresponding action (permission) that allows the operation. You specify actions in a policy's `Action` element. The remaining columns provide the following additional information:

- The type of policy you must use to allow permissions to perform the operation. When the key policy is required, you can allow the permissions directly in the key policy, or you can ensure the key policy contains the policy statement that enables IAM policies (p. 16) and then allow the permissions in an IAM policy.
- The resource or resources for which you can allow the operation. You specify resources in a policy's `Resource` element. For key policies, you always specify `"*"` for the resource, which effectively means "this CMK." A key policy applies only to the CMK it is attached to. For IAM policies, you can specify the Amazon Resource Name (ARN) for a specific resource or set of resources.

- The AWS KMS condition keys you can use to control access to the operation. You specify conditions in a policy's `Condition` element. For more information, see AWS KMS Condition Keys (p. 36).

### AWS KMS API Operations and Permissions

| API Operations and Actions (Permissions) | Policy Type | Resources and ARNs (for IAM Policies) | AWS KMS Condition Keys |
|---|---|---|---|
| CancelKeyDeletion<br>`kms:CancelKeyDeletion` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br>`kms:ViaService` |
| CreateAlias<br>`kms:CreateAlias`<br>This operation requires access to two resources, an alias and a CMK, and requires permissions for both. | IAM policy (for the alias) | Alias<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:alias/`*`alias_name`* | None (when controlling access to the alias) |
| | Key policy (for the CMK) | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br>`kms:ViaService` |
| CreateGrant<br>`kms:CreateGrant` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:GrantConstraintType`<br>`kms:GrantIsForAWSResource`<br>`kms:GrantOperations`<br>`kms:CallerAccount`<br>`kms:ViaService` |
| CreateKey<br>`kms:CreateKey` | IAM policy | * | `kms:BypassPolicyLockoutSafetyChe` |
| Decrypt<br>`kms:Decrypt` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:EncryptionContext:`<br>`kms:EncryptionContextKeys`<br>`kms:CallerAccount`<br>`kms:ViaService` |
| DeleteAlias<br>`kms:DeleteAlias`<br>This operation requires access to two resources, an alias and a CMK, and requires permissions for both. | IAM policy (for the alias) | Alias<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:alias/`*`alias_name`* | None (when controlling access to the alias) |
| | Key policy (for the CMK) | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br>`kms:ViaService` |
| DescribeKey<br>`kms:DescribeKey` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br>`kms:ViaService` |

| API Operations and Actions (Permissions) | Policy Type | Resources and ARNs (for IAM Policies) | AWS KMS Condition Keys |
|---|---|---|---|
| DisableKey<br>`kms:DisableKey` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br>`kms:ViaService` |
| DisableKeyRotation<br>`kms:DisableKeyRotation` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br>`kms:ViaService` |
| EnableKey<br>`kms:EnableKey` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br>`kms:ViaService` |
| EnableKeyRotation<br>`kms:EnableKeyRotation` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br>`kms:ViaService` |
| Encrypt<br>`kms:Encrypt` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:EncryptionContext:`<br>`kms:EncryptionContextKeys`<br>`kms:CallerAccount`<br>`kms:ViaService` |
| GenerateDataKey<br>`kms:GenerateDataKey` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:EncryptionContext:`<br>`kms:EncryptionContextKeys`<br>`kms:CallerAccount`<br>`kms:ViaService` |
| GenerateDataKeyWithoutPlaintext<br>`kms:GenerateDataKeyWithoutPlaintext` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:EncryptionContext:`<br>`kms:EncryptionContextKeys`<br>`kms:CallerAccount`<br>`kms:ViaService` |
| GenerateRandom<br>`kms:GenerateRandom` | IAM policy | * | None |
| GetKeyPolicy<br>`kms:GetKeyPolicy` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br>`kms:ViaService` |
| GetKeyRotationStatus<br>`kms:GetKeyRotationStatus` | Key policy | CMK<br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br>`kms:ViaService` |

| API Operations and Actions (Permissions) | Policy Type | Resources and ARNs (for IAM Policies) | AWS KMS Condition Keys |
|---|---|---|---|
| ListAliases<br><br>`kms:ListAliases` | IAM policy | * | None |
| ListGrants<br><br>`kms:ListGrants` | Key policy | CMK<br><br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br><br>`kms:ViaService` |
| ListKeyPolicies<br><br>`kms:ListKeyPolicies` | Key policy | CMK<br><br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br><br>`kms:ViaService` |
| ListKeys<br><br>`kms:ListKeys` | IAM policy | * | None |
| ListRetirableGrants<br><br>`kms:ListRetirableGrants` | IAM policy | * | None |
| PutKeyPolicy<br><br>`kms:PutKeyPolicy` | Key policy | CMK<br><br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br><br>`kms:ViaService` |
| ReEncrypt<br><br>`kms:ReEncryptFrom`<br><br>`kms:ReEncryptTo`<br><br>This operation requires access to two CMKs, one for the decryption (`kms:ReEncryptFrom`) and one for the subsequent encryption (`kms:ReEncryptTo`). Users need permissions for both. | Key policy | CMK<br><br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:EncryptionContext:`*`KeyName`*<br><br>`kms:EncryptionContextKeys`<br><br>`kms:ReEncryptOnSameKey`<br><br>`kms:CallerAccount`<br><br>`kms:ViaService` |
| RetireGrant<br><br>Permission to retire a grant is specified in the grant. You cannot control access to this operation in a policy. For more information, see RetireGrant in the *AWS Key Management Service API Reference*. | Not applicable | Not applicable | Not applicable |

| API Operations and Actions (Permissions) | Policy Type | Resources and ARNs (for IAM Policies) | AWS KMS Condition Keys |
|---|---|---|---|
| RevokeGrant<br><br>`kms:RevokeGrant` | Key policy | CMK<br><br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br><br>`kms:ViaService` |
| ScheduleKeyDeletion<br><br>`kms:ScheduleKeyDeletion` | Key policy | CMK<br><br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br><br>`kms:ViaService` |
| UpdateAlias<br><br>`kms:UpdateAlias`<br><br>This operation requires access to three resources, one alias and two CMKs, the one that the alias currently points to, and the new target CMK specified in the `UpdateAlias` request. Users need permissions for all three resources. | IAM policy (for the alias) | Alias<br><br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:alias/`*`alias_name`* | None (when controlling access to the alias) |
| | Key policy (for the CMKs) | CMK<br><br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br><br>`kms:ViaService` |
| UpdateKeyDescription<br><br>`kms:UpdateKeyDescription` | Key policy | CMK<br><br>`arn:aws:kms:`*`AWS_region`*`:`*`AWS_account_ID`*`:key/`*`CMK_key_ID`* | `kms:CallerAccount`<br><br>`kms:ViaService` |

# Using Policy Conditions with AWS KMS

When you use key policies (p. 14) and IAM policies (p. 27) to control access to your AWS KMS resources, you can use a policy statement's `Condition` element to specify the circumstances in which the statement takes effect. For example, you might want a policy statement to take effect only after a specific date. Or, you might want a policy statement to control access based on whether a specific value exists in the API request.

To specify conditions, you use predefined *condition keys* in a policy statement's `Condition` element. Some condition keys apply generally to AWS, and some are specific to AWS KMS.

Topics

- AWS Condition Keys (p. 35)
- AWS KMS Condition Keys (p. 36)

## AWS Condition Keys

AWS provides a set of predefined condition keys, called *AWS-wide condition keys*, for all AWS services that support IAM for access control. For example, you can use the `aws:MultiFactorAuthPresent` condition key to require multi-factor authentication (MFA). For more

information and a list of the AWS-wide condition keys, see Available Keys for Conditions in the *IAM User Guide*.

## Using the IP Address Condition in Policies with AWS KMS Permissions

If you use AWS KMS to protect your data in an integrated service (p. 78), use caution when specifying the IP address condition operators or the `aws:SourceIp` condition key in the same policy statement that allows or denies access to AWS KMS. For example, a policy like the one shown at Block Requests That Don't Come From an Approved IP Address or Range in the *IAM User Guide* will deny access to the AWS services that make requests to AWS KMS on your behalf. Consider this scenario:

1. You attach a policy like the one shown at Block Requests That Don't Come From an Approved IP Address or Range to an IAM user. This IAM user has other policies attached that allow it to use Amazon EBS, Amazon EC2, and AWS KMS.
2. The user attempts to attach an encrypted EBS volume to an EC2 instance. This action fails with an authorization error despite the user being allowed to use all of the relevant services.

The action at step 2 fails because when the user attempts to attach the encrypted EBS volume to an EC2 instance, Amazon EC2 sends a request to AWS KMS to decrypt the volume's encrypted data key. This request comes from an IP address associated with the Amazon EC2 infrastructure, not the IP address of the originating user. The policy you attached at step 1 contains an explicit deny statement that denies all requests that don't come from the approved IP addresses, which means that Amazon EC2 is denied the access it needs to decrypt the EBS volume's encrypted data key.

# AWS KMS Condition Keys

AWS KMS provides an additional set of predefined condition keys that you can use in key policies and IAM policies. These condition keys are specific to AWS KMS. For example, you can use the `kms:EncryptionContext` condition key to require a particular encryption context (p. 153) when controlling access to a KMS customer master key (CMK).

The following table lists the AWS KMS condition keys, and the following information about each:

- The condition type for the condition key, to indicate which condition operators you can use with the condition key.
- The AWS KMS API operations that include the condition key in the API request, to indicate which permissions (actions) you can use in a policy statement with the condition key.
- Whether you can use the condition key with key policies, IAM policies, or both.

Following this table is more information about each condition key, including example policy statements that demonstrate policy syntax.

| AWS KMS Condition Keys | Condition Type | API Operations | Policy Type |
| --- | --- | --- | --- |
| kms:BypassPolicyLockoutSafetyCheck (p. 37) | Boolean | CreateKey | IAM policies only |
| kms:CallerAccount (p. 38) | String | The `kms:CallerAccount` condition key exists for all AWS KMS operations *except* for | Key policies only |

| AWS KMS Condition Keys | Condition Type | API Operations | Policy Type |
|---|---|---|---|
| | | these: CreateKey, GenerateRandom, ListAliases, ListKeys, ListRetirableGrants, RetireGrant. | |
| kms:EncryptionContext: (p. 38) | String | Encrypt<br>Decrypt<br>GenerateDataKey<br>GenerateDataKeyWithoutPlaintext<br>ReEncrypt | IAM and key policies |
| kms:EncryptionContextKeys (p. 39) | String | Encrypt<br>Decrypt<br>GenerateDataKey<br>GenerateDataKeyWithoutPlaintext<br>ReEncrypt | IAM and key policies |
| kms:GrantConstraintType (p. 40) | String | CreateGrant | IAM and key policies |
| kms:GrantIsForAWSResource (p. 40) | Boolean | CreateGrant | IAM and key policies |
| kms:GrantOperations (p. 41) | String | CreateGrant | IAM and key policies |
| kms:ReEncryptOnSameKey (p. 41) | Boolean | ReEncrypt | IAM and key policies |
| kms:ViaService (p. 42) | String | The `kms:ViaService` condition key exists for all AWS KMS operations *except* for these: CreateKey, GenerateRandom, ListAliases, ListKeys, ListRetirableGrants, RetireGrant. | IAM and key policies |

## kms:BypassPolicyLockoutSafetyCheck

You can use this condition key to control access to the CreateKey operation based on the `BypassPolicyLockoutSafetyCheck` parameter in the request. For example, you can prevent users from bypassing the policy lockout safety check by denying them permission to create CMKs when the request's `BypassPolicyLockoutSafetyCheck` parameter is true, as in the following example policy statement. The following example shows a policy statement in an IAM policy.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
```

```
    "Action": "kms:CreateKey",
    "Resource": "*",
    "Condition": {
      "Bool": {
        "kms:BypassPolicyLockoutSafetyCheck": true
      }
    }
  }
}
```

## kms:CallerAccount

You can use this condition key to allow or deny access to all identities (IAM users and roles) in an AWS account. In key policies, you use the `Principal` element to specify the identities to which the policy statement applies. The syntax for the `Principal` element does not provide a way to specify all identities in an AWS account, but you can achieve this effect by combining this condition key with a `Principal` element that specifies all AWS identities.

For example, the following policy statement demonstrates how to use the `kms:CallerAccount` condition key. This policy statement is in the key policy for the AWS-managed CMK for Amazon EBS. It combines a `Principal` element that specifies all AWS identities with the `kms:CallerAccount` condition key to effectively allow access to all identities in AWS account 111122223333. It contains an additional AWS KMS condition key (`kms:ViaService`) to further limit the permissions by only allowing requests that come through Amazon EBS. For more information, see kms:ViaService (p. 42).

```
{
  "Sid": "Allow access through EBS for all principals in the account that are
 authorized to use EBS",
  "Effect": "Allow",
  "Principal": {"AWS": "*"},
  "Condition": {
    "StringEquals": {
      "kms:CallerAccount": "111122223333",
      "kms:ViaService": "ec2.us-west-2.amazonaws.com"
    }
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:CreateGrant",
    "kms:DescribeKey"
  ],
  "Resource": "*"
}
```

## kms:EncryptionContext:

You can use this condition key prefix to control access based on the encryption context (p. 153) in the AWS KMS API request. Encryption context is a set of key–value pairs that you can include with AWS KMS API operations that perform encryption and decryption (Encrypt, Decrypt, GenerateDataKey, GenerateDataKeyWithoutPlaintext, and ReEncrypt). Use this condition key prefix to check both sides of the encryption context; that is, both the key and the value. To use this condition key prefix, pair it with the encryption context key to form a custom condition key, like this:

`kms:EncryptionContext:`*`encryption_context_key`*

The following example policy statement uses the `kms:EncryptionContext:` condition key prefix to allow access to use a CMK only when the encryption context contains the following key–value pairs:

- `AppName` = `ExampleApp`
- `FilePath` = `/var/opt/secrets/`

To do this, the `kms:EncryptionContext:` condition key prefix is paired with each encryption context key to form custom condition keys (`kms:EncryptionContext:AppName` and `kms:EncryptionContext:FilePath`).

The following example shows a policy statement in a key policy.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:GenerateDataKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:AppName": "ExampleApp",
      "kms:EncryptionContext:FilePath": "/var/opt/secrets/"
    }
  }
}
```

## kms:EncryptionContextKeys

You can use this condition key to control access based on the encryption context (p. 153) in the AWS KMS API request. Encryption context is a set of key–value pairs that you can include with AWS KMS API operations that perform encryption and decryption (Encrypt, Decrypt, GenerateDataKey, GenerateDataKeyWithoutPlaintext, and ReEncrypt). Use this condition key to check only the encryption context keys, not the values.

The following example policy statement uses the `kms:EncryptionContextKeys` condition key with the Null condition operator to allow access to use a CMK only when the request contains encryption context. It does this by allowing access only when the `kms:EncryptionContextKeys` condition key exists (is not null) in the API request. It does not check the keys or values of the encryption context, only that encryption context exists. The following example shows a policy statement in a key policy.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": [
    "kms:Encrypt",
    "kms:GenerateDataKey*"
  ],
  "Resource": "*",
  "Condition": {
    "Null": {
      "kms:EncryptionContextKeys": false
    }
  }
```

```
}
```

## kms:GrantConstraintType

You can use this condition key to control access to the CreateGrant operation based on the type of grant constraint in the request. When you create a grant, you can optionally specify a grant constraint to allow the operations permitted by the grant only when a particular encryption context is present. The grant constraint can be one of two types: `EncryptionContextEquals` or `EncryptionContextSubset`. You can use this condition key to check that the request contains one type or the other. For more information about grant constraints, see GrantConstraints in the *AWS Key Management Service API Reference*.

The following example policy statement uses the `kms:GrantConstraintType` condition key to allow a user to create grants only when the request includes an `EncryptionContextEquals` grant constraint. The following example shows a policy statement in a key policy.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": "kms:CreateGrant",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:GrantConstraintType": "EncryptionContextEquals"
    }
  }
}
```

## kms:GrantIsForAWSResource

You can use this condition key to control access to the CreateGrant operation based on whether the grant is created in the context of an AWS service integrated with AWS KMS (p. 78). This condition key is set to `true` when one of the following integrated services is used to create the grant:

- Amazon Elastic Block Store (Amazon EBS) – For more information, see How Amazon Elastic Block Store (Amazon EBS) Uses AWS KMS (p. 85).
- Amazon Relational Database Service (Amazon RDS) – For more information, see How Amazon Relational Database Service (Amazon RDS) Uses AWS KMS (p. 96).
- Amazon Redshift – For more information, see How Amazon Redshift Uses AWS KMS (p. 95).
- AWS Certificate Manager (ACM) – For more information, see ACM Private Key Security in the *AWS Certificate Manager User Guide*.

For example, the following policy statement uses the `kms:GrantIsForAWSResource` condition key to allow a user to create grants only through one of the integrated services in the preceding list. It does not allow the user to create grants directly. The following example shows a policy statement in a key policy.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
```

```
  "Action": "kms:CreateGrant",
  "Resource": "*",
  "Condition": {
    "Bool": {
      "kms:GrantIsForAWSResource": true
    }
  }
}
```

## kms:GrantOperations

You can use this condition key to control access to the CreateGrant operation based on the grant operations in the request. For example, you can allow a user to create grants that delegate permission to encrypt but not decrypt.

The following example policy statement uses the `kms:GrantOperations` condition key to allow a user to create grants that delegate permission to encrypt and to re-encrypt when this CMK is the destination CMK. The following example shows a policy statement in a key policy.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": "kms:CreateGrant",
  "Resource": "*",
  "Condition": {
    "ForAllValues:StringEquals": {
      "kms:GrantOperations": [
        "Encrypt",
        "ReEncryptTo"
      ]
    }
  }
}
```

## kms:ReEncryptOnSameKey

You can use this condition key to control access to the ReEncrypt operation based on whether the request specifies a destination CMK that is the same one used for the original encryption. For example, the following policy statement uses the `kms:ReEncryptOnSameKey` condition key to allow a user to re-encrypt only when the destination CMK is the same one used for the original encryption. The following example shows a policy statement in a key policy.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": "ReEncrypt*",
  "Resource": "*",
  "Condition": {
    "Bool": {
      "kms:ReEncryptOnSameKey": true
    }
  }
```

```
}
```

## kms:ViaService

You can use this condition key to control access to a CMK based on whether it is used in the context of an AWS service integrated with AWS KMS (p. 78). For example, you can allow a user to use a CMK only through an integrated service. To do this, use the `kms:ViaService` condition key with the service principal name of the service you want to allow access through. You can use the following service principal names with the `kms:ViaService` condition key:

- acm.*AWS_region*.amazonaws.com – For AWS Certificate Manager (ACM)
- codecommit.*AWS_region*.amazonaws.com – For AWS CodeCommit
- dms.*AWS_region*.amazonaws.com – For AWS Database Migration Service (AWS DMS)
- ec2.*AWS_region*.amazonaws.com – For Amazon Elastic Block Store (Amazon EBS)
- importexport.*AWS_region*.amazonaws.com – For AWS Import/Export
- rds.*AWS_region*.amazonaws.com – For Amazon Relational Database Service (Amazon RDS)
- redshift.*AWS_region*.amazonaws.com – For Amazon Redshift
- s3.*AWS_region*.amazonaws.com – For Amazon Simple Storage Service (Amazon S3)
- ses.*AWS_region*.amazonaws.com – For Amazon Simple Email Service (Amazon SES)
- workmail.*AWS_region*.amazonaws.com – For Amazon WorkMail
- workspaces.*AWS_region*.amazonaws.com – For Amazon WorkSpaces

The following example policy statement uses the `kms:ViaService` condition key to allow access to a set of AWS KMS actions only when the CMK is used through Amazon EBS or Amazon RDS in the US West (Oregon) region. The following example shows a policy statement in a key policy.

> **Important**
> When you control access to a CMK with the `kms:ViaService` condition key, ensure the policy statement allows the permissions necessary for the integrated service to use the CMK. For more information, see How AWS Services use AWS KMS (p. 78). Note also that users need permission to use the integrated services themselves. For details about giving users access to an AWS service that integrates with AWS KMS, consult the documentation for the integrated service.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:CreateGrant",
    "kms:ListGrants",
    "kms:DescribeKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": [
        "ec2.us-west-2.amazonaws.com",
        "rds.us-west-2.amazonaws.com"
      ]
```

```
    }
  }
}
```

# Using Grants

AWS KMS supports two resource-based access control mechanisms: *key policies* and *grants*. Grants enable you to programmatically delegate the use of KMS customer master keys (CMKs) to other AWS principals. You can also use key policies to allow other principals to access a CMK, but key policies work best for relatively static assignments of permissions.

Key policy changes follow the same permissions model used for policy editing elsewhere in AWS. That is, users either have permission to change the key policy or they do not. Users with the `kms:PutKeyPolicy` permission for a CMK can completely replace the key policy for a CMK with a different key policy of their choice. To enable more granular permissions management, use grants.

You call the CreateGrant API operation to create a grant. You pass the identifier of the CMK for which the grant is to be created, the grantee principal being given permission to use the CMK, and a list of operations to be allowed. The `CreateGrant` operation returns a grant ID that you can use to identify the grant in subsequent operations. To further customize the grant permissions, you can also pass optional parameters that define grant constraints. After the grant has been created, the principal identified in the grant can execute the permitted operations, subject to the defined constraints, for as long as the grant is active. Grants can be explicitly revoked by a user who has the `kms:RevokeGrant` permission on the CMK, or they can be retired by the principal designated as the retiring principal for the grant.

There are two supported grant constraints: `EncryptionContextEquals` and `EncryptionContextSubset`. `EncryptionContextEquals` specifies that the grant applies only when the exact specified encryption context is present in the request. `EncryptionContextSubset` specifies that the grant applies as long as all the entries in the `EncryptionContextSubset` constraint are matched by the request. In this case, the request can contain additional encryption context entries. For example, a grant that allows the encrypt and decrypt operations with an `EncryptionContextSubset` constraint of `{"Department":"Finance","Classification":"Public"}` allows encryption and decryption when the request contains an encryption context of either `{"Department":"Finance","Classification":"Public"}` or `{"Department":"Finance","Classification":"Public","Customer":"12345"}`, but not when the request contains an encryption context of `{"Department":"Finance"}`.

When the grant includes `CreateGrant` as an allowed operation, the grant only allows creation of equally or more restrictive grants. That is, the grant operations passed with a subsequent `CreateGrant` API request can include any subset of the currently-allowed grant operations, and the grant constraints can be the same or more restrictive (fields can be added to an `EncryptionContextSubset` constraint, or an `EncryptionContextSubset` constraint can be turned into an `EncryptionContextEquals` constraint).

For Java code samples that demonstrate how to work with grants, see Working with Grants (p. 145).

# Determining Access to an AWS KMS Customer Master Key

To determine the full extent of who or what currently has access to a customer master key (CMK) in AWS KMS, you must examine the CMK's key policy, all grants (p. 43) that apply to the CMK, and

potentially all AWS Identity and Access Management (IAM) policies. You might do this to determine the scope of potential usage of a CMK, or to help you meet compliance or auditing requirements. The following topics can help you generate a complete list of the AWS principals (identities) that currently have access to a CMK.

Topics



# Understanding Policy Evaluation

When authorizing access to a CMK, AWS KMS evaluates the key policy attached to the CMK, all grants that apply to the CMK, and all IAM policies attached to the IAM user or role making the request. In many cases, AWS KMS must evaluate the CMK's key policy and IAM policies together to determine whether access to the CMK is allowed or denied. To do this, AWS KMS uses a process similar to the one described at Determining Whether a Request is Allowed or Denied in the *IAM User Guide*. Remember, though, that IAM policies by themselves are not sufficient to allow access to a KMS CMK. The CMK's key policy must also allow access.

For example, assume that you have two CMKs and three users, all in the same AWS account. The CMKs and users have the following policies:

- CMK1's key policy allows access to the AWS account (root user) and thereby enables IAM policies to allow access to CMK1 (p. 16).
- CMK2's key policy allows access to Alice and Charlie.
- Alice has no IAM policy.
- Bob's IAM policy allows all AWS KMS actions for all CMKs.
- Charlie's IAM policy denies all AWS KMS actions for all CMKs.

Alice cannot access CMK1 because CMK1's key policy does not explicitly allow her access, and she has no IAM policy that allows access. Alice can access CMK2 because the CMK's key policy explicitly allows her access.

Bob can access CMK1 because CMK1's key policy enables IAM policies to allow access, and Bob has an IAM policy that allows access. Bob cannot access CMK2 because the key policy for CMK2 does not allow access to the account, so Bob's IAM policy does not by itself allow access to CMK2.

Charlie cannot access CMK1 or CMK2 because all AWS KMS actions are denied in his IAM policy. The explicit deny in Charlie's IAM policy overrides the explicit allow in CMK2's key policy.

# Examining the Key Policy

You can examine the key policy in two ways:

- If the CMK was created in the AWS Management Console, you can use the console's *default view* on the key details page to view the principals listed in the key policy. If you can view the key policy in this way, it means the key policy allows access with IAM policies (p. 16). Be sure to examine IAM policies (p. 47) to determine the complete list of principals that can access the CMK.
- You can use the GetKeyPolicy operation in the AWS KMS API to retrieve a copy of the key policy document, and then examine the document. You can also view the policy document in the AWS Management Console.

Ways to examine the key policy

- Examining the Key Policy in the AWS Management Console (p. 45)
- Examining the Key Policy Document (p. 45)

## Examining the Key Policy in the AWS Management Console

**To view a customer master key (CMK)'s permissions on the key details page (console)**

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. In the list of keys, choose the alias of the key that you want to examine.
4. In the **Key Policy** section of the key details page, find the list of IAM users and roles in the **Key Administrators** section, and another list in the **Key Users** section. The listed users, roles, and AWS accounts all have access to manage or use this CMK.

   **Important**
   The IAM users, roles, and AWS accounts listed here are the ones that have been explicitly granted access in the key policy. If you use IAM policies to allow access to CMKs, other IAM users and roles might have access to this CMK, even if they are not listed here. Take care to examine all IAM policies (p. 47) in this account to determine if they allow access to this CMK.
5. (Optional) To view the key policy document, choose **Switch to policy view**.

## Examining the Key Policy Document

You can view the key policy document in a couple of ways:

- Use the key details page of the AWS Management Console (see the preceding section for instructions).
- Use the GetKeyPolicy operation in the AWS KMS API to retrieve a copy of the key policy document.

Examine the key policy document and take note of all principals specified in each policy statement's `Principal` element. The IAM users, IAM roles, and AWS accounts in the `Principal` elements are those that have access to this CMK.

The following examples use the policy statements found in the default key policy (p. 15) to demonstrate how to do this.

### Example Policy Statement 1

```
{
  "Sid": "Enable IAM User Permissions",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:root"},
  "Action": "kms:*",
  "Resource": "*"
}
```

In the preceding policy statement, `arn:aws:iam::111122223333:root` refers to the AWS account 111122223333. By default, a policy statement like this one is present in the key policy document when you create a new CMK with the console, and when you create a new CMK programmatically but do not provide a key policy.

> **Note**
> A key policy document with a statement that allows access to the AWS account (root user) enables IAM policies in the account to allow access to the CMK (p. 16). This means that IAM users and roles in the account might have access to the CMK even if they are not explicitly listed as principals in the key policy document. Take care to examine all IAM policies (p. 47) in all AWS accounts listed as principals to determine whether they allow access to this CMK.

### Example Policy Statement 2

```
{
  "Sid": "Allow access for Key Administrators",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:user/KMSKeyAdmin"},
  "Action": [
    "kms:Describe*",
    "kms:Put*",
    "kms:Create*",
    "kms:Update*",
    "kms:Enable*",
    "kms:Revoke*",
    "kms:List*",
    "kms:Disable*",
    "kms:Get*",
    "kms:Delete*",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "*"
}
```

In the preceding policy statement, `arn:aws:iam::111122223333:user/KMSKeyAdmin` refers to the IAM user named KMSKeyAdmin in AWS account 111122223333. This user is allowed to perform the actions listed in the policy statement, which are the administrative actions for managing a CMK.

**Example Policy Statement 3**

```
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:role/EncryptionApp"},
  "Action": [
    "kms:DescribeKey",
    "kms:GenerateDataKey*",
    "kms:Encrypt",
    "kms:ReEncrypt*",
    "kms:Decrypt"
  ],
  "Resource": "*"
}
```

In the preceding policy statement, `arn:aws:iam::111122223333:role/EncryptionApp` refers to the IAM role named EncryptionApp in AWS account 111122223333. Principals that can assume this role are allowed to perform the actions listed in the policy statement, which are the cryptographic actions for encrypting and decrypting data with a CMK.

**Example Policy Statement 4**

```
{
  "Sid": "Allow attachment of persistent resources",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:role/EncryptionApp"},
  "Action": [
    "kms:ListGrants",
    "kms:CreateGrant",
    "kms:RevokeGrant"
  ],
  "Resource": "*",
  "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
}
```

In the preceding policy statement, `arn:aws:iam::111122223333:role/EncryptionApp` refers to the IAM role named EncryptionApp in AWS account 111122223333. Principals that can assume this role are allowed to perform the actions listed in the policy statement. These actions, when combined with the actions allowed in **Example policy statement 3**, are those necessary to delegate use of the CMK to most AWS services that integrate with AWS KMS (p. 78), specifically the services that use grants (p. 43). The `Condition` element ensures that the delegation is allowed only when the delegate is an AWS service that integrates with AWS KMS and uses grants for authorization.

To learn all the different ways you can specify a principal in a key policy document, see Specifying a Principal in the *IAM User Guide*.

To learn more about AWS KMS key policies, see Using Key Policies in AWS KMS (p. 14).

# Examining IAM Policies

In addition to the key policy and grants, you can also use IAM policies in combination with a CMK's key policy to allow access to a CMK. For more information about how IAM policies and key policies work together, see Understanding Policy Evaluation (p. 44).

To determine which principals currently have access to a CMK through IAM policies, you can use the browser-based IAM Policy Simulator tool, or you can make requests to the IAM API.

Ways to examine IAM policies

- Examining IAM Policies with the IAM Policy Simulator (p. 48)
- Examining IAM Policies with the IAM API (p. 48)

## Examining IAM Policies with the IAM Policy Simulator

The IAM Policy Simulator can help you learn which principals have access to a KMS CMK through an IAM policy.

### To use the IAM Policy Simulator to determine access to a KMS CMK

1. Sign in to the AWS Management Console and then open the IAM Policy Simulator at https://policysim.aws.amazon.com/.
2. In the **Users, Groups, and Roles** pane, choose the user, group, or role whose policies you want to simulate.
3. (Optional) Clear the check box next to any policies that you want to omit from the simulation. To simulate all policies, leave all policies selected.
4. In the **Policy Simulator** pane, do the following:
   a. For **Select service**, choose **Key Management Service**.
   b. To simulate specific AWS KMS actions, for **Select actions**, choose the actions to simulate. To simulate all AWS KMS actions, choose **Select All**.
5. (Optional) The Policy Simulator simulates access to all KMS CMKs by default. To simulate access to a specific KMS CMK, select **Simulation Settings** and then type the Amazon Resource Name (ARN) of the KMS CMK to simulate.
6. Select **Run Simulation**.

You can view the results of the simulation in the **Results** section. Repeat steps 2 through 6 for every IAM user, group, and role in the AWS account.

## Examining IAM Policies with the IAM API

You can use the IAM API to examine IAM policies programmatically. The following steps provide a general overview of how to do this:

1. For each AWS account listed as a principal in the CMK's key policy (that is, each *root account* listed in this format: `"Principal": {"AWS": "arn:aws:iam::111122223333:root"}`), use the ListUsers and ListRoles operations in the IAM API to retrieve a list of every IAM user and role in the account.
2. For each IAM user and role in the list, use the SimulatePrincipalPolicy operation in the IAM API, passing in the following parameters:
   - For `PolicySourceArn`, specify the Amazon Resource Name (ARN) of a user or role from your list. You can specify only one `PolicySourceArn` for each `SimulatePrincipalPolicy` API request, so you must call this API multiple times, once for each IAM user and role in your list.
   - For the `ActionNames` list, specify every AWS KMS API action to simulate. To simulate all AWS KMS API actions, use `kms:*`. To test individual AWS KMS API actions, precede each API action with "`kms:`", for example "`kms:ListKeys`". For a complete list of all AWS KMS API actions, see Actions in the *AWS Key Management Service API Reference*.
   - (Optional) To determine whether the IAM users or roles have access to specific KMS CMKs, use the `ResourceArns` parameter to specify a list of the Amazon Resource Names (ARNs) of the CMKs. To determine whether the IAM users or roles have access to any CMK, do not use the `ResourceArns` parameter.

IAM responds to each `SimulatePrincipalPolicy` API request with an evaluation decision: `allowed`, `explicitDeny`, or `implicitDeny`. For each response that contains an evaluation decision of `allowed`, the response will also contain the name of the specific AWS KMS API action that is allowed and, if applicable, the ARN of the CMK that was used in the evaluation.

# Examining Grants

Grants are advanced mechanisms for specifying permissions that you or an AWS service integrated with AWS KMS can use to specify how and when a CMK can be used. Grants are attached to a CMK, and each grant contains the principal who receives permission to use the CMK and a list of operations that are allowed. Grants are an alternative to the key policy, and are useful for specific use cases. For more information, see Using Grants (p. 43).

To retrieve a list of grants attached to a CMK, use the AWS KMS ListGrants API (or list-grants AWS CLI command). You can examine the grants for a CMK to determine who or what currently has access to use the CMK via those grants. For example, the following is a JSON representation of a grant that was obtained from the list-grants command in the AWS CLI.

```
{"Grants": [{
  "Operations": ["Decrypt"],
  "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
  "Name": "0d8aa621-43ef-4657-b29c-3752c41dc132",
  "RetiringPrincipal": "arn:aws:iam::123456789012:root",
  "GranteePrincipal": "arn:aws:sts::111122223333:assumed-role/aws:ec2-
infrastructure/i-5d476fab",
  "GrantId":
 "dc716f53c93acacf291b1540de3e5a232b76256c83b2ecb22cdefa26576a2d3e",
  "IssuingAccount": "arn:aws:iam::111122223333:root",
  "CreationDate": 1.444151834E9,
  "Constraints": {"EncryptionContextSubset": {"aws:ebs:id": "vol-5cccfb4e"}}
}]}
```

To find out who or what has access to use the CMK, look for the `"GranteePrincipal"` element. In the preceding example, the grantee principal is an assumed role user associated with the EC2 instance i-5d476fab, which the EC2 infrastructure uses to attach the encrypted EBS volume vol-5cccfb4e to the instance. In this case, the EC2 infrastructure role has permission to use the CMK because you previously created an encrypted EBS volume protected by this CMK, and then attached the volume to an EC2 instance.

The following is another example of a JSON representation of a grant that was obtained from the list-grants command in the AWS CLI. In the following example, the grantee principal is another AWS account.

```
{"Grants": [{
  "Operations": ["Encrypt"],
  "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
  "Name": "",
  "GranteePrincipal": "arn:aws:iam::444455556666:root",
  "GrantId":
 "f271e8328717f8bde5d03f4981f06a6b3fc18bcae2da12ac38bd9186e7925d11",
  "IssuingAccount": "arn:aws:iam::111122223333:root",
  "CreationDate": 1.444151269E9
}]}
```

# Rotating Keys

When you request AWS KMS to create a customer master key (CMK), the service creates a key ID for the CMK and key material referred to as a backing key that is tied to the key ID of the CMK. If you choose to enable key rotation for a given CMK, AWS KMS will create a new version of the backing key for each rotation. It is the backing key that is used to perform cryptographic operations such as encryption and decryption. When you choose a CMK to encrypt new data, AWS KMS automatically uses the latest version of the backing key to perform the encryption. When you want to decrypt data, AWS KMS automatically determines the correct version of the backing key to use. From your point of view, your CMK is simply a logical resource that does not change regardless of whether or of how many times the underlying backing keys have been rotated.

Automated key rotation currently retains all prior backing keys so that decryption of encrypted data can take place transparently. If you want to prevent decryption of old ciphertexts, you can create a new CMK and change your alias to point to the new key. You can then control when you choose to disable the old key. Disabling a CMK prevents the backing keys tied to it from being used to encrypt or to decrypt.

For more detailed information about backing keys and rotation, see the KMS Cryptographic Details whitepaper.

# Importing Key Material in AWS Key Management Service (AWS KMS)

A customer master key (CMK) is a logical representation of a master key in AWS KMS. In addition to the master key's identifiers and other metadata including its creation date, description, and key state (p. 75), a CMK contains the *key material* used to encrypt and decrypt data. When you create a CMK (p. 6), by default AWS KMS generates the key material for that CMK. But you can choose to create a CMK without key material and then import your own key material into that CMK.

When you use imported key material, you remain responsible for the key material while allowing AWS KMS to use a copy of it. You might choose to do this for one or more of the following reasons:

- To prove that you generated the key material using a source of randomness that meets your requirements.
- To use key material from your own infrastructure with AWS services, and to use AWS KMS to manage the lifecycle of that key material within AWS.
- To gain the ability to set an expiration time for the key material in AWS and to manually delete it (p. 60), but to also make it available again in the future. In contrast, scheduling key deletion (p. 63) requires a waiting period of 7 to 30 days, after which you cannot recover the deleted CMK.
- To own the original copy of the key material, and to keep it outside of AWS for additional durability and disaster recovery during the complete lifecycle of the key material.

For information about important differences between CMKs with imported key material and those with key material generated by AWS KMS, see Considerations for Imported Key Material (p. 52).

The key material you import must be a 256-bit symmetric encryption key.

### Topics

# How To Import Key Material

The following overview describes the process to import your key material into AWS KMS. For more details about each step in the process, see the corresponding topic.

1. Create a CMK with no key material (p. 53) – To get started with importing key material, first create a CMK whose *origin* is EXTERNAL. This indicates that the key material was generated outside of AWS KMS and prevents AWS KMS from generating key material for the CMK. In a later step you will import your own key material into this CMK.

2. Download the public key and import token (p. 55) – After completing step 1, download a public key and an import token. These items protect the import of your key material to AWS KMS.

3. Encrypt the key material (p. 58) – Use the public key that you downloaded in step 2 to encrypt the key material that you created on your own system.

4. Import the key material (p. 59) – Upload the encrypted key material that you created in step 3 and the import token that you downloaded in step 2.

# Considerations for Imported Key Material

Before you decide to import key material into AWS KMS, you should understand the following characteristics of imported key material.

Availability and durability

You remain responsible for the key material's overall availability and durability. AWS KMS is designed to keep imported key material highly available, but the service does not maintain the durability of imported key material at the same level as key material generated on your behalf. This difference is meaningful in the following cases:

- When you set an expiration time for your imported key material, AWS KMS deletes the key material after it expires. AWS KMS does not delete the CMK or its metadata. You cannot set an expiration time for key material generated by AWS KMS.
- When you manually delete imported key material (p. 60), AWS KMS deletes the key material but does not delete the CMK or its metadata. In contrast, scheduling key deletion (p. 63) requires a waiting period of 7 to 30 days, after which AWS KMS deletes the key material and all of the CMK's metadata.
- In the unlikely event of certain regionwide failures that affect the service (such as a total loss of power), AWS KMS cannot automatically restore your imported key material. However, AWS KMS can restore the CMK and its metadata.

To make your key material available again after any of these events, you must retain a copy of the key material in a system that you control so that you can reimport it into the original CMK.

Secure key generation

You are responsible for generating the key material using a source of randomness that meets your security requirements.

One key per CMK

When you import key material into a CMK, the CMK is permanently associated with that key material. You cannot import different key material into that CMK, and you cannot enable automatic key rotation (p. 50) for a CMK with imported key material. However, you can manually rotate a CMK with imported key material. To do so, create a new CMK (p. 53) and then import the new key material into that CMK. Then change the CMK identifier in your applications or AWS service configurations to the key ID for the new CMK. Also, if you have a KMS alias that points to the old CMK, you can update it to point to the new one to complete the rotation process.

Ciphertexts are not portable between CMKs

When you encrypt data under a KMS CMK, the ciphertext cannot be decrypted with any other CMK. This applies to all KMS CMKs, and remains true even when you import the same key material into a different CMK.

# Importing Key Material Step 1: Create an AWS KMS Customer Master Key (CMK) With No Key Material

By default, AWS KMS creates key material for you when you create a customer master key (CMK). To instead import your own key material, start by creating a CMK with no key material. You distinguish between these two types of CMKs by the CMK's *origin*. When AWS KMS creates the key material for you, the CMK's origin is `AWS_KMS`. When you create a CMK with no key material, the CMK's origin is `EXTERNAL`, which indicates that the key material was generated outside of AWS KMS.

A CMK with no key material is in the *pending import* state and is not available for use. To use it, you must import key material as explained later. When you import key material, the CMK's key state changes to *enabled*. For more information about key state, see How Key State Affects Use of a Customer Master Key (p. 75).

To create a CMK with no key material, you can use the AWS Management Console or the AWS KMS API. You can use the API directly by making HTTP requests, or through one of the AWS SDKs or command line tools.

Topics

- Create a CMK with No Key Material (AWS Management Console) (p. 53)
- Create a CMK with No Key Material (AWS KMS API) (p. 55)

## Create a CMK with No Key Material (AWS Management Console)

You can use the AWS Management Console to create a CMK with no key material. Before you do this, you can configure the console to show additional columns in the list of KMS CMKs to more easily distinguish your CMKs with imported key material.

### To show additional columns in the list of KMS CMKs

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Choose the settings button (  ) in the upper-right corner of the page.

4. Select the check boxes for **Expiration Date** and **Origin**, and then choose **Close**.

### To create a CMK with no key material (console)

You need to create a CMK for the imported key material only once. To reimport the same key material into an existing CMK, see Step 2: Download the Public Key and Import Token (p. 55).

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Choose **Create key**.
4. Type an alias and (optionally) a description for the CMK.
5. Choose **Advanced Options**.
6. For **Key Material Origin**, choose **External**. Then select the check box next to **I understand the security, availability, and durability implications of using an imported key** to indicate that you understand the implications of using imported key material. To read about these implications, choose the **security, availability, and durability implications** link.

   Choose **Next Step**.
7. Select which IAM users and roles can administer the CMK. For more information, see Allows Key Administrators to Administer the CMK (p. 16).

   **Note**
   All IAM users and roles with IAM policies that specify the appropriate permissions can also administer the CMK.

   Choose **Next Step**.
8. Select which IAM users and roles can use the CMK to encrypt and decrypt data. For more information, see Allows Key Users to Use the CMK (p. 18).

   **Note**
   All IAM users and roles with IAM policies that specify the appropriate permissions can also use the CMK.
9. (Optional) At the bottom of the page, you can give permissions to other AWS accounts to use the CMK to encrypt and decrypt data. Choose **Add an External Account** and then type the AWS account ID of the account to give permissions to. Repeat as necessary to add more than one external account.

   **Note**
   Administrators of the external accounts must also allow access to the CMK by creating IAM policies for their users. For more information, see Allowing External AWS Accounts to Access a CMK (p. 26).

   Choose **Next Step**.
10. Choose **Finish** to create the CMK.

    After you complete this step, the console displays the **Import key material** wizard. To continue the process now, see Download the Public Key and Import Token (AWS Management Console) (p. 56).

    Otherwise, choose **Skip and do this later**. To find your CMK in the list, you can show additional columns as described previously, or you can look in the **Status** column for **Pending Import**. Your new CMK remains in the **Pending Import** state until you import key material as described in the following steps.

Proceed to Step 2: Download the Public Key and Import Token (p. 55).

## Create a CMK with No Key Material (AWS KMS API)

To use the AWS KMS API to create a CMK with no key material, send a CreateKey request with the `Origin` parameter set to `EXTERNAL`. The following example shows how to do this with the AWS Command Line Interface (AWS CLI).

```
$ aws kms create-key --origin EXTERNAL
```

When the command is successful, you see output similar to the following. The CMK's `Origin` is `EXTERNAL` and its `KeyState` is `PendingImport`.

```
{
    "KeyMetadata": {
        "Origin": "EXTERNAL",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Description": "",
        "Enabled": false,
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "PendingImport",
        "CreationDate": 1470811233.761,
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "AWSAccountId": "111122223333"
    }
}
```

Copy the CMK's key ID from your command output to use in later steps, and then proceed to Step 2: Download the Public Key and Import Token (p. 55).

# Importing Key Material Step 2: Download the Public Key and Import Token

After you create a customer master key (CMK) with no key material (p. 53), you download a public key and import token for that CMK. You need these items to import your key material, and you can download both items in one step using the AWS Management Console or the AWS KMS API.

You also download these items when you want to reimport key material into a CMK. You might do this to change the expiration time for the key material, or to restore a CMK after the key material has expired or been deleted.

Use of the public key

When you import key material, you don't upload the raw key material to AWS KMS. You must first encrypt the key material with the public key that you download in this step, and then upload the encrypted key material to AWS KMS. When AWS KMS receives your encrypted key material, it uses the corresponding private key to decrypt it. The public key you receive from AWS KMS is a 2048-bit RSA public key and is always unique to your AWS account.

Use of the import token

The import token contains metadata to ensure that your key material is imported correctly. When you upload your encrypted key material to AWS KMS, you must upload the same import token that you download in this step.

Plan ahead

Before you download a public key and import token, you must determine how you will encrypt your key material. Typically, you choose an option based on the capabilities of the hardware security module (HSM) or key management system that protects your key material. You must use the RSA PKCS #1 encryption scheme with one of three padding options, represented by the following choices. These choices are listed in order of AWS preference. The technical details of the schemes represented by these choices are explained in section 7 of the PKCS #1 Version 2.1 standard.

- **RSAES_OAEP_SHA_256** – The RSA encryption algorithm with Optimal Asymmetric Encryption Padding (OAEP) with the SHA-256 hash function.
- **RSAES_OAEP_SHA_1** – The RSA encryption algorithm with Optimal Asymmetric Encryption Padding (OAEP) with the SHA-1 hash function.
- **RSAES_PKCS1_V1_5** – The RSA encryption algorithm with the padding format defined in PKCS #1 Version 1.5.

If your HSM or key management system supports it, we recommending using RSAES_OAEP_SHA_256 to encrypt your key material. If that option is not available, you should use RSAES_OAEP_SHA_1. If neither of the OAEP options are available, you must use RSAES_PKCS1_V1_5. For information about how to encrypt your key material, see the documentation for the hardware security module or key management system that protects your key material.

The public key and import token are valid for 24 hours. If you don't use them to import key material within 24 hours of downloading them, you must download new ones.

To download the public key and import token, you can use the AWS Management Console or the AWS KMS API. You can use the API directly by making HTTP requests, or through one of the AWS SDKs or command line tools.

Topics



# Download the Public Key and Import Token (AWS Management Console)

You can use the AWS Management Console to download the public key and import token. If you just completed the steps to create a CMK with no key material (p. 53), skip to Step 5 (p. 56).

### To download the public key and import token (console)

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Choose the alias of the CMK for which you are downloading the public key and import token.

   **Tip**
   The CMK's **Origin** must be **External**. If you don't see the **Origin** column, choose the settings button (  ) in the upper-right corner of the page. Select the check box next to **Origin**, and then choose **Close**.
4. In the **Key Material** section of the page, choose **Download wrapping key and import token**.
5. For **Select wrapping algorithm**, choose the option that you will use to encrypt your key material. For more information about the options, see the preceding section.

6. Choose **Download wrapping key and import token**, and then save the file.
7. Decompress the `.zip` file that you saved in the previous step (`ImportParameters.zip`).

   The folder contains the following files:

   - The wrapping key (public key), in a file named wrappingKey_*CMK_key_ID_timestamp* (for example, `wrappingKey_f44c4e20-f83c-48f4-adc6-a1ef38829760_0809092909`). This is a 2048-bit RSA public key.
   - The import token, in a file named importToken_*CMK_key_ID_timestamp* (for example, `importToken_f44c4e20-f83c-48f4-adc6-a1ef38829760_0809092909`).
   - A text file named README_*CMK_key_ID_timestamp*.txt (for example, `README_f44c4e20-f83c-48f4-adc6-a1ef38829760_0809092909.txt`). This file contains information about the wrapping key (public key), the wrapping algorithm to use to encrypt your key material, and the date and time when the wrapping key (public key) and import token expire.

   To continue the process now, proceed to the next step. Otherwise, choose **Skip and do this later** and then proceed to Step 3: Encrypt the Key Material (p. 58).
8. (Optional) To continue the process now, encrypt your key material (p. 58). Then do one of the following:

   - If you are in the **Import key material** wizard, select the check box for **I am ready to upload my exported key material** and choose **Next**.
   - If you are in the key details page, choose **Upload key material**.

After you complete this step, proceed to Step 3: Encrypt the Key Material (p. 58).

# Download the Public Key and Import Token (AWS KMS API)

To use the AWS KMS API to download the public key and import token, send a GetParametersForImport request that specifies the CMK for which you are downloading these items. The following example shows how to do this with the AWS CLI.

This example specifies `RSAES_OAEP_SHA_1` as the encryption option. To specify a different option, replace `RSAES_OAEP_SHA_1` with `RSAES_OAEP_SHA_256` or `RSAES_PKCS1_V1_5`. Replace *1234abcd-12ab-34cd-56ef-1234567890ab* with the key ID of the CMK for which to download the public key and import token. You can use the CMK's key ID or Amazon Resource Name (ARN), but you cannot use an alias for this operation.

```
$ aws kms get-parameters-for-import --key-
id 1234abcd-12ab-34cd-56ef-1234567890ab \
                                    --wrapping-algorithm RSAES_OAEP_SHA_1 \
                                    --wrapping-key-spec RSA_2048
```

When the command is successful, you see output similar to the following:

```
{
    "ParametersValidTo": 1470933314.949,
    "PublicKey": "public key base64 encoded data",
    "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "ImportToken": "import token base64 encoded data"
}
```

When you receive this output, save the base64 encoded public key and import token in separate files. Then base64 decode each file into binary data and save the binary data in new files. Doing so prepares these items for later steps. See the following example.

**To prepare the public key and import token for later steps**

1. Copy the public key's base64 encoded data (represented by *`public key base64 encoded data`* in the example output), paste it into a new file, and then save the file. Give the file a descriptive name, for example `PublicKey.b64`.
2. Use OpenSSL to base64 decode the file's contents and save the decoded data to a new file. The following example decodes the data in the file that you saved in the previous step (`PublicKey.b64`) and saves the output to a new file named `PublicKey.bin`.

```
$ openssl enc -d -base64 -A -in PublicKey.b64 -out PublicKey.bin
```

Repeat these two steps for the import token, and then proceed to Step 3: Encrypt the Key Material (p. 58).

# Importing Key Material Step 3: Encrypt the Key Material

After you download the public key and import token (p. 55), you use the public key to encrypt your key material. Typically, you encrypt your key material when exporting it from your hardware security module (HSM) or key management system. For information about how to do this, see the documentation for your HSM or key management system. You can also refer to the following section that provides a proof of concept demonstration using OpenSSL. When you encrypt your key material, use the RSA PKCS #1 encryption scheme with the padding option that you specified when you downloaded the public key and import token (p. 55) (RSAES_OAEP_SHA_256, RSAES_OAEP_SHA_1, or RSAES_PKCS1_V1_5).

## Encrypt Key Material with OpenSSL

The following example demonstrates how to use OpenSSL to generate a 256-bit symmetric key and then encrypt this key material for import into a KMS customer master key (CMK).

> **Important**
> This example is a proof of concept demonstration only. For production systems, use a more secure method (such as a commercial HSM or key management system) to generate and store your key material.

**To use OpenSSL to generate key material and encrypt it for import into AWS KMS**

1. Use the following command to generate a 256-bit symmetric key and save it in a file named `PlaintextKeyMaterial.bin`.

```
$ openssl rand -out PlaintextKeyMaterial.bin 32
```

2. Use the following command to encrypt the key material with the public key that you downloaded previously (see Download the Public Key and Import Token (AWS KMS API) (p. 57)) and save it in a file named `EncryptedKeyMaterial.bin`. Replace *`PublicKey.bin`* with the name of the file that contains the public key. If you downloaded the public key from the console, this file is named wrappingKey_*`CMK_key_ID_timestamp`* (for example, `wrappingKey_f44c4e20-f83c-48f4-adc6-a1ef38829760_0809092909`).

```
$ openssl rsautl -encrypt \
                 -in PlaintextKeyMaterial.bin \
                 -oaep \
                 -inkey PublicKey.bin \
                 -keyform DER \
                 -pubin \
                 -out EncryptedKeyMaterial.bin
```

Proceed to Step 4: Import the Key Material (p. 59).

# Importing Key Material Step 4: Import the Key Material

After you encrypt your key material (p. 58), you can import the key material to use with an AWS KMS customer master key (CMK). To import key material, you upload the encrypted key material from Step 3: Encrypt the Key Material (p. 58) and the import token that you downloaded at Step 2: Download the Public Key and Import Token (p. 55). You must import key material into the same CMK that you specified when you downloaded the public key and import token.

When you import key material, you can optionally specify a time at which the key material expires. When the key material expires, AWS KMS deletes the key material and the CMK becomes unusable. To use the CMK again, you must reimport key material.

After you successfully import key material, the CMK's key state changes to enabled, and you can use the CMK.

To import key material, you can use the AWS Management Console or the AWS KMS API. You can use the API directly by making HTTP requests, or through one of the AWS SDKs or command line tools.

Topics

- Import Key Material (AWS Management Console) (p. 59)
- Import Key Material (AWS KMS API) (p. 60)

## Import Key Material (AWS Management Console)

You can use the AWS Management Console to import key material. If you just completed the optional final step of downloading the public key and import token with the console (p. 56), skip to Step 5 (p. 59).

### To import key material (console)

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Choose the alias of the CMK for which you downloaded the public key and import token.
4. In the **Key Material** section of the page, choose **Upload key material**.
5. In the **Specify key material details** section, for **Encrypted key material**, choose the file that contains your encrypted key material. For **Import token**, choose the file that contains the import token that you downloaded previously (p. 56).

6. In the **Choose an expiration option** section, choose whether the key material expires. If you choose expiration, type a date and a time in the corresponding boxes.
7. Choose **Finish** or **Upload key material**.

## Import Key Material (AWS KMS API)

To use the AWS KMS API to import key material, send an ImportKeyMaterial request. The following example shows how to do this with the AWS CLI.

This example specifies an expiration time for the key material. To import key material with no expiration, replace `KEY_MATERIAL_EXPIRES` with `KEY_MATERIAL_DOES_NOT_EXPIRE` and omit the `--valid-to` parameter. Replace *`1234abcd-12ab-34cd-56ef-1234567890ab`* with the key ID of the CMK for which you downloaded the public key and import token. You can use the CMK's key ID or ARN but you cannot use an alias for this operation. Replace *`EncryptedKeyMaterial.bin`* and *`ImportToken.bin`* with the names of the files that contain the encrypted key material and the import token, respectively.

```
$ aws kms import-key-material --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
                              --encrypted-key-material
 fileb://EncryptedKeyMaterial.bin \
                              --import-token fileb://ImportToken.bin \
                              --expiration-model KEY_MATERIAL_EXPIRES \
                              --valid-to 2016-11-08T12:00:00-08:00
```

# Deleting Imported Key Material

When you import key material, you have the option of specifying a time at which the key material expires. When the key material expires, AWS KMS deletes the key material and the customer master key (CMK) becomes unusable. You can also delete key material on demand. Whether you wait for the key material to expire or you delete it manually, the effect is the same. AWS KMS deletes the key material, the CMK's key state (p. 75) changes to *pending import*, and the CMK is unusable. To use the CMK again, you must reimport the same key material.

Deleting key material affects the CMK right away, but you can reverse the deletion of key material by reimporting the same key material into the CMK. In contrast, scheduling key deletion (p. 63) for a CMK is irreversible. It deletes the key material and all metadata associated with the CMK, and requires a waiting period of between 7 and 30 days.

To delete key material, you can use the AWS Management Console or the AWS KMS API. You can use the API directly by making HTTP requests, or through one of the AWS SDKs or command line tools.

Topics

- How Deleting Key Material Affects AWS Services Integrated With AWS KMS (p. 60)
- Delete Key Material (AWS Management Console) (p. 61)
- Delete Key Material (AWS KMS API) (p. 62)

## How Deleting Key Material Affects AWS Services Integrated With AWS KMS

When you delete key material, the CMK becomes unusable right away. However, *data encryption keys* that are actively in use by AWS services are not immediately affected. This means that deleting key

material might not immediately affect all of the data and AWS resources protected under the CMK, though they are affected eventually.

Several AWS services integrate with AWS KMS (p. 78) to protect your data, such as Amazon Elastic Block Store (Amazon EBS), Amazon Relational Database Service (Amazon RDS), Amazon Simple Storage Service (Amazon S3), and others. Most of these services use *envelope encryption* to protect your data. Envelope encryption means that the CMK protects a data encryption key (or *data key*), and the data key protects your data. These data keys persist in memory on the AWS service host while the data they are protecting are actively in use. For more information about how envelope encryption works, see How Envelope Encryption Works with Supported AWS Services (p. 78).

For example, consider this scenario:

1. You create an encrypted EBS volume, protected under a CMK with imported key material. This action creates a corresponding request to AWS KMS to generate a unique data key for that volume.

2. AWS KMS generates a new data key, encrypts it with the specified CMK, and then sends the encrypted data key to Amazon EBS to store with the volume until you attach the volume to an Amazon Elastic Compute Cloud (Amazon EC2) instance.

3. You attach the EBS volume to an EC2 instance. This action creates a corresponding request to AWS KMS to decrypt the EBS volume's encrypted data key.

4. AWS KMS decrypts the encrypted data key and sends the decrypted (plaintext) data key to Amazon EC2.

5. Amazon EC2 uses the plaintext data key in hypervisor memory to encrypt disk I/O to the EBS volume while the volume is attached to the EC2 instance.

6. You delete the CMK's imported key material. This has no immediate effect on the EC2 instance or the EBS volume, because Amazon EC2 is using the plaintext data key—not the CMK—to encrypt all disk I/O while the volume is attached to the instance.

7. When the EBS volume is detached (due to your explicit request or an event that affects the EBS volume or the EC2 instance), the plaintext data key is deleted from memory. The next attempt to attach the encrypted EBS volume to an EC2 instance fails. It fails because the CMK needed to decrypt the EBS volume's encrypted data key (step 3 in this scenario) is unusable (it has no key material). To use the EBS volume again, you must reimport the same key material into the CMK.

# Delete Key Material (AWS Management Console)

You can use the AWS Management Console to delete key material.

**To delete key material (console)**

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Choose one of the following:

- Select the check box for the CMK whose key material you want to delete. Choose **Key actions**, **Delete key material**.
- Choose the alias of the CMK whose key material you want to delete. In the **Key Material** section of the page, choose **Delete key material**.

4. Confirm that you want to delete the key material and then choose **Delete key material**. The CMK's key state changes to `Pending Import`.

# Delete Key Material (AWS KMS API)

To use the AWS KMS API to delete key material, send a DeleteImportedKeyMaterial request. The following example shows how to do this with the AWS CLI.

Replace *1234abcd-12ab-34cd-56ef-1234567890ab* with the key ID of the CMK whose key material you want to delete. You can use the CMK's key ID or ARN but you cannot use an alias for this operation.

```
$ aws kms delete-imported-key-material --key-
id 1234abcd-12ab-34cd-56ef-1234567890ab
```

# Deleting Customer Master Keys

Deleting a customer master key (CMK) in AWS Key Management Service (AWS KMS) is destructive and potentially dangerous. It deletes the key material and all metadata associated with the CMK, and is irreversible. After a CMK is deleted you can no longer decrypt the data that was encrypted under that key, which means that data becomes unrecoverable. You should delete a CMK only when you are sure that you don't need to use it anymore. If you are not sure, consider disabling the CMK (p. 9) instead of deleting it. You can re-enable a disabled CMK if you need to use it again later, but you cannot recover a deleted CMK.

Before deleting a CMK, you might want to know how many ciphertexts were encrypted under that CMK. AWS KMS does not store this information, and does not store any of the ciphertexts. To obtain this information, you must determine on your own the past usage of a CMK. For some guidance that might help you do this, go to Determining Past Usage of a Customer Master Key (p. 71).

You might choose to delete a CMK for one or more of the following reasons:

- To complete the key lifecycle for CMKs that you no longer need
- To avoid the management overhead and costs associated with maintaining unused CMKs
- To reduce the number of CMKs that count against your limit (p. 159)

Topics



## How Deleting Customer Master Keys Works

Because it is destructive and potentially dangerous to delete a customer master key (CMK), AWS KMS enforces a waiting period. To delete a CMK in AWS KMS you *schedule key deletion*. You can set the

waiting period from a minimum of 7 days up to a maximum of 30 days. The default waiting period is 30 days. During the waiting period, the CMK is *pending deletion* which means it can't be used. After the waiting period ends, AWS KMS deletes the CMK and all AWS KMS data associated with it, including all aliases that point to it.

When you schedule key deletion, AWS KMS reports the date and time when the waiting period ends. This date and time is at least the specified number of days from when you scheduled key deletion, but it can be up to 24 hours longer. For example, when you schedule key deletion and specify a waiting period of 7 days, the end of the waiting period occurs no earlier than 7 days and no more than 8 days from the time of your request. You can confirm the exact date and time when the waiting period ends in the AWS Management Console, AWS CLI, or AWS KMS API.

Use the waiting period to ensure that you don't need the CMK now or in the future. You can configure an Amazon CloudWatch alarm (p. 69) to warn you if a person or application attempts to use the CMK during the waiting period. To recover the CMK, you can cancel key deletion before the waiting period ends. After the waiting period ends you cannot cancel key deletion, and AWS KMS deletes the CMK.

# How Deleting Customer Master Keys Affects AWS Services Integrated With AWS KMS

Several AWS services integrate with AWS KMS to protect your data. Some of these services, such as Amazon EBS and Amazon Redshift, continually modify data in your AWS account while they are in use. These services protect your data using *envelope encryption*, which means the customer master key (CMK) in AWS KMS encrypts a data key, and the data key encrypts your data. These data keys persist in memory as long as the data they are protecting is actively in use. For more information about how envelope encryption works, go to How Envelope Encryption Works with Supported AWS Services (p. 78).

When you schedule a CMK for deletion it becomes unusable. However, data keys that are actively in use are unaffected. This means that scheduling a CMK for deletion does not immediately affect resources and data that are actively in use.

For example, consider this scenario:

1. You create an encrypted EBS volume, at which time Amazon EBS requests a unique data key encrypted with the CMK that you specified when creating the volume.
2. AWS KMS creates a new data key, encrypts it with the specified CMK, and then sends the encrypted data key to Amazon EBS to store with the volume.
3. You attach the EBS volume to an EC2 instance, at which time Amazon EC2 calls the AWS KMS `Decrypt` API to decrypt the EBS volume's encrypted data key. AWS KMS sends the decrypted (plaintext) data key to Amazon EC2.
4. Amazon EC2 uses the plaintext data key in hypervisor memory to encrypt disk I/O to the EBS volume. The data key persists in memory as long as the EBS volume is attached to the EC2 instance.
5. You schedule the CMK for deletion. This has no immediate effect on the EC2 instance or the EBS volume, because Amazon EC2 is using the plaintext data key—not the CMK—to encrypt the EBS volume.
6. The key deletion waiting period ends, and AWS KMS deletes the CMK. This has no immediate effect on the EC2 instance or the EBS volume, because Amazon EC2 is using the plaintext data key—not the CMK—to encrypt the EBS volume.

However, the next time the encrypted EBS volume is attached to an EC2 instance, the attachment fails because at that time Amazon EC2 calls the AWS KMS `Decrypt` API (step 3 in the preceding scenario) but the CMK needed for decryption is unusable (it is pending deletion or deleted).

# Scheduling and Canceling Key Deletion

The following procedures describe how to schedule key deletion and cancel key deletion in AWS Key Management Service (AWS KMS) using the AWS Management Console, the AWS CLI, or the AWS SDK for Java.

> **Warning**
> Deleting a customer master key (CMK) in AWS KMS is destructive and potentially dangerous. You should proceed only when you are sure that you don't need to use the CMK anymore and won't need to use it in the future. If you are not sure, you should disable the CMK (p. 9) instead of deleting it.

Before you can delete a CMK, you must have permission to do so. If you rely on the key policy alone to specify AWS KMS permissions, you might need to add additional permissions before you can delete the CMK. For information about adding these permissions, go to Adding Permission to Schedule and Cancel Key Deletion (p. 67).

Ways to schedule and cancel key deletion

- Using the AWS Management Console (p. 65)
- Using the AWS CLI (p. 66)
- Using the AWS SDK for Java (p. 66)

## Scheduling and Canceling Key Deletion (AWS Management Console)

You can schedule and cancel key deletion in the AWS Management Console.

### To schedule key deletion

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Select the check box next to the key that you want to delete.
4. Choose **Key Actions**, **Schedule key deletion**.
5. For **Waiting period (in days)**, type a number of days between 7 and 30. Choose **Schedule deletion**.

The key status changes to Pending Deletion.

### To cancel key deletion

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Select the check box next to the key that you want to recover.
4. Choose **Key Actions**, **Cancel key deletion**.

The key status changes from Pending Deletion to Disabled. To use the key, you must enable it (p. 9).

# Scheduling and Canceling Key Deletion (AWS CLI)

Use the `aws kms schedule-key-deletion` command to schedule key deletion from the AWS CLI as shown in the following example.

```
$ aws kms schedule-key-deletion --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
 --pending-window-in-days 10
```

When used successfully, the command responds with the following example output:

```
{
    "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "DeletionDate": 1442102400.0
}
```

Use the `aws kms cancel-key-deletion` command to cancel key deletion from the AWS CLI as shown in the following example.

```
$ aws kms cancel-key-deletion --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
```

When used successfully, the command responds with the following example output:

```
{
    "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
}
```

# Scheduling and Canceling Key Deletion (AWS SDK for Java)

The following example demonstrates how to schedule a key for deletion with the AWS SDK for Java. This example requires that you previously instantiated an `AWSKMSClient` as `kms`.

```
String KeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

int PendingWindowInDays = 10;

ScheduleKeyDeletionRequest scheduleKeyDeletionRequest =
new
 ScheduleKeyDeletionRequest().withKeyId(KeyId).withPendingWindowInDays(PendingWindowInDays)
kms.scheduleKeyDeletion(scheduleKeyDeletionRequest);
```

The following example demonstrates how to cancel key deletion with the AWS SDK for Java. This example requires that you previously instantiated an `AWSKMSClient` as `kms`.

```
String KeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
```

```
CancelKeyDeletionRequest cancelKeyDeletionRequest =
new CancelKeyDeletionRequest().withKeyId(KeyId);
kms.cancelKeyDeletion(cancelKeyDeletionRequest);
```

# Adding Permission to Schedule and Cancel Key Deletion

If you use IAM policies to allow AWS KMS permissions, all IAM users and roles that have AWS administrator access (`"Action": "*"`) or AWS KMS full access (`"Action": "kms:*"`) are already allowed to schedule and cancel key deletion for AWS KMS CMKs. If you rely on the key policy alone to allow AWS KMS permissions, you might need to add additional permissions to allow your IAM users and roles to delete CMKs. To add those permissions, see the following steps.

The following procedures describe how to add permissions to a key policy using the AWS Management Console or the AWS CLI.

Ways to add permission to schedule and cancel key deletion

- Using the AWS Management Console (p. 67)
- Using the AWS CLI (p. 68)

## Adding Permission to Schedule and Cancel Key Deletion (AWS Management Console)

You can use the AWS Management Console to add permissions for scheduling and canceling key deletion.

### To add permission to schedule and cancel key deletion

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Choose the alias of the key whose permissions you want to change.
4. In the **Key Policy** section, under **Key Deletion**, select **Allow key administrators to delete this key** and then choose **Save Changes**.

   **Note**
   If you do not see the **Allow key administrators to delete this key** option, this likely means that you have previously modified this key policy using the AWS KMS API. In this case you must update the key policy document manually. Add the `kms:ScheduleKeyDeletion` and `kms:CancelKeyDeletion` permissions to the key administrators statement (`"Sid": "Allow access for Key Administrators"`) in the key policy, and then choose **Save Changes**.


# Adding Permission to Schedule and Cancel Key Deletion (AWS CLI)

You can use the AWS Command Line Interface to add permissions for scheduling and canceling key deletion.

**To add permission to schedule and cancel key deletion**

1. Use the `aws kms get-key-policy` command to retrieve the existing key policy, and then save the policy document to a file.
2. Open the policy document in your preferred text editor, add the `kms:ScheduleKeyDeletion` and `kms:CancelKeyDeletion` permissions to the policy statement that gives permissions to the key administrators (for example, the policy statement with `"Sid": "Allow access for Key Administrators"`), and save the file. The following example shows a policy statement with these two permissions:

```
{
  "Sid": "Allow access for Key Administrators",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:user/KMSKeyAdmin"},
  "Action": [
```

```
    "kms:Create*",
    "kms:Describe*",
    "kms:Enable*",
    "kms:List*",
    "kms:Put*",
    "kms:Update*",
    "kms:Revoke*",
    "kms:Disable*",
    "kms:Get*",
    "kms:Delete*",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "*"
}
```

3. Use the `aws kms put-key-policy` command to apply the key policy to the key.

# Creating an Amazon CloudWatch Alarm to Detect Usage of a Customer Master Key that is Pending Deletion

You can use a combination of AWS CloudTrail, Amazon CloudWatch Logs, and Amazon Simple Notification Service (Amazon SNS) to create an alarm that notifies you of AWS KMS API requests that attempt to use a customer master key (CMK) that is pending deletion. If you receive a notification from such an alarm, you might want to cancel deletion of the CMK to give yourself more time to determine whether you want to delete it.

The following procedures explain how to receive a notification whenever an AWS KMS API request that results in the "*Key ARN* `is pending deletion`" error message is written to your CloudTrail log files. Note that not all AWS KMS API requests that use a CMK that is pending deletion result in this error message. For example, you can successfully use the `ListKeys`, `CancelKeyDeletion`, and `PutKeyPolicy` APIs (and others) with CMK that are pending deletion. The intent of this CloudWatch alarm is to detect usage that might indicate that a person or application still expects to use the CMK for cryptographic operations (`Encrypt`, `Decrypt`, `GenerateDataKey`, and others). To see a list of which AWS KMS APIs do and don't result in this error message, go to How Key State Affects Use of a Customer Master Key (p. 75).

Before you begin these procedures, you must have already turned on CloudTrail in the AWS account and region where you intend to monitor AWS KMS API requests. For instructions, go to Creating a Trail for the First Time in the *AWS CloudTrail User Guide*.

Steps



## Part 1: Configure CloudTrail Log File Delivery to CloudWatch Logs

You must configure delivery of your CloudTrail log files to CloudWatch Logs so that CloudWatch Logs can monitor them for AWS KMS API requests that attempt to use a customer master key (CMK) that is pending deletion.

### To configure CloudTrail log file delivery to CloudWatch Logs

1. Sign in to the AWS Management Console and open the CloudTrail console at https://console.aws.amazon.com/cloudtrail/.
2. On the top navigation bar, choose the AWS region in which you intend to monitor activity.
3. If necessary, in the left navigation pane, choose **Configuration**.
4. In the content pane, in the **CloudWatch Logs (Optional)** section, choose **Configure**.
5. If necessary, for **New or existing log group**, type a name for the log group, such as **`CloudTrail/DefaultLogGroup`**, and then choose **Continue**.
6. On the **AWS CloudTrail will deliver CloudTrail events associated with API activity in your account to your CloudWatch Logs log group** page, choose **Allow**.

# Part 2: Create the CloudWatch Alarm

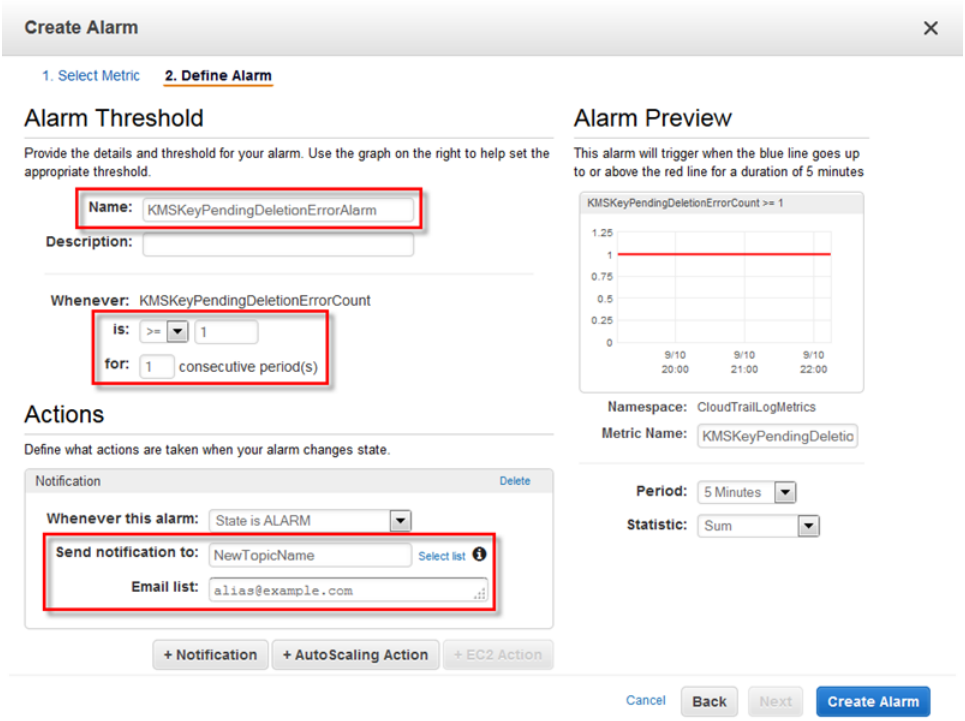
To receive a notification when AWS KMS API requests attempt to use a customer master key (CMK) that is pending deletion, you must create a CloudWatch alarm and configure notification.

### To create a CloudWatch alarm that monitors attempted usage of a KMS CMK that is pending deletion

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. On the top navigation bar, choose the AWS region in which you intend to monitor activity.
3. In the left navigation pane, choose **Logs**.
4. In the list of **Log Groups**, select the check box next to the log group that you created previously, such as **CloudTrail/DefaultLogGroup**. Then choose **Create Metric Filter**.
5. For **Filter Pattern**, type or paste the following:

   ```
   { $.eventSource = kms* && $.errorMessage = "* is pending deletion." }
   ```

   Choose **Assign Metric**.
6. On the **Create Metric Filter and Assign a Metric** page, do the following:
   a. For **Metric Namespace**, type **`CloudTrailLogMetrics`**.
   b. For **Metric Name**, type **`KMSKeyPendingDeletionErrorCount`**.
   c. Choose **Show advanced metric settings**, and then if necessary for **Metric Value**, type **`1`**.
   d. Choose **Create Filter**.
7. In the filter box, choose **Create Alarm**.
8. In the **Create Alarm** window, do the following:
   a. For **Name**, type **`KMSKeyPendingDeletionErrorAlarm`**.
   b. Following **Whenever:**, for **is:**, choose **`>=`** and then type **`1`**.
   c. For **for consecutive period(s)**, if necessary, type **`1`**.
   d. Next to **Send notification to:**, do one of the following:
      - To use a new Amazon SNS topic, choose **New list**, and then type a new topic name. For **Email list:**, type at least one email address. You can type more than one email address by separating them with commas.
      - To use an existing Amazon SNS topic, choose the name of the topic to use.
   e. Choose **Create Alarm**.




9. If you chose to send notifications to an email address, open the email message you receive from no-reply@sns.amazonaws.com with a subject "AWS Notification - Subscription Confirmation." Confirm your email address by choosing the **Confirm subscription** link in the email message.

   **Note**
   You will not receive email notifications until after you have confirmed your email address.

After you complete this procedure, you will receive a notification each time this CloudWatch alarm enters the ALARM state. If you receive a notification for this alarm, it might mean that someone or something still needs to use this CMK. In that case, you should cancel deletion of the CMK (p. 65) to give yourself more time to determine whether you really want to delete it.

# Determining Past Usage of a Customer Master Key

Before deleting a customer master key (CMK), you might want to know how many ciphertexts were encrypted under that key. AWS KMS does not store this information, and does not store any of the ciphertexts. To obtain this information, you must determine on your own the past usage of a CMK. Knowing how a CMK was used in the past might help you decide whether or not you will need it in the future. The following guidance can help you determine the past usage of a CMK.

Topics

- Examining CMK Permissions to Determine the Scope of Potential Usage (p. 72)
- Examining AWS CloudTrail Logs to Determine Actual Usage (p. 72)

# Examining CMK Permissions to Determine the Scope of Potential Usage

Determining who or what currently has access to a customer master key (CMK) might help you determine how widely the CMK was used and whether it is still needed. To learn how to determine who or what currently has access to a CMK, go to Determining Access to an AWS KMS Customer Master Key (p. 43).

# Examining AWS CloudTrail Logs to Determine Actual Usage

AWS KMS is integrated with AWS CloudTrail, so all AWS KMS API activity is recorded in CloudTrail log files. If you have CloudTrail turned on in the region where your customer master key (CMK) is located, you can examine your CloudTrail log files to view a history of all AWS KMS API activity for a particular CMK, and thus its usage history. You might be able to use a CMK's usage history to help you determine whether or not you still need it.

The following examples show CloudTrail log entries that are generated when a KMS CMK is used to protect an object stored in Amazon Simple Storage Service (Amazon S3). In this example, the object is uploaded to Amazon S3 using server-side encryption with AWS KMS-managed keys (SSE-KMS) (p. 99). When you upload an object to Amazon S3 with SSE-KMS, you specify the KMS CMK to use for protecting the object. Amazon S3 uses the AWS KMS `GenerateDataKey` API to request a unique data key for the object, and this API event is logged in CloudTrail with an entry similar to the following:

```
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROACKCEVSQ6C2EXAMPLE:example-user",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admins/example-user",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2015-09-10T23:12:48Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/Admins",
        "accountId": "111122223333",
        "userName": "Admins"
      }
    },
    "invokedBy": "internal.amazonaws.com"
  },
  "eventTime": "2015-09-10T23:58:18Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "GenerateDataKey",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "internal.amazonaws.com",
  "userAgent": "internal.amazonaws.com",
  "requestParameters": {
```

```
    "encryptionContext": {"aws:s3:arn": "arn:aws:s3:::example_bucket/
example_object"},
    "keySpec": "AES_256",
    "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  },
  "responseElements": null,
  "requestID": "cea04450-5817-11e5-85aa-97ce46071236",
  "eventID": "80721262-21a5-49b9-8b63-28740e7ce9c9",
  "readOnly": true,
  "resources": [{
    "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "accountId": "111122223333"
  }],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

When you later download this object from Amazon S3, Amazon S3 sends a `Decrypt` API request to AWS KMS to decrypt the object's data key using the specified CMK. When you do this, your CloudTrail log files include an entry similar to the following:

```
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROACKCEVSQ6C2EXAMPLE:example-user",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admins/example-user",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2015-09-10T23:12:48Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/Admins",
        "accountId": "111122223333",
        "userName": "Admins"
      }
    },
    "invokedBy": "internal.amazonaws.com"
  },
  "eventTime": "2015-09-10T23:58:39Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "internal.amazonaws.com",
  "userAgent": "internal.amazonaws.com",
  "requestParameters": {
    "encryptionContext": {"aws:s3:arn": "arn:aws:s3:::example_bucket/
example_object"}},
  "responseElements": null,
  "requestID": "db750745-5817-11e5-93a6-5b87e27d91a0",
  "eventID": "ae551b19-8a09-4cfc-a249-205ddba330e3",
```

```
  "readOnly": true,
  "resources": [{
    "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "accountId": "111122223333"
  }],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

All AWS KMS API activity is logged by CloudTrail. By evaluating these log entries, you might be able to determine the past usage of a particular CMK, and this might help you determine whether or not you want to delete it.

To see more examples of how AWS KMS API activity appears in your CloudTrail log files, go to Logging AWS KMS API Calls Using AWS CloudTrail (p. 114). For more information about CloudTrail go to the AWS CloudTrail User Guide.

# How Key State Affects Use of a Customer Master Key

Each customer master key (CMK) in AWS KMS exists in one of three states: enabled, disabled, or pending deletion. Enabled CMKs are available for use. CMKs in the disabled or pending deletion state are available for some APIs, but are unavailable for others. CMKs in the disabled or pending deletion state cannot be used for cryptographic operations.

Consult the following table to learn the result of each AWS KMS API on existing CMKs in each state. The `CreateKey` and `GenerateRandom` APIs do not affect existing CMKs, so they are not applicable. The results in this table assume that the API caller is authorized to use the CMK.

**Legend**

 – Succeeds.

 – Results in `DisabledException` with the message "*`Key ARN`* `is disabled.`"

 – Results in `DisabledException` with the message "*`Key ARN`* `is pending deletion.`"

 – Results in `KMSInvalidStateException` with the message "*`Key ARN`* `is pending deletion.`"

 – Results in `KMSInvalidStateException` with the message "*`Key ARN`* `is not pending deletion.`"

 – Applies only to the `UpdateAlias` API request. When a CMK that is pending deletion is the "source" key, the request succeeds. When a CMK that is pending deletion is the target key, the request results in `KMSInvalidStateException` with the message "*`Key ARN`* `is pending deletion.`"

N/A – Not applicable

| API | Enabled | Disabled | Pending Deletion |
|---|---|---|---|
| CancelKeyDeletion |  |  |  |

| API | Enabled | Disabled | Pending Deletion |
|---|---|---|---|
| CreateAlias |  |  |  |
| CreateGrant |  |  |  |
| CreateKey | N/A | N/A | N/A |
| Decrypt |  |  |  |
| DeleteAlias |  |  |  |
| DescribeKey |  |  |  |
| DisableKey |  |  |  |
| DisableKeyRotation |  |  |  |
| EnableKey |  |  |  |
| EnableKeyRotation |  |  |  |
| Encrypt |  |  |  |
| GenerateDataKey |  |  |  |
| GenerateDataKeyWithoutPlaintext |  |  |  |
| GenerateRandom | N/A | N/A | N/A |
| GetKeyPolicy |  |  |  |
| GetKeyRotationStatus |  |  |  |
| ListAliases |  |  |  |
| ListGrants |  |  |  |

| API | Enabled | Disabled | Pending Deletion |
|---|---|---|---|
| ListKeyPolicies | ✓ | ✓ | ✓ |
| ListKeys | ✓ | ✓ | ✓ |
| ListRetirableGrants | ✓ | ✓ | ✓ |
| PutKeyPolicy | ✓ | ✓ | ✓ |
| ReEncrypt | ✓ |  |  |
| RetireGrant | ✓ | ✓ | ✓ |
| RevokeGrant | ✓ | ✓ | ✓ |
| ScheduleKeyDeletion | ✓ | ✓ |  |
| UpdateAlias | ✓ | ✓ |  |
| UpdateKeyDescription | ✓ | ✓ |  |

# How AWS Services use AWS KMS

Several AWS services use AWS KMS to provide encryption of customer data. The following topics discuss the details of how each service uses AWS KMS. The first topic in the list describes in general terms how envelope encryption works in the context of the supported services.

Topics



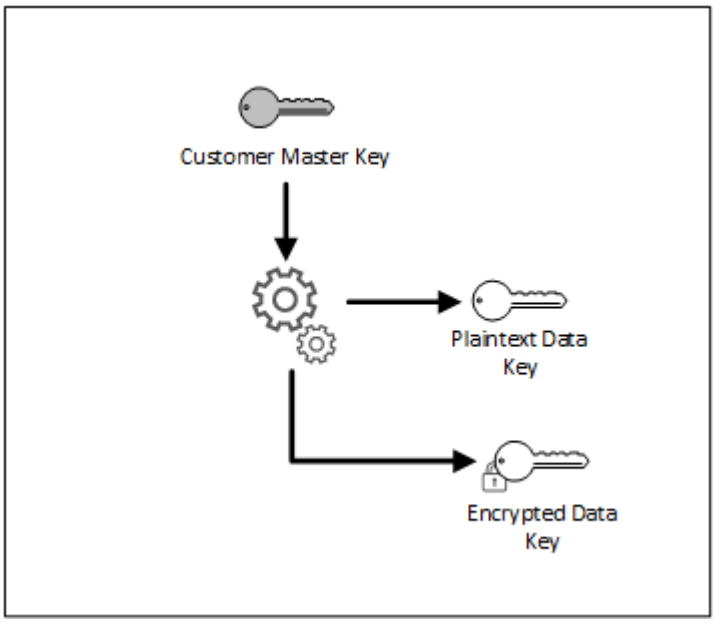
## How Envelope Encryption Works with Supported AWS Services

This topic describes how and when AWS KMS generates, encrypts, and decrypts keys that can be used to encrypt your data in a supported AWS service.

### Envelope Encryption

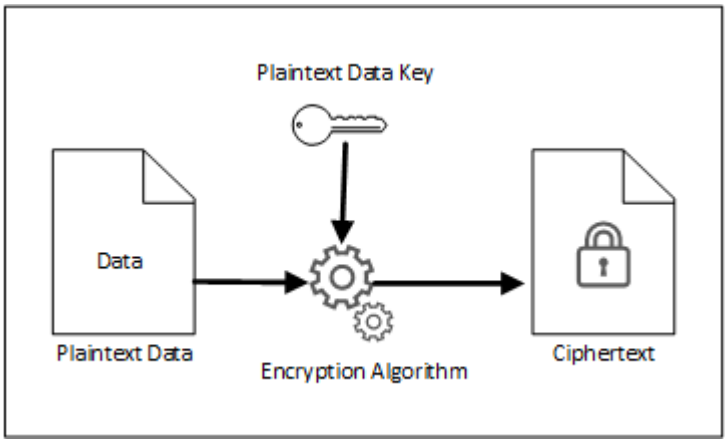
AWS KMS supports two kinds of keys — *master keys* and *data keys*. Master keys can be used to directly encrypt and decrypt up to 4 kilobytes of data and can also be used to protect data keys. The data keys are then used to encrypt and decrypt customer data.

Customer master keys are stored securely in AWS KMS. They can never be exported from AWS KMS. Data keys created inside of AWS KMS can be exported and are protected for export by being encrypted under a master key. The data key encryption process is illustrated by the following diagram:




# Encrypting User Data

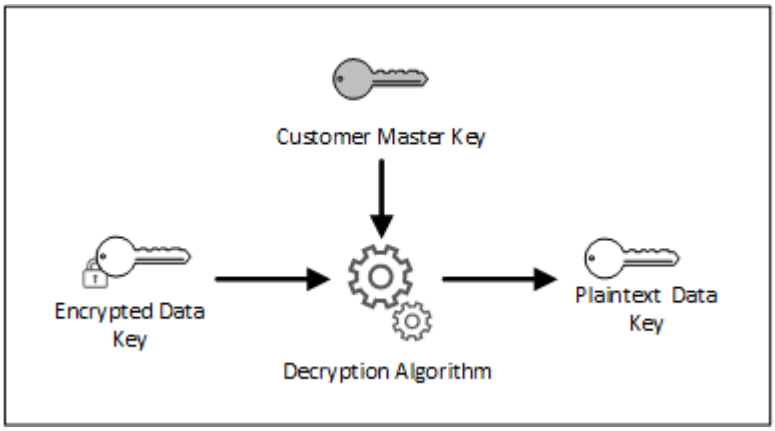
When a data key is requested, AWS KMS returns both the encrypted and plaintext key back to the service or application that requested it. The plaintext data key is used to encrypt the user's data in memory. This key should never be written to disk and should be deleted from memory as soon as practical. The encrypted copy of the data key should be written to disk alongside of the encrypted data. This is acceptable and simplifies management of the encrypted data key.



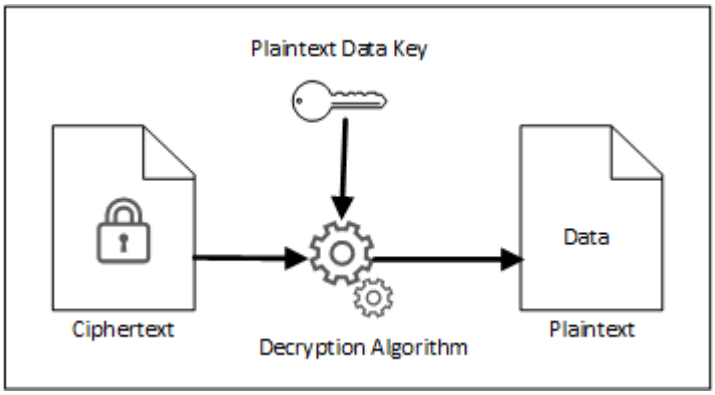

# Decrypting User Data

Decryption reverses the encryption process. When a service or application needs to decrypt data, it sends AWS KMS the encrypted data key. AWS KMS decrypts the data key by automatically using the correct customer master key and then sends the plaintext key back to the service or application that requested it. The plaintext key is used to decrypt the data. This key should never be written to disk and

should be deleted as soon as is practical. The following illustration shows the customer master key used with the symmetric decryption algorithm to decrypt the data key.




The next illustration shows the plaintext data key and symmetric algorithm used together to decrypt the user's encrypted data. The plaintext data key should be removed from memory as soon as is practical.




## Managed Keys in AWS Services and in Custom Applications

You can choose to encrypt data in one of the services integrated with AWS KMS by using the AWS-managed key for that service under your account. In this case, all users who have access to that service can use the key. For more granular control, you can choose to create a customer-managed key and set policies that define who can use the key and what actions the users can perform.

# How AWS CloudTrail Uses AWS KMS

You can use AWS CloudTrail to record AWS API calls and other activity for your AWS account and to save the recorded information to log files in an Amazon Simple Storage Service (Amazon S3) bucket that you choose. By default, the log files delivered by CloudTrail to your S3 bucket are encrypted using server-side encryption with Amazon S3–managed encryption keys (SSE-S3). But you can choose instead to use server-side encryption with AWS KMS–managed keys (SSE-KMS). To learn how to encrypt your CloudTrail log files with AWS KMS, see Encrypting CloudTrail Log Files with AWS KMS–Managed Keys (SSE-KMS) in the *AWS CloudTrail User Guide*.

Topics



# Understanding When Your CMK is Used

Encrypting CloudTrail log files with AWS KMS builds on the Amazon S3 feature called server-side encryption with AWS KMS–managed keys (SSE-KMS). To learn more about SSE-KMS, see How Amazon Simple Storage Service (Amazon S3) Uses AWS KMS (p. 99) in this guide or Protecting Data Using Server-Side Encryption with AWS KMS–Managed Keys (SSE-KMS) in the *Amazon Simple Storage Service Developer Guide*.

When you configure AWS CloudTrail to use SSE-KMS to encrypt your log files, CloudTrail and Amazon S3 use your KMS customer master key (CMK) when you perform certain actions with those services. The following sections explain when and how those services can use your CMK, and provide additional information that you can use to validate this explanation.

Actions that cause CloudTrail and Amazon S3 to use your CMK



## You Configure CloudTrail to Encrypt Log Files with Your Customer Master Key (CMK)

When you update your CloudTrail configuration to use your CMK, CloudTrail sends a `GenerateDataKey` request to AWS KMS to verify that the CMK exists and that CloudTrail has permission to use it for encryption. CloudTrail does not use the resulting data key.

The `GenerateDataKey` request includes the following information for the encryption context (p. 153):

- The Amazon Resource Name (ARN) of the CloudTrail trail
- The ARN of the S3 bucket and path where the CloudTrail log files are delivered

The `GenerateDataKey` request results in an entry in your CloudTrail logs similar to the following example. When you see a log entry like this one, you can determine that CloudTrail (1) called the AWS KMS (2) `GenerateDataKey` API (3) for a specific trail (4). AWS KMS created the data key under a specific CMK (5).

**Note**
You might need to scroll to the right to see some of the callouts in the following example log entry.

```
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::086441151436:user/AWSCloudTrail",
    "accountId": "086441151436",
```

```
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "userName": "AWSCloudTrail",
    "sessionContext": {"attributes": {
      "mfaAuthenticated": "false",
      "creationDate": "2015-11-11T21:15:33Z"
    }},
    "invokedBy": "internal.amazonaws.com"
  },
  "eventTime": "2015-11-11T21:15:33Z",
  "eventSource": "kms.amazonaws.com",(2)
  "eventName": "GenerateDataKey",(3)
  "awsRegion": "us-west-2",
  "sourceIPAddress": "internal.amazonaws.com",
  "userAgent": "internal.amazonaws.com",
  "requestParameters": {
    "keyId": "arn:aws:kms:us-west-2:111122223333:alias/
ExampleAliasForCloudTrailCMK",
    "encryptionContext": {
      "aws:cloudtrail:arn": "arn:aws:cloudtrail:us-west-2:111122223333:trail/
Default",(4)
      "aws:s3:arn": "arn:aws:s3:::example-bucket-for-CT-logs/
AWSLogs/111122223333/"
    },
    "keySpec": "AES_256"
  },
  "responseElements": null,
  "requestID": "581f1f11-88b9-11e5-9c9c-595a1fb59ac0",
  "eventID": "3cdb2457-c035-4890-93b6-181832b9e766",
  "readOnly": true,
  "resources": [{
    "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",(5)
    "accountId": "111122223333"
  }],
  "eventType": "AwsServiceEvent",
  "recipientAccountId": "111122223333"
}
```

## CloudTrail Puts a Log File into Your S3 Bucket

Each time CloudTrail puts a log file into your S3 bucket, Amazon S3 sends a `GenerateDataKey` request to AWS KMS on behalf of CloudTrail. In response to this request, AWS KMS generates a unique data key and then sends Amazon S3 two copies of the data key, one in plaintext and one that is encrypted with the specified CMK. Amazon S3 uses the plaintext data key to encrypt the CloudTrail log file and then removes the plaintext data key from memory as soon as possible after use. Amazon S3 stores the encrypted data key as metadata with the encrypted CloudTrail log file.

The `GenerateDataKey` request includes the following information for the encryption context (p. 153):

- The Amazon Resource Name (ARN) of the CloudTrail trail
- The ARN of the S3 object (the CloudTrail log file)

Each `GenerateDataKey` request results in an entry in your CloudTrail logs similar to the following example. When you see a log entry like this one, you can determine that CloudTrail (1) called the

AWS KMS (2) `GenerateDataKey` API (3) for a specific trail (4) to protect a specific log file (5). AWS KMS created the data key under the specified CMK (6), shown twice in the same log entry.

**Note**

You might need to scroll to the right to see some of the callouts in the following example log entry.

```
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROACKCEVSQ6C2EXAMPLE:i-34755b85",
    "arn": "arn:aws:sts::086441151436:assumed-role/AWSCloudTrail/
i-34755b85",1
    "accountId": "086441151436",
    "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2015-11-11T20:45:25Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::086441151436:role/AWSCloudTrail",
        "accountId": "086441151436",
        "userName": "AWSCloudTrail"
      }
    },
    "invokedBy": "internal.amazonaws.com"
  },
  "eventTime": "2015-11-11T21:15:58Z",
  "eventSource": "kms.amazonaws.com",2
  "eventName": "GenerateDataKey",3
  "awsRegion": "us-west-2",
  "sourceIPAddress": "internal.amazonaws.com",
  "userAgent": "internal.amazonaws.com",
  "requestParameters": {
    "encryptionContext": {
      "aws:cloudtrail:arn": "arn:aws:cloudtrail:us-west-2:111122223333:trail/
Default",4
      "aws:s3:arn": "arn:aws:s3:::example-bucket-
for-CT-logs/AWSLogs/111122223333/CloudTrail/us-
west-2/2015/11/11/111122223333_CloudTrail_us-
west-2_20151111T2115Z_7JREEBimdK8d2nC9.json.gz"5
    },
    "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",6
    "keySpec": "AES_256"
  },
  "responseElements": null,
  "requestID": "66f3f74a-88b9-11e5-b7fb-63d925c72ffe",
  "eventID": "7738554f-92ab-4e27-83e3-03354b1aa898",
  "readOnly": true,
```

```
  "resources": [{
    "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "accountId": "111122223333"
  }],
  "eventType": "AwsServiceEvent",
  "recipientAccountId": "111122223333"
}
```

## You Get an Encrypted Log File from Your S3 Bucket

Each time you get an encrypted CloudTrail log file from your S3 bucket, Amazon S3 sends a `Decrypt` request to AWS KMS on your behalf to decrypt the log file's encrypted data key. In response to this request, AWS KMS uses your CMK to decrypt the data key and then sends the plaintext data key to Amazon S3. Amazon S3 uses the plaintext data key to decrypt the CloudTrail log file and then removes the plaintext data key from memory as soon as possible after use.

The `Decrypt` request includes the following information for the encryption context (p. 153):

- The Amazon Resource Name (ARN) of the CloudTrail trail
- The ARN of the S3 object (the CloudTrail log file)

Each `Decrypt` request results in an entry in your CloudTrail logs similar to the following example. When you see a log entry like this one, you can determine that an IAM user in your AWS account (1) called the AWS KMS (2) `Decrypt` API (3) for a specific trail (4) and a specific log file (5). AWS KMS decrypted the data key under a specific CMK (6).

**Note**
You might need to scroll to the right to see some of the callouts in the following example log entry.

```
{
  "eventVersion": "1.02",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE",
    "arn": "arn:aws:iam::111122223333:user/cloudtrail-admin",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "cloudtrail-admin",
    "sessionContext": {"attributes": {
      "mfaAuthenticated": "false",
      "creationDate": "2015-11-11T20:48:04Z"
    }},
    "invokedBy": "signin.amazonaws.com"
  },
  "eventTime": "2015-11-11T21:20:52Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "internal.amazonaws.com",
  "userAgent": "internal.amazonaws.com",
  "requestParameters": {
    "encryptionContext": {
```

```
      "aws:cloudtrail:arn": "arn:aws:cloudtrail:us-west-2:111122223333:trail/
Default",(4)
      "aws:s3:arn": "arn:aws:s3:::example-bucket-
for-CT-logs/AWSLogs/111122223333/CloudTrail/us-
west-2/2015/11/11/111122223333_CloudTrail_us-
west-2_20151111T2115Z_7JREEBimdK8d2nC9.json.gz"(5)
    }
  },
  "responseElements": null,
  "requestID": "16a0590a-88ba-11e5-b406-436f15c3ac01",
  "eventID": "9525bee7-5145-42b0-bed5-ab7196a16daa",
  "readOnly": true,
  "resources": [{
    "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",(6)
    "accountId": "111122223333"
  }],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

## Understanding How Often Your CMK is Used

To predict costs and better understand your AWS bill, you might want to know how often CloudTrail uses your CMK. AWS KMS charges for all API requests to the service that exceed the free tier. For the exact charges, see AWS Key Management Service Pricing.

When you encrypt CloudTrail log files with AWS KMS–Managed Keys (SSE-KMS), each time CloudTrail puts a log file into your S3 bucket (p. 82) it results in an AWS KMS API request. Typically, CloudTrail puts a log file into your S3 bucket once every five minutes, which results in approximately 288 AWS KMS API requests per day, per region, and per AWS account. For example:

- If you enable this feature in two regions in a single AWS account, you can expect approximately 576 AWS KMS API requests per day (2 x 288).
- If you enable this feature in two regions in each of three AWS accounts, you can expect approximately 1,728 AWS KMS API requests per day (6 x 288).

These numbers represent only the AWS KMS API requests that occur when CloudTrail puts a log file into your S3 bucket. Each time you get an encrypted log file from your S3 bucket (p. 84) it results in an additional AWS KMS API request.

# How Amazon Elastic Block Store (Amazon EBS) Uses AWS KMS

This topic discusses how Amazon Elastic Block Store (Amazon EBS) uses AWS KMS to encrypt volumes and snapshots.

Topics

- Using AWS CloudFormation to Create Encrypted Amazon EBS Volumes (p. 87)

# Amazon EBS Encryption

When you create an encrypted Amazon EBS volume and attach it to a supported Amazon Elastic Compute Cloud (Amazon EC2) instance type, data stored at rest on the volume, disk I/O, and snapshots created from the volume are all encrypted. The encryption occurs on the servers that host Amazon EC2 instances.

This feature is supported on all Amazon EBS volume types. You access encrypted volumes the same way you access other volumes; encryption and decryption are handled transparently and they require no additional action from you, your EC2 instance, or your application. Snapshots of encrypted volumes are automatically encrypted, and volumes that are created from encrypted snapshots are also automatically encrypted.

To create an encrypted Amazon EBS volume, you select the appropriate box in the Amazon EBS section of the Amazon EC2 console. You can use a custom customer master key (CMK) by choosing one from the list that appears below the encryption box. If you do not specify a custom CMK, Amazon EBS uses the AWS-managed CMK for Amazon EBS in your account. If there is no AWS-managed CMK for Amazon EBS in your account, Amazon EBS creates one.

The following explains how Amazon EBS uses your CMK:

1. When you create an encrypted EBS volume, Amazon EBS sends a GenerateDataKeyWithoutPlaintext request to AWS KMS, specifying the CMK that you chose for EBS volume encryption.
2. AWS KMS generates a new data key, encrypts it under the specified CMK, and then sends the encrypted data key to Amazon EBS to store with the volume metadata.
3. When you attach the encrypted volume to an EC2 instance, Amazon EC2 sends the encrypted data key to AWS KMS with a Decrypt request.
4. AWS KMS decrypts the encrypted data key and then sends the decrypted (plaintext) data key to Amazon EC2.
5. Amazon EC2 uses the plaintext data key in hypervisor memory to encrypt disk I/O to the EBS volume. The data key persists in memory as long as the EBS volume is attached to the EC2 instance.

# Amazon EBS Encryption Context

Amazon EBS sends encryption context (p. 153) when making AWS KMS API requests to generate data keys and decrypt. Amazon EBS uses the volume ID as encryption context for all volumes and for encrypted snapshots created with the CreateSnapshot operation in the Amazon EC2 API. In the `requestParameters` field of a CloudTrail log entry, the encryption context looks similar to the following:

```
"encryptionContext": {
  "aws:ebs:id": "vol-0cfb133e847d28be9"
}
```

Amazon EBS uses the snapshot ID as encryption context for encrypted snapshots created with the CopySnapshot operation in the Amazon EC2 API. In the `requestParameters` field of a CloudTrail log entry, the encryption context looks similar to the following:

```
"encryptionContext": {
```

```
  "aws:ebs:id": "snap-069a655b568de654f"
}
```

## Detecting Amazon EBS Failures

To create an encrypted EBS volume or attach the volume to an EC2 instance, Amazon EBS and the Amazon EC2 infrastructure must be able to use the CMK that you specified for EBS volume encryption. When the CMK is not usable—for example, when it is not in the enabled key state (p. 75)—the volume creation or volume attachment fails. In this case, Amazon EBS sends an *event* to Amazon CloudWatch Events to notify you about the failure. With CloudWatch Events, you can establish rules that trigger automatic actions in response to these events. For more information, see Amazon CloudWatch Events for Amazon EBS in the *Amazon EC2 User Guide for Linux Instances*, especially the following sections:

- Invalid Encryption Key on Volume Attach or Reattach
- Invalid Encryption Key on Create Volume

To fix these failures, ensure that the CMK that you specified for EBS volume encryption is enabled. To do this, first view the CMK (p. 7) to determine its current key state (the **Status** column in the AWS Management Console). Then, see the information at one of the following links:

- If the CMK's key state is disabled, enable it (p. 9).
- If the CMK's key state is pending import, import key material (p. 52).
- If the CMK's key state is pending deletion, cancel key deletion (p. 65).

## Using AWS CloudFormation to Create Encrypted Amazon EBS Volumes

You can use AWS CloudFormation to create encrypted Amazon EBS volumes. For more information, see AWS::EC2::Volume in the *AWS CloudFormation User Guide*.

# How Amazon Elastic Transcoder Uses AWS KMS

You can use Amazon Elastic Transcoder to convert media files stored in an Amazon S3 bucket into formats required by consumer playback devices. Both input and output files can be encrypted and decrypted. The following sections discuss how AWS KMS is used for both processes.

Topics



## Encrypting the input file

Before you can use Elastic Transcoder, you must create an Amazon S3 bucket and upload your media file into it. You can encrypt the file before uploading by using Amazon S3 server-side encryption or AES client-side encryption.

If you choose client-side encryption using AES, you are responsible for encrypting the file before uploading it to Amazon S3, and you must provide Elastic Transcoder access to the encryption key. You do this by using an AWS KMS customer master key (CMK) to protect the AES encryption key you used to encrypt the media file.

If you choose server-side encryption, you are allowing Amazon S3 to perform all encryption and decryption of files on your behalf. You can configure Amazon S3 to use one of three different master keys to protect the unique data key used to encrypt your file:

- The Amazon S3 master key, a key that is owned and managed by AWS
- The AWS-managed CMK for Amazon S3, a master key that is owned by your account but managed by AWS
- Any customer-managed CMK that you create by using AWS KMS

You can request encryption and the master key you want by using the Amazon S3 console or the appropriate Amazon S3 APIs. For more information about how Amazon S3 performs encryption, see Protecting Data Using Encryption in the *Amazon Simple Storage Service Developer Guide*.

When you use an AWS KMS CMK as the master key (the AWS-managed CMK for Amazon S3 in your account or a customer-managed CMK) to protect the input file, Amazon S3 and AWS KMS interact in the following manner:

1. Amazon S3 requests a plaintext data key and a copy of the data key encrypted under the specified CMK.
2. AWS KMS creates a data key, encrypts it under the CMK, and then sends both the plaintext data key and the encrypted data key to Amazon S3.
3. Amazon S3 encrypts the media file using the plaintext data key and stores the file in the specified Amazon S3 bucket.
4. Amazon S3 stores the encrypted data key alongside of the encrypted media file.

# Decrypting the input file

If you choose Amazon S3 server-side encryption to encrypt the input file, Elastic Transcoder does not decrypt the file. Instead, Elastic Transcoder relies on Amazon S3 to perform decryption depending on the settings you specify when you create a job and a pipeline. The following combination of settings are available.

| Encryption mode | AWS KMS key | Meaning |
|---|---|---|
| **S3** | Default | Amazon S3 creates and manages the keys used to encrypt and decrypt the media file. The process is opaque to the user. |
| **S3-AWS-KMS** | Default | Amazon S3 uses a data key encrypted by the default AWS-managed CMK for Amazon S3 in your account to encrypt the media file. |
| **S3-AWS-KMS** | Custom (with ARN) | Amazon S3 uses a data key encrypted by the specified customer-managed CMK to encrypt the media file. |

When **S3-AWS-KMS** is specified, Amazon S3 and AWS KMS work together in the following manner to perform the decryption.

1. Amazon S3 sends the encrypted data key to AWS KMS.
2. AWS KMS decrypts the data key by using the appropriate CMK, and then sends the plaintext data key back to Amazon S3.
3. Amazon S3 uses the plaintext data key to decrypt the ciphertext.

If you choose client-side encryption using an AES key, Elastic Transcoder retrieves the encrypted file from the Amazon S3 bucket and decrypts it. Elastic Transcoder uses the CMK you specified when you created the pipeline to decrypt the AES key and then uses the AES key to decrypt the media file.

# Encrypting the output file

Elastic Transcoder encrypts the output file depending on how you specify the encryption settings when you create a job and a pipeline. The following options are available.

| Encryption mode | AWS KMS key | Meaning |
|---|---|---|
| **S3** | Default | Amazon S3 creates and manages the keys used to encrypt the output file. |
| **S3-AWS-KMS** | Default | Amazon S3 uses a data key created by AWS KMS and encrypted by the AWS-managed CMK for Amazon S3 in your account. |
| **S3-AWS-KMS** | Custom (with ARN) | Amazon S3 uses a data key encrypted by using the customer-managed CMK specified by the ARN to encrypt the media file. |
| **AES-** | Default | Elastic Transcoder uses the AWS-managed CMK for Amazon S3 in your account to decrypt the specified AES key you provide and uses that key to encrypt the output file. |
| **AES-** | Custom (with ARN) | Elastic Transcoder uses the customer-managed CMK specified by the ARN to decrypt the specified AES key you provide and uses that key to encrypt the output file. |

When you specify that an AWS KMS CMK (the AWS-managed CMK for Amazon S3 in your account or a customer-managed CMK) be used to encrypt the output file, Amazon S3 and AWS KMS interact in the following manner:

1. Amazon S3 requests a plaintext data key and a copy of the data key encrypted by using the specified CMK.

2. AWS KMS creates a data key, encrypts it under the CMK, and sends both the plaintext data key and the encrypted data key to Amazon S3.
3. Amazon S3 encrypts the media using the data key and stores it in the specified Amazon S3 bucket.
4. Amazon S3 stores the encrypted data key alongside the encrypted media file.

When you specify that your provided AES key be used to encrypt the output file, the AES key must be encrypted using a CMK in AWS KMS. Elastic Transcoder, AWS KMS, and you interact in the following manner:

1. You encrypt your AES key by calling the Encrypt operation in the AWS KMS API. AWS KMS encrypts the key by using the specified CMK. You specify which CMK to use when you are creating the pipeline.
2. You specify the file containing the encrypted AES key when you create the Elastic Transcoder job.
3. Elastic Transcoder decrypts the key by calling the Decrypt operation in the AWS KMS API, passing the encrypted key as ciphertext.
4. Elastic Transcoder uses the decrypted AES key to encrypt the output media file and then deletes the decrypted AES key from memory. Only the encrypted copy you originally defined in the job is saved to disk.
5. You can download the encrypted output file and decrypt it locally by using the original AES key that you defined.

**Important**
Your private encryption keys are never stored by AWS. Therefore, it is important that you safely and securely manage your keys. If you lose them, you won't be able to decrypt your data.

# HLS Content Protection

HTTP Live Streaming (HLS) is an adaptive streaming protocol. Elastic Transcoder supports HLS by breaking your input file into smaller individual files called media segments. A set of corresponding individual media segments contain the same material encoded at different bit rates, thereby enabling the player to select the stream that best fits the available bandwidth. Elastic Transcoder also creates playlists that contain metadata for the various segments that are available to be streamed.

You can use AES-128 encryption to protect the transcoded media segments. When you enable HLS content protection, each media segment is encrypted using an AES-128 encryption key. When the content is viewed, the player downloads the key and decrypts the media segments during the playback process.

Two types of keys are used: an AWS KMS CMK and a data key. You must create a CMK to use to encrypt and decrypt the data key. Elastic Transcoder uses the data key to encrypt and decrypt media segments. The data key must be AES-128. All variations and segments of the same content are encrypted using the same data key. You can provide a data key or have Elastic Transcoder create it for you.

The CMK can be used to encrypt the data key at the following points:

- If you provide your own data key, you must encrypt it before passing it to Elastic Transcoder.
- If you request that Elastic Transcoder generate the data key, then Elastic Transcoder encrypts the data key for you.

The CMK can be used to decrypt the data key at the following points:

- Elastic Transcoder decrypts your provided data key when it needs to use the data key to encrypt the output file or decrypt the input file.

- You decrypt a data key generated by Elastic Transcoder and use it to decrypt output files.

For more information, see HLS Content Protection in the *Amazon Elastic Transcoder Developer Guide*.

## Elastic Transcoder Encryption Context

Elastic Transcoder sends encryption context (p. 153) when making AWS KMS API requests to generate data keys, encrypt, and decrypt. The encryption context is written to CloudTrail logs to help you understand why a given AWS KMS CMK was used. Elastic Transcoder uses the service name as encryption context. In the `requestParameters` field of a CloudTrail log file, the encryption context looks similar to the following:

```
"encryptionContext": {
  "service" : "elastictranscoder.amazonaws.com"
}
```

For more information about how to configure Elastic Transcoder jobs to use one of the supported encryption options, see Data Encryption Options in the *Amazon Elastic Transcoder Developer Guide*.

# How Amazon EMR Uses AWS KMS

When you use an Amazon EMR cluster, you can configure the cluster to encrypt data *at rest*, which means the cluster encrypts data before saving it to a persistent storage location. You can encrypt data at rest on the EMR File System (EMRFS), on the storage volumes of cluster nodes, or both. To encrypt data at rest, you can use a customer master key (CMK) in AWS KMS. The following topics explain how an Amazon EMR cluster uses a CMK to encrypt data at rest.

Amazon EMR clusters also encrypt data *in transit*, which means the cluster encrypts data before sending it through the network. You cannot use a CMK to encrypt data in transit. For more information, see In-Transit Data Encryption in the *Amazon EMR Release Guide*.

For more information about all the encryption options available in Amazon EMR, see Understanding Encryption Options with Amazon EMR in the *Amazon EMR Release Guide*.

Topics

- Encrypting Data on the EMR File System (EMRFS) (p. 91)
- Encrypting Data on the Storage Volumes of Cluster Nodes (p. 93)
- Encryption Context (p. 94)

## Encrypting Data on the EMR File System (EMRFS)

Amazon EMR clusters use two distributed files systems:

- The Hadoop Distributed File System (HDFS). HDFS encryption does not use a CMK in AWS KMS.

- The EMR File System (EMRFS). EMRFS is an implementation of HDFS that allows Amazon EMR clusters to store data in Amazon Simple Storage Service (Amazon S3). EMRFS supports four encryption options, two of which use a CMK in AWS KMS. For more information about all four of the EMRFS encryption options, see At-Rest Encryption for Amazon S3 with EMRFS in the *Amazon EMR Release Guide*.

The two EMRFS encryption options that use a CMK use the following encryption features offered by Amazon S3:

- Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS). With SSE-KMS, the Amazon EMR cluster sends data to Amazon S3, and then Amazon S3 uses a CMK to encrypt the data before saving it to an S3 bucket. For more information about how this works, see Process for Encrypting Data on EMRFS with SSE-KMS (p. 92).

- Client-Side Encryption with AWS KMS-Managed Keys (CSE-KMS). With CSE-KMS, the Amazon EMR cluster uses a CMK to encrypt data before sending it to Amazon S3 for storage. For more information about how this works, see Process for Encrypting Data on EMRFS with CSE-KMS (p. 93).

When you configure an Amazon EMR cluster to encrypt data on EMRFS with SSE-KMS or CSE-KMS, you choose the CMK in AWS KMS that you want Amazon S3 or the Amazon EMR cluster to use. With SSE-KMS, you can choose the AWS-managed CMK for Amazon S3 with the alias **aws/s3**, or a custom CMK that you create. With CSE-KMS, you must choose a custom CMK that you create. When you choose a custom CMK, you must ensure that your Amazon EMR cluster has permission to use the CMK. For more information, see Add the EMR Instance Role to an AWS KMS CMK in the *Amazon EMR Release Guide*.

For both SSE-KMS and CSE-KMS, the CMK you choose is the master key in an envelope encryption (p. 78) workflow. This means the data is encrypted with a unique data encryption key (or *data key*), and this data key is encrypted under the CMK in AWS KMS. The encrypted data and an encrypted copy of its data key are stored together as a single encrypted object in an S3 bucket. For more information about how this works, see the following topics.

Topics

- Process for Encrypting Data on EMRFS with SSE-KMS (p. 92)
- Process for Encrypting Data on EMRFS with CSE-KMS (p. 93)

## Process for Encrypting Data on EMRFS with SSE-KMS

When you configure an Amazon EMR cluster to use SSE-KMS, the encryption process works like this:

1. The cluster sends data to Amazon S3 for storage in an S3 bucket.
2. Amazon S3 sends a GenerateDataKey request to AWS KMS, specifying the key ID of the CMK that you chose when you configured the cluster to use SSE-KMS. The request includes encryption context; for more information, see Encryption Context (p. 94).
3. AWS KMS generates a unique data encryption key (data key) and then sends two copies of this data key to Amazon S3. One copy is unencrypted (plaintext), and the other copy is encrypted under the CMK.
4. Amazon S3 uses the plaintext data key to encrypt the data that it received in step 1, and then removes the plaintext data key from memory as soon as possible after use.
5. Amazon S3 stores the encrypted data and the encrypted copy of the data key together as a single encrypted object in an S3 bucket.

The decryption process works like this:

1. The cluster requests an encrypted data object from an S3 bucket.
2. Amazon S3 extracts the encrypted data key from the S3 object, and then sends the encrypted data key to AWS KMS with a Decrypt request. The request includes encryption context; for more information, see Encryption Context (p. 94).

3. AWS KMS decrypts the encrypted data key using the same CMK that was used to encrypt it, and then sends the decrypted (plaintext) data key to Amazon S3.
4. Amazon S3 uses the plaintext data key to decrypt the encrypted data, and then removes the plaintext data key from memory as soon as possible after use.
5. Amazon S3 sends the decrypted data to the cluster.

## Process for Encrypting Data on EMRFS with CSE-KMS

When you configure an Amazon EMR cluster to use CSE-KMS, the encryption process works like this:

1. When it's ready to store data in Amazon S3, the cluster sends a GenerateDataKey request to AWS KMS, specifying the key ID of the CMK that you chose when you configured the cluster to use CSE-KMS. The request includes encryption context; for more information, see Encryption Context (p. 94).
2. AWS KMS generates a unique data encryption key (data key) and then sends two copies of this data key to the cluster. One copy is unencrypted (plaintext), and the other copy is encrypted under the CMK.
3. The cluster uses the plaintext data key to encrypt the data, and then removes the plaintext data key from memory as soon as possible after use.
4. The cluster combines the encrypted data and the encrypted copy of the data key together into a single encrypted object.
5. The cluster sends the encrypted object to Amazon S3 for storage.

The decryption process works like this:

1. The cluster requests the encrypted data object from an S3 bucket.
2. Amazon S3 sends the encrypted object to the cluster.
3. The cluster extracts the encrypted data key from the encrypted object, and then sends the encrypted data key to AWS KMS with a Decrypt request. The request includes encryption context; for more information, see Encryption Context (p. 94).
4. AWS KMS decrypts the encrypted data key using the same CMK that was used to encrypt it, and then sends the decrypted (plaintext) data key to the cluster.
5. The cluster uses the plaintext data key to decrypt the encrypted data, and then removes the plaintext data key from memory as soon as possible after use.

# Encrypting Data on the Storage Volumes of Cluster Nodes

An Amazon EMR cluster is a collection of Amazon Elastic Compute Cloud (Amazon EC2) instances. Each instance in the cluster is called a *cluster node* or *node*. Each node can have two types of storage volumes: instance store volumes, and Amazon Elastic Block Store (Amazon EBS) volumes. You can configure the cluster to use Linux Unified Key Setup (LUKS) to encrypt both types of storage volumes on the nodes (but not the boot volume of each node). This is called *local disk encryption*.

When you enable local disk encryption for a cluster, you can choose to encrypt the LUKS master key with a CMK in AWS KMS. You must choose a custom CMK that you create; you cannot use an AWS-managed CMK. When you choose a custom CMK, you must ensure that your Amazon EMR cluster has permission to use the CMK. For more information, see Add the EMR Instance Role to an AWS KMS CMK in the *Amazon EMR Release Guide*.

When you enable local disk encryption using a CMK, the encryption process works like this:

1. When each cluster node launches, it sends a GenerateDataKey request to AWS KMS, specifying the key ID of the CMK that you chose when you enabled local disk encryption for the cluster.
2. AWS KMS generates a unique data encryption key (data key) and then sends two copies of this data key to the node. One copy is unencrypted (plaintext), and the other copy is encrypted under the CMK.
3. The node uses a base64-encoded version of the plaintext data key as the password that protects the LUKS master key. The node saves the encrypted copy of the data key on its boot volume.
4. If the node reboots, the rebooted node sends the encrypted data key to AWS KMS with a Decrypt request.
5. AWS KMS decrypts the encrypted data key using the same CMK that was used to encrypt it, and then sends the decrypted (plaintext) data key to the node.
6. The node uses the base64-encoded version of the plaintext data key as the password to unlock the LUKS master key.

# Encryption Context

Each AWS service that is integrated with AWS KMS can specify *encryption context* when it uses AWS KMS to generate data keys or to encrypt or decrypt data. Encryption context is additional authenticated information that AWS KMS uses to check for data integrity. When a service specifies encryption context for an encryption operation, it must specify the same encryption context for the corresponding decryption operation or decryption will fail. Encryption context is also written to AWS CloudTrail log files, which can help you understand why a given CMK was used. For more information about encryption context, see Encryption Context (p. 153).

The following section explain the encryption context that is used in each Amazon EMR encryption scenario that uses a CMK.

## Encryption Context for EMRFS Encryption with SSE-KMS

With SSE-KMS, the Amazon EMR cluster sends data to Amazon S3, and then Amazon S3 uses a CMK to encrypt the data before saving it to an S3 bucket. In this case, Amazon S3 uses the Amazon Resource Name (ARN) of the S3 object as encryption context with each GenerateDataKey and Decrypt request that it sends to AWS KMS. The following example shows a JSON representation of the encryption context that Amazon S3 uses.

```
{ "aws:s3:arn" : "arn:aws:s3:::S3_bucket_name/S3_object_key" }
```

## Encryption Context for EMRFS Encryption with CSE-KMS

With CSE-KMS, the Amazon EMR cluster uses a CMK to encrypt data before sending it to Amazon S3 for storage. In this case, the cluster uses the Amazon Resource Name (ARN) of the CMK as encryption context with each GenerateDataKey and Decrypt request that it sends to AWS KMS. The following example shows a JSON representation of the encryption context that the cluster uses.

```
{ "kms_cmk_id" : "arn:aws:kms:us-
east-2:111122223333:key/0987ab65-43cd-21ef-09ab-87654321cdef" }
```

## Encryption Context for Local Disk Encryption with LUKS

When an Amazon EMR cluster uses local disk encryption with LUKS, the cluster nodes do not specify encryption context with the GenerateDataKey and Decrypt requests that they send to AWS KMS.

# How Amazon Redshift Uses AWS KMS

This topic discusses how Amazon Redshift uses AWS KMS to encrypt data.

Topics



## Amazon Redshift Encryption

An Amazon Redshift data warehouse is a collection of computing resources called nodes, which are organized into a group called a cluster. Each cluster runs an Amazon Redshift engine and contains one or more databases.

Amazon Redshift uses a four-tier, key-based architecture for encryption. The architecture consists of data encryption keys, a database key, a cluster key, and a master key.

Data encryption keys encrypt data blocks in the cluster. Each data block is assigned a randomly-generated AES-256 key. These keys are encrypted by using the database key for the cluster.

The database key encrypts data encryption keys in the cluster. The database key is a randomly-generated AES-256 key. It is stored on disk in a separate network from the Amazon Redshift cluster and passed to the cluster across a secure channel.

The cluster key encrypts the database key for the Amazon Redshift cluster. You can use AWS KMS, AWS CloudHSM, or an external hardware security module (HSM) to manage the cluster key. See the Amazon Redshift Database Encryption documentation for more details.

If the master key resides in AWS KMS, it encrypts the cluster key. You can request encryption by checking the appropriate box in the Amazon Redshift console. You can specify a customer-managed master key to use by choosing one from the list that appears below the encryption box. If you do not specify a customer-managed key, the AWS-managed key for Amazon Redshift under your account will be used.

## Encryption Context

Each service that is integrated with AWS KMS specifies an encryption context when requesting data keys, encrypting, and decrypting. The encryption context is additional authenticated information that AWS KMS uses to check for data integrity. That is, when an encryption context is specified for an encryption operation, the service also specifies it for the decryption operation or decryption will not succeed. Amazon Redshift uses the cluster ID and the creation time for the encryption context. In the `requestParameters` field of a CloudTrail log file, the encryption context will look similar to this.

```
"encryptionContext": {
    "aws:redshift:arn":
 "arn:aws:redshift:region:account_ID:cluster:cluster_name",
    "aws:redshift:createtime": "20150206T1832Z"
},
```

You can search on the cluster name in your CloudTrail logs to understand what operations were performed by using a customer master key. The operations include cluster encryption, cluster decryption, and generating data keys.

For more information, see Encryption Context (p. 153).

# How Amazon Relational Database Service (Amazon RDS) Uses AWS KMS

You can use the Amazon Relational Database Service (Amazon RDS) to set up, operate, and scale a relational database in the cloud. Optionally, you can choose to encrypt the data stored on your Amazon RDS DB instance under a customer master key (CMK) in AWS KMS. To learn how to encrypt your Amazon RDS resources under a KMS CMK, see Encrypting Amazon RDS Resources in the *Amazon Relational Database Service User Guide*.

Amazon RDS builds on Amazon Elastic Block Store (Amazon EBS) encryption to provide full disk encryption for database volumes. For more information about how Amazon EBS uses AWS KMS to encrypt volumes, see How Amazon Elastic Block Store (Amazon EBS) Uses AWS KMS (p. 85).

When you create an encrypted DB instance with Amazon RDS, Amazon RDS creates an encrypted EBS volume on your behalf to store the database. Data stored at rest on the volume, database snapshots, automated backups, and read replicas are all encrypted under the KMS CMK that you specified when you created the DB instance.

## Amazon RDS Encryption Context

When Amazon RDS uses your KMS CMK, or when Amazon EBS uses it on behalf of Amazon RDS, the service specifies an encryption context (p. 153). The encryption context is additional authenticated information that AWS KMS uses to ensure data integrity. That is, when an encryption context is specified for an encryption operation, the service must specify the same encryption context for the decryption operation or decryption will not succeed. The encryption context is also written to your AWS CloudTrail logs to help you understand why a given CMK was used. Your CloudTrail logs might contain many entries describing the use of a CMK, but the encryption context in each log entry can help you determine the reason for that particular use.

At minimum, Amazon RDS always uses the DB instance ID for the encryption context, as in the following JSON-formatted example:

```
{ "aws:rds:db-id": "db-CQYSMDPBRZ7BPMH7Y3RTDG5QY" }
```

This encryption context can help you identify the DB instance for which your CMK was used.

When your CMK is used for a specific DB instance and a specific EBS volume, both the DB instance ID and the EBS volume ID are used for the encryption context, as in the following JSON-formatted example:

```
{
  "aws:rds:db-id": "db-BRG7VYS3SVIFQW7234EJQOM5RQ",
  "aws:ebs:id": "vol-ad8c6542"
}
```

# How Amazon Simple Email Service (Amazon SES) Uses AWS KMS

You can use Amazon Simple Email Service (Amazon SES) to receive email, and (optionally) to encrypt the received email messages before storing them in an Amazon Simple Storage Service (Amazon S3) bucket that you choose. When you configure Amazon SES to encrypt email messages, you must choose the KMS customer master key (CMK) under which Amazon SES encrypts the messages.

You can choose the default CMK in your account for Amazon SES with the alias **aws/ses**, or you can choose a custom CMK that you created separately in AWS KMS.

For more information about receiving email using Amazon SES, go to Receiving Email with Amazon SES in the *Amazon Simple Email Service Developer Guide*.

Topics



# Overview of Amazon SES Encryption Using AWS KMS

When you configure Amazon SES to receive email and encrypt the email messages before saving them to your S3 bucket, the process works like this:

1. You create a receipt rule for Amazon SES, specifying the S3 action, an S3 bucket for storage, and a KMS customer master key (CMK) for encryption.
2. Amazon SES receives an email message that matches your receipt rule.
3. Amazon SES requests a unique data key encrypted with the KMS CMK that you specified in the applicable receipt rule.
4. AWS KMS creates a new data key, encrypts it with the specified CMK, and then sends the encrypted and plaintext copies of the data key to Amazon SES.
5. Amazon SES uses the plaintext data key to encrypt the email message and then removes the plaintext data key from memory as soon as possible after use.
6. Amazon SES puts the encrypted email message and the encrypted data key in the specified S3 bucket. The encrypted data key is stored as metadata with the encrypted email message.

To accomplish steps 3 (p. 97) through 6 (p. 97) in the preceding list, Amazon SES uses the AWS–provided Amazon S3 encryption client. Use the same client to retrieve your encrypted email messages from Amazon S3 and decrypt them. For more information, see Retrieving and Decrypting Email Messages (p. 98).

# Amazon SES Encryption Context

When Amazon SES requests a data key to encrypt your received email messages (step 3 (p. 97) in the Overview of Amazon SES Encryption Using AWS KMS (p. 97)), it includes encryption context in the request. The encryption context provides additional authenticated information that AWS KMS uses to ensure data integrity. The encryption context is also written to your AWS CloudTrail log files, which can help you understand why a given customer master key (CMK) was used. Amazon SES uses the following for the encryption context:

- The ID of the AWS account in which you've configured Amazon SES to receive email messages
- The rule name of the Amazon SES receipt rule that invoked the S3 action on the email message
- The Amazon SES message ID for the email message

The following example shows a JSON representation of the encryption context that Amazon SES uses:

```
{
```

```
  "aws:ses:source-account": "111122223333",
  "aws:ses:rule-name": "example-receipt-rule-name",
  "aws:ses:message-id": "d6iitobk75ur44p8kdnnp7g2n800"
}
```

For more information about encryption context, go to Encryption Context (p. 153).

# Giving Amazon SES Permission to Use Your AWS KMS Customer Master Key (CMK)

You can use the default customer master key (CMK) in your account for Amazon SES with the alias **aws/ses**, or you can use a custom CMK you create. If you use the default CMK for Amazon SES, you don't need to perform any steps to give Amazon SES permission to use it. However, to specify a custom CMK when you add the S3 action to your Amazon SES receipt rule, you must ensure that Amazon SES has permission to use the CMK to encrypt your email messages. To give Amazon SES permission to use your custom CMK, add the following statement to your CMK's key policy (p. 14):

```
{
  "Sid": "Allow SES to encrypt messages using this master key",
  "Effect": "Allow",
  "Principal": {"Service": "ses.amazonaws.com"},
  "Action": [
    "kms:Encrypt",
    "kms:GenerateDataKey*"
  ],
  "Resource": "*",
  "Condition": {
    "Null": {
      "kms:EncryptionContext:aws:ses:rule-name": false,
      "kms:EncryptionContext:aws:ses:message-id": false
    },
    "StringEquals": {"kms:EncryptionContext:aws:ses:source-account":
 "ACCOUNT-ID-WITHOUT-HYPHENS"}
  }
}
```

Replace *ACCOUNT-ID-WITHOUT-HYPHENS* with the 12-digit ID of the AWS account in which you've configured Amazon SES to receive email messages. This policy statement allows Amazon SES to encrypt data with this CMK only under these condition:

- Amazon SES must specify `aws:ses:rule-name` and `aws:ses:message-id` in the `EncryptionContext` of their AWS KMS API requests.
- Amazon SES must specify `aws:ses:source-account` in the `EncryptionContext` of their AWS KMS API requests, and the value for `aws:ses:source-account` must match the AWS account ID specified in the key policy.

For more information about the encryption context that Amazon SES uses when encrypting your email messages, go to Amazon SES Encryption Context (p. 97). For general information about encryption context, go to Encryption Context (p. 153).

# Retrieving and Decrypting Email Messages

Amazon SES does not have permission to decrypt your encrypted email messages and cannot decrypt them for you. You must write code to retrieve your email messages from Amazon S3 and decrypt

them. To make this easier, use the Amazon S3 encryption client. The following AWS SDKs include the Amazon S3 encryption client:

- AWS SDK for Java – See `AmazonS3EncryptionClient` in the *AWS SDK for Java API Reference*.
- AWS SDK for Ruby – See `Aws::S3::Encryption::Client` in the *AWS SDK for Ruby API Reference*.

The Amazon S3 encryption client simplifies the work of constructing the necessary requests to Amazon S3 to retrieve the encrypted email message and to AWS KMS to decrypt the message's encrypted data key, and of decrypting the email message. For example, to successfully decrypt the encrypted data key you must pass the same encryption context that Amazon SES passed when requesting the data key from AWS KMS (step 3 (p. 97) in the Overview of Amazon SES Encryption Using AWS KMS (p. 97)). The Amazon S3 encryption client handles this, and much of the other work, for you.

To see sample code that uses the Amazon S3 encryption client in the AWS SDK for Java to do client-side decryption, go to:

- Example: Client-Side Encryption (Option 1: Using an AWS KMS–Managed Customer Master Key (AWS SDK for Java)) in the *Amazon Simple Storage Service Developer Guide*
- Amazon S3 Encryption with AWS Key Management Service on the AWS Java Development Blog

# How Amazon Simple Storage Service (Amazon S3) Uses AWS KMS

This topic discusses how to protect data at rest within Amazon S3 data centers by using AWS KMS. There are two ways to use AWS KMS with Amazon S3. You can use server-side encryption to protect your data with a customer master key or you can use a AWS KMS customer master key with the Amazon S3 encryption client to protect your data on the client side.

Topics

- Server-Side Encryption: Using SSE-KMS (p. 99)
- Using the Amazon S3 Encryption Client (p. 100)
- Encryption Context (p. 100)

## Server-Side Encryption: Using SSE-KMS

You can protect data at rest in Amazon S3 by using three different modes of server-side encryption: SSE-S3, SSE-C, or SSE-KMS.

- SSE-S3 requires that Amazon S3 manage the data and master encryption keys. For more information about SSE-S3, see Protecting Data Using Server-Side Encryption with AWS-Managed Encryption Keys.
- SSE-C requires that you manage the encryption key. For more information about SSE-C, see Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys (SSE-C).
- SSE-KMS requires that AWS manage the data key but you manage the master key in AWS KMS. The remainder of this topic discusses how to protect data by using server-side encryption with AWS KMS-managed keys (SSE-KMS).

You can request encryption and the master key you want by using the Amazon S3 console or API. In the console, check the appropriate box to perform encryption and select your key from the list. For the Amazon S3 API, specify encryption and choose your key by setting the appropriate headers in a GET

or PUT request. For more information, see Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS).

You can choose a specific customer-managed master key or accept the AWS-managed key for Amazon S3 under your account. If you choose to encrypt your data, AWS KMS and Amazon S3 perform the following actions:

- Amazon S3 requests a plaintext data key and a copy of the key encrypted by using the specified customer-managed master key or the AWS-managed master key.
- AWS KMS creates a data key, encrypts it by using the master key, and sends both the plaintext data key and the encrypted data key to Amazon S3.
- Amazon S3 encrypts the data using the data key and removes the plaintext key from memory as soon as possible after use.
- Amazon S3 stores the encrypted data key as metadata with the encrypted data.

Amazon S3 and AWS KMS perform the following actions when you request that your data be decrypted.

- Amazon S3 sends the encrypted data key to AWS KMS.
- AWS KMS decrypts the key by using the appropriate master key and sends the plaintext key back to Amazon S3.
- Amazon S3 decrypts the ciphertext and removes the plaintext data key from memory as soon as possible.

# Using the Amazon S3 Encryption Client

You can use the Amazon S3 encryption client in the AWS SDK from your own application to encrypt objects and upload them to Amazon S3. This method allows you to encrypt your data locally to ensure its security as it passes to the Amazon S3 service. The S3 service receives your encrypted data and does not play a role in encrypting or decrypting it.

The Amazon S3 encryption client encrypts the object by using envelope encryption. The client calls AWS KMS as a part of the encryption call you make when you pass your data to the client. AWS KMS verifies that you are authorized to use the customer master key and, if so, returns a new plaintext data key and the data key encrypted under the customer master key. The encryption client encrypts the data by using the plaintext key and then deletes the key from memory. The encrypted data key is sent to Amazon S3 to store alongside your encrypted data.

# Encryption Context

Each service that is integrated with AWS KMS specifies an encryption context when requesting data keys, encrypting, and decrypting. The encryption context is additional authenticated information that AWS KMS uses to check for data integrity. That is, when an encryption context is specified for an encryption operation, the service also specifies it for the decryption operation or decryption will not succeed. If you are using SSE-KMS or the Amazon S3 encryption client to perform encryption, Amazon S3 uses the bucket path as the encryption context. In the `requestParameters` field of a CloudTrail log file, the encryption context will look similar to this.

```
"encryptionContext": {
    "aws:s3:arn": "arn:aws:s3:::bucket_name/file_name"
},
```

For more information, see Encryption Context (p. 153).

# How Amazon WorkMail Uses AWS KMS

This topic discusses how Amazon WorkMail uses AWS KMS to encrypt email messages.

Topics



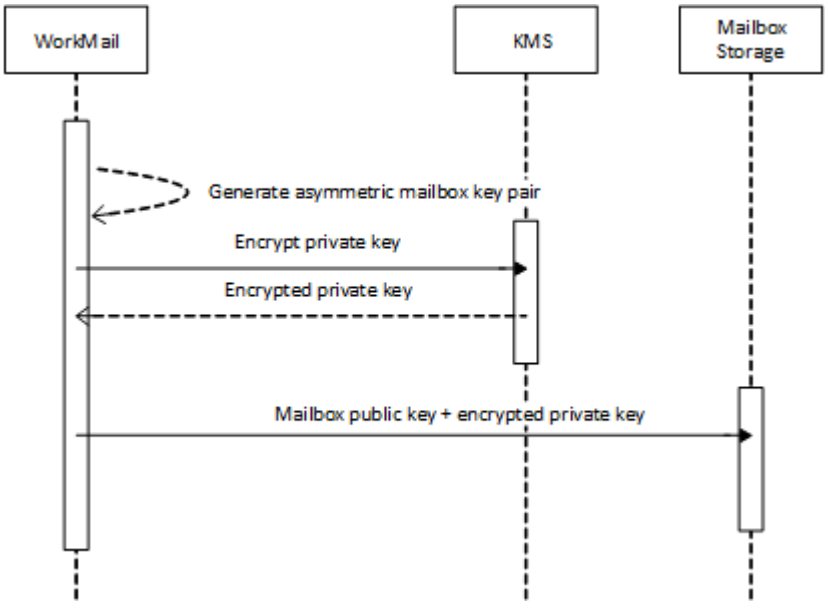
## Amazon WorkMail Overview

Amazon WorkMail is an email service in the cloud that provides a cost-effective way for your organization to receive and send email and use calendars. Amazon WorkMail supports existing desktop and mobile clients and integrates with your existing corporate directory. Users can leverage their existing credentials to sign on to their email by using Microsoft Outlook, a mobile device, or a browser.

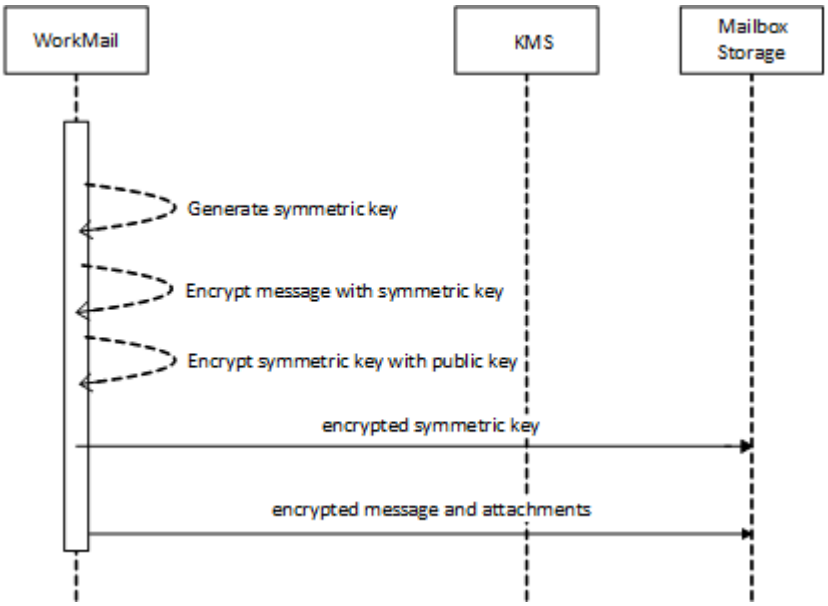
Using the Amazon WorkMail console, you can create an Amazon WorkMail organization and optionally assign to it one or more email domains that you own. Then you can create new email users and email distribution groups. Users can then send and receive messages. The messages are encrypted and stored until ready to be viewed.

## Amazon WorkMail Encryption

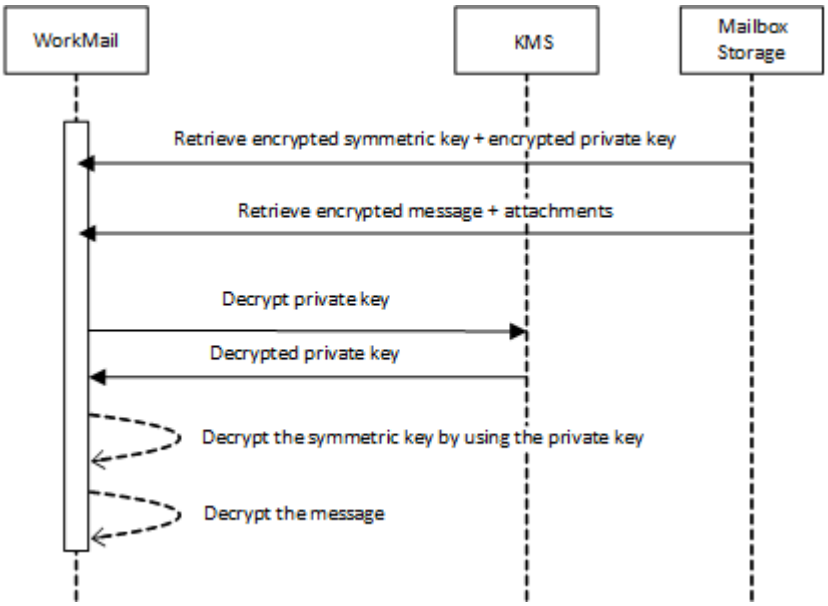
Each end user you create is associated with one mailbox. Amazon WorkMail creates an asymmetric key pair for each mailbox and sends the private key portion of the key pair to AWS KMS to be encrypted under a customer master key (CMK). The CMK can be a custom key that you choose for your organization or the default Amazon WorkMail service CMK. The encrypted private key and unencrypted public key is then saved for later use.




Each message received is encrypted by using a symmetric key dynamically generated by Amazon WorkMail. The symmetric key is then encrypted by using the public key associated with the user's mailbox. The encrypted symmetric key and the encrypted message and attachments are then stored.




In asymmetric cryptography, data that is encrypted by using the public key can be decrypted only by using the corresponding private key. As mentioned above, however, Amazon WorkMail encrypts the private key by using an AWS KMS CMK. To make the private key ready to use, it must therefore be decrypted by using the same CMK used to encrypt it. Thus, when a user is ready to retrieve email messages, Amazon WorkMail sends the private key to AWS KMS for decryption and uses the plaintext private key returned by AWS KMS to decrypt the symmetric key that was used to encrypt the email message. Amazon WorkMail then uses the symmetric key to decrypt the message before presenting it to the user.




# Amazon WorkMail Encryption Context

Each service that is integrated with AWS KMS specifies an encryption context when requesting data keys, encrypting, and decrypting. The encryption context is additional authenticated information that AWS KMS uses to check for data integrity. That is, when an encryption context is specified for an

encryption operation, the service also specifies it for the decryption operation or decryption will not succeed. The encryption context is written to your CloudTrail logs to help you understand why a given AWS KMS key was used. Amazon WorkMail uses the organization ID for the encryption context. In the `requestParameters` field of a CloudTrail log file, the encryption context will look similar to this.

```
"encryptionContext": {
    "aws:workmail:arn":"arn:aws:workmail:region:account
 ID:organization/organization ID"
}
```

The organization ID is a unique identifier that Amazon WorkMail generates when an organization is created. A customer can have multiple organizations in an AWS account. The following example shows an organization ID in the us-east-1 region.

```
    "aws:workmail:arn":"arn:aws:workmail:us-east-1:123456789012:organization/
m-68755160c4cb4e29a2b2f8fb58f359d7"
```

For more information about the encryption context, see Encryption Context (p. 153).

# How Amazon WorkSpaces Uses AWS KMS

You can use Amazon WorkSpaces to provision a cloud-based desktop (a *WorkSpace*) for each of your end users. When you launch a new WorkSpace, you can choose to encrypt its volumes and decide which AWS KMS customer master key (CMK) to use for the encryption. You can choose your account's default CMK for Amazon WorkSpaces (use the alias **aws/workspaces**), or you can choose a custom CMK that you created separately in AWS KMS.

> **Note**
> You can encrypt up to 30 WorkSpace volumes under a single CMK. This limit applies to the default CMK and to custom CMKs.

For more information about creating WorkSpaces with encrypted volumes, go to Encrypt a WorkSpace in the *Amazon WorkSpaces Administration Guide*.

Topics



## Overview of Amazon WorkSpaces Encryption Using AWS KMS

When you create WorkSpaces with encrypted volumes, Amazon WorkSpaces uses Amazon Elastic Block Store (Amazon EBS) to create and manage those volumes. Both services use your KMS customer master key (CMK) to work with the encrypted volumes. For more information about EBS volume encryption, see the following documentation:

- How Amazon Elastic Block Store (Amazon EBS) Uses AWS KMS (p. 85) in this guide
- Amazon EBS Encryption in the *Amazon EC2 User Guide for Windows Instances*

When you launch WorkSpaces with encrypted volumes, the end-to-end process works like this:

1. You specify the CMK to use for encryption as well as the WorkSpace's user and directory. This action creates a grant (p. 43) that allows Amazon WorkSpaces to use your CMK only for this WorkSpace—that is, only for the WorkSpace associated with the specified user and directory.
2. Amazon WorkSpaces creates an encrypted EBS volume for the WorkSpace and specifies the CMK to use as well as the volume's user and directory (the same information that you specified at step 1 (p. 104)). This action creates a grant (p. 43) that allows Amazon EBS to use your CMK only for this WorkSpace and volume—that is, only for the WorkSpace associated with the specified user and directory, and only for the specified volume.
3. Amazon EBS requests a volume data key that is encrypted under your CMK and specifies the WorkSpace user's `Sid` and directory ID as well as the volume ID as encryption context.
4. AWS KMS creates a new data key, encrypts it under your CMK, and then sends the encrypted data key to Amazon EBS.
5. Amazon WorkSpaces uses Amazon EBS to attach the encrypted volume to your WorkSpace, at which time Amazon EBS sends the encrypted data key to AWS KMS with a `Decrypt` request and specifies the WorkSpace user's `Sid` and directory ID as well as the volume ID as encryption context.
6. AWS KMS uses your CMK to decrypt the data key, and then sends the plaintext data key to Amazon EBS.
7. Amazon EBS uses the plaintext data key to encrypt all data going to and from the encrypted volume. Amazon EBS keeps the plaintext data key in memory for as long as the volume is attached to the WorkSpace.
8. Amazon EBS stores the encrypted data key (received at step 4 (p. 104)) with the volume metadata for future use in case you reboot or rebuild the WorkSpace.
9. When you use the AWS Management Console to remove a WorkSpace (or use the `TerminateWorkspaces` action in the Amazon WorkSpaces API), Amazon WorkSpaces and Amazon EBS retire the grants that allowed them to use your CMK for that WorkSpace.

# Amazon WorkSpaces Encryption Context

Amazon WorkSpaces doesn't use your customer master key (CMK) directly for cryptographic operations (such as `Encrypt`, `Decrypt`, `GenerateDataKey`, etc.), which means Amazon WorkSpaces doesn't send requests to AWS KMS that include encryption context. However, when Amazon EBS requests an encrypted data key for the encrypted volumes of your WorkSpaces (step 3 (p. 104) in the Overview of Amazon WorkSpaces Encryption Using AWS KMS (p. 103)) and when it requests a plaintext copy of that data key (step 5 (p. 104)), it includes encryption context in the request. The encryption context provides additional authenticated information that AWS KMS uses to ensure data integrity. The encryption context is also written to your AWS CloudTrail log files, which can help you understand why a given customer master key (CMK) was used. Amazon EBS uses the following for the encryption context:

- The `sid` of the AWS Directory Service user that is associated with the WorkSpace
- The directory ID of the AWS Directory Service directory that is associated with the WorkSpace
- The volume ID of the encrypted volume

The following example shows a JSON representation of the encryption context that Amazon EBS uses:

```
{
  "aws:workspaces:sid-directoryid":
 "[S-1-5-21-277731876-1789304096-451871588-1107]@[d-1234abcd01]",
  "aws:ebs:id": "vol-1234abcd"
}
```

For more information about encryption context, see Encryption Context (p. 153).

# Giving Amazon WorkSpaces Permission to Use A CMK On Your Behalf

You can use your account's default customer master key (CMK) for Amazon WorkSpaces with the alias **aws/workspaces**, or you can use a custom CMK that you create. If you use the default CMK for Amazon WorkSpaces, you don't need to perform any steps to give Amazon WorkSpaces permission to use it. AWS KMS automatically specifies the necessary permissions in the key policy (p. 14) for the default CMK.

To use a custom CMK, the WorkSpaces administrators who create WorkSpaces with encrypted volumes must have permission to use the CMK. The WorkSpaces administrators don't use the CMK directly. Simply creating a WorkSpace with encrypted volumes implicitly creates the grant (p. 43) that gives Amazon WorkSpaces permission to use the CMK on the administrator's behalf.

Even though the WorkSpaces administrators don't use the CMK directly, they need permission to use the CMK because they can only grant permissions that they have. To give WorkSpaces administrators permission to use a CMK, do these things:

1. Add the WorkSpaces administrators to the list of key users in the CMK's key policy (p. 105)
2. Give the WorkSpaces administrators extra permissions with an IAM policy (p. 106)

WorkSpaces administrators also need permission to use Amazon WorkSpaces. For more information about these permissions, go to Controlling Access to Amazon WorkSpaces Resources in the *Amazon WorkSpaces Administration Guide*.

## Part 1: Adding WorkSpaces Administrators to a CMK's Key Users

To add WorkSpaces administrators to the list of key users in a CMK's key policy, you can use the AWS Management Console or the AWS Command Line Interface (AWS CLI).

### To add WorkSpaces administrators as key users for a CMK (console)

1. Open the **Encryption Keys** section of the Identity and Access Management (IAM) console at https://console.aws.amazon.com/iam/home#encryptionKeys.
2. For **Filter**, choose the appropriate AWS region. Do not use the region selector in the menu bar (top right corner).
3. Choose the alias of the CMK that WorkSpaces administrators will use.
4. In the **Key Policy** section, under **Key Users**, choose **Add**.
5. In the list of IAM users and roles, select the users and roles that correspond to your WorkSpaces administrators, and then choose **Attach**.

### To add WorkSpaces administrators as key users for a CMK (AWS CLI)

1. Use the `aws kms get-key-policy` command to retrieve the existing key policy, and then save the policy document to a file.
2. Open the policy document in your preferred text editor. Add the IAM users and roles that correspond to your WorkSpaces administrators to the policy statements that give permission to key users (p. 18). Then save the file.
3. Use the `aws kms put-key-policy` command to apply the key policy to the CMK.

## Part 2: Giving WorkSpaces Administrators Extra Permissions with an IAM Policy

In addition to the permissions in the key users section of the default key policy (p. 15), WorkSpaces administrators need some permissions in an IAM user policy that applies to them. For information about creating and editing IAM user policies, go to Working with Managed Policies and Working with Inline Policies in the *IAM User Guide*.

At minimum, WorkSpaces administrators need permission to create grants (p. 43) for the custom CMK(s) that they will use with Amazon WorkSpaces. To use the AWS Management Console to create WorkSpaces with encrypted volumes, WorkSpaces administrators also need permission to list aliases and list keys, which are actions that the console performs on behalf of WorkSpaces administrators to display a list of available CMKs.

To give these permissions to your WorkSpaces administrators, add an IAM policy similar to the following example to your WorkSpaces administrators. Replace *arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab* in the first policy statement with the ARN(s) of the CMK(s) that WorkSpaces administrators will use when they create WorkSpaces with encrypted volumes. If your WorkSpaces administrators will launch WorkSpaces with only the Amazon WorkSpaces API (not with the console), you can omit the second statement with the `"kms:ListAliases"` and `"kms:ListKeys"` permissions.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "kms:CreateGrant",
      "Resource": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:ListAliases",
        "kms:ListKeys"
      ],
      "Resource": "*"
    }
  ]
}
```

# Monitoring Customer Master Keys

Monitoring is an important part of understanding the availability, state, and usage of your customer master keys (CMKs) in AWS KMS and maintaining the reliability, availability, and performance of your AWS solutions. Collecting monitoring data from all the parts of your AWS solution will help you debug a multipoint failure if one occurs. Before you start monitoring your CMKs, however, create a monitoring plan that includes answers to the following questions:

- What are your monitoring goals?
- What resources will you monitor?
- How often will you monitor these resources?
- What monitoring tools (p. 108) will you use?
- Who will perform the monitoring tasks?
- Who should be notified when something happens?

The next step is to monitor your CMKs over time to establish a baseline for normal AWS KMS usage and expectations in your environment. As you monitor your CMKs, store historical monitoring data so that you can compare it with current data, identify normal patterns and anomalies, and devise methods to address issues.

For example, you can monitor AWS KMS API activity and events that affect your CMKs. When data falls above or below your established norms, you might need to investigate or take corrective action.

To establish a baseline for normal patterns, monitor the following items:

- AWS KMS API activity for *data plane* operations. These are cryptographic operations that use a CMK, such as Decrypt, Encrypt, ReEncrypt, and GenerateDataKey.
- AWS KMS API activity for *control plane* operations that are important to you. These operations manage a CMK, and you might want to monitor those that change a CMK's availability (such as ScheduleKeyDeletion, CancelKeyDeletion, DisableKey, EnableKey, ImportKeyMaterial, and DeleteImportedKeyMaterial) or modify a CMK's access control (such as PutKeyPolicy and RevokeGrant).
- Other AWS KMS metrics (such as the amount of time remaining until your imported key material (p. 51) expires) and events (such as the expiration of imported key material or the deletion or key rotation of a CMK).

# Monitoring Tools

AWS provides various tools that you can use to monitor your CMKs. You can configure some of these tools to do the monitoring for you, while some of the tools require manual intervention. We recommend that you automate monitoring tasks as much as possible.

## Automated Monitoring Tools

You can use the following automated monitoring tools to watch your CMKs and report when something has changed.

- **Amazon CloudWatch Alarms** – Watch a single metric over a time period that you specify, and perform one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon Simple Notification Service (Amazon SNS) topic or Auto Scaling policy. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods. For more information, see Monitoring with Amazon CloudWatch (p. 109).
- **Amazon CloudWatch Events** – Match events and route them to one or more target functions or streams to capture state information and, if necessary, make changes or take corrective action. For more information, see AWS KMS Events (p. 112) and the Amazon CloudWatch Events User Guide.
- **Amazon CloudWatch Logs** – Monitor, store, and access your log files from AWS CloudTrail or other sources. For more information, see the Amazon CloudWatch Logs User Guide.
- **AWS CloudTrail Log Monitoring** – Share log files between accounts, monitor CloudTrail log files in real time by sending them to CloudWatch Logs, write log processing applications in Java, and validate that your log files have not changed after delivery by CloudTrail. For more information, see Working with CloudTrail Log Files in the *AWS CloudTrail User Guide*.

## Manual Monitoring Tools

Another important part of monitoring CMKs involves manually monitoring those items that the CloudWatch alarms and events don't cover. The AWS KMS, CloudWatch, AWS Trusted Advisor, and other AWS console dashboards provide an at-a-glance view of the state of your AWS environment.

The AWS KMS console dashboard shows the following information about each CMK:

- Status
- Creation date
- Origin
- Expiration date (for CMKs whose origin is `EXTERNAL`)
- Scheduled deletion date (for CMKs that are pending deletion)

The CloudWatch console dashboard shows the following:

- Current alarms and status
- Graphs of alarms and resources
- Service health status

In addition, you can use CloudWatch to do the following:

- Create customized dashboards to monitor the services you care about
- Graph metric data to troubleshoot issues and discover trends
- Search and browse all your AWS resource metrics
- Create and edit alarms to be notified of problems

AWS Trusted Advisor can help you monitor your AWS resources to improve performance, reliability, security, and cost effectiveness. Four Trusted Advisor checks are available to all users; more than 50 checks are available to users with a Business or Enterprise support plan. For more information, see AWS Trusted Advisor.

# Monitoring with Amazon CloudWatch

You can monitor your customer master keys (CMKs) using Amazon CloudWatch, which collects and processes raw data from AWS KMS into readable, near real-time metrics. These data are recorded for a period of two weeks so that you can access historical information and gain a better understanding of the usage of your CMKs and their changes over time. For more information about Amazon CloudWatch, see the Amazon CloudWatch User Guide.

Topics



## AWS KMS Metrics and Dimensions

When you import key material into a CMK (p. 51) and set it to expire, AWS KMS sends metrics and dimensions to CloudWatch. You can view the AWS KMS metrics using the AWS Management Console and the Amazon CloudWatch API.

### AWS KMS Metrics

The `AWS/KMS` namespace includes the following metrics.

**SecondsUntilKeyMaterialExpiration**
: This metric tracks the number of seconds remaining until imported key material expires. This metric is valid only for CMKs whose origin is `EXTERNAL` and whose key material is or was set to expire. The most useful statistic for this metric is `Minimum`, which tells you the smallest amount of time remaining for all data points in the specified statistic period. The only valid unit for this metric is `Seconds`.

  Use this metric to track the amount of time that remains until your imported key material expires. When that amount of time falls below a threshold that you define, you might want to take action such as reimporting the key material with a new expiration date. You can create a CloudWatch alarm to notify you when that happens. For more information, see Creating CloudWatch Alarms to Monitor AWS KMS Metrics (p. 110).

### Dimensions for AWS KMS Metrics

AWS KMS metrics use the `AWS/KMS` namespace and have only one valid dimension: `KeyId`. You can use this dimension to view metric data for a specific CMK or set of CMKs.

## How Do I View AWS KMS Metrics?

You can view the AWS KMS metrics using the AWS Management Console and the Amazon CloudWatch API.

**To view metrics using the CloudWatch console**

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. If necessary, change the region. From the navigation bar, select the region where your AWS resources reside.
3. In the navigation pane, choose **Metrics**.
4. In the content pane, choose the **All metrics** tab. Then, below **AWS Namespaces**, choose **KMS**.
5. Choose **Per-Key Metrics** to view the individual metrics and dimensions.

To view metrics using the Amazon CloudWatch API

To view AWS KMS metrics using the CloudWatch API, send a ListMetrics request with `Namespace` set to `AWS/KMS`. The following example shows how to do this with the AWS Command Line Interface (AWS CLI).

```
$ aws cloudwatch list-metrics --namespace AWS/KMS
```

# Creating CloudWatch Alarms to Monitor AWS KMS Metrics

You can create a CloudWatch alarm that sends an Amazon SNS message when the value of the metric changes and causes the alarm to change state. An alarm watches a single metric over a time period you specify, and performs one or more actions based on the value of the metric relative to a given threshold over a number of time periods. The action is a notification sent to an Amazon SNS topic or Auto Scaling policy. Alarms invoke actions for sustained state changes only. CloudWatch alarms do not invoke actions simply because they are in a particular state; the state must have changed and been maintained for a specified number of periods.

Topics



## Create a CloudWatch Alarm to Monitor the Expiration of Imported Key Material

When you import key material into a CMK (p. 51), you can optionally specify a time at which the key material expires. When the key material expires, AWS KMS deletes the key material and the CMK becomes unusable. To use the CMK again, you must reimport key material. You can create a CloudWatch alarm to notify you when the amount of time that remains until your imported key material expires falls below a threshold that you define (for example, 10 days). If you receive a notification from such an alarm, you might want to take action such as reimporting the key material with a new expiration date.

**To create an alarm to monitor the expiration of imported key material (AWS Management Console)**

1. Open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.

2. If necessary, change the region. From the navigation bar, select the region where your AWS resources reside.
3. In the navigation pane, choose **Alarms**. Then choose **Create Alarm**.
4. Choose **Browse Metrics** and then choose **KMS**.
5. Select the check box next to the key ID of the CMK to monitor.
6. In the lower pane, use the menus to change the statistic to **Minimum** and the time period to **1 Minute**. Then choose **Next**.
7. In the **Create Alarm** window, do the following:

   a. For **Name**, type a name such as **`KeyMaterialExpiresSoon`**.

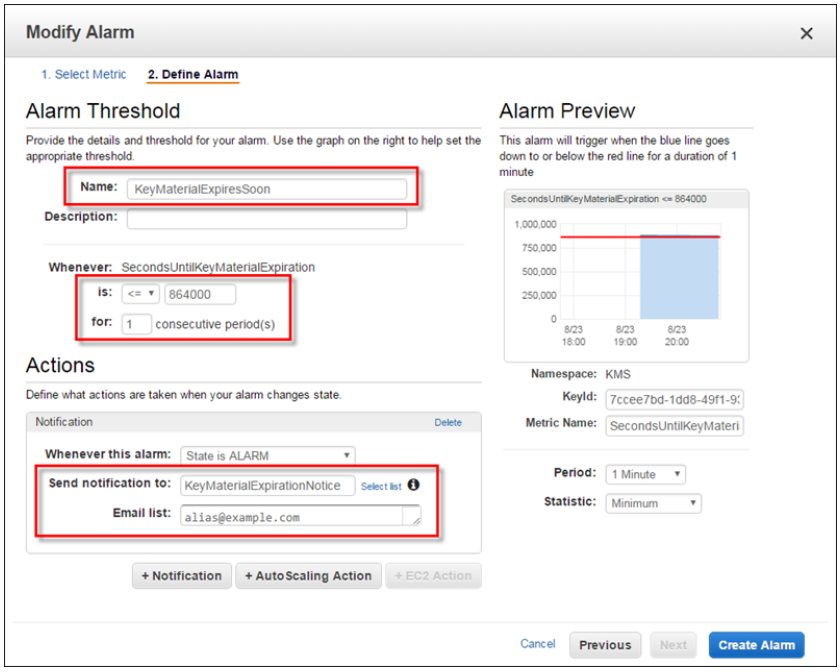
   b. Following **Whenever:**, for **is:**, choose **<=** and then type the number of seconds for your threshold value. For example, to be notified when the time that remains until your imported key material expires is 10 days or less, type **`864000`**.

   c. For **for consecutive period(s)**, if necessary, type **1**.

   d. For **Send notification to:**, do one of the following:

      - To use a new Amazon SNS topic, choose **New list** and then type a new topic name. For **Email list:**, type at least one email address. You can type more than one email address by separating them with commas.
      - To use an existing Amazon SNS topic, choose the name of the topic to use.

   e. Choose **Create Alarm**.




8. If you chose to send notifications to an email address, open the email message you receive from no-reply@sns.amazonaws.com with subject "AWS Notification - Subscription Confirmation." Confirm your email address by choosing the **Confirm subscription** link in the email message.

   **Important**
   You will not receive email notifications until after you have confirmed your email address.

## Create a CloudWatch Alarm to Monitor Usage of CMKs that are Pending Deletion

When you schedule key deletion (p. 63) for a CMK, AWS KMS enforces a waiting period before deleting the CMK. You can use the waiting period to ensure that you don't need the CMK now or in the future. You can also configure a CloudWatch alarm to warn you if a person or application attempts to use the CMK during the waiting period. If you receive a notification from such an alarm, you might want to cancel deletion of the CMK.

For more information, see Creating an Amazon CloudWatch Alarm to Detect Usage of a Customer Master Key that is Pending Deletion (p. 69).

# AWS KMS Events

AWS KMS integrates with Amazon CloudWatch Events to notify you of certain events that affect your CMKs. Each event is represented in JSON (JavaScript Object Notation) and contains the event name, the date and time when the event occurred, the CMK affected, and more. You can use CloudWatch Events to collect these events and set up rules that route them to one or more *targets* such as AWS Lambda functions, Amazon SNS topics, Amazon SQS queues, streams in Amazon Kinesis Streams, or built-in targets.

For more information about using CloudWatch Events with other kinds of events, including those emitted by AWS CloudTrail when it records a read/write API request, see the Amazon CloudWatch Events User Guide.

For more information about the AWS KMS events, see the following topics.

Topics

- KMS CMK Rotation (p. 112)
- KMS Imported Key Material Expiration (p. 113)
- KMS CMK Deletion (p. 113)

## KMS CMK Rotation

When you enable annual rotation of a CMK's key material (p. 50), AWS KMS creates new key material for the CMK each year and sends a corresponding event to CloudWatch Events. The following is an example of this event.

```
{
  "version": "0",
  "id": "6a7e8feb-b491-4cf7-a9f1-bf3703467718",
  "detail-type": "KMS CMK Rotation",
  "source": "aws.kms",
  "account": "111122223333",
  "time": "2016-08-25T21:05:33Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  ],
  "detail": {
    "key-id": "1234abcd-12ab-34cd-56ef-1234567890ab"
  }
}
```

## KMS Imported Key Material Expiration

When you import key material into a CMK (p. 51), you can optionally specify a time at which the key material expires. When the key material expires, AWS KMS deletes the key material and sends a corresponding event to CloudWatch Events. The following is an example of this event.

```
{
  "version": "0",
  "id": "9da9af57-9253-4406-87cb-7cc400e43465",
  "detail-type": "KMS Imported Key Material Expiration",
  "source": "aws.kms",
  "account": "111122223333",
  "time": "2016-08-22T20:12:19Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  ],
  "detail": {
    "key-id": "1234abcd-12ab-34cd-56ef-1234567890ab"
  }
}
```

## KMS CMK Deletion

When you schedule key deletion (p. 63) for a CMK, AWS KMS enforces a waiting period before deleting the CMK. After the waiting period ends, AWS KMS deletes the CMK and sends a corresponding event to CloudWatch Events. The following is an example of this event.

```
{
  "version": "0",
  "id": "e9ce3425-7d22-412a-a699-e7a5fc3fbc9a",
  "detail-type": "KMS CMK Deletion",
  "source": "aws.kms",
  "account": "111122223333",
  "time": "2016-08-19T03:23:45Z",
  "region": "us-west-2",
  "resources": [
    "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
  ],
  "detail": {
    "key-id": "1234abcd-12ab-34cd-56ef-1234567890ab"
  }
}
```

# Logging AWS KMS API Calls Using AWS CloudTrail

AWS KMS is integrated with CloudTrail, a service that captures API calls made by or on behalf of AWS KMS in your AWS account and delivers the log files to an Amazon S3 bucket that you specify. CloudTrail captures API calls from the AWS KMS console or from the AWS KMS API. Using the information collected by CloudTrail, you can determine what request was made, the source IP address from which the request was made, who made the request, when it was made, and so on. To learn more about CloudTrail, including how to configure and enable it, see the AWS CloudTrail User Guide.

When you enable CloudTrail logging in your AWS account, API calls made to AWS KMS actions are tracked in log files. AWS KMS records are written together with other AWS service records in a log file. CloudTrail determines when to create and write to a new log file based on a time period and file size.

CloudTrail logs all of the AWS KMS actions. For example, calls to the `CreateKey`, `Encrypt`, and `Decrypt` actions generate entries in the CloudTrail log files.

Every log entry contains information about who generated the request. The user identity information in the log helps you determine whether the request was made with root or IAM user credentials, with temporary security credentials for a role or federated user, or by another AWS service. For more information, see CloudTrail userIdentity Element in the CloudTrail Event Reference chapter in the *AWS CloudTrail User Guide*.

You can store your log files in your bucket for as long as you want, but you can also define Amazon S3 lifecycle rules to archive or delete log files automatically. By default, your log files are encrypted by using Amazon S3 server-side encryption (SSE) with a key managed by the Amazon S3 service.

You can choose to have CloudTrail publish Amazon SNS notifications when new log files are delivered if you want to take quick action upon log file delivery. For more information, see Configuring Amazon SNS Notifications for CloudTrail in the *AWS CloudTrail User Guide*.

You can also aggregate AWS KMS log files from multiple AWS regions and multiple AWS accounts into a single Amazon S3 bucket. For more information, see Receiving CloudTrail Log Files from Multiple Regions in the *AWS CloudTrail User Guide*.

CloudTrail log files can contain one or more log entries where each entry is made up of multiple JSON-formatted events. A log entry represents a single request from any source and includes information about the requested action, any parameters, the date and time of the action, and so on. The log entries are not guaranteed to be in any particular order. That is, they are not an ordered stack trace of the

public API calls. For more information about the fields that make up a log entry, see the CloudTrail Event Reference in the *AWS CloudTrail User Guide*.

For examples of what these CloudTrail log entries look like, see the following topics.



# CreateAlias

The following example shows a log file generated by calling `CreateAlias`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice",
            "sessionContext": {
                "attributes": {
                    "mfaAuthenticated": "false",
                    "creationDate": "2014-11-04T00:52:27Z"
                }
            }
        },
        "eventTime": "2014-11-04T00:52:27Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "CreateAlias",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
```

```
        "requestParameters": {
            "aliasName": "alias/my_alias",
            "targetKeyId": "arn:aws:kms:us-
east-1:123456789012:key/64e07f97-2489-4d04-bfdf-41723ad130bd"
        },
        "responseElements": null,
        "requestID": "d9472f40-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "f72d3993-864f-48d6-8f16-e26e1ae8dff0",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-
east-1:123456789012:key/64e07f97-2489-4d04-bfdf-41723ad130bd",
            "accountId": "123456789012"
        },
        {
            "ARN": "arn:aws:kms:us-east-1:123456789012:alias/my_alias",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# CreateGrant

The following example shows a log file generated by calling `CreateGrant`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:53:12Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "CreateGrant",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "keyId": "arn:aws:kms:us-east-1:123456789012:key/65f61d18-
c45c-41ca-90c9-179982e9b716",
            "constraints": {
                "encryptionContextSubset": {
                    "ContextKey1": "Value1"
                }
            },
            "operations": ["Encrypt",
            "RetireGrant"],
```

```
            "granteePrincipal": "EX_PRINCIPAL_ID"
        },
        "responseElements": {
            "grantId":
 "f020fe75197b93991dc8491d6f19dd3cebb24ee62277a05914386724f3d48758"
        },
        "requestID": "f3c08808-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "5d529779-2d27-42b5-92da-91aaea1fc4b5",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/65f61d18-
c45c-41ca-90c9-179982e9b716",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# CreateKey

The following example shows a log file generated by calling `CreateKey`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:59Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "CreateKey",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "policy": "{\n  \"Version\":\"2012-10-17\",\n  \"Statement
\":[{\n    \"Effect\":\"Allow\",\n    \"Principal\":{\"AWS\":
\"arn:aws:iam::123456789012:user/Alice\"},\n    \"Action\":\"kms:*\",\n
   \"Resource\":\"*\"\n  }, {\n    \"Effect\":\"Allow\",\n    \"Principal
\":{\"AWS\":\"arn:aws:iam::012345678901:user/Bob\"},\n    \"Action\":
\"kms:CreateGrant\",\n    \"Resource\":\"*\"\n  }, {\n    \"Effect\":\"Allow
\",\n    \"Principal\":{\"AWS\":\"arn:aws:iam::012345678901:user/Charlie\"},
\n    \"Action\":\"kms:Encrypt\",\n    \"Resource\":\"*\"\n}]\n}",
            "description": "",
            "keyUsage": "ENCRYPT_DECRYPT"
        },
        "responseElements": {
            "keyMetadata": {
```

```
                "AWSAccountId": "123456789012",
                "enabled": true,
                "creationDate": "Nov 4, 2014 12:52:59 AM",
                "keyId": "06dc80ca-1bdc-4d0b-be5b-b7009cd14f13",
                "keyUsage": "ENCRYPT_DECRYPT",
                "description": "",
                "arn": "arn:aws:kms:us-
east-1:123456789012:key/06dc80ca-1bdc-4d0b-be5b-b7009cd14f13"
            }
        },
        "requestID": "ebe8ee68-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "ba116326-1792-4784-87dd-a688d1cb42ec",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-
east-1:123456789012:key/06dc80ca-1bdc-4d0b-be5b-b7009cd14f13",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# Decrypt

The following example shows a log file generated by calling `Decrypt`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:20Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "Decrypt",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "errorCode": "InvalidCiphertextException",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "d5239dea-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "954983cf-7da9-4adf-aeaa-261a1292c0aa",
        "readOnly": true,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/e17cebae-
e7a6-4864-b92f-0365f2feff38",
            "accountId": "123456789012"
```

```
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# DeleteAlias

The following example shows a log file generated by calling `DeleteAlias`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice",
            "sessionContext": {
                "attributes": {
                    "mfaAuthenticated": "false",
                    "creationDate": "2014-11-04T00:52:27Z"
                }
            }
        },
        "eventTime": "2014-11-04T00:52:27Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "DeleteAlias",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "aliasName": "alias/my_alias"
        },
        "responseElements": null,
        "requestID": "d9542792-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "12f48554-bb04-4991-9cfc-e7e85f68eda0",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:alias/my_alias",
            "accountId": "123456789012"
        },
        {
            "ARN": "arn:aws:kms:us-
east-1:123456789012:key/64e07f97-2489-4d04-bfdf-41723ad130bd",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# DescribeKey

The following example shows a log file generated by multiple calls to `DescribeKey` in response to viewing keys in the IAM management console.


```
{
  "Records": [
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T20:51:21Z"
          }
        },
        "invokedBy": "signin.amazonaws.com"
      },
      "eventTime": "2014-11-05T20:51:34Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "DescribeKey",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "signin.amazonaws.com",
      "requestParameters": {
        "keyId": "30a9a1e7-2a84-459d-9c61-04cbeaebab95"
      },
      "responseElements": null,
      "requestID": "874d4823-652d-11e4-9a87-01af2a1ddecb",
      "eventID": "f715da9b-c52c-4824-99ae-88aa1bb58ae4",
      "readOnly": true,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-
east-1:123456789012:key/30a9a1e7-2a84-459d-9c61-04cbeaebab95",
          "accountId": "123456789012"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice",
        "sessionContext": {
          "attributes": {
```

```
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T20:51:21Z"
          }
        },
        "invokedBy": "signin.amazonaws.com"
      },
      "eventTime": "2014-11-05T20:51:55Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "DescribeKey",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "192.0.2.0",
      "userAgent": "signin.amazonaws.com",
      "requestParameters": {
        "keyId": "e7b6d35a-b551-4c8f-b51a-0460ebc04565"
      },
      "responseElements": null,
      "requestID": "9400c720-652d-11e4-9a87-01af2a1ddecb",
      "eventID": "939fcefb-dc14-4a52-b918-73045fe97af3",
      "readOnly": true,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-east-1:123456789012:key/e7b6d35a-b551-4c8f-
b51a-0460ebc04565",
          "accountId": "123456789012"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    }
  ]
}
```

# DisableKey

The following example shows a log file generated by calling `DisableKey`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:43Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "DisableKey",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "keyId": "262d9fcb-f1a0-4447-af16-3714cff61ec1"
```

```
        },
        "responseElements": null,
        "requestID": "e26552bc-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "995c4653-3c53-4a06-a0f0-f5531997b741",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/262d9fcb-
f1a0-4447-af16-3714cff61ec1",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# EnableKey

The following example shows a log file generated by calling `EnableKey`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:20Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "EnableKey",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "keyId": "e17cebae-e7a6-4864-b92f-0365f2feff38"
        },
        "responseElements": null,
        "requestID": "d528a6fb-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "be393928-3629-4370-9634-567f9274d52e",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/e17cebae-
e7a6-4864-b92f-0365f2feff38",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# Encrypt

The following example shows a log file generated by calling `Encrypt`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:53:11Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "Encrypt",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "encryptionContext": {
                "ContextKey1": "Value1"
            },
            "keyId": "arn:aws:kms:us-
east-1:012345678901:key/8d3acf57-6bba-480a-9459-ed1b8e79d3d0"
        },
        "responseElements": null,
        "requestID": "f3423043-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "91235988-eb87-476a-ac2c-0cdc244e6dca",
        "readOnly": true,
        "resources": [{
            "ARN": "arn:aws:kms:us-
east-1:012345678901:key/8d3acf57-6bba-480a-9459-ed1b8e79d3d0",
            "accountId": "012345678901"
        }],
        "eventType": "AwsServiceEvent",
        "recipientAccountId": "012345678901"
    }
  ]
}
```


# GenerateDataKey

The following example shows a log file created by calling `GenerateDataKey`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
```

```
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:40Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "GenerateDataKey",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "keyId": "637e8678-3d08-4922-a650-e77eb1591db5",
            "numberOfBytes": 32
        },
        "responseElements": null,
        "requestID": "e0eb83e3-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "a9dea4f9-8395-46c0-942c-f509c02c2b71",
        "readOnly": true,
        "resources": [{
            "ARN": "arn:aws:kms:us-
east-1:123456789012:key/637e8678-3d08-4922-a650-e77eb1591db5",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# GenerateDataKeyWithoutPlaintext

The following example shows a log file created by calling `GenerateDataKeyWithoutPlaintext`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:23Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "GenerateDataKeyWithoutPlaintext",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "errorCode": "InvalidKeyUsageException",
        "requestParameters": {
            "keyId": "d4f2a88d-5f9c-4807-b71d-4d0ee5225156",
            "numberOfBytes": 16
```

```
        },
        "responseElements": null,
        "requestID": "d6b8e411-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "f7734272-9ec5-4c80-9f36-528ebbe35e4a",
        "readOnly": true,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/
d4f2a88d-5f9c-4807-b71d-4d0ee5225156",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# GenerateRandom

The following example shows a log file created by calling `GenerateRandom`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:37Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "GenerateRandom",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "df1e3de6-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "239cb9f7-ae05-4c94-9221-6ea30eef0442",
        "readOnly": true,
        "resources": [],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# GetKeyPolicy

The following example shows a log file generated by calling `GetKeyPolicy`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:50:30Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "GetKeyPolicy",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "keyId": "arn:aws:kms:us-east-1:123456789012:key/e923fe55-
d3ef-4f9c-89a1-2752f98c3a70",
            "policyName": "default"
        },
        "responseElements": null,
        "requestID": "93746dd6-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "4aa7e4d5-d047-452a-a5a6-2cce282a7e82",
        "readOnly": true,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/e923fe55-
d3ef-4f9c-89a1-2752f98c3a70",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# ListAliases

The following example shows a log file generated by calling `ListAliases`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:51:45Z",
```

```
        "eventSource": "kms.amazonaws.com",
        "eventName": "ListAliases",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "limit": 5,
            "marker":
 "eyJiIjoiYWxpYXMvZTU0Y2MxOTMtYTMwNC00YzEwLTliZWItYTJjZjA3NjA2OTJhIiwiYSI6ImFsaWFzL2U1NGNjM
        },
        "responseElements": null,
        "requestID": "bfe6c190-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "a27dda7b-76f1-4ac3-8b40-42dfba77bcd6",
        "readOnly": true,
        "resources": [],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```


# ListGrants

The following example shows a log file generated by calling `ListGrants`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:49Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "ListGrants",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "keyId": "arn:aws:kms:us-east-1:123456789012:key/ea22a751-
e707-40d0-92ac-13a28fa9eb11",
            "marker":
 "eyJncmFudElkIjoiMWY4M2U2ZmM0YTY2NDgxYjQ2Yzc4MTdhM2Y4YmQwMDFkZDNiYmQ1MGVlYTMyY2RmOWFiNWY1N
\u003d\u003d",
            "limit": 10
        },
        "responseElements": null,
        "requestID": "e5c23960-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "d24380f5-1b20-4253-8e92-dd0492b3bd3d",
        "readOnly": true,
        "resources": [{
```

```
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/ea22a751-
e707-40d0-92ac-13a28fa9eb11",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# ReEncrypt

The following example shows a log file generated by calling `ReEncrypt`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:19Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "ReEncrypt",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "destinationKeyId": "arn:aws:kms:us-
east-1:123456789012:key/116b8956-a086-40f1-96d6-4858ef794ba5"
        },
        "responseElements": null,
        "requestID": "d3eeee63-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "627c13b4-8791-4983-a80b-4c28807b964c",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/ff0c0fc1-
cbaa-41ab-a267-69481da8a4c8",
            "accountId": "123456789012"
        },
        {
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/116b8956-
a086-40f1-96d6-4858ef794ba5",
            "accountId": "123456789012"
        }],
        "eventType": "AwsServiceEvent",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

# Amazon EC2 Example One

The following example demonstrates an IAM user creating an encrypted volume using the default volume key in the Amazon EC2 management console.

The following example shows a CloudTrail log entry that demonstrates the user Alice creating an encrypted volume using a default volume key in AWS EC2 Management Console. The EC2 log file record includes a `volumeId` field with a value of `"vol-13439757"`. The AWS KMS record contains an `encryptionContext` field with a value of `"aws:ebs:id": "vol-13439757"`. Similarly, the `principalId` and `accountId` between the two records match. The records reflect the fact that creating an encrypted volume generates a data key that is used to encrypt the volume content.


```
{
  "Records": [
     {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T20:40:44Z"
          }
        },
        "invokedBy": "signin.amazonaws.com"
      },
      "eventTime": "2014-11-05T20:50:18Z",
      "eventSource": "ec2.amazonaws.com",
      "eventName": "CreateVolume",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "72.72.72.72",
      "userAgent": "signin.amazonaws.com",
      "requestParameters": {
        "size": "10",
        "zone": "us-east-1a",
        "volumeType": "gp2",
        "encrypted": true
      },
      "responseElements": {
        "volumeId": "vol-13439757",
        "size": "10",
        "zone": "us-east-1a",
        "status": "creating",
        "createTime": 1415220618876,
        "volumeType": "gp2",
        "iops": 30,
        "encrypted": true
      },
      "requestID": "1565210e-73d0-4912-854c-b15ed349e526",
      "eventID": "a3447186-135f-4b00-8424-bc41f1a93b4f",
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
```

```
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T20:40:44Z"
          }
        },
        "invokedBy": "AWS Internal"
      },
      "eventTime": "2014-11-05T20:50:19Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "GenerateDataKeyWithoutPlaintext",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "AWS Internal",
      "userAgent": "AWS Internal",
      "requestParameters": {
        "encryptionContext": {
          "aws:ebs:id": "vol-13439757"
        },
        "numberOfBytes": 64,
        "keyId": "alias/aws/ebs"
      },
      "responseElements": null,
      "requestID": "create-123456789012-758241111-1415220618",
      "eventID": "4bd2a696-d833-48cc-b72c-05e61b608399",
      "readOnly": true,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-east-1:123456789012:key/
e29ddfd4-1bf6-4e1b-8ecb-08216bd70d07",
          "accountId": "123456789012"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    }
  ]
}
```


# Amazon EC2 Example Two

The following example shows an IAM user running an Amazon EC2 instance that mounts a data volume encrypted by using a default volume key. The action taken by the user generates multiple AWS KMS log records. Creating the encrypted volume generates a data key, and the Amazon EC2 service generates a grant, on behalf of the customer, that enables it to decrypt the data key. The `instanceId`, `"i-81e2f56c"`, is referred to in the `granteePrincipal` field of the `CreateGrant` record as `"123456789012:aws:ec2-infrastructure:i-81e2f56c"` as well as in the identity of the principal calling Decrypt, `"arn:aws:sts::123456789012:assumed-`

role/aws:ec2-infrastructure/i-81e2f56c". The key identified by the UUID "e29ddfd4-1bf6-4e1b-8ecb-08216bd70d07" is common across all three KMS calls.


```
{
  "Records": [
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T21:34:36Z"
          }
        },
        "invokedBy": "signin.amazonaws.com"
      },
      "eventTime": "2014-11-05T21:35:27Z",
      "eventSource": "ec2.amazonaws.com",
      "eventName": "RunInstances",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "72.72.72.72",
      "userAgent": "signin.amazonaws.com",
      "requestParameters": {
        "instancesSet": {
          "items": [
            {
              "imageId": "ami-b66ed3de",
              "minCount": 1,
              "maxCount": 1
            }
          ]
        },
        "groupSet": {
          "items": [
            {
              "groupId": "sg-98b6e0f2"
            }
          ]
        },
        "instanceType": "m3.medium",
        "blockDeviceMapping": {
          "items": [
            {
              "deviceName": "/dev/xvda",
              "ebs": {
                "volumeSize": 8,
                "deleteOnTermination": true,
                "volumeType": "gp2"
              }
            },
            {
              "deviceName": "/dev/sdb",
              "ebs": {
```

```
            "volumeSize": 8,
            "deleteOnTermination": false,
            "volumeType": "gp2",
            "encrypted": true
          }
        }
      ]
    },
    "monitoring": {
      "enabled": false
    },
    "disableApiTermination": false,
    "instanceInitiatedShutdownBehavior": "stop",
    "clientToken": "XdKUT141516171819",
    "ebsOptimized": false
  },
  "responseElements": {
    "reservationId": "r-5ebc9f74",
    "ownerId": "123456789012",
    "groupSet": {
      "items": [
        {
          "groupId": "sg-98b6e0f2",
          "groupName": "launch-wizard-2"
        }
      ]
    },
    "instancesSet": {
      "items": [
        {
          "instanceId": "i-81e2f56c",
          "imageId": "ami-b66ed3de",
          "instanceState": {
            "code": 0,
            "name": "pending"
          },
          "amiLaunchIndex": 0,
          "productCodes": {

          },
          "instanceType": "m3.medium",
          "launchTime": 1415223328000,
          "placement": {
            "availabilityZone": "us-east-1a",
            "tenancy": "default"
          },
          "monitoring": {
            "state": "disabled"
          },
          "stateReason": {
            "code": "pending",
            "message": "pending"
          },
          "architecture": "x86_64",
          "rootDeviceType": "ebs",
          "rootDeviceName": "/dev/xvda",
          "blockDeviceMapping": {

          },
```

```
              "virtualizationType": "hvm",
              "hypervisor": "xen",
              "clientToken": "XdKUT1415223327917",
              "groupSet": {
                "items": [
                  {
                    "groupId": "sg-98b6e0f2",
                    "groupName": "launch-wizard-2"
                  }
                ]
              },
              "networkInterfaceSet": {

              },
              "ebsOptimized": false
            }
          ]
        }
      },
      "requestID": "41c4b4f7-8bce-4773-bf0e-5ae3bb5cbce2",
      "eventID": "cd75a605-2fee-4fda-b847-9c3d330ebaae",
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T21:34:36Z"
          }
        },
        "invokedBy": "AWS Internal"
      },
      "eventTime": "2014-11-05T21:35:35Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "CreateGrant",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "AWS Internal",
      "userAgent": "AWS Internal",
      "requestParameters": {
        "constraints": {
          "encryptionContextSubset": {
            "aws:ebs:id": "vol-f67bafb2"
          }
        },
        "granteePrincipal": "123456789012:aws:ec2-infrastructure:i-81e2f56c",
        "keyId": "arn:aws:kms:us-east-1:123456789012:key/
e29ddfd4-1bf6-4e1b-8ecb-08216bd70d07"
      },
      "responseElements": {
```

```
        "grantId":
 "6caf442b4ff8a27511fb6de3e12cc5342f5382112adf75c1a91dbd221ec356fe"
      },
      "requestID": "41c4b4f7-8bce-4773-bf0e-5ae3bb5cbce2",
      "eventID": "c1ad79e3-0d3f-402a-b119-d5c31d7c6a6c",
      "readOnly": false,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-east-1:123456789012:key/
e29ddfd4-1bf6-4e1b-8ecb-08216bd70d07",
          "accountId": "123456789012"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T21:34:36Z"
          }
        },
        "invokedBy": "AWS Internal"
      },
      "eventTime": "2014-11-05T21:35:32Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "GenerateDataKeyWithoutPlaintext",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "AWS Internal",
      "userAgent": "AWS Internal",
      "requestParameters": {
        "encryptionContext": {
          "aws:ebs:id": "vol-f67bafb2"
        },
        "numberOfBytes": 64,
        "keyId": "alias/aws/ebs"
      },
      "responseElements": null,
      "requestID": "create-123456789012-758247346-1415223332",
      "eventID": "ac3cab10-ce93-4953-9d62-0b6e5cba651d",
      "readOnly": true,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-east-1:123456789012:key/
e29ddfd4-1bf6-4e1b-8ecb-08216bd70d07",
          "accountId": "123456789012"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
```

```
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "123456789012:aws:ec2-infrastructure:i-81e2f56c",
        "arn": "arn:aws:sts::123456789012:assumed-role/aws:ec2-
infrastructure/i-81e2f56c",
        "accountId": "123456789012",
        "accessKeyId": "",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T21:35:38Z"
          },
          "sessionIssuer": {
            "type": "Role",
            "principalId": "123456789012:aws:ec2-infrastructure",
            "arn": "arn:aws:iam::123456789012:role/aws:ec2-infrastructure",
            "accountId": "123456789012",
            "userName": "aws:ec2-infrastructure"
          }
        }
      },
      "eventTime": "2014-11-05T21:35:47Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "Decrypt",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "172.172.172.172",
      "requestParameters": {
        "encryptionContext": {
          "aws:ebs:id": "vol-f67bafb2"
        }
      },
      "responseElements": null,
      "requestID": "b4b27883-6533-11e4-b4d9-751f1761e9e5",
      "eventID": "edb65380-0a3e-4123-bbc8-3d1b7cff49b0",
      "readOnly": true,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-east-1:123456789012:key/
e29ddfd4-1bf6-4e1b-8ecb-08216bd70d07",
          "accountId": "123456789012"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    }
  ]
}
```

# Programming the AWS KMS API

You can use the AWS KMS API to perform the following actions, and more.

- Create, describe, list, enable, and disable keys.
- Create, delete, list, and update aliases.
- Encrypt, decrypt, and re-encrypt content.
- Set, list, and retrieve key policies.
- Create, retire, revoke, and list grants.
- Retrieve key rotation status.
- Update key descriptions.
- Generate data keys with or without plaintext.
- Generate random data.

For example code in Java that uses the AWS SDK for Java to call the AWS KMS API, see the following topics.

- Creating a Client (p. 136)
- Working With Keys (p. 137)
- Encrypting and Decrypting Data (p. 141)
- Working with Key Policies (p. 143)
- Working with Grants (p. 145)
- Working with Aliases (p. 147)

## Creating a Client

Before you can program to the AWS Key Management Service, you must create a client, as shown by the following example. The client object, `kms`, is used throughout all of the code in the sections that follow.

```
package com.amazon.kms;
```

```
import java.io.File;
import java.io.IOException;
import java.nio.ByteBuffer;

import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.PropertiesCredentials;
import com.amazonaws.services.kms.AWSKMS;
import com.amazonaws.services.kms.AWSKMSClient;
import com.amazonaws.services.kms.model.*;

public class kmsSDKExample {

    private final AWSKMSClient kms;

    public kmsSDKExample() {
        kms = getClient();
    }

    public static void main(String[] args) {
        new kmsSDKExample();
    }

    private AWSKMS getClient() {
        final AWSCredentials creds;

        AWSKMSClient kms = new AWSKMSClient(creds);

        // Use the endpoint that corresponds to your region. This example
 uses the US West (Oregon)
        // region. For more information, see http://amzn.to/1mKTMmG
 (docs.aws.amazon.com)
        kms.setEndpoint("https://kms.us-west-2.amazonaws.com");

        return kms;
    }
}
```

# Working With Keys

This topic discusses how to create, describe, list, enable, and disable keys.

Topics



## Creating a Customer Master Key

Call the `CreateKey` function to create a customer master key. The function takes three optional parameters, as shown in the following example.

```
// Creating a key.
//
// Input Parameters:
//   The function takes three optional parameters.
//      Description  - Contains a string description for the key
//      KeyUsage     - Use the default value (ENCRYPT_DECRYPT)
//      Policy       - Use the default policy, which grants rights to all key
 actions
//
// Return Values:
//   The function returns a CreateKeyResult structure that contains the
 following:
//      AWSAccountId - Account ID of the account the key is associated with
//      ARN          - Amazon Resource Name for the key
//      CreationDate - Date the key was created in UTC format
//      Description  - Key description
//      Enabled      - A Boolean value that specifies whether the key is
 enabled
//      KeyID        - A unique value that can be used to identify the key in
 other operations
//      KeyUsage     - A value that shows what the key can be used for
//
String desc = "Key for protecting critical data";

CreateKeyRequest req = new CreateKeyRequest().withDescription(desc);
CreateKeyResult result = kms.createKey(req);
```

# Generating a Data Key

Call the `GenerateDataKey` function to create a data key. The function takes up to five parameters, as shown in the following example.

```
// Generate a data key
//
// Input Parameters:
//   The function takes five parameters.
//     KeyId             - Unique identifier for the key to be used for
 encryption
//     EncryptionContext - Authenticated data
//     NumberOfBytes     - The number of bytes of data key being requested
//     KeySpec           - The key specification being requested ("AES_128"
 or "AES_256")
//     GrantTokens       - List of grant tokens
//
//  Return Values:
//    The function returns a byte buffer that contains the encrypted key, a
 byte buffer
//    of the plaintext key, and the KeyID of the master key under which the
 key is encrypted.
//
// Replace the following string with a real key ID.
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

GenerateDataKeyRequest dataKeyRequest = new GenerateDataKeyRequest();
dataKeyRequest.setKeyId(keyId);
dataKeyRequest.setKeySpec("AES_128");
```

```
GenerateDataKeyResult dataKeyResult =
 kmsClient.generateDataKey(dataKeyRequest);

ByteBuffer plaintextKey = dataKeyResult.getPlaintext();

ByteBuffer encryptedKey = dataKeyResult.getCiphertextBlob();
```

# Describing a Key

Call the `DescribeKey` function to retrieve detailed information about a customer master key.

```
// Describing a key.
//
// Input Parameters:
//   The function takes one required parameter.
//      KeyId        - Unique identifier of the key. This can be an ARN, an
 alias, or a globally unique
//                     identifier.
//
// Return Values:
//   The function returns a DescribeKeyResult object that contains metadata
 about
//   the key.
//      AWSAccountId - ID of the account the key is associated with
//      ARN          - Amazon Resource Name for the key
//      CreationDate - Date the key was created in UTC format
//      Description  - Key description
//      Enabled      - A Boolean value that specifies whether the key is
 enabled
//      KeyId        - A unique value that can be used to identify the key in
 other operations
//      KeyUsage     - A value that shows what the key can be used for
//
// Replace the following string with a real key ID.
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

DescribeKeyRequest req = new DescribeKeyRequest().withKeyId(keyId);
DescribeKeyResult result = kms.describeKey(req);
```

# Listing Keys

Call the `ListKeys` function to retrieve a list of the customer master keys.

```
// Listing keys.
//
// Input Parameters:
//   The function takes two required parameters.
//      Limit      -  Specify this parameter only when paginating results to
 indicate the
//                    maximum number of keys you want listed in the response.
 If there are
//                    additional keys beyond the maximum you specify,
 the Truncated
```

```
//                    response element will be set to true.
//      Marker     -  Use this parameter only when paginating results, and
 only in a subsequent
//                    request after you've received a response where the
 results are truncated.
//                    Set it to the value of the NextMarker in the response
 you
//                    just received.
//
// Return Values:
//   The function returns a ListKeysResult object that contains the following
//   values:
//      Keys       - A list of keys
//      NextMarker - If Truncated is true, this value is present and contains
 the value
//                   to use for the Marker request parameter in a subsequent
 pagination
//                   request.
//      Truncated  - A flag that indicates whether there are more items in
 the list. If your results
//                   were truncated, you can make a subsequent pagination
 request using the
//                   Marker request parameter to retrieve more keys in the
 list.
//
Integer limit = 10;
String marker = null;

ListKeysRequest req = new
 ListKeysRequest().withMarker(marker).withLimit(limit);
ListKeysResult result = kms.listKeys(req);
```

# Enabling Keys

Call the `EnableKey` function to mark a key as enabled.

```
// Enabling a key.
//
// Input Parameters:
//   The function takes one required parameter.
//      KeyId    - Unique identifier of the customer master key to be
 enabled. This can be an
//                 ARN, an alias, or a globally unique identifier.
//
// Return Values:
//   The function does not return a value.
// Replace the following string with a real key ID.
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

EnableKeyRequest req = new EnableKeyRequest().withKeyId(keyId);
kms.enableKey(req);
```

# Disabling Keys

Call the `DisableKey` function to prevent a key from being used.

```
// Disabling a key.
//
// Input Parameters:
//   The function takes one required parameter.
//      KeyId    - Unique identifier of the customer master key to be
 disabled. This can be an
//                 ARN, an alias, or a globally unique identifier.
//
// Return Values:
//   The function does not return a value.
//
// Replace the following string with a real key ID.
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

DisableKeyRequest req = new DisableKeyRequest().withKeyId(keyId);
kms.disableKey(req);
```

# Encrypting and Decrypting Data

This topic discusses how to encrypt, decrypt, and re-encrypt content.

Topics



## Encrypting Data

Call the `Encrypt` function to encrypt plaintext data.

```
// Encrypting content
//
// Input Parameters:
//   The function takes four parameters.
//     KeyId             - Unique identifier for the key to be used for
 encryption
//     Plaintext         - Byte buffer that contains the content to be
 encrypted
//     EncryptionContext - Authenticated data
//     GrantTokens       - List of grant tokens
//
//  Return Values:
//    The function returns a byte buffer that contains the encrypted content
 and the key ID
//    of the master key used.
//
// Replace the following string with a real key ID.
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
ByteBuffer plaintext = ByteBuffer.wrap(new byte[]{1,2,3,4,5,6,7,8,9,0});

EncryptRequest req = new
 EncryptRequest().withKeyId(keyId).withPlaintext(plaintext);
```

```
ByteBuffer ciphertext = kms.encrypt(req).getCiphertextBlob();
```

# Decrypting Data

Call the `Decrypt` function to decrypt ciphertext. The data to decrypt must be valid ciphertext that you receive from the `Encrypt` function.

```
// Decrypting content
//
// Input Parameters:
//   The function takes three parameters.
//     CipherTextBlob    - Ciphertext to be decrypted
//     EncryptionContext - Authenticated data
//     GrantTokens       - List of grant tokens
//
// Return Values:
//   The function returns a byte buffer that contains the decrypted content.
//
ByteBuffer ciphertextBlob = Place your ciphertext here;

DecryptRequest req = new DecryptRequest().withCiphertextBlob(ciphertextBlob);
ByteBuffer plainText = kms.decrypt(req).getPlaintext();
```

# Re-Encrypting Data Under a Different Key

Call the `ReEncrypt` function to encrypt previously encrypted data by using a new key. This function decrypts your ciphertext and re-encrypts it by using a different key that you specify. The function never exposes your plaintext outside of AWS KMS.

```
// ReEncrypt content
// Input parameters:
//   The function takes three parameters.
//     CipherTextBlob               - Ciphertext to be re-encrypted
//     SourceEncryptionContext      - Authenticated data used for the
 original encryption
//     DestinationKeyId             - Key identifier for the re-encrypted
 data
//     DestinationEncryptionContext - encryption context for the re-encrypted
 data
//     GrantTokens                  - List of grant tokens
//
// Return Values:
//   The function returns a byte buffer that contains the re-encrypted
 content.
//
ByteBuffer sourceCiphertextBlob = Place your ciphertext here;
// Replace the following string with a real key ID.
String destinationKeyId = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";

ReEncryptRequest req = new ReEncryptRequest();
req.setCiphertextBlob(sourceCiphertextBlob);
req.setDestinationKeyId(destinationKeyId);
ByteBuffer destinationCipherTextBlob =
 kms.reEncrypt(req).getCiphertextBlob();
```

# Working with Key Policies

Use the AWS SDK for Java and the following sample code to list, retrieve, and set key policies for AWS KMS customer master keys (CMKs). This sample code requires that you previously instantiated an `AWSKMSClient` as `kms`.

Topics



## Listing Key Policies

Use a `ListKeyPoliciesRequest` to list the key policies for a CMK.

```
// Listing key policies
//
// Input Parameters:
//   keyId  -  A unique identifier for the CMK. This value can be a globally
 unique identifier, a
//             fully specified ARN to either an alias or a key, or an alias
 name prefixed by
//             "alias/".
//               -  Key ARN Example - arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
//               -  Alias ARN Example - arn:aws:kms:us-
west-2:111122223333:alias/ExampleAlias
//               -  Globally Unique Key ID Example -
 1234abcd-12ab-34cd-56ef-1234567890ab
//               -  Alias Name Example - alias/ExampleAlias
//
// Return Values:
//   A list of key policies.
//
// Replace the following string with a real key ID.
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

ListKeyPoliciesRequest req = new ListKeyPoliciesRequest().withKeyId(keyId);
ListKeyPoliciesResult result = kms.listKeyPolicies(req);
```

## Retrieving a Key Policy

Use a `GetKeyPolicyRequest` to retrieve a key policy.

```
// Retrieving a key policy
//
// Input Parameters:
//   keyId       -  A unique identifier for the CMK for which to return the
 key policy. This value
//                  can be a globally unique identifier or the fully
 specified ARN for a CMK.
//                    -  Key ARN Example - arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

```
//                    -  Globally Unique Key ID Example -
 1234abcd-12ab-34cd-56ef-1234567890ab
//   policyName  -  String that contains the name of the policy. Currently,
 this must be "default".
//
// Return Values:
//   A key policy.
//
// Replace the following string with a real key ID.
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String policyName = "default";

GetKeyPolicyRequest req = new
 GetKeyPolicyRequest().withKeyId(keyId).withPolicyName(policyName);
GetKeyPolicyResult result = kms.getKeyPolicy(req);
```

# Setting a Key Policy

Use a `PutKeyPolicyRequest` to set a key policy for a CMK.

```
// Setting a key policy for a CMK
//
// Input Parameters:
//   keyId       -   A unique identifier for the CMK. This value can be a
 globally unique identifier
//                   or the fully specified ARN to a CMK.
//                     -  Key ARN Example - arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
//                     -  Globally Unique Key ID Example -
 1234abcd-12ab-34cd-56ef-1234567890ab
//   policyName  -  Name of the policy to use. Currently, the only supported
 name is "default".
//   policy      -  The policy to use. This is required and delegates back to
 the account. The CMK
//                  is the root of trust. The policy size limit is 32 KiB
 (32768 bytes).
//
// Replace the following string with a real key ID.
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String policyName = "default";
String policy = "{" +
                "  \"Version\": \"2012-10-17\"," +
                "  \"Statement\": [{" +
                "    \"Sid\": \"Allow access for ExampleUser\"," +
                "    \"Effect\": \"Allow\"," +
                // Replace the following user ARN with one for a real user.
                "    \"Principal\": {\"AWS\":
 \"arn:aws:iam::111122223333:user/ExampleUser\"}," +
                "    \"Action\": [" +
                "      \"kms:Encrypt\"," +
                "      \"kms:GenerateDataKey*\"," +
                "      \"kms:Decrypt\"," +
                "      \"kms:DescribeKey\"," +
                "      \"kms:ReEncrypt*\"" +
                "    ]," +
                "    \"Resource\": \"*\"" +
```

```
                "  }]" +
                "}";

PutKeyPolicyRequest req = new
 PutKeyPolicyRequest().withKeyId(keyId).withPolicy(policy).withPolicyName(policyName);
kms.putKeyPolicy(req);
```

# Working with Grants

This topic discusses how to create, retire, revoke, and list grants.

Topics



## Creating a Grant

Call the `CreateGrant` function to create a grant.

```
// Creating a grant
//
// Input Parameters:
//   The function takes up to six parameters.
//     KeyId              - Unique identifier for the key. This can be an
 ARN, an alias, or a globally unique value.
//     GranteePrincipal   - Principal given permission to use the key
 identified by the KeyId parameter
//     RetiringPrincipal  - Principal given permission to retire the grant
//     Operations         - List of operations permitted by the grant
//     Constraints        - The conditions under which the actions specified
 by the Operations parameter are allowed
//     GrantTokens        - List of grant tokens
//
// Return Values:
//   The function returns two values.
//     GrantToken         - Signed and encrypted string value that contains
 all of the information needed to create the grant
//     GrantID            - Globally unique identifier of the grant
//
// Replace the following string with a real key ID.
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String granteePrincipal = "arn:aws:iam::111122223333:user/Alice";
String operation = GrantOperation.Encrypt;

CreateGrantRequest req = new CreateGrantRequest();
req.setKeyId(keyId);
req.setGranteePrincipal(granteePrincipal);
req.setOperation(operation);

CreateGrantResult result = kms.createGrant(req);
```

# Retiring a Grant

Call the `RetireGrant` function to retire a grant. You should retire a grant to clean up after you are done using it.

```
// Retiring a grant
//
// Input Parameters:
//   GrantToken - unique grant identifier
//
// Return Values:
//   The function does not return a value.
//
String grantToken = Place your grant token here;

RetireGrantRequest req = new RetireGrantRequest().withGrantToken(grantToken);
kms.retireGrant(req);
```

# Revoking Grants

Call the `RevokeGrant` function to revoke a grant. You should revoke a grant to deny operations that depend on it.

```
// Revoking a grant
//
// Input Parameters:
//   KeyId   - Unique identifier for the key
//   GrantId - Unique identifier for the grant
//
// Return Values:
//   The function does not return a value.
//
// Replace the following string with a real key ID.
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String grantId = "grant1";

RevokeGrantRequest req = new
 RevokeGrantRequest().withKeyId(keyId).withGrantId(grantId);
kms.revokeGrant(req);
```

# Listing Grants

Call the `ListGrants` function to list all of the grants on a given key.

```
// Listing grants
//
// Input Parameters:
//   The function takes three parameters.
//     KeyId    - Unique identifier for the key
//     Limit    - Specify this parameter only when paginating results to
 indicate the
//                maximum number of grants you want listed in the response.
 If there are
```

```
//                additional grants beyond the maximum you specify,
 the Truncated
//                response element will be set to true.
//     Marker    - Use this parameter only when paginating results, and only
 in a subsequent
//                request after you've received a response where the results
 are truncated.
//                Set it to the value of the NextMarker in the response you
//                just received.
//
// Return Values:
//   The function returns a list of grants for the key.
//
// Replace the following string with a real key ID.
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
Integer limit = 10;
String marker = null;

ListGrantsRequest req = new
 ListGrantsRequest().withKeyId(keyId).withMarker(marker).withLimit(limit);
ListGrantsResult result = kms.listGrants(req);
```

# Working with Aliases

This topic discusses how to create, delete, and update an alias.

An alias is a display name for a key. It can be used in place of a `KeyId` for the following operations:

- `DescribeKey`
- `Encrypt`
- `GenerateDataKey`
- `GenerateDataKeyWithoutPlaintext`
- `ListKeyPolicies`
- `ReEncrypt`

You can use a full ARN to specify an alias or just the alias name as shown in the following example. If you use the alias name, be sure to prepend "alias/" to it.

```
// Fully specified ARN
arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias

// Alias name (prefixed with "alias/")
alias/ExampleAlias
```

An alias is not a property of a key, and therefore can be associated with and disassociated from an existing key without changing the properties of the key. Deleting an alias does not delete the underlying key.

Topics

# Creating an Alias

Call the `CreateAlias` function to create an alias. The alias should be unique.

```
// Creating an alias
//
// Input Parameters:
//   The function takes two parameters.
//     AliasName    - String that contains a display name for a key. This is
 of the format
//                    "alias/[a-zA-Z0-9/_-]+". That is, the alias name can be
 an alphanumeric
//                    value and contain an underscore or a dash. Alias names
 that begin with
//                    "alias/aws..." are reserved for AWS use.
//     TargetKeyId  - Unique key identifier of the key to which the display
 name will
//                    be associated
//
// Return Values:
//   The function does not return a value.
//
String aliasName = "alias/projectKey1";
String targetKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

CreateAliasRequest req = new
 CreateAliasRequest().withAliasName(aliasName).withTargetKeyId(targetKeyId);
kms.createAlias(req);
```

# Deleting an Alias

Call the `DeleteAlias` function to delete an alias.

```
// Deleting an alias
//
// Input Parameters:
//   The function takes one parameter.
//     AliasName - String that contains a display name for a key
//
// Return Values:
//   The function does not return a value.
//
String aliasName = "alias/projectKey1";

DeleteAliasRequest req  = new DeleteAliasRequest().withAliasName(aliasName);
kms.deleteAlias(req);
```

# Listing Aliases

Call the `ListAliases` function to list all of the key aliases for your account.

```
// Listing aliases
//
// Input Parameters:
//   The function takes three parameters.
//     Limit    - Specify this parameter only when paginating results to
 indicate the
//                maximum number of aliases you want listed in the response.
 If there are
//                additional aliases beyond the maximum you specify,
 the Truncated
//                response element will be set to true.
//     Marker   - Use this parameter only when paginating results, and only
 in a subsequent
//                request after you've received a response where the results
 are truncated.
//                Set it to the value of the NextMarker in the response you
//                just received.
//
// Return Values:
//   The function returns a list of aliases for the keys in your account.
//
Integer limit = 10;

ListAliasesRequest req = new ListAliasesRequest().withLimit(limit);
ListAliasesResult result = kms.listAliases(req);
```

# Updating an Alias

Call the `UpdateAlias` function to associate an alias with a different key.

```
// Updating an alias
//
// Input Parameters:
//   The function takes two parameters.
//     AliasName   - String that contains the name of the alias to be
 modified. An alias name can
//                   contain only alphanumeric characters, forward slashes,
 underscores, and dashes.
//                   An alias must start with the word "alias" followed by a
 forward slash (alias/).
//                   An alias that begins with "aws" after the forward slash
 is reserved by
//                   Amazon Web Services (AWS).
//     TargetKeyId - Unique identifier of the customer master key to be
 associated with the alias.
//                   This value can be a globally unique identifier or the
 fully specified ARN of
//                   a key.
//
// Return Values:
//   The function does not return a value.
//
String aliasName = "alias/projectKey1";
String targetKeyId = "arn:aws:kms:us-
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";

UpdateAliasRequest req = new UpdateAliasRequest()
```

```
      .withAliasName(aliasName)
      .withTargetKeyId(targetKeyId);

kms.updateAlias(req);
```

# Cryptography Basics

Following are some basic terms and concepts in cryptography that you'll encounter when you work with AWS KMS.

**Plaintext and Ciphertext**
*Plaintext* refers to information or data in an unencrypted, or unprotected, form. *Ciphertext* refers to the output of an encryption algorithm operating on plaintext. Ciphertext is unreadable without knowledge of the algorithm and a secret key.

**Algorithms and Keys**
An *encryption algorithm* is a step-by-step set of instructions that specifies precisely how plaintext is transformed into ciphertext. Encryption algorithms require a secret key. AWS KMS uses the Advanced Encryption Standard (AES) algorithm in Galois/Counter Mode (GCM), known as AES-GCM. AWS KMS uses this algorithm with 256-bit secret keys.

**Symmetric and Asymmetric Encryption**
Encryption algorithms are either symmetric or asymmetric. *Symmetric encryption* uses the same secret key to perform both the encryption and decryption processes. *Asymmetric encryption*, also known as *public-key encryption*, uses two keys, a public key for encryption and a corresponding private key for decryption. The public key and private key are mathematically related so that when the public key is used for encryption, the corresponding private key must be used for decryption. AWS KMS uses only symmetric encryption.

For a more detailed introduction to cryptography and AWS KMS, see the following topics.

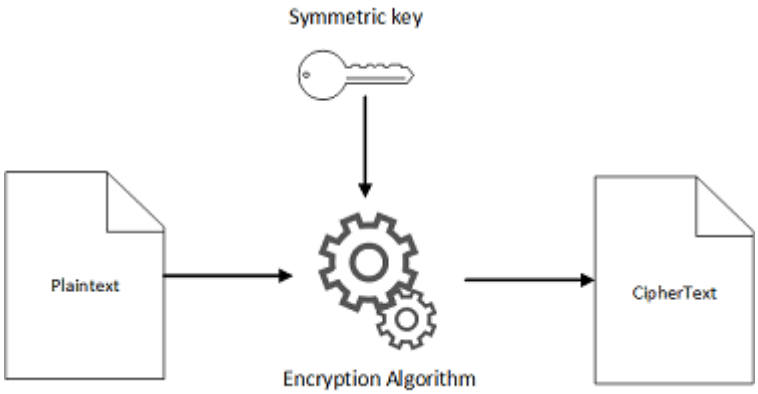
- How Symmetric Key Cryptography Works (p. 151)
- Authenticated Encryption (p. 152)
- Encryption Context (p. 153)
- Reference: AWS KMS and Cryptography Terminology (p. 154)

# How Symmetric Key Cryptography Works

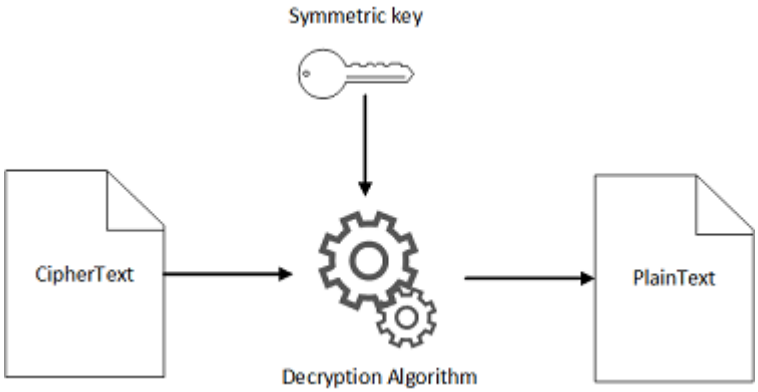
This topic provides a high-level introduction to how symmetric key cryptography uses algorithms to encrypt and decrypt data, the difference between block and stream ciphers, and how block ciphers use encryption modes to expand the effectiveness of the generic encryption schemes.

# Encryption and Decryption

AWS KMS uses symmetric key cryptography to perform encryption and decryption. Symmetric key cryptography uses the same algorithm and key to both encrypt and decrypt digital data. The unencrypted data is typically called plaintext whether it is text or not. The encrypted data is typically called ciphertext. The following illustration shows a secret (symmetric) key and a symmetric algorithm being used to turn plaintext into ciphertext.




The next illustration shows the same secret key and symmetric algorithm being used to turn ciphertext back into plaintext.




# Authenticated Encryption

*Authenticated encryption* provides confidentiality, data integrity, and authenticity assurances on encrypted data. The `Encrypt` API takes plaintext, a customer master key (CMK) identifier, and an encryption context (p. 153) and returns ciphertext. The encryption context represents additional authenticated data (AAD). The encryption process uses the AAD only to generate an authentication tag. The tag is included with the output ciphertext and used as input to the decryption process. This means that the encryption context that you supply to the `Decrypt` API must be the same as the encryption context you supply to the `Encrypt` API. Otherwise, the encryption and decryption tags will not match, and the decryption process will fail to produce plaintext. Further, if any one of the parameters has been tampered with—specifically if the ciphertext has been altered—the authentication tag will not compute to the same value that it did during encryption. The decryption process will fail and the ciphertext will not be decrypted.

Encryption context (p. 153) is AWS KMS's implementation of authenticated encryption or AAD. To learn how to use encryption context to protect the integrity of encrypted data, see the post How to Protect the Integrity of Your Encrypted Data by Using AWS Key Management Service and EncryptionContext on the AWS Security Blog.

# Encryption Context

*Encryption context* is a set of key-value pairs that you can pass to AWS KMS when you call the `Encrypt`, `Decrypt`, `ReEncrypt`, `GenerateDataKey`, and `GenerateDataKeyWithoutPlaintext` APIs. It is checked for integrity but not stored as part of the ciphertext that is returned. Although the encryption context is not included in the ciphertext, it is cryptographically bound to the ciphertext during encryption and must be passed again when you call the `Decrypt` (or `ReEncrypt`) API. Decryption only succeeds if the encryption context you pass for decryption is exactly the same as the encryption context you passed during encryption. The encryption context is logged by AWS CloudTrail.

To learn how to use encryption context to protect the integrity of encrypted data, see the post How to Protect the Integrity of Your Encrypted Data by Using AWS Key Management Service and EncryptionContext on the AWS Security Blog.

Encryption context can consist of any values that you want. However, because it is not encrypted and because it is logged if CloudTrail logging is turned on, your encryption context should not include sensitive information. We further recommend that your encryption context describe the data being encrypted or decrypted so that you can better understand the CloudTrail log entries produced by AWS KMS. For example, Amazon EBS uses the ID of the encrypted volume as the encryption context for server-side encryption. If you are encrypting a file, you might use part of the file path as encryption context.

Topics



## Encryption Context in Grants and Key Policies

In addition to using encryption context to check ciphertext integrity and authenticity, AWS KMS supports authorization by using grants and key policies that incorporate encryption context. Authorization that uses encryption context more tightly controls access to encrypted resources. When you create a grant, for example, you can optionally specify a set of constraints that specify an encryption context that unambiguously identifies the resource to which long-term access is being granted. For example, consider encrypted volumes with Amazon EBS and Amazon EC2: When an EBS volume is attached to an EC2 instance, a grant is created that allows only that instance to decrypt only that volume. This is accomplished by encoding the volume ID as the encryption context in the `Constraint` that is passed to the `CreateGrant` API. Without the encryption context in the constraint, the Amazon EC2 instance would obtain access to all volumes encrypted under the customer master key (CMK) rather than a specific volume.

## Logging Encryption Context

AWS KMS uses AWS CloudTrail to log the encryption context so you can determine which CMKs and data have been accessed. That is, the log entry shows exactly which CMK was used to encrypt or decrypt specific data referenced by the encryption context in the log entry.

**Important**
Because the encryption context is logged, it must not contain sensitive information.

## Storing Encryption Context

You should store the encryption context alongside the encrypted data to simplify using it when you call the `Decrypt` (or `ReEncrypt`) API. One security enhancement you might consider is to store only enough of the encryption context to help you create the full encryption context on the fly when needed for encryption or decryption. For example, if you are encrypting a file and decide that the encryption context should be the full file path, store only part of that path alongside the encrypted file contents. Then, when you need the full encryption context, reconstruct it from the stored fragment. If someone then moves the file to a different location, when you recreate the encryption context, the context will be different and the decryption process will fail, indicating that your data has been tampered with.

# Reference: AWS KMS and Cryptography Terminology

This section provides a brief glossary of terms for working with encryption in AWS KMS.

- **Additional authenticated data (AAD)**

  Offers both data-integrity and authenticity by using additional authenticated data during the encryption process. The AAD is authenticated but not encrypted. Using AAD with authenticated encryption enables the decryption process to detect any changes that may have been made to either the ciphertext or the additional authenticated data after encryption.
- **Authentication**

  The process of determining whether an entity is who it claims to be, or that information has not been manipulated by unauthorized entities.
- **Authorization**

  Specifies an entity's legitimate access to a resource.
- **Block cipher modes**

  Encrypts plaintext to ciphertext where the plaintext and cipher text are of arbitrary length. Modes are typically used to encrypt something that is longer than one block.
- **Block ciphers**

  An algorithm that operates on blocks of data, one block at a time.
- **Data key**

  A symmetric key generated by AWS KMS for your service. Inside of your service or custom application, the data key is used to encrypt or decrypt data. It can be considered a resource by a service or application, or it can simply be metadata associated with the encrypted data.
- **Decryption**

  The process of turning ciphertext back into the form it had before encryption. A decrypted message is called plaintext.
- **Encryption**

  The process of providing data confidentiality to a plaintext message. An encrypted message is called ciphertext.
- **Encryption context**

  AWS KMS specific AAD in the form of a "key":"value" pair. Although not encrypted, it is bound to the ciphertext during encryption and must be passed again during decryption. If the encryption

context passed for encryption is not the same as the encryption context passed for decryption or the ciphertext has been changed, the decryption process will fail.

- **Master key**

  A key created by AWS KMS that can only be used within the AWS KMS service. The master key is commonly used to encrypt data keys so that the encrypted key can be securely stored by your service. However, AWS KMS master keys can also be used to encrypt or decrypt arbitrary chunks of data that are no greater than 4 KiB. Master keys are categorized as either customer managed keys or AWS managed keys. Customer managed keys are created by a customer for use by a service or application. AWS managed keys are the default keys used by AWS services that support encryption.
- **Symmetric key cryptography**

  Uses a single secret key to encrypt and decrypt a message.

# Document History

The following table describes the important changes to the documentation since the last release of AWS Key Management Service.

- **Current API version**: 2014-11-01
- **Latest documentation update**: August 31, 2016

| Change | Description | Release Date |
|---|---|---|
| New content | Added documentation about Monitoring Customer Master Keys (p. 107) and Monitoring with Amazon CloudWatch (p. 109). | August 31, 2016 |
| New content | Added documentation about Importing Key Material (p. 51). | August 11, 2016 |
| New content | Added the following documentation: Overview of Managing Access (p. 12), Using IAM Policies (p. 27), AWS KMS API Permissions Reference (p. 31), and Using Policy Conditions (p. 35). | July 5, 2016 |
| Update | Updated portions of the documentation in the Authentication and Access Control (p. 10) chapter. | July 5, 2016 |
| Update | Updated the Limits (p. 159) page to reflect new default limits. | May 31, 2016 |
| Update | Updated the Limits (p. 159) page to reflect new default limits, and updated the Grant Tokens (p. 4) documentation to improve clarity and accuracy. | April 11, 2016 |

| Change | Description | Release Date |
| --- | --- | --- |
| New content | Added documentation about Allowing Multiple IAM Users to Access a CMK (p. 25) and Using the IP Address Condition (p. 36). | February 17, 2016 |
| Update | Updated the Using Key Policies in AWS KMS (p. 14) and Modifying a Key Policy (p. 22) pages to improve clarity and accuracy. | February 17, 2016 |
| Update | Updated the Getting Started (p. 6) topic pages to improve clarity. | January 5, 2016 |
| New content | Added documentation about How AWS CloudTrail Uses AWS KMS (p. 80). | November 18, 2015 |
| New content | Added instructions for Modifying a Key Policy (p. 22). | November 18, 2015 |
| Update | Updated the documentation about How Amazon Relational Database Service (Amazon RDS) Uses AWS KMS (p. 96). | November 18, 2015 |
| New content | Added documentation about How Amazon WorkSpaces Uses AWS KMS (p. 103). | November 6, 2015 |
| Update | Updated the Using Key Policies in AWS KMS (p. 14) page to improve clarity. | October 22, 2015 |
| New content | Added documentation about Deleting Customer Master Keys (p. 63), including supporting documentation about Creating an Amazon CloudWatch Alarm (p. 69) and Determining Past Usage of a Customer Master Key (p. 71). | October 15, 2015 |
| New content | Added documentation about Determining Access to an AWS KMS Customer Master Key (p. 43). | October 15, 2015 |
| New content | Added documentation about How Key State Affects Use of a Customer Master Key (p. 75). | October 15, 2015 |
| New content | Added documentation about How Amazon Simple Email Service (Amazon SES) Uses AWS KMS (p. 96). | October 1, 2015 |

| Change | Description | Release Date |
| --- | --- | --- |
| Update | Updated the Limits (p. 159) page to explain the new requests per second limits. | August 31, 2015 |
| New content | Added information about the charges for using AWS KMS. See AWS KMS Pricing (p. 2). | August 14, 2015 |
| New content | Added requests per second to the AWS KMS Limits (p. 159). | June 11, 2015 |
| New content | Added a new Java code sample demonstrating use of the `UpdateAlias` API. See Updating an Alias (p. 149). | June 1, 2015 |
| Update | Moved the AWS Key Management Service regions table to the *AWS General Reference*. | May 29, 2015 |
| New content | Added documentation about How Amazon EMR Uses AWS KMS (p. 91). | January 28, 2015 |
| New content | Added documentation about How Amazon WorkMail Uses AWS KMS (p. 101). | January 28, 2015 |
| New content | Added documentation about How Amazon Relational Database Service (Amazon RDS) Uses AWS KMS (p. 96). | January 6, 2015 |
| New content | Added documentation about How Amazon Elastic Transcoder Uses AWS KMS (p. 87). | November 24, 2014 |
| New guide | Introduced the *AWS Key Management Service Developer Guide*. | November 12, 2014 |

# Limits

All AWS KMS objects have limits that apply to each region and each AWS account. If you need to exceed these limits, please visit the AWS Support Center and create a case.

| Resource | Default Limit |
| --- | --- |
| Customer Master Keys (CMKs) (p. 159) | 1000 |
| Aliases (p. 159) | 1100 |
| Grants per CMK (p. 159) | 2500 |
| Grants for a given principal per CMK (p. 160) | 30 |
| Requests per second (p. 160) | Varies by API operation; see table (p. 160). |

**Customer Master Keys (CMKs): 1000**

You can have up to 1000 CMKs per region. All CMKs count towards this limit regardless of their status (enabled, disabled, or pending deletion). You can request more CMKs in a region; however, managing a large number of CMKs from the AWS Management Console may be slower than acceptable. If you have a large number of CMKs in a region, we recommend managing them programmatically with the AWS SDKs or AWS Command Line Tools.

**Aliases: 1100**

An alias is an independent display name that you can map to a CMK. It is not a property of a CMK. You can map multiple aliases to a single CMK, so the limit for aliases is higher than the limit for CMKs. If you request an increase in the number of CMKs, you might also need to request an increase in the number of aliases.

**Grants per CMK: 2500**

Using Grants (p. 43) are advanced mechanisms for specifying permissions that you or an AWS service integrated with AWS KMS can use to limit how and when a CMK can be used. Grants are attached to a CMK, and each grant contains the principal who receives permission to use the CMK, the ID of the CMK, and a list of operations that can be performed. Grants are an alternative to the key policy (p. 14).

Each CMK can have up to 2500 grants, including the grants created by AWS services that are integrated with AWS KMS. For a list of these services, see How AWS Services use AWS KMS (p. 78). One effect of this limit is that you cannot create more than 2500 resources that use

the same CMK. For example, you cannot create more than 2500 encrypted EBS volumes (p. 85) that use the same CMK.

**Grants for a given principal per CMK: 30**

For a given CMK, no more than 30 grants can specify the same grantee principal. For example, assume that you want to encrypt multiple Amazon EBS volumes and attach them to a single Amazon Elastic Compute Cloud (Amazon EC2) instance. In this case, a unique grant is created for each encrypted volume and all of these grants have the same grantee principal: an IAM assumed-role user associated with the EC2 instance. Each grant gives permission to use the specified CMK to decrypt an EBS volume's unique data encryption key (DEK). For each CMK, you can have up to 30 grants that specify the same EC2 instance as the grantee principal. This effectively means that you can have no more than 30 encrypted EBS volumes per EC2 instance for a given CMK.

**Requests per second: varies**

The point at which AWS KMS begins to throttle API requests differs depending on the API operation. The following table lists each AWS KMS API operation and the point at which AWS KMS begins to throttle API requests for that operation.

The API operations in the first row share a limit of 100 requests per second. The remaining API operations each have a unique limit for requests per second, which means the limit is not shared.

For example, when you make 50 `GenerateDataKey` requests and 30 `Decrypt` requests per second, you can make an additional 20 requests per second using any of the API operations in the first row of the table before AWS KMS begins to throttle your requests. When you make 70 `Encrypt` requests and 40 `GenerateRandom` requests per second, AWS KMS will throttle your requests because you are making more than 100 requests per second using API operations in the first row of the table.

| API operation | Requests-per-second limit |
|---|---|
| `Encrypt`<br>`Decrypt`<br>`ReEncrypt`<br>`GenerateRandom`<br>`GenerateDataKey`<br>`GenerateDataKeyWithoutPlaintext` | 100 (shared) |
| `CancelKeyDeletion` | 5 |
| `CreateAlias` | 5 |
| `CreateGrant` | 15 |
| `CreateKey` | 5 |
| `DeleteAlias` | 5 |
| `DescribeKey` | 30 |
| `DisableKey` | 5 |
| `DisableKeyRotation` | 5 |
| `EnableKey` | 5 |
| `EnableKeyRotation` | 5 |
| `GetKeyPolicy` | 30 |
| `GetKeyRotationStatus` | 5 |

| API operation | Requests-per-second limit |
| --- | --- |
| `ListAliases` | 5 |
| `ListGrants` | 5 |
| `ListKeyPolicies` | 5 |
| `ListKeys` | 5 |
| `ListRetirableGrants` | 5 |
| `PutKeyPolicy` | 5 |
| `RetireGrant` | 15 |
| `RevokeGrant` | 15 |
| `ScheduleKeyDeletion` | 5 |
| `UpdateAlias` | 5 |
| `UpdateKeyDescription` | 5 |

Note that API requests can be made directly or by an integrated AWS service on your behalf. For example, you might use server-side encryption with AWS KMS (SSE-KMS) when you store data in Amazon S3. Each time you upload or download an S3 object that's encrypted with SSE-KMS, Amazon S3 makes a `GenerateDataKey` (for uploads) or `Decrypt` (for downloads) API request to AWS KMS on your behalf. Each API request that Amazon S3 makes on your behalf counts towards your limit, and your requests will be throttled if you exceed 100 uploads or downloads per second of S3 objects encrypted with SSE-KMS.