
AWS Key Management Service

Developer Guide



AWS Key Management Service: Developer Guide

Copyright © 2018 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is AWS Key Management Service?	1
Concepts	2
Customer Master Keys (CMKs)	2
Data Keys	3
Envelope Encryption	5
Encryption Context	6
Key Policies	7
Grants	7
Grant Tokens	7
Auditing CMK Usage	7
Key Management Infrastructure	7
Getting Started	8
Creating Keys	8
Creating CMKs (Console)	8
Creating CMKs (KMS API)	10
Viewing Keys	11
Viewing CMKs (Console)	11
Viewing CMKs (KMS API)	13
Finding the Key ID and ARN	16
Editing Keys	18
Editing CMKs (Console)	18
Editing CMKs (KMS API)	23
Tagging Keys	24
Managing CMK Tags (Console)	24
Managing CMK Tags (KMS API)	25
Enabling and Disabling Keys	26
Enabling and Disabling CMKs (Console)	27
Enabling and Disabling CMKs (KMS API)	27
Authentication and Access Control	29
Authentication	29
Access Control	30
Overview of Managing Access	30
AWS KMS Resources and Operations	31
Managing Access to AWS KMS CMKs	31
Specifying Permissions in a Policy	32
Specifying Conditions in a Policy	33
Using Key Policies	33
Overview of Key Policies	33
Default Key Policy	34
Example Key Policy	40
Changing a Key Policy	43
Keeping Key Policies Up to Date	48
Using IAM Policies	50
Overview of IAM Policies	51
Permissions Required to Use the AWS KMS Console	51
AWS Managed (Predefined) Policies for AWS KMS	52
Customer Managed Policy Examples	52
AWS KMS API Permissions Reference	54
Using Policy Conditions	59
AWS Global Condition Keys	60
AWS KMS Condition Keys	61
Using Grants	78
Determining Access	80
Understanding Policy Evaluation	80

Examining the Key Policy	81
Examining IAM Policies	84
Examining Grants	85
Rotating Keys	87
How Automatic Key Rotation Works	88
How to Enable and Disable Automatic Key Rotation	88
Enabling and Disabling Key Rotation (Console)	89
Enabling and Disabling Key Rotation (KMS API)	90
Rotating Keys Manually	90
Importing Key Material	93
About Imported Key Material	93
How To Import Key Material	94
How to Reimport Key Material	94
How to Identify CMKs with Imported Key Material	95
To identify the value of the <code>Origin</code> property of CMKs (Console)	95
To identify the value of the <code>Origin</code> property of CMKs (KMS API)	96
Step 1: Create a CMK with No Key Material	96
Creating a CMK with No Key Material (Console)	97
Creating a CMK with No Key Material (KMS API)	99
Step 2: Download the Public Key and Import Token	99
Downloading the Public Key and Import Token (Console)	101
Downloading the Public Key and Import Token (KMS API)	103
Step 3: Encrypt the Key Material	104
Example: Encrypt Key Material with OpenSSL	104
Step 4: Import the Key Material	105
Import Key Material (Console)	105
Import Key Material (KMS API)	106
Deleting Key Material	107
How Deleting Key Material Affects AWS Services Integrated With AWS KMS	107
Delete Key Material (Console)	108
Delete Key Material (KMS API)	108
Deleting Customer Master Keys	110
How Deleting CMKs Works	110
How Deleting CMKs Affects Integrated AWS Services	111
Scheduling and Canceling Key Deletion	111
Using the AWS Management Console	112
Using the AWS CLI	113
Using the AWS SDK for Java	114
Adding Permission to Schedule and Cancel Key Deletion	114
Using the AWS Management Console	114
Using the AWS CLI	116
Creating an Amazon CloudWatch Alarm	117
Requirements for a CloudWatch Alarm	117
Create the CloudWatch Alarm	118
Determining Past Usage of a CMK	120
Examining CMK Permissions to Determine the Scope of Potential Usage	120
Examining AWS CloudTrail Logs to Determine Actual Usage	120
How Key State Affects Use of a Customer Master Key	123
How AWS Services use AWS KMS	127
AWS CloudTrail	127
Understanding When Your CMK is Used	127
Understanding How Often Your CMK is Used	131
Amazon DynamoDB	132
Using CMKs and Data Keys	132
Authorizing Use of the Service Default Key	134
DynamoDB Encryption Context	135
Monitoring DynamoDB Interaction with AWS KMS	136

Amazon Elastic Block Store (Amazon EBS)	139
Amazon EBS Encryption	140
Using CMKs and Data Keys	140
Amazon EBS Encryption Context	140
Detecting Amazon EBS Failures	141
Using AWS CloudFormation to Create Encrypted Amazon EBS Volumes	141
Amazon Elastic Transcoder	141
Encrypting the input file	142
Decrypting the input file	142
Encrypting the output file	143
HLS Content Protection	144
Elastic Transcoder Encryption Context	145
Amazon EMR	145
Encrypting Data on the EMR File System (EMRFS)	146
Encrypting Data on the Storage Volumes of Cluster Nodes	148
Encryption Context	148
Amazon Redshift	149
Amazon Redshift Encryption	149
Encryption Context	150
Amazon Relational Database Service (Amazon RDS)	150
Amazon RDS Encryption Context	150
AWS Secrets Manager	151
Protecting the Secret Value	151
Encrypting and Decrypting Secrets	151
Using Your AWS KMS CMK	153
Authorizing Use of the CMK	154
Secrets Manager Encryption Context	155
Monitoring Secrets Manager Interaction with AWS KMS	156
Amazon Simple Email Service (Amazon SES)	158
Overview of Amazon SES Encryption Using AWS KMS	159
Amazon SES Encryption Context	159
Giving Amazon SES Permission to Use Your AWS KMS Customer Master Key (CMK)	160
Retrieving and Decrypting Email Messages	160
Amazon Simple Storage Service (Amazon S3)	161
Server-Side Encryption: Using SSE-KMS	161
Using the Amazon S3 Encryption Client	162
Encryption Context	162
AWS Systems Manager Parameter Store	162
Encrypting and Decrypting Secure String Parameters	163
Setting Permissions to Encrypt and Decrypt Parameter Values	164
Parameter Store Encryption Context	165
Troubleshooting CMK Issues in Parameter Store	166
Amazon WorkMail	167
Amazon WorkMail Overview	167
Amazon WorkMail Encryption	167
Amazon WorkMail Encryption Context	169
Amazon WorkSpaces	169
Overview of Amazon WorkSpaces Encryption Using AWS KMS	170
Amazon WorkSpaces Encryption Context	170
Giving Amazon WorkSpaces Permission to Use A CMK On Your Behalf	171
Monitoring Customer Master Keys	173
Monitoring Tools	173
Automated Tools	173
Manual Tools	174
Monitoring with CloudWatch	174
Metrics and Dimensions	175
Creating Alarms	176

AWS KMS Events	177
Logging AWS KMS API Calls with AWS CloudTrail	180
AWS KMS Information in CloudTrail	180
Understanding AWS KMS Log File Entries	181
CreateAlias	181
CreateGrant	182
CreateKey	183
Decrypt	184
DeleteAlias	184
DescribeKey	185
DisableKey	187
EnableKey	187
Encrypt	188
GenerateDataKey	189
GenerateDataKeyWithoutPlaintext	189
GenerateRandom	190
GetKeyPolicy	190
ListAliases	191
ListGrants	192
ReEncrypt	192
Amazon EC2 Example One	193
Amazon EC2 Example Two	195
Using a VPC Endpoint	200
Create a VPC Endpoint	201
Creating a VPC Endpoint (Console)	201
Creating an AWS KMS VPC Endpoint (AWS CLI)	202
Connecting to a VPC Endpoint	203
Using a VPC Endpoint in a Policy Statement	204
Audit the CMK Use for your VPC	206
Programming the AWS KMS API	207
Creating a Client	207
Working With Keys	208
Creating a Customer Master Key	208
Generating a Data Key	210
Viewing a Custom Master Key	212
Getting Key IDs and Key ARNs of Customer Master Keys	213
Enabling Customer Master Keys	214
Disabling Customer Master Keys	216
Encrypting and Decrypting Data Keys	217
Encrypting a Data Key	218
Decrypting a Data Key	220
Re-Encrypting a Data Key Under a Different Customer Master Key	221
Working with Key Policies	223
Listing Key Policy Names	223
Getting a Key Policy	225
Setting a Key Policy	227
Working with Grants	230
Creating a Grant	230
Viewing a Grant	232
Retiring a Grant	234
Revoking a Grant	235
Working with Aliases	237
Creating an Alias	238
Listing Aliases	239
Updating an Alias	242
Deleting an Alias	244
Cryptography Basics	246

How Symmetric Key Cryptography Works	246
Encryption and Decryption	246
Authenticated Encryption	247
Encryption Context	248
Encryption Context in Grants and Key Policies	248
Logging Encryption Context	248
Storing Encryption Context	249
Reference: AWS KMS and Cryptography Terminology	249
Document History	251
Recent Updates	251
Earlier Updates	251
Limits	255
Customer Master Keys (CMKs): 1000	255
Aliases: 1100	255
Key policy document size: 32 KB	256
Grants per CMK: 2500	256
Grants for a given principal per CMK: 500	256
Requests per second: varies	256

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. The master keys that you create in AWS KMS are protected by [FIPS 140-2 validated cryptographic modules](#).

AWS KMS is integrated with most [other AWS services](#) that encrypt your data with encryption keys that you manage. AWS KMS is also integrated with [AWS CloudTrail](#) to provide encryption key usage logs to help meet your auditing, regulatory and compliance needs.

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

- Create, describe, and list master keys
- Enable and disable master keys
- Create and view grants and access control policies for your master keys
- Enable and disable automatic rotation of the cryptographic material in a master key
- Import cryptographic material into an AWS KMS master key
- Tag your master keys for easier identification, categorizing, and tracking
- Create, delete, list, and update *aliases*, which are friendly names associated with 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.

Learn More

- For a more detailed introduction to AWS KMS, see [AWS KMS Concepts \(p. 2\)](#).
- For information about the AWS KMS API, see the [AWS Key Management Service API Reference](#).
- For detailed technical information 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 \(CMKs\) \(p. 2\)](#)
- [Data Keys \(p. 3\)](#)
- [Envelope Encryption \(p. 5\)](#)
- [Encryption Context \(p. 6\)](#)
- [Key Policies \(p. 7\)](#)
- [Grants \(p. 7\)](#)
- [Grant Tokens \(p. 7\)](#)
- [Auditing CMK Usage \(p. 7\)](#)
- [Key Management Infrastructure \(p. 7\)](#)

Customer Master Keys (CMKs)

The primary resources in AWS KMS are *customer master keys* (CMKs). You can use a CMK to encrypt and decrypt up to 4 KB (4096 bytes) of data. Typically, you use CMKs to generate, encrypt, and decrypt the [data keys \(p. 3\)](#) that you use outside of AWS KMS to encrypt your data. This strategy is known as [envelope encryption \(p. 5\)](#).

CMKs are created in AWS KMS and never leave AWS KMS unencrypted. To use or manage your CMK, you access them through AWS KMS. This strategy differs from [data keys \(p. 3\)](#). AWS KMS does not store, manage, or track your data keys. You must use them outside of AWS KMS.

There are three types of CMKs in AWS accounts: customer managed CMKs, AWS managed CMKs, and AWS owned CMKs.

Type of CMK	Can view	Can manage	Used only for my AWS account
Customer managed CMK (p. 3)	Yes	Yes	Yes
AWS managed CMK (p. 3)	Yes	No	Yes
AWS owned CMK (p. 3)	No	No	No

[AWS services that integrate with AWS KMS \(p. 127\)](#) differ in their support for CMKs. Some services encrypt your data by default with an AWS owned CMK. Some encrypt under AWS managed CMKs that they create in your account. Other services allow you specify a customer managed CMK that you have created. And others support all types of CMKs to allow you the ease of an AWS owned CMK, the visibility of an AWS managed CMK, or the control of a customer-managed CMK.

Customer managed CMKs

Customer managed CMKs are CMKs in your AWS account that you create, own, and manage. You have full control over these CMKs, including establishing and maintaining their [key policies](#), [IAM policies](#), and [grants](#) (p. 29), [enabling and disabling](#) (p. 26) them, [rotating their cryptographic material](#) (p. 87), [adding tags](#) (p. 24), [creating aliases](#) (p. 237) that refer to the CMK, and [scheduling the CMKs for deletion](#) (p. 110).

You can use your customer managed CMKs in cryptographic operations and audit their use in AWS CloudTrail logs. In addition, many [AWS services that integrate with AWS KMS](#) (p. 127) let you specify a customer managed CMK to protect the data that they store and manage for you.

Customer managed CMKs incur a monthly fee and a fee for use in excess of the free tier. They are counted against the AWS KMS [limits](#) (p. 255) for your account. For details, see [AWS Key Management Service Pricing](#) and [Limits](#) (p. 255).

AWS managed CMKs

AWS managed CMKs are CMKs in your account that are created, managed, and used on your behalf by an AWS service that integrates with AWS KMS. You can identify AWS managed CMKs by their aliases, which have the format `aws/service-name`, such as `aws/redshift`.

You can view the AWS managed CMKs in your account, view their key policies, and audit their use in AWS CloudTrail logs. However, you cannot manage these CMKs or change their permissions. And, you cannot use AWS managed CMKs in cryptographic operations directly; the service that creates them uses them on your behalf. To view the key policy for an AWS managed CMK, use the [GetKeyPolicy](#) operation. You cannot view the key policy in the AWS Management Console, or change it by any means.

You do not pay a monthly fee for AWS managed CMKs. They can be subject to fees for use in excess of the free tier, but some AWS services cover these costs for you. For details, see the encryption section of the service documentation. AWS managed CMKs do not count against limits on the number of CMKs in each region of your account, but when they are used on behalf of a principal in your account, they count against request rate limits. For details, see [AWS Key Management Service Pricing](#) and [Limits](#) (p. 255).

AWS owned CMKs

AWS owned CMKs are not in your AWS account. They are part of a collection of CMKs that AWS owns and manages for use in multiple AWS accounts. AWS services can use AWS owned CMKs to protect your data.

You cannot view, manage, or use AWS owned CMKs, or audit their use. However, you do not need to do any work or change any programs to protect the keys that encrypt your data.

You are not charged a monthly fee or a usage fee for use of AWS owned CMKs and they do not count against AWS KMS limits for your account.

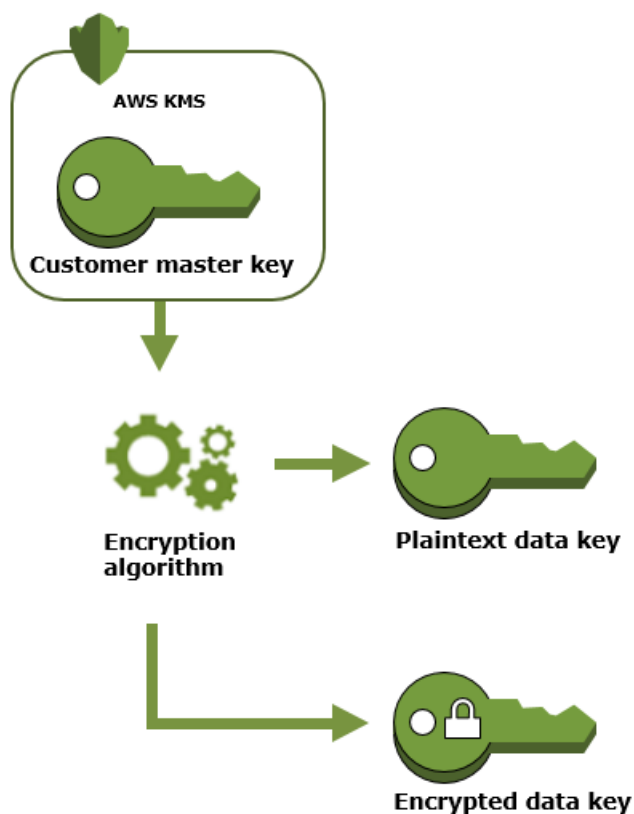
Data Keys

Data keys are encryption keys that you can use to encrypt data, including large amounts of data and other data encryption keys.

You can use AWS KMS [customer master keys](#) (p. 2) (CMKs) to generate, encrypt, and decrypt data keys. However, AWS KMS does not store, manage, or track your data keys, or perform cryptographic operations with data keys. You must use and manage data keys outside of AWS KMS.

Create a data key

To create a data key, call the [GenerateDataKey](#) operation. AWS KMS uses the CMK that you specify to generate a data key. The operation returns a plaintext copy of the data key and a copy of the data key encrypted under the CMK, as shown in the following image.

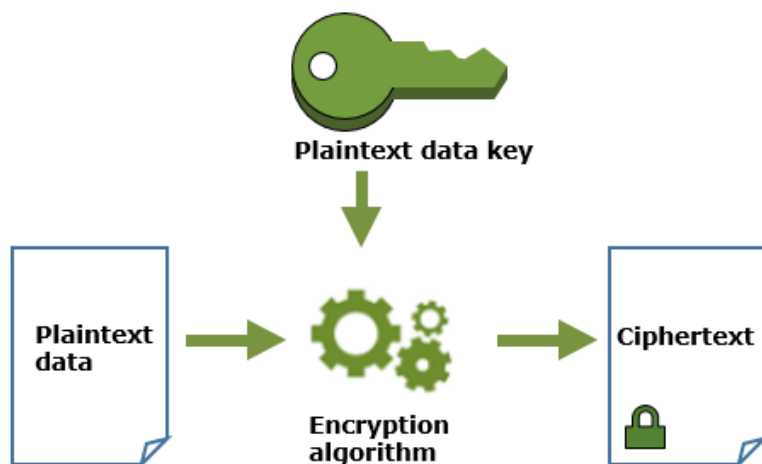


AWS KMS also supports the [GenerateDataKeyWithoutPlaintext](#) operation, which returns only an encrypted data key. When you need to use the data key, ask AWS KMS to [decrypt](#) it.

Encrypt data with a data key

AWS KMS cannot use a data key to encrypt data, but you can use the data key outside of KMS, such as by using OpenSSL or a cryptographic library like the [AWS Encryption SDK](#).

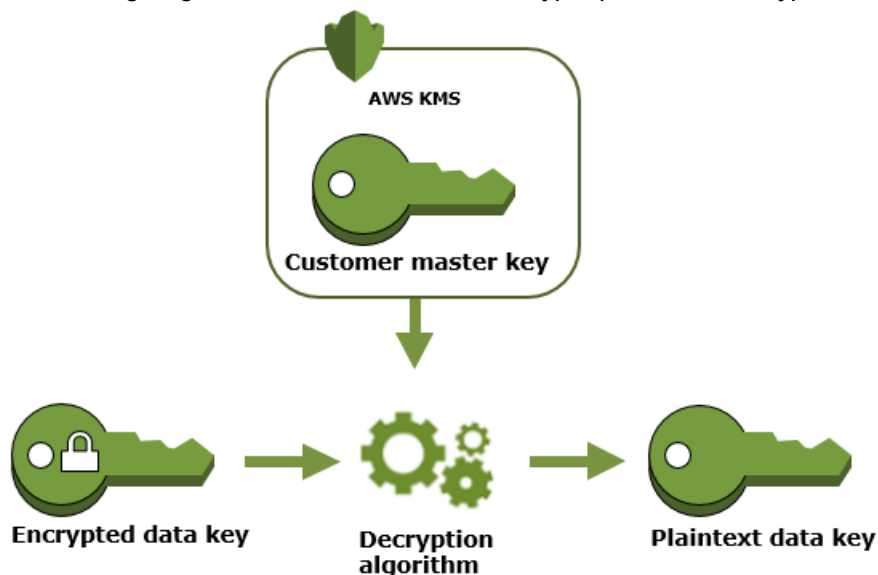
After using the plaintext data key to encrypt data, remove it from memory as soon as possible. You can safely store the encrypted data key with the encrypted data so it is available to decrypt the data.



Decrypt data with a data key

To decrypt your data, pass the encrypted data key to the [Decrypt](#) operation. AWS KMS uses your CMK to decrypt the data key and then it returns the plaintext data key. Use the plaintext data key to decrypt your data and then remove the plaintext data key from memory as soon as possible.

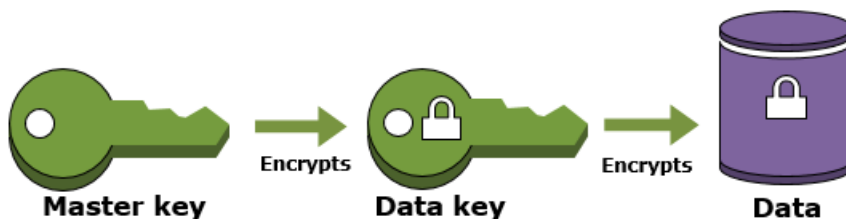
The following diagram shows how to use the Decrypt operation to decrypt an encrypted data key.



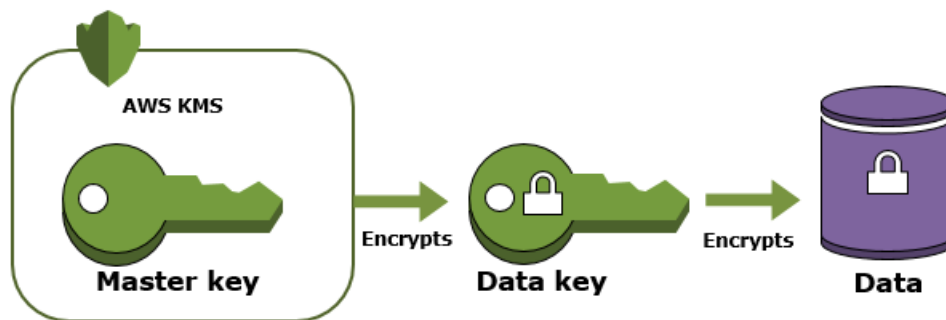
Envelope Encryption

When you encrypt your data, your data is protected, but you have to protect your encryption key. One strategy is to encrypt it. *Envelope encryption* is the practice of encrypting plaintext data with a data key, and then encrypting the data key under another key.

You can even encrypt the data encryption key under another encryption key, and encrypt that encryption key another encryption key. But, eventually, one key must remain in plaintext so you can decrypt the keys and your data. This top-level plaintext key encryption key is known as the *master key*.



AWS KMS helps you to protect your master keys by storing and managing them securely. Master keys stored in AWS KMS, known as [customer master keys \(p. 2\)](#) (CMKs), never leave the AWS KMS [FIPS validated hardware security modules](#) unencrypted. To use an AWS KMS CMK, you must call AWS KMS.



Envelope encryption offers several benefits:

- **Protecting data keys**

When you encrypt a data key, you don't have to worry about storing the encrypted data key, because the 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 re-encrypting raw data multiple times with different keys, you can re-encrypt 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. Envelope encryption lets you combine the strengths of each strategy.

Encryption Context

All AWS KMS cryptographic operations accept an *encryption context*, an optional set of key–value pairs that can contain additional contextual information about the data. When you provide an encryption context to an AWS KMS encryption operation, you must supply the same encryption context to the corresponding decryption operation. Otherwise, the request to decrypt fails.

The encryption context is not secret. It appears in plaintext in [AWS CloudTrail Logs \(p. 180\)](#) so you can use it to identify and categorize your cryptographic operations in logs and audits.

For example, [Amazon Simple Storage Service \(p. 162\)](#) (Amazon S3) uses an encryption context in which the key is `aws:s3:arn` and the value is the S3 bucket path to the file that is being encrypted.

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

You can also use the encryption context to refine or limit access to customer master keys (CMKs) in your account. You can use the encryption context [as a constraint in grants \(p. 78\)](#) and as a [condition in policy statements \(p. 59\)](#).

For detailed information about the encryption context, see [Encryption Context \(p. 248\)](#).

Key Policies

When you create a CMK, you determine who can use and manage that CMK. These permissions are contained in a document called the *key policy*. You can use the key policy to add, remove, or change 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 \(p. 29\)](#).

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 \(p. 78\)](#).

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 requester, the IP address that the request came from, and so on. For more information, see [Logging AWS KMS API Calls with AWS CloudTrail \(p. 180\)](#) 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 [AWS Management Console](#) and [AWS KMS API operations](#) to create, view, edit, tag, enable, disable, topics.

Topics

- [Creating Keys](#) (p. 8)
- [Viewing Keys](#) (p. 11)
- [Editing Keys](#) (p. 18)
- [Tagging Keys](#) (p. 24)
- [Enabling and Disabling Keys](#) (p. 26)

Creating Keys

You can create customer master key (CMK) in the AWS Management Console or by using the [CreateKey](#) operation.

Topics

- [Creating CMKs \(Console\)](#) (p. 8)
- [Creating CMKs \(KMS API\)](#) (p. 10)

Creating CMKs (Console)

You can use the AWS Management Console to create customer master keys (CMKs).

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To create a new CMK (new console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose **Create key**.
5. Type an alias for the CMK. The alias name cannot begin with **aws/**. The **aws/** prefix is reserved by Amazon Web Services to represent AWS-managed CMKs in your account.

An alias is a display name that you can use to identify the CMK. We recommend that you choose an alias that indicates the type of data you plan to protect or the application you plan to use with the CMK.

Aliases are required when you create a CMK in the AWS Management Console. They are optional when you use the [CreateKey](#) operation.

6. (Optional) Type a description for the CMK.

We recommend that you choose a description that explains the type of data you plan to protect or the application you plan to use with the CMK.

7. Choose **Next**.
8. (Optional) Type a tag key and an optional tag value. To add more than one tag to the CMK, choose **Add tag**.

When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. For information about tagging CMKs, see [Tagging Keys \(p. 24\)](#).

9. Choose **Next**.
10. Select the IAM users and roles that can administer the CMK.

Note

IAM policies can give other IAM users and roles permission to manage the CMK.

11. (Optional) To prevent the selected IAM users and roles from deleting this CMK, in the **Key deletion** section at the bottom of the page, clear the **Allow key administrators to delete this key** check box.
12. Choose **Next**.
13. Select the IAM users and roles that can use the CMK for cryptographic operations.

Note

The AWS account (root user) has full permissions by default. As a result, any IAM policies can also give users and roles permission use the CMK for cryptographic operations.

14. (Optional) You can allow other AWS accounts to use this CMK for cryptographic operations. To do so, in the **Other AWS accounts** section at the bottom of the page, choose **Add another AWS account** and enter the AWS account identification number of an external account. To add multiple external accounts, repeat this step.

Note

To allow principals in the external accounts to use the CMK, Administrators of the external account must create IAM policies that provide these permissions. For more information, see [Allowing External AWS Accounts to Access a CMK \(p. 47\)](#).

15. Choose **Next**.
16. Review the key policy document that was created from your choices. You can edit it, too.
17. Choose **Finish** to create the CMK.

Tip

To use your new CMK programmatically and in command line interface operations, you need a key ID or key ARN. For detailed instructions, see [Finding the Key ID and ARN \(p. 16\)](#)

To create a new CMK (original console)

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
3. Choose **Create key**.
4. Type an alias for the CMK. The alias name cannot begin with `aws`. The `aws` prefix is reserved by Amazon Web Services to identify [AWS managed CMKs \(p. 2\)](#) in your account.

An alias is a display name that you can use to identify the CMK. We recommend that you choose an alias that indicates the type of data you plan to protect or the application you plan to use with the CMK.

Aliases are required when you create a CMK in the AWS Management Console. They are optional when you use the [CreateKey](#) operation.

5. (Optional) Type a description for the CMK.

We recommend that you choose a description that explains the type of data you plan to protect or the application you plan to use with the CMK.

6. Choose **Next Step**.
7. (Optional) Type a [tag key](#) (p. 24) and an optional tag value. To add more than one tag to the CMK, choose **Add tag**.
8. Choose **Next Step**.
9. 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 allow the appropriate permissions can also administer the CMK.

10. (Optional) To prevent the IAM users and roles that you chose in the previous step from deleting this CMK, clear the box at the bottom of the page for **Allow key administrators to delete this key**.
11. Choose **Next Step**.
12. 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 allow the appropriate permissions can also use the CMK.

13. (Optional) You can use the controls at the bottom of the page to specify other AWS accounts that can use this CMK to encrypt and decrypt data. To do so, choose **Add an External Account** and then type the intended AWS account ID. 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. 47).

14. Choose **Next Step**.
15. Choose **Finish** to create the CMK.

Tip

To use your new CMK programmatically and in command line interface operations, you need a key ID or key ARN. For detailed instructions, see [Finding the Key ID and ARN](#) (p. 16)

Creating CMKs (KMS API)

The [CreateKey](#) operation creates a new AWS KMS customer master key (CMK). These examples use the [AWS Command Line Interface \(AWS CLI\)](#), but you can use any supported programming language.

This operation has no required parameters. However, if you are creating a key with no key material, the `Origin` parameter is required. You might also want to use the `Policy` parameter to specify a key policy. You can change the key policy ([PutKeyPolicy](#)) and add optional elements, such as a [description](#) and [tags](#) at any time.

The following is an example of a call to the `CreateKey` operation with no parameters.

```
$ aws kms create-key
{
```

```
    "KeyMetadata": {
      "Origin": "AWS_KMS",
      "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
      "Description": "",
      "KeyManager": "CUSTOMER",
      "Enabled": true,
      "KeyUsage": "ENCRYPT_DECRYPT",
      "KeyState": "Enabled",
      "CreationDate": 1502910355.475,
      "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "AWSAccountId": "111122223333"
    }
  }
}
```

If you do not specify a key policy for your new CMK, the [default key policy \(p. 34\)](#) that `CreateKey` applies differs from the default key policy that the console applies when you use it to create a new CMK.

For example, this call to the [GetKeyPolicy](#) operation returns the key policy that `CreateKey` applies. It gives the AWS account root user access to the CMK and allows it to create AWS Identity and Access Management (IAM) policies for the CMK. For detailed information about IAM policies and key policies for CMKs, see [Authentication and Access Control for AWS KMS \(p. 29\)](#)

```
$ aws kms get-key-policy --key-id 1234abcd-12ab-34cd-56ef-1234567890ab --policy-name
default --output text
{
  "Version" : "2012-10-17",
  "Id" : "key-default-1",
  "Statement" : [ {
    "Sid" : "Enable IAM User Permissions",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : "kms:*",
    "Resource" : "*"
  } ]
}
```

Viewing Keys

You can use [AWS Management Console](#) or the [AWS Key Management Service \(AWS KMS\) API](#) to view customer master keys (CMKs), including CMKs that you manage and CMKs that are managed by AWS.

Topics

- [Viewing CMKs \(Console\) \(p. 11\)](#)
- [Viewing CMKs \(KMS API\) \(p. 13\)](#)
- [Finding the Key ID and ARN \(p. 16\)](#)

Viewing CMKs (Console)

You can see a list of your customer managed keys in the AWS Management Console.

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except

for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.


To view your CMKs (new console)

To navigate to the CMK display

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**. To view the keys in your account that AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**.

The display shows all the CMKs of each type in your AWS account and region. By default, the page displays the alias, key ID, status, and creation date for each CMK, but you can customize it to show the information that you need.

To customize the CMK display

1. On the **AWS managed keys** or **Customer managed keys** page, choose the settings icon () in the upper-right corner of the page.
2. On the **Preferences** page, choose your preferred settings, and then choose **Confirm**.

To display CMK details

- On the **AWS managed keys** or **Customer managed keys** page, choose the alias or key ID of the CMK.

The details include the CMK ID, Amazon Resource Name (ARN), alias, description, key policy, tags, and key rotation settings of a CMK.

The Alias section lists only one alias. To find all aliases associated with the CMK, use the [ListAliases](#) operation.

To find CMKs by ID or alias name

- On the **AWS managed keys** or **Customer managed keys** page, in the **Filter** box, enter all or part of the alias name or key ID of a CMK. The filter searches all CMKs of each type, even if you have too many to display on the current page.


To view your CMKs (original console)

To navigate to the CMK display

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).

The **Encryption Keys** page lists the AWS managed and customer managed CMKs in your AWS account and region. By default, the page displays the alias, key ID, status, and creation date for each CMK, but you can customize it to meet your needs.

To customize the CMK display (optional)

1. Choose the settings button () in the upper-right corner of the page.
2. On the **Preferences** page, select your preferred options, and then choose **Close**.

To show detailed information about the CMK

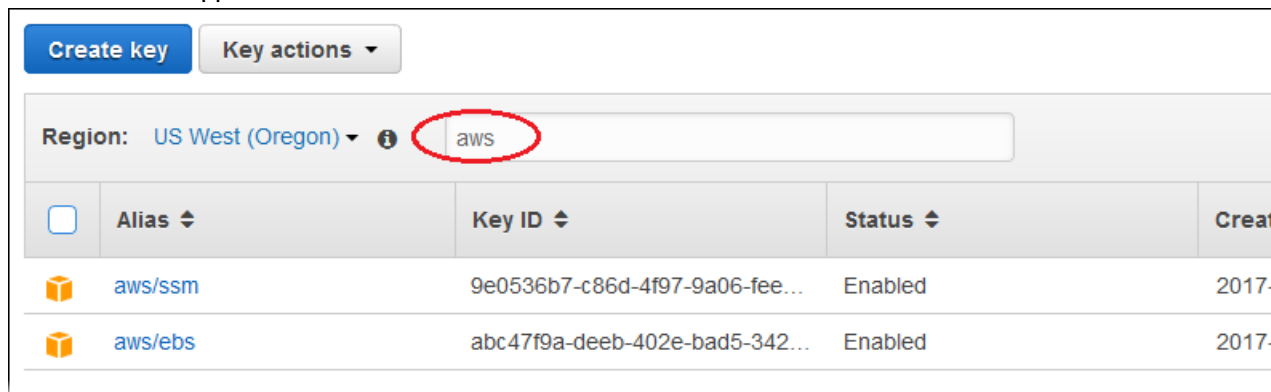
The details include the Amazon Resource Name (ARN), description, key policy, tags, and key rotation settings of the CMK.

- On the **Encryption keys** page, choose the alias or key ID of the CMK.

To find CMKs

You can use the **Filter** box to find CMKs based on their aliases.

- In the **Filter** box, type all or part of the alias name of a CMK. Only the CMKs with alias names that match the filter appear.



Viewing CMKs (KMS API)

You can use the [AWS Key Management Service \(AWS KMS\) API](#) to view your CMKs. This section demonstrates several operations that return details about existing CMKs. The examples use the [AWS Command Line Interface \(AWS CLI\)](#), but you can use any supported programming language.

Topics

- [ListKeys: Get the ID and ARN of All CMKs \(p. 13\)](#)
- [DescribeKey: Get Detailed Information About a CMK \(p. 14\)](#)
- [GetKeyPolicy: Get the Key Policy Attached to a CMK \(p. 14\)](#)
- [ListAliases: View CMKs by Alias Name \(p. 15\)](#)

ListKeys: Get the ID and ARN of All CMKs

The [ListKeys](#) operation returns the ID and Amazon Resource Name (ARN) of all CMKs in the account and region. To see the aliases and key IDs of your CMKs that have aliases, use the [ListAliases](#) operation.

For example, this call to the `ListKeys` operation returns the ID and ARN of each CMK in this fictitious account.

```
$ aws kms list-keys

{
  "Keys": [
    {
      "KeyArn": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
      "KeyArn": "arn:aws:kms:us-west-2:111122223333:key/0987dcb-a09fe-87dc-65ba-ab0987654321",
      "KeyId": "0987dcb-a09fe-87dc-65ba-ab0987654321"
    },
    {
      "KeyArn": "arn:aws:kms:us-east-2:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d",
      "KeyId": "1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d"
    }
  ]
}
```

DescribeKey: Get Detailed Information About a CMK

The `DescribeKey` operation returns details about the specified CMK. To identify the CMK, use its key ID, key ARN, alias name, or alias ARN.

For example, this call to `DescribeKey` returns information about an existing CMK. The fields in the response vary with the key state and the key origin.

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

{
  "KeyMetadata": {
    "Origin": "AWS_KMS",
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "Description": "",
    "KeyManager": "CUSTOMER",
    "Enabled": true,
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "CreationDate": 1499988169.234,
    "Arn": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "AWSAccountId": "111122223333"
  }
}
```

You can use the `DescribeKey` operation on a predefined AWS alias, that is, an AWS alias with no key ID. When you do, AWS KMS associates the alias with an [AWS managed CMK \(p. 2\)](#) and returns its `KeyId` and `Arn` in the response.

GetKeyPolicy: Get the Key Policy Attached to a CMK

The `GetKeyPolicy` operation gets the key policy that is attached to the CMK. To identify the CMK, use its key ID or key ARN. You must also specify the policy name, which is always `default`. (If your output is difficult to read, add the `--output text` option to your command.)

```
$ aws kms get-key-policy --key-id 1234abcd-12ab-34cd-56ef-1234567890ab --policy-name default

{
  "Version" : "2012-10-17",
  "Id" : "key-default-1",
  "Statement" : [ {
    "Sid" : "Enable IAM User Permissions",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : "kms:*",
    "Resource" : "*"
  } ]
}
```

ListAliases: View CMKs by Alias Name

The [ListAliases](#) operation returns aliases in the account and region. The `TargetKeyId` in the response displays the key ID of the CMK that the alias refers to, if any.

By default, the **ListAliases** command returns all aliases in the account and region. This includes [aliases that you created](#) and associated with your [customer-managed CMKs \(p. 2\)](#), and aliases that AWS created and associated with [AWS managed CMKs \(p. 2\)](#) in your account. You can recognize AWS aliases because their names have the format `aws/<service-name>`, such as `aws/dynamodb`.

The response might also include aliases that have no `TargetKeyId` field, such as the `aws/redshift` alias in this example. These are predefined aliases that AWS has created but has not yet associated with a CMK.

```
$ aws kms list-aliases

{
  "Aliases": [
    {
      "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/ImportedKey",
      "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
      "AliasName": "alias/ExampleKey"
    },
    {
      "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/test-key",
      "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
      "AliasName": "alias/test-key"
    },
    {
      "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/financeKey",
      "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
      "AliasName": "alias/financeKey"
    },
    {
      "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/aws/dynamodb",
      "TargetKeyId": "1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d",
      "AliasName": "alias/aws/dynamodb"
    },
    {
      "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/aws/redshift",
      "AliasName": "alias/aws/redshift"
    },
    {
      "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/aws/s3",
      "TargetKeyId": "0987ab65-43cd-21ef-09ab-87654321cdef",
    }
  ]
}
```

```
        "AliasName": "alias/aws/s3"
    }
  ]
}
```

To get the aliases that refer to a particular CMK, use the `KeyId` parameter. The parameter value can be the Amazon Resource Name (ARN) of the CMK or the CMK ID. You cannot specify an alias or alias ARN.

The command in the following example gets the aliases that refer to a customer managed CMK. But you can use a command like this one to find the aliases that refer to AWS managed CMKs, too.

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

{
  "Aliases": [
    {
      "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/test-key",
      "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
      "AliasName": "alias/test-key"
    },
    {
      "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/financeKey",
      "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
      "AliasName": "alias/financeKey"
    }
  ]
}
```

Finding the Key ID and ARN

To identify your AWS KMS CMKs in programs, scripts, and command line interface (CLI) commands, you use the ID of the CMK or its Amazon Resource Name (ARN). Cryptographic operations also let you use the CMK alias.

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To find the CMK ID and ARN (new console)

1. Open the AWS KMS console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**. To view the keys in your account that AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**.
4. To find the key ID, see the row that begins with the CMK alias. Each row displays the key ID and alias, along with the status and creation date of each CMK.

Customer managed keys					Key actions ▼	Create
<input type="text"/>						
<input type="checkbox"/>	Alias ▲	Key ID ▼	Status	Creation date		
<input type="checkbox"/>	master-key-test	1234abcd-12ab-34cd-56ef-1234567890ab	Enabled	2018-11-06 15:11 PST		

5. To find the Amazon Resource Name (ARN) of the CMK, choose the key ID or alias. This opens a page of details that includes the ARN.

1234abcd-12ab-34dc-56ef-1234567890ab		Key actions ▼	Create
General configuration			
ARN	arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34dc-56ef-1234567890ab	Status	Enabled
Alias	master-key-test	Description	-
Origin	AWS_KMS	Creation date	2018-11-06 15:11 PST

To find the CMK ID and ARN (original console)

To navigate to the CMK display

1. Go to the original AWS KMS console at <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
3. To find key ID, look on the row for the CMK alias. Each row displays the key ID and alias, along with the status and creation date of each CMK.

Create key

Key actions ▾

Region: US West (Oregon) ▾ ⓘ

Filter

<input type="checkbox"/>	Alias ▴ ▾	Key ID ▴	Status ▴ ▾	Creation Date ▴ ▾
<input type="checkbox"/>	master-key-test	1234abcd-12ab-34cd-56ef-1234567890ab	Enabled	2017-07-05 11:12 PDT

4. To find the CMK ARN (key ARN), choose the alias name. This opens a page of details that includes the key ARN.

IAM > Encryption Keys > test-key

▼ Summary

Region	us-west-2
ARN	arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1
Status	Enabled
Alias	test-key
Description	Key for testing KMS APIs

To find the CMK ID and ARN (KMS API)

Use the ListKeys API operation

- To find the CMK ID and ARN, use the [ListKeys](#) (p. 13) operation.

Editing Keys

You can use the AWS KMS API and the key detail page of the AWS Management Console to edit some of the properties of your customer managed [customer master keys](#) (p. 2) (CMKs). You can change the description, add and remove administrators and users, manage tags, and enable and disable key rotation.

You cannot change the properties of [AWS managed CMKs](#) (p. 2).

Topics

- [Editing CMKs \(Console\)](#) (p. 18)
- [Editing CMKs \(KMS API\)](#) (p. 23)

Editing CMKs (Console)

Users who have the required permissions can change the properties of a customer managed CMK, including its description, tags, policies and grants, and rotation status in the AWS Management Console.

You can [view](#) (p. 11), but not edit, the properties of AWS managed CMKs. To view the key policy for an AWS managed CMK, use the [GetKeyPolicy](#) operation.

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To edit a customer managed CMK (new console)

Navigate to the CMK details page

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**. (You cannot edit the properties of AWS managed keys.)
4. Choose the alias or key ID of the CMK that you want to edit. Now, use the controls on the key details page to view and change the properties of the CMK.

Change the CMK description

You can change the description of your CMK unless it is pending deletion. The description is optional.

1. In the upper-right corner, choose **Edit**.
2. For **Description**, type a brief description of the CMK.
3. To save your changes, choose **Save**.

Change CMK administrators and users

You can change the key policy for your CMK. Key policies define the IAM users, groups, and roles that can manage the CMK and use it for cryptographic operations.

The AWS account (root user) has full permissions by default. As a result, any IAM users and roles whose attached policies allow the appropriate permissions can also administer the CMK. For detailed information about setting key policies and IAM policies, see [Authentication and Access Control for AWS KMS \(p. 29\)](#).

1. Under **General configuration**, choose the **Key policy** tab.

If the key policy for the CMK is a default policy, the **Key policy** tab displays the default view with **Key administrators**, **Key deletion**, **Key users**, and **Other AWS accounts** sections. Otherwise, the tab displays the key policy document.

To edit the key policy document directly, choose **Switch to policy view** (if applicable), choose **Edit**, edit the document, then choose **Save**.

The remaining steps in this procedure explain how to edit the key policy using the default view.

2. To change the users and roles who can manage the CMK, use the **Key administrators** section.
 - To add a key administrator, choose **Add**, choose or type a user or role, then choose **Add**.
 - To remove a key administrator, check the box for the user or role, then choose **Remove**.
3. To prevent the key administrators from scheduling deletion of the CMK, in the **Key deletion** section, clear the **Allow key administrators to delete this key** check box.
4. To change the users and roles who can use the CMK in cryptographic operations, use the **Key users** section.
 - To add a key user, choose **Add**, choose a user or role, then choose **Add**.
 - To remove a key user, check the box for the user or role, then choose **Remove**.
5. To change the other AWS accounts that can use the CMK in cryptographic operations, in the **Other AWS accounts** section, choose **Add other AWS accounts**.

Note

Adding an external account does not allow users and roles in the account to use the CMK. To allow users and roles in an external account to use the CMK, an administrator

of the external account must add IAM policies that provide these permissions. For more information, see [Allowing External AWS Accounts to Access a CMK \(p. 47\)](#).

- To add accounts, choose **Add another AWS account**, type the account number.
- To remove accounts, on the row with the account number, choose **Remove**.

When you are done, choose **Save changes**, then click the **X** to close the window.

Add, edit, and delete tags

You can change the tags for your CMK. Each tag is a name–value pair. The tag name must be unique in the account and region.

You can use tags to identify and categorize your CMKs. When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. For more information about CMK tags, see [Tagging Keys \(p. 24\)](#).

- Under **General configuration**, choose the **Tags** tab.
 - To create your first tag, choose **Create tag**, type a tag name and tag value, and then choose **Save**.
 - To add a tag, choose **Edit**, choose **Add tag**, type a tag name and tag value, and then choose **Save**.
 - To change the name or value of a tag, choose **Edit**, make your changes, and then choose **Save**.
 - To delete a tag, choose **Edit**. On the tag row, choose **Remove**, and then choose **Save**.

Enable or disable rotation

You can enable and disable [automatic rotation \(p. 87\)](#) of the cryptographic material in a [customer managed CMK \(p. 2\)](#). This feature is not supported for CMKs with imported key material.

[AWS managed CMKs \(p. 2\)](#) are automatically rotated every three years. You cannot enable or disable this feature.

1. Under **General configuration**, choose the **Key rotation** tab.
2. To enable automatic key rotation, check the **Automatically rotate this CMK every year** check box. To disable automatic key rotation, clear the check box.
3. To save your changes, choose **Save**.

To edit a customer managed CMK (original console)

Navigate to the CMK details page

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
3. Choose the alias of the CMK whose details you want to see.

Note

You cannot edit AWS managed CMKs, which are denoted by the orange AWS icon.

On the key details page, you can view and edit the CMK.

Change the description

In the **Summary** section, type a brief description of the CMK in the **Description** box. To save your changes, choose **Save Changes**.



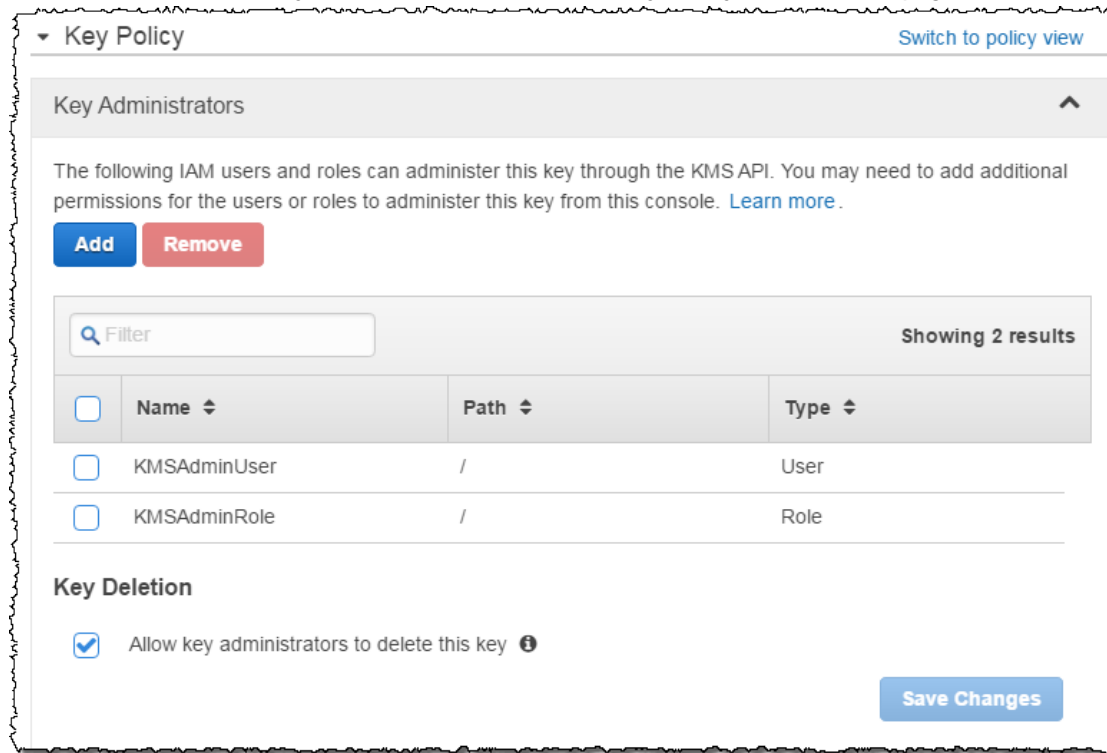
The screenshot shows the AWS KMS console interface for editing a CMK. The breadcrumb navigation at the top reads "IAM > Encryption Keys > example-key", with a "Back to Encryption Keys" link. The "Summary" section is expanded, displaying the following details:

- Region:** us-west-2
- ARN:** arn:aws:kms:us-west-2:111122223333:key/abcd1234-ef56-ab12-cd34-ef56abcd1234
- Alias:** example-key
- Description:** A text input field containing "Example CMK".

A red rectangular box highlights the "Description" input field and the "Save Changes" button located at the bottom right of the section.

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.



The screenshot shows the "Key Policy" section of the AWS KMS console, specifically the "Key Administrators" area. The breadcrumb navigation at the top reads "Key Policy", with a "Switch to policy view" link. The "Key Administrators" section is expanded, showing the following controls:

- Key Administrators:** A section header with an upward arrow icon.
- Description:** A text block stating: "The following IAM users and roles can administer this key through the KMS API. You may need to add additional permissions for the users or roles to administer this key from this console. [Learn more](#)."
- Buttons:** "Add" (blue) and "Remove" (red) buttons.
- Filter:** A search input field with a magnifying glass icon and the text "Filter".
- Results:** A table showing 2 results. The table has columns for "Name", "Path", and "Type".

<input type="checkbox"/>	Name ↕	Path ↕	Type ↕
<input type="checkbox"/>	KMSAdminUser	/	User
<input type="checkbox"/>	KMSAdminRole	/	Role

Below the table, the "Key Deletion" section is expanded, showing a checked checkbox for "Allow key administrators to delete this key" with an information icon. A "Save Changes" button is located at the bottom right of the section.

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.

Key Users

This Account

The following IAM users and roles can use this key to encrypt and decrypt data from within applications and when using AWS services integrated with KMS. [Learn more](#).

Add

Remove

Filter

Showing 2 results

<input type="checkbox"/>	Name	Path	Type
<input type="checkbox"/>	KMSUser	/	User
<input type="checkbox"/>	KMSRole	/	Role

External Accounts

The following external accounts can use this key to encrypt and decrypt data. Administrators of the accounts shown below are responsible for managing the permissions that allow their IAM users and roles to use this key.

arn:aws:iam::444455556666:root

Remove

+ Add External Account

Save Changes

Add, edit, and remove tags

Use the controls in the **Tags** section of the page.

Tags

Add tags to help organize and identify this key, and to help track your AWS costs. [Learn more](#).

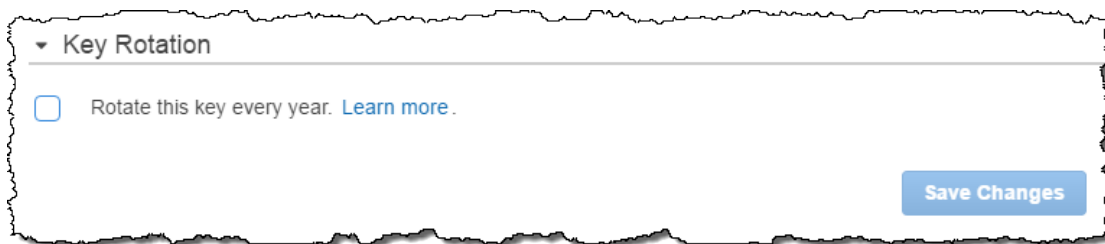
Tag key	Tag value	Remove
CreatedBy	ExampleUser	<input type="checkbox"/>
CostCenter	87654	<input type="checkbox"/>
Add unique key	Add value	<input type="checkbox"/>

Add tag

Save Changes

Enable or disable rotation

Use the controls in the **Key Rotation** section of the page to enable and disable [automatic rotation](#) (p. 87) of the cryptographic material in a customer managed CMK.



Editing CMKs (KMS API)

You can use the [AWS Key Management Service \(AWS KMS\) API](#) to edit the properties of your [customer managed CMKs \(p. 2\)](#). These examples use the [AWS Command Line Interface \(AWS CLI\)](#), but you can use any supported programming language. This section demonstrates several operations that return details about existing CMKs.

You cannot edit the properties of [AWS managed CMKs \(p. 2\)](#).

Topics

- [UpdateKeyDescription: Change the Description of a CMK \(p. 23\)](#)
- [PutKeyPolicy: Change the Key Policy for a CMK \(p. 24\)](#)
- [Enable and Disable Key Rotation \(p. 24\)](#)

Tip

For information about adding, deleting, and editing tags, see [Tagging Keys \(p. 24\)](#).

UpdateKeyDescription: Change the Description of a CMK

The [UpdateKeyDescription](#) operation adds or changes the description of a CMK. To see the description, use the [DescribeKey](#) operation.

For example, this call to the [UpdateKeyDescription](#) operation changes the description of the specified CMK.

```
$ aws kms update-key-description --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
                                --description "Example key"
```

To get the description of a key, use the [DescribeKey](#) operation, as shown in the following example.

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

{
  "KeyMetadata": {
    "Origin": "AWS_KMS",
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "Description": "Example key",
    "KeyManager": "CUSTOMER",
    "Enabled": true,
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "CreationDate": 1499988169.234,
    "Arn": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "AWSAccountId": "111122223333"
  }
}
```

```
}
```

PutKeyPolicy: Change the Key Policy for a CMK

The [PutKeyPolicy](#) operation changes the key policy of the CMK to the policy that you specify. The policy includes permissions for administrators, users, and roles. For a detailed example, see [PutKeyPolicy Examples](#).

Enable and Disable Key Rotation

The [EnableKeyRotation](#) operation enables [automatic rotation \(p. 87\)](#) of the cryptographic material in a CMK. The [DisableKeyRotation](#) operation disables it. The [GetKeyRotationStatus](#) operation returns a Boolean value that tells you whether automatic key rotation is enabled (**true**) or disabled (**false**).

For an example, see [Rotating Customer Master Keys \(p. 87\)](#).

Tagging Keys

You can add, change, and delete tags for [customer managed CMKs \(p. 2\)](#). Each tag consists of a *tag key* and a *tag value* that you define. For example, the tag key might be "Cost Center" and the tag value might be "87654." You cannot tag [AWS managed CMKs \(p. 2\)](#).

When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. You can use this feature to track AWS KMS costs for a project, application, or cost center.

For more information about using tags for cost allocation, see [Using Cost Allocation Tags](#) in the *AWS Billing and Cost Management User Guide*. For information about the rules that apply to tag keys and tag values, see [User-Defined Tag Restrictions](#) in the *AWS Billing and Cost Management User Guide*.

Topics

- [Managing CMK Tags \(Console\) \(p. 24\)](#)
- [Managing CMK Tags \(KMS API\) \(p. 25\)](#)

Managing CMK Tags (Console)

You can add, edit, and delete tags for your customer managed CMKs in the AWS Management Console. You can add tags to a CMK when you [create it \(p. 8\)](#) and edit them at any time. You cannot edit the tags of CMKs that are pending deletion. For more information, see [Editing Keys \(p. 18\)](#).

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To manage tags for your CMKs (new console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.

2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**. (You cannot manage the tags of an AWS managed CMK.)
4. Select the check box next to the alias of a CMK.
5. Choose **Key actions, Add or edit tags**.
6. Use the controls to add, edit, or delete tags. The tag name must be unique in the account and region.
7. To save your changes, choose **Save changes**.

To manage tags for your CMKs (original console)

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
3. Select the check box next to the alias of the CMKs whose tags you want to manage.

Note

You cannot tag AWS managed CMKs, which are denoted by the orange AWS icon.

4. Choose **Key actions, Add or edit tags**.
5. Use the controls in the **Add or edit tags** window. To save your changes, choose **Save Changes**.

Tag key	Tag value	Remove
CostCenter	87654	
CreatedBy	ExampleUser	

Managing CMK Tags (KMS API)

You can use the [AWS Key Management Service \(AWS KMS\) API](#) to add, delete, and list tags for the CMKs that you manage. These examples use the [AWS Command Line Interface \(AWS CLI\)](#), but you can use any supported programming language.

You cannot tag AWS managed CMKs.

Topics

- [TagResource: Add or Change Tags for a CMK \(p. 26\)](#)
- [ListResourceTags: Get the Tags for a CMK \(p. 26\)](#)
- [UntagResource: Delete Tags from a CMK \(p. 26\)](#)

TagResource: Add or Change Tags for a CMK

The [TagResource](#) operation adds one or more tags to a CMK.

You can also use **TagResource** to change the values for an existing tag. To replace tag values, specify the same tag key with different values. To add values to a tag, specify the tag key with both new and existing values.

For example, this call to the **TagResource** operation adds **Purpose** and **Department** tags to the specified CMK. You can use any keys and values as CMK tags.

```
$ aws kms tag-resource --key-id 1234abcd-12ab-34cd-56ef-1234567890ab /  
                        --tags TagKey=Purpose,TagValue=Test /  
                        TagKey=Department,TagValue=Finance
```

When this command is successful, it does not return any output. To view the tags on a CMK, use the [ListResourceTags](#) operation.

ListResourceTags: Get the Tags for a CMK

The [ListResourceTags](#) operation gets the tags for a CMK. The `key-id` parameter is required.

For example, the following command gets the tags for the specified CMK.

```
$ aws kms list-resource-tags --key-id 1234abcd-12ab-34cd-56ef-1234567890ab  
  
    "Truncated": false,  
    "Tags": [  
      {  
        "TagKey": "Purpose",  
        "TagValue": "Test"  
      },  
      {  
        "TagKey": "Department",  
        "TagValue": "Finance"  
      }  
    ]  
  }
```

UntagResource: Delete Tags from a CMK

The [UntagResource](#) operation deletes tags from a CMK. The `key-id` and `tag-keys` parameters are required.

For example, this command deletes the **Purpose** tag and all of its values from the specified CMK.

```
$ aws kms untag-resource --tag-keys Purpose --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
```

When this command is successful, it does not return any output.

Enabling and Disabling Keys

You can disable and reenable the AWS Key Management Service (AWS KMS) customer master keys (CMKs) that you manage. You cannot enable or disable AWS managed 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 until you re-enable it. AWS managed CMKs are permanently enabled for use by [services that use AWS KMS \(p. 127\)](#). You cannot disable them.

You can also delete CMKs. For more information, see [Deleting Customer Master Keys \(p. 110\)](#).

Note

AWS KMS does not rotate the backing keys of customer-managed CMKs while they are disabled. For more information, see [How Automatic Key Rotation Works \(p. 88\)](#).

Topics

- [Enabling and Disabling CMKs \(Console\) \(p. 27\)](#)
- [Enabling and Disabling CMKs \(KMS API\) \(p. 27\)](#)

Enabling and Disabling CMKs (Console)

You can enable and disable customer managed CMKs from the IAM section of the AWS Management Console.

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To enable or disable a CMK (new console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Select the check box for the CMKs that you want to enable or disable.
5. To enable a CMK, choose **Key actions, Enable**. To disable a CMK, choose **Key actions, Disable**.

To enable a CMK (original console)

To enable a CMK (console)

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
3. Select the check box next to the alias of the CMKs that you want to enable or disable.

Note

You cannot disable AWS managed CMKs, which are denoted by the orange AWS icon.

4. To enable a CMK, choose **Key actions, Enable**. To disable a CMK, choose **Key actions, Disable**.

Enabling and Disabling CMKs (KMS API)

The [EnableKey](#) operation enables a disabled AWS KMS customer master key (CMK). These examples use the [AWS Command Line Interface \(AWS CLI\)](#), but you can use any supported programming language. The `key-id` parameter is required.

This operation does not return any output. To see the key status, use the [DescribeKey](#) operation.

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

The [DisableKey](#) operation disables an enabled CMK. The `key-id` parameter is required.

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

This operation does not return any output. To see the key status, use the [DescribeKey](#) operation, and see the `Enabled` field.

```
$ aws kms describe-key --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
  "KeyMetadata": {
    "Origin": "AWS_KMS",
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "Description": "",
    "KeyManager": "CUSTOMER",
    "Enabled": false,
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Disabled",
    "CreationDate": 1502910355.475,
    "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "AWSAccountId": "111122223333"
  }
}
```

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 \(p. 29\)](#)
- [Access Control \(p. 30\)](#)

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. 30)
- [Using Key Policies](#) (p. 33)
- [Using IAM Policies](#) (p. 50)
- [AWS KMS API Permissions Reference](#) (p. 54)
- [Using Policy Conditions](#) (p. 59)
- [Using Grants](#) (p. 78)

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 \(p. 31\)](#)
- [Managing Access to AWS KMS CMKs \(p. 31\)](#)
- [Specifying Permissions in a Policy \(p. 32\)](#)
- [Specifying Conditions in a Policy \(p. 33\)](#)

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 \(p. 54\)](#).

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 policies*), 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 policies 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 \(p. 33\)](#).

For more information about using IAM policies, see [Using IAM Policies \(p. 50\)](#).

For more information about using grants, see [Using Grants \(p. 78\)](#).

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 \(p. 54\)](#).

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 \(p. 31\)](#). 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 \(p. 33\)](#) and [Using IAM Policies \(p. 50\)](#).

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 \(p. 59\)](#).

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 without them. For more information, see [Managing Access to AWS KMS CMKs \(p. 31\)](#).

Topics

- [Overview of Key Policies \(p. 33\)](#)
- [Default Key Policy \(p. 34\)](#)
- [Example Key Policy \(p. 40\)](#)

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 \(p. 34\)](#) and [Changing a Key Policy \(p. 43\)](#).

A key policy document cannot exceed 32 KB (32,768 bytes). Key policy documents use the same JSON syntax as 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. You might do this to make sure that a user cannot access it, even when a different policy allows access.

- **Principal** – (Required) The [principal](#) is the identity that gets the permissions specified in the policy statement. 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. 54).
- **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. 59).

For more information about AWS policy syntax, see [AWS IAM Policy Reference](#) in the *IAM User Guide*.

Default Key Policy

Default key policy when you create a CMK programmatically

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 policy statement that gives the AWS account (root user) that owns the CMK full access to the CMK and enables IAM policies in the account to allow access to the CMK. For more information about this policy statement, see [Allows Access to the AWS Account and Enables IAM Policies](#) (p. 34).

Default key policy when you create a CMK with the AWS Management Console

When you [create a CMK with the AWS Management Console](#) (p. 8), you can 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 default 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. 34)
- [Allows Key Administrators to Administer the CMK](#) (p. 35)
- [Allows Key Users to Use the CMK](#) (p. 37)

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 the following two things.

1. Reduces the risk of the CMK becoming unmanageable.

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 the CMK, because the root user can access a CMK

only when the key policy explicitly allows it. This is different from most other resources in AWS, which implicitly allow access to the root user.

2. Enables IAM policies to allow access to the CMK.

IAM policies by themselves are not sufficient to allow access to a CMK. However, 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 does this; it 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. 31).

The following example shows the policy statement that allows access to the AWS account and thereby enables IAM policies.

```
{
  "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.

Warning

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. This 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. You can also 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 available on the key details page for each CMK.

Key Policy

Switch to policy view

Key Administrators

The following IAM users and roles can administer this key through the KMS API. You may need to add additional permissions for the users or roles to administer this key from this console. [Learn more](#).

Add

Remove

Filter

Showing 2 results

<input type="checkbox"/>	Name ↕	Path ↕	Type ↕
<input type="checkbox"/>	KMSAdminUser	/	User
<input type="checkbox"/>	KMSAdminRole	/	Role

Key Deletion

☒ Allow key administrators to delete this key ⓘ

Save Changes

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. This statement is 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:TagResource",
    "kms:UntagResource",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "*"
}
```

The key administrators statement allows the following permissions:

- **kms:Create*** – Allows key administrators to create aliases and [grants \(p. 78\)](#) for this CMK.
- **kms:Describe*** – Allows key administrators to retrieve information about this 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 this CMK's state to enabled and to specify [annual rotation of the CMK's key material \(p. 87\)](#).
- **kms:List*** – Allows key administrators to retrieve lists of the aliases, grants, key policies, and tags 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 this CMK's description.
- **kms:Revoke*** – Allows key administrators to revoke the permissions for this CMK that are allowed by a [grant \(p. 78\)](#).
- **kms:Disable*** – Allows key administrators to set this CMK's state to disabled and to disable [annual rotation of this CMK's key material \(p. 87\)](#).
- **kms:Get*** – Allows key administrators to retrieve the key policy for this CMK and to determine whether this CMK's key material is rotated annually. If this CMK's origin is external, it also allows key administrators to download the public key and import token for this CMK. For more information about CMK origin, see [Importing Key Material \(p. 93\)](#).
- **kms>Delete*** – Allows key administrators to delete an alias that points to this CMK and, if this CMK's origin is external, to delete the imported key material. For more information about imported key material, see [Importing Key Material \(p. 93\)](#). Note that this permission does not allow key administrators to [delete the CMK \(p. 110\)](#).
- **kms:ImportKeyMaterial** – Allows key administrators to import key material into the CMK.

Note

This permission is not shown in the preceding example policy statement. This permission is applicable only to CMKs whose origin is external. It is automatically included in the key administrators statement when you use the console to [create a CMK with no key material \(p. 96\)](#). For more information, see [Importing Key Material \(p. 93\)](#).

- **kms:TagResource** – Allows key administrators to add and update tags for this CMK.
- **kms:UntagResource** – Allows key administrators to remove tags from this CMK.
- **kms:ScheduleKeyDeletion** – Allows key administrators to [delete this CMK \(p. 110\)](#).
- **kms:CancelKeyDeletion** – Allows key administrators to cancel the pending deletion of this CMK.

The final two permissions in the preceding list, `kms:ScheduleKeyDeletion` and `kms:CancelKeyDeletion`, are included by default when you [create a CMK \(p. 8\)](#). However, you can optionally remove them from the key policy when you create a CMK by clearing the box for **Allow key administrators to delete this key**. In the same way, you can use the key details page to remove them from the default key policy for existing CMKs. For more information, see [Editing Keys \(p. 18\)](#).

Many of these permissions contain the wildcard character (*). That 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.

Note

The key administrators statement described in the preceding section is in the latest version of the default key policy. For information about previous versions of the default key policy, see [Keeping Key Policies Up to Date \(p. 48\)](#).

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

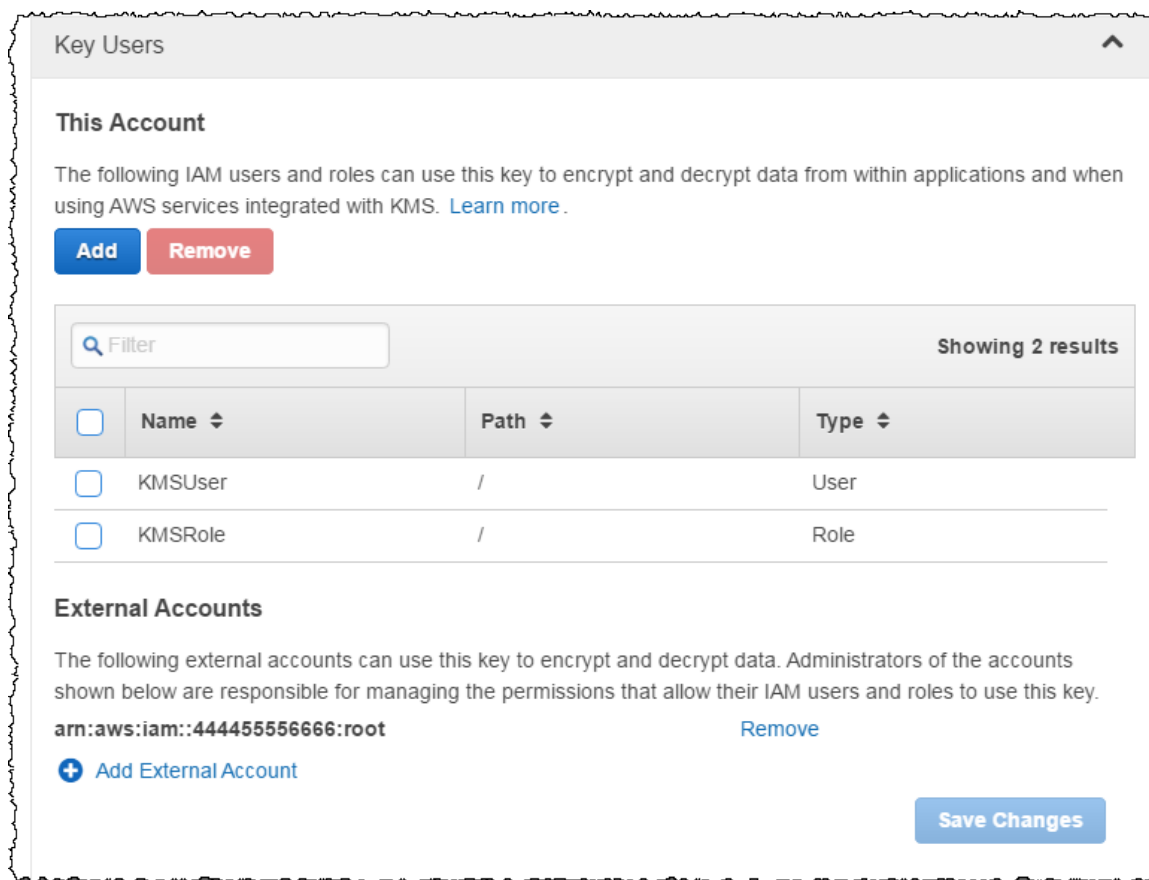
directly for encryption and decryption. They also have permission to delegate a subset of their own permissions to some of the [AWS services that are integrated with AWS KMS \(p. 127\)](#). Key users can implicitly give these services permissions to use the CMK in specific and limited ways. This implicit delegation is done using [grants \(p. 78\)](#). 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 \(p. 139\)](#).
- 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 \(p. 149\)](#).
- Use this CMK with other [AWS services integrated with AWS KMS \(p. 127\)](#), 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. 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 the integrated services but not others. However, you can create a custom key policy to do this. For more information, see the [kms:ViaService \(p. 75\)](#) condition key.

You can add IAM users, IAM roles, and external AWS accounts to the list of key users when you create the CMK. You can also 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 key 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. These statements are called 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",
  ] },
  "Action": [
    "kms:CreateKey",
    "kms:CreateAlias",
    "kms:UpdateAlias",
    "kms:DeleteAlias",
    "kms:ScheduleKeyDeletion",
    "kms:CancelKeyDeletion"
  ],
  "Resource": "*"
}
```

```
"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 this CMK.
- **kms:Decrypt** – Allows key users to successfully request that AWS KMS decrypt data with this CMK.
- **kms:ReEncrypt*** – Allows key users to successfully request that AWS KMS re-encrypt 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 (data keys) to use for client-side encryption. Key users can choose to receive two copies of the data key—one in plaintext form and one that is encrypted with this 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 \(p. 127\)](#), 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 \(p. 59\)](#).

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 \(p. 34\)](#) 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:TagResource",
      "kms:UntagResource",
      "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.

Key Policy

Switch to policy view

Key Administrators

The following IAM users and roles can administer this key through the KMS API. You may need to add additional permissions for the users or roles to administer this key from this console. [Learn more](#).

Add

Remove

Filter

Showing 2 results

<input type="checkbox"/>	Name ↕	Path ↕	Type ↕
<input type="checkbox"/>	KMSAdminUser	/	User
<input type="checkbox"/>	KMSAdminRole	/	Role

Key Deletion

☒ Allow key administrators to delete this key ⓘ

Save Changes

Key Users

This Account

The following IAM users and roles can use this key to encrypt and decrypt data from within applications and when using AWS services integrated with KMS. [Learn more](#).

Add

Remove

Filter

Showing 2 results

<input type="checkbox"/>	Name ↕	Path ↕	Type ↕
<input type="checkbox"/>	KMSUser	/	User
<input type="checkbox"/>	KMSRole	/	Role

External Accounts

The following external accounts can use this key to encrypt and decrypt data. Administrators of the accounts shown below are responsible for managing the permissions that allow their IAM users and roles to use this key.

arn:aws:iam::444455556666:root

Remove

+ Add External Account

Save Changes

Changing a Key Policy

To change the permissions for a customer master key (CMK) in AWS KMS, you change the CMK's [key policy](#) (p. 33).

When changing a key policy, keep in mind the following rules:

- 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. 33).
- You cannot add IAM groups to a key policy, but you can add multiple IAM users. For more information, see [Allowing Multiple IAM Users to Access a CMK](#) (p. 46).
- If 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. 47).
- The resulting key policy document cannot exceed 32 KB (32,768 bytes).

Topics

- [How to Change a Key Policy](#) (p. 43)
- [Allowing Multiple IAM Users to Access a CMK](#) (p. 46)
- [Allowing External AWS Accounts to Access a CMK](#) (p. 47)

How to Change a Key Policy

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

Topics

- [Using the AWS Management Console Default View](#) (p. 43)
- [Using the AWS Management Console Policy View](#) (p. 45)
- [Using the AWS KMS API](#) (p. 46)

Using the AWS Management Console Default View

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

If the following steps don't match what you see in the console, it might mean that this key policy was not created by the console. Or it might mean that 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 Policy View](#) (p. 45) or [Using the AWS KMS API](#) (p. 46).

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM

console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To change a key policy using the console default view (new console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**. (You cannot change the key policies of AWS managed keys.)
4. Choose the alias or key ID of the CMK whose key policy you want to change.
5. Scroll down to the **Key policy** tab.
6. Decide what to change.
 - To add or remove [key administrators \(p. 35\)](#), and to allow or prevent key administrators from [deleting the CMK \(p. 110\)](#), use the controls in the **Key administrators** section of the page. Key administrators manage the CMK, including enabling and disabling it, setting key policy, and [enabling key rotation \(p. 87\)](#).
 - To add or remove [key users \(p. 37\)](#), and to allow or disallow external AWS accounts to use the CMK, use the controls in the **Key users** section of the page. Key users can use the CMK in cryptographic operations, such as encrypting, decrypting, re-encrypting, and generating data keys.

To change a key policy using the console default view (original console)

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
3. Choose the alias of the CMK whose key policy you want to change.
4. Decide what to change.
 - To add or remove [key administrators \(p. 35\)](#), and to allow or disallow key administrators to [delete the CMK \(p. 110\)](#), use the controls in the **Key Administrators** area in the **Key Policy** section of the page.

The screenshot shows the 'Key Policy' section of the AWS console. At the top, there's a tab labeled 'Key Policy' and a link 'Switch to policy view'. Below this is the 'Key Administrators' section, which includes a description: 'The following IAM users and roles can administer this key through the KMS API. You may need to add additional permissions for the users or roles to administer this key from this console. [Learn more](#).' There are 'Add' and 'Remove' buttons. Below this is a table with a search filter and 'Showing 2 results'. The table has columns for 'Name', 'Path', and 'Type'. It lists 'KMSAdminUser' (User) and 'KMSAdminRole' (Role). At the bottom, there's a 'Key Deletion' section with a checkbox 'Allow key administrators to delete this key' which is checked. A 'Save Changes' button is at the bottom right.

	Name	Path	Type
<input type="checkbox"/>	KMSAdminUser	/	User
<input type="checkbox"/>	KMSAdminRole	/	Role

- To add or remove [key users \(p. 37\)](#), 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.

The screenshot shows the 'Key Users' interface in the AWS Management Console. It has a title bar 'Key Users' with an upward arrow. Below is a section 'This Account' with a description: 'The following IAM users and roles can use this key to encrypt and decrypt data from within applications and when using AWS services integrated with KMS. [Learn more.](#)' There are 'Add' and 'Remove' buttons. Below this is a table with a search filter and 'Showing 2 results'. The table has columns: Name, Path, and Type. It lists 'KMSUser' (User) and 'KMSRole' (Role). Below the table is an 'External Accounts' section with a description: 'The following external accounts can use this key to encrypt and decrypt data. Administrators of the accounts shown below are responsible for managing the permissions that allow their IAM users and roles to use this key.' It shows an account 'arn:aws:iam::444455556666:root' with a 'Remove' button. At the bottom, there is an 'Add External Account' button with a plus icon and a 'Save Changes' button.

Name	Path	Type
KMSUser	/	User
KMSRole	/	Role

Using the AWS Management Console Policy View

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

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To change a key policy document using the console policy view (new console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**. (You cannot change the key policy of an AWS managed CMK.)
4. Choose the alias or key ID of the CMK that you want to change.
5. Scroll down to the **Key policy** tab.
6. In the **Key Policy** section, choose **Switch to policy view**.
7. Edit the key policy document, and then choose **Save changes**.

To change a key policy document using the console policy view (original console)

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.

2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation 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 change 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 \(p. 223\)](#).

To change a key policy document (KMS 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 \(p. 34\)](#). 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 change 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 \(p. 80\)](#).

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. Change the key policy for the CMK in account 444455556666.
2. Add an IAM policy (or change 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.

Changing 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 Default View \(p. 43\)](#).

You can also change the key policy document directly using the console's policy view or the AWS KMS API, as described in [Using the AWS Management Console Policy View \(p. 45\)](#) and [Using the AWS KMS API \(p. 46\)](#).

- To add the external account to a key policy as a *key administrator* or give custom permissions, you must change 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 Policy View \(p. 45\)](#) or [Using the AWS KMS API \(p. 46\)](#).

For an example of JSON syntax that adds an external account to the `Principal` element of a key policy document, see [the policy statement in the default console key policy \(p. 37\)](#) that allows key users to use the CMK.

Adding or Changing 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 change 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 \(p. 127\)](#), 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 \(ARN\) \(p. 31\)](#) 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 \(p. 50\)](#).

Keeping Key Policies Up to Date

When you [use the AWS Management Console to create a customer master key \(CMK\) \(p. 8\)](#), you can choose the IAM users, IAM roles, and AWS accounts that you want to have access to the CMK. These users, roles, and accounts are added to a [default key policy \(p. 34\)](#) that controls access to the CMK. Occasionally, the default key policy for new CMKs is updated. Typically, these updates correspond to new AWS KMS features.

When you create a new CMK, the latest version of the default key policy is added to the CMK. However, existing CMKs continue to use their existing key policy—that is, new versions of the default key policy are *not* automatically applied to existing CMKs. Instead, the console alerts you that a newer version is available and prompts you to upgrade it.

Note

The console alerts you only when you are using the default key policy that was applied when you created the CMK. If you manually modified the key policy document using the console's

policy view or the [PutKeyPolicy](#) API operation, the console does not alert you when new permissions are available.

For information about the permissions that are added to a key policy when you upgrade it, see [Changes to the Default Key Policy \(p. 50\)](#). Upgrading to the latest version of the key policy should not cause problems because it only adds permissions; it doesn't remove any. We recommend keeping your key policies up to date unless you have a specific reason not to.

Determining whether a newer version of the default key policy is available

You can use the AWS Management Console to learn whether a newer version of the default key policy is available.

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To find a newer version of the default key policy (new console)

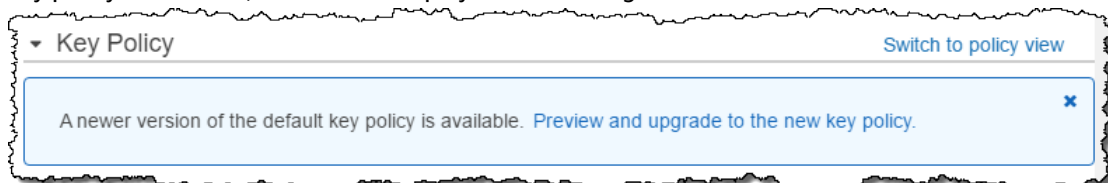
1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose the alias or key ID of a CMK that uses the default key policy.
5. Scroll down to the **Key policy** section of the page.

When a newer version of the default key policy is available, the console displays the following alert.

A newer version of the default key policy is available. Preview and upgrade to the new key policy.

To find a newer version of the default key policy (original console)

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
3. Choose the alias of the CMK whose key policy you want to see. When a newer version of the default key policy is available, the console displays the following alert.



Upgrading to the latest version of the default key policy

When a new default key policy is available, the following alert is displayed in the Key Policy section of the console page.

A newer version of the default key policy is available. Preview and upgrade to the new key policy.

To upgrade to the latest version of the default key policy

1. If you see an alert announcing a newer version of the default key policy, choose **Preview and upgrade to the new key policy**.
2. Review the key policy document for the latest version of the default key policy. For more information about the difference between the latest version and previous versions, see [Changes to the Default Key Policy \(p. 50\)](#). After reviewing the key policy, choose **Upgrade key policy**.

Changes to the Default Key Policy

In the [current version of the default key policy \(p. 34\)](#), the *key administrators statement* contains more permissions than those in previous versions. These additional permissions correspond to new AWS KMS features.

CMKs that use an earlier version of the default key policy might be missing the following permissions. When you upgrade to the latest version of the default key policy, they're added to the key administrators statement.

kms:TagResource and kms:UntagResource

These permissions allow key administrators to add, update, and remove tags from the CMK. They were added to the default key policy when AWS KMS released the [tagging feature \(p. 24\)](#).

kms:ScheduleKeyDeletion and kms:CancelKeyDeletion

These permissions allow key administrators to schedule and cancel deletion for the CMK. They were added to the default key policy when AWS KMS released the [CMK deletion feature \(p. 110\)](#).

Note

The `kms:ScheduleKeyDeletion` and `kms:CancelKeyDeletion` permissions are included by default when you [create a CMK \(p. 8\)](#) and when you upgrade to the latest version of the default key policy. However, you can optionally remove them from the default key policy when you create a CMK by clearing the box for **Allow key administrators to delete this key**. In the same way, you can use the key details page to remove them from the default key policy for existing CMKs. That includes CMKs whose key policy you upgraded to the latest version.

Using IAM Policies with AWS KMS

You can use IAM policies in combination with [key policies \(p. 33\)](#) 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 \(p. 34\)](#).

Topics

- [Overview of IAM Policies \(p. 51\)](#)
- [Permissions Required to Use the AWS KMS Console \(p. 51\)](#)
- [AWS Managed \(Predefined\) Policies for AWS KMS \(p. 52\)](#)
- [Customer Managed Policy Examples \(p. 52\)](#)

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 \(p. 54\)](#).

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 \(p. 52\)](#).

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. 54).

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. 33). 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. 54).

Examples

- [Allow a User Read-Only Access to All CMKs through the AWS KMS Console](#) (p. 52)
- [Allow a User to Encrypt and Decrypt with Any CMK in a Specific AWS Account](#) (p. 53)
- [Allow a User to Encrypt and Decrypt with Any CMK in a Specific AWS Account and Region](#) (p. 53)
- [Allow a User to Encrypt and Decrypt with Specific CMKs](#) (p. 53)
- [Prevent a User from Disabling or Deleting Any CMKs](#) (p. 54)

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 11112223333.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt"
    ],
    "Resource": [
      "arn:aws:kms:*:11112223333: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 11112223333 in the US West (Oregon) region.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt"
    ],
    "Resource": [
      "arn:aws:kms:us-west-2:11112223333: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

The Actions and Resources Table is designed to help you define [access control \(p. 30\)](#) in [key policies \(p. 33\)](#) and [IAM policies \(p. 50\)](#). The columns provide the following information:

- **API Operations and Actions (Permissions)** lists each AWS KMS API operation and the corresponding action (permission) that allows the operation. You specify actions in a policy's `Action` element.
- **Policy Type** indicates whether the permission can be used in a key policy or IAM policy. When the type is *key policy*, you can specify the permission explicitly in the key policy. Or, if the key policy contains the [policy statement that enables IAM policies \(p. 34\)](#), you can specify the permission in an IAM policy. When the type is *IAM policy*, you can specify the permission only in an IAM policy.
- **Resources and ARNs** lists the resources for which you can allow the operation. To specify a resource in an IAM policy, type the Amazon Resource Name (ARN) in the `Resource` element. Because a key policy applies only to the CMK that it is attached to, the value of its `Resource` element is always `"*"`.
- **AWS KMS Condition Keys** lists the AWS KMS condition keys that 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 \(p. 61\)](#). This column also includes [AWS global condition keys](#) that are supported by AWS KMS, but not by all AWS services.

AWS KMS API Operations and Permissions

API Operations and Actions (Permissions)	Policy Type	Resources and ARNs (for IAM Policies)	AWS KMS Condition Keys
CancelKeyDeletion <code>kms:CancelKeyDeletion</code>	Key policy	CMK <code>arn:aws:kms:<i>AWS_region</i>:<i>AWS_account_ID</i>:key/<i>CMK_key_ID</i></code>	<code>kms:CallerAccount</code> <code>kms:ViaServiceID:key/</code>
CreateAlias <code>kms:CreateAlias</code> To use this operation, the caller needs <code>kms:CreateAlias</code> permission on two resources: <ul style="list-style-type: none"> The alias (in an IAM policy) The CMK (in a key policy) 	IAM policy (for the alias)	Alias <code>arn:aws:kms:<i>AWS_region</i>:<i>AWS_account_ID</i>:alias/<i>alias_name</i></code>	None (when controlling access to the alias)
	Key policy (for the CMK)	CMK <code>arn:aws:kms:<i>AWS_region</i>:<i>AWS_account_ID</i>:key/<i>CMK_key_ID</i></code>	<code>kms:CallerAccount</code> <code>kms:ViaServiceID:key/</code>
CreateGrant <code>kms:CreateGrant</code>	Key policy	CMK <code>arn:aws:kms:<i>AWS_region</i>:<i>AWS_account_ID</i>:key/<i>CMK_key_ID</i></code>	<code>kms:CallerAccount</code> <code>kms:EncryptionContextKey</code> <code>kms:EncryptionContextKeys</code> <code>kms:GrantConstraintType</code> <code>kms:GranteePrincipal</code> <code>kms:GrantIsForAWSResource</code> <code>kms:GrantOperations</code> <code>kms:RetiringPrincipal</code> <code>kms:ViaService</code>
CreateKey <code>kms:CreateKey</code>	IAM policy	*	<code>kms:BypassPolicyLockoutSafety</code> <code>kms:KeyOrigin</code>
Decrypt <code>kms:Decrypt</code>	Key policy	CMK <code>arn:aws:kms:<i>AWS_region</i>:<i>AWS_account_ID</i>:key/<i>CMK_key_ID</i></code>	<code>kms:CallerAccount</code> <code>kms:EncryptionContextKey</code> <code>kms:EncryptionContextKeys</code> <code>kms:ViaService</code>
DeleteAlias <code>kms>DeleteAlias</code> To use this operation, the caller needs	IAM policy (for the alias)	Alias <code>arn:aws:kms:<i>AWS_region</i>:<i>AWS_account_ID</i>:alias/<i>alias_name</i></code>	None (when controlling access to the alias)
	Key policy (for the CMK)	CMK	<code>kms:CallerAccount</code>

API Operations and Actions (Permissions)	Policy Type	Resources and ARNs (for IAM Policies)	AWS KMS Condition Keys
kms:DeleteAlias permission on two resources: <ul style="list-style-type: none"> The alias (in an IAM policy) The CMK (in a key policy) 		arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:ViaServiceID:key/
DeleteImportedKeyMaterial kms:DeleteImportedKeyMaterial	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaServiceID:key/
DescribeKey kms:DescribeKey	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaServiceID:key/
DisableKey kms:DisableKey	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaServiceID:key/
DisableKeyRotation kms:DisableKeyRotation	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaServiceID:key/
EnableKey kms:EnableKey	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaServiceID:key/
EnableKeyRotation kms:EnableKeyRotation	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaServiceID:key/
Encrypt kms:Encrypt	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:EncryptionContextKey/ kms:EncryptionContextKeys kms:ViaService
GenerateDataKey kms:GenerateDataKey	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:EncryptionContextKey/ kms:EncryptionContextKeys kms:ViaService

API Operations and Actions (Permissions)	Policy Type	Resources and ARNs (for IAM Policies)	AWS KMS Condition Keys
GenerateDataKeyWithoutPlaintext kms:GenerateDataKeyWithoutPlaintext	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_id</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:EncryptionContextKey kms:EncryptionContextKeys kms:ViaService
GenerateRandom kms:GenerateRandom	IAM policy	*	None
GetKeyPolicy kms:GetKeyPolicy	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_id</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaService_ID:key/
GetKeyRotationStatus kms:GetKeyRotationStatus	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_id</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaService_ID:key/
GetParametersForImport kms:GetParametersForImport	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_id</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaService_ID:key/ kms:WrappingAlgorithm kms:WrappingKeySpec
ImportKeyMaterial kms:ImportKeyMaterial	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_id</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ExpirationMode kms:ValidTo kms:ViaService
ListAliases kms:ListAliases	IAM policy	*	None
ListGrants kms:ListGrants	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_id</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaService_ID:key/
ListKeyPolicies kms:ListKeyPolicies	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_id</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaService_ID:key/
ListKeys kms:ListKeys	IAM policy	*	None

API Operations and Actions (Permissions)	Policy Type	Resources and ARNs (for IAM Policies)	AWS KMS Condition Keys
ListResourceTags kms:ListResourceTags	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaService
ListRetirableGrants kms:ListRetirableGrants	IAM policy	*	None
PutKeyPolicy kms:PutKeyPolicy	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:BypassPolicyLockoutSafety kms:CallerAccount kms:ViaService
ReEncrypt kms:ReEncryptFrom kms:ReEncryptTo <p>To use this operation, the caller needs permission on two CMKs:</p> <ul style="list-style-type: none"> kms:ReEncryptFrom on the CMK used to decrypt kms:ReEncryptTo on the CMK used to encrypt 	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:EncryptionContextKey kms:EncryptionContextKeys kms:ReEncryptOnSameKey kms:ViaService
RetireGrant <p>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 <i>AWS Key Management Service API Reference</i>.</p>	Not applicable	Not applicable	Not applicable
RevokeGrant kms:RevokeGrant	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaService
ScheduleKeyDeletion kms:ScheduleKeyDeletion	Key policy	CMK arn:aws:kms: <i>AWS_region</i> : <i>AWS_account_ID</i> :key/ <i>CMK_key_ID</i>	kms:CallerAccount kms:ViaService

API Operations and Actions (Permissions)	Policy Type	Resources and ARNs (for IAM Policies)	AWS KMS Condition Keys
TagResource <code>kms:TagResource</code>	Key policy	CMK <code>arn:aws:kms:<i>AWS_region</i>:<i>AWS_account_ID</i>:key/<i>CMK_key_ID</i></code>	<code>kms:CallerAccount</code> <code>kms:AWSServiceID:key/</code> <code>aws:RequestTag</code> (AWS global condition key) <code>aws:TagKeys</code> (AWS global condition key)
UntagResource <code>kms:UntagResource</code>	Key policy	CMK <code>arn:aws:kms:<i>AWS_region</i>:<i>AWS_account_ID</i>:key/<i>CMK_key_ID</i></code>	<code>kms:CallerAccount</code> <code>kms:AWSServiceID:key/</code> <code>aws:RequestTag</code> (AWS global condition key) <code>aws:TagKeys</code> (AWS global condition key)
UpdateAlias <code>kms:UpdateAlias</code> To use this operation, the caller needs <code>kms:UpdateAlias</code> permission on three resources: <ul style="list-style-type: none"> • The alias • The CMK that that alias currently points to • The CMK that is specified in the <code>UpdateAlias</code> request 	IAM policy (for the alias)	Alias <code>arn:aws:kms:<i>AWS_region</i>:<i>AWS_account_ID</i>:alias/<i>alias_name</i></code>	None (when controlling access to the alias)
	Key policy (for the CMKs)	CMK <code>arn:aws:kms:<i>AWS_region</i>:<i>AWS_account_ID</i>:key/<i>CMK_key_ID</i></code>	<code>kms:CallerAccount</code> <code>kms:AWSServiceID:key/</code>
UpdateKeyDescription <code>kms:UpdateKeyDescription</code>	Key policy	CMK <code>arn:aws:kms:<i>AWS_region</i>:<i>AWS_account_ID</i>:key/<i>CMK_key_ID</i></code>	<code>kms:CallerAccount</code> <code>kms:AWSServiceID:key/</code>

Using Policy Conditions with AWS KMS

You can specify conditions in the [key policies](#) (p. 33) and [IAM policies](#) (p. 50) that control access to AWS KMS resources. The policy statement is effective only when the conditions are true. 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 only when a specific value appears in an API request.

To specify conditions, you use predefined *condition keys* in the `Condition` element of a policy statement with [IAM condition policy operators](#). Some condition keys apply generally to AWS; others are specific to AWS KMS.

Topics

- [AWS Global Condition Keys \(p. 60\)](#)
- [AWS KMS Condition Keys \(p. 61\)](#)

AWS Global Condition Keys

AWS provides [global condition keys](#), a set of predefined condition keys for all AWS services that use IAM for access control. For example, you can use the `aws:PrincipalType` condition key to allow access only when the principal in the request is the type you specify.

AWS KMS supports all global condition keys, including the `aws:TagKeys` and `aws:RequestTag` condition keys that control access based on the resource tag in the request. This condition key is supported by some, but not all, AWS services.

Topics

- [Using the IP Address Condition in Policies with AWS KMS Permissions \(p. 60\)](#)
- [Using VPC Endpoint Conditions in Policies with AWS KMS Permissions \(p. 61\)](#)

Using the IP Address Condition in Policies with AWS KMS Permissions

You can use AWS KMS to protect your data in an [integrated AWS service \(p. 127\)](#). But 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, the policy in [AWS: Denies Access to AWS Based on the Source IP](#) restricts AWS actions to requests from the specified IP range.

Consider this scenario:

1. You attach a policy like the one shown at [AWS: Denies Access to AWS Based on the Source IP](#) to an IAM user. You set the value of the `aws:SourceIp` condition key to the range of IP addresses for the user's company. 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 even though the user has permission to use all the relevant services.

Step 2 fails because the request to AWS KMS to decrypt the volume's encrypted data key comes from an IP address that is associated with the Amazon EC2 infrastructure. To succeed, the request must come from the IP address of the originating user. Because the policy in step 1 explicitly denies all requests from IP addresses other than those specified, Amazon EC2 is denied permission to decrypt the EBS volume's encrypted data key.

Also, the `aws:sourceIP` condition key is not effective when the request comes from an [Amazon VPC endpoint](#). To restrict requests to a VPC endpoint, including an [AWS KMS VPC endpoint \(p. 200\)](#), use the `aws:sourceVpce` or `aws:sourceVpc` condition keys. For more information, see [VPC Endpoints - Controlling the Use of Endpoints](#) in the *Amazon VPC User Guide*.

Using VPC Endpoint Conditions in Policies with AWS KMS Permissions

AWS KMS supports Amazon Virtual Private Cloud (Amazon VPC) endpoints (p. 200) that are powered by AWS PrivateLink. You can use the following [global condition keys](#) in IAM policies to allow or deny access to a particular VPC or VPC endpoint.

You can also use these condition keys in [AWS KMS key policies](#) (p. 204) to restrict access to AWS KMS CMKs to requests from the VPC or VPC endpoint.

- `aws:SourceVpc` limits access to requests from the specified VPC.
- `aws:SourceVpce` limits access to requests from the specified VPC endpoint.

If you use these condition keys in a key policy statement that allows or denies access to AWS KMS CMKs, you might inadvertently deny access to services that use AWS KMS on your behalf.

Take care to avoid a situation like the [IP address condition keys](#) (p. 60) example. If you restrict requests for a CMK to a VPC or VPC endpoint, calls to AWS KMS from an integrated service, such as Amazon S3 or Amazon EBS, might fail. This can happen even if the source request ultimately originates in the VPC or from the VPC endpoint.

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. 248) when controlling access to a KMS customer master key (CMK).

The following topics describe each AWS KMS condition key and include example policy statements that demonstrate policy syntax.

Topics

- [kms:BypassPolicyLockoutSafetyCheck](#) (p. 62)
- [kms:CallerAccount](#) (p. 63)
- [kms:EncryptionContext](#) (p. 64)
- [kms:EncryptionContextKeys](#) (p. 66)
- [kms:ExpirationModel](#) (p. 68)
- [kms:GrantConstraintType](#) (p. 69)
- [kms:GrantIsForAWSResource](#) (p. 70)
- [kms:GrantOperations](#) (p. 71)
- [kms:GranteePrincipal](#) (p. 71)
- [kms:KeyOrigin](#) (p. 72)
- [kms:ReEncryptOnSameKey](#) (p. 73)
- [kms:RetiringPrincipal](#) (p. 73)
- [kms:ValidTo](#) (p. 74)
- [kms:ViaService](#) (p. 75)
- [kms:WrappingAlgorithm](#) (p. 77)
- [kms:WrappingKeySpec](#) (p. 77)

kms:BypassPolicyLockoutSafetyCheck

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
kms:BypassPolicyLockoutSafetyCheck	Boolean	CreateKey PutKeyPolicy	CreateKey: IAM policies only PutKeyPolicy: IAM and key policies

The `kms:BypassPolicyLockoutSafetyCheck` condition key controls access to the [CreateKey](#) and [PutKeyPolicy](#) operations based on the value of the `BypassPolicyLockoutSafetyCheck` parameter in the request.

The following example IAM policy statement prevents users from bypassing the policy lockout safety check by denying them permission to create CMKs when the value of the `BypassPolicyLockoutSafetyCheck` parameter in the `CreateKey` request is `true`.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": "kms:CreateKey",
    "Resource": "*",
    "Condition": {
      "Bool": {
        "kms:BypassPolicyLockoutSafetyCheck": true
      }
    }
  }
}
```

You can also use the `kms:BypassPolicyLockoutSafetyCheck` condition key in an IAM policy or key policy to control access to the `PutKeyPolicy` operation. The following example policy statement from a key policy prevents users from bypassing the policy lockout safety check when changing the policy of a CMK.

Instead of using an explicit `Deny`, this policy statement uses `Allow` with the [Null condition operator](#) to allow access only when the request does not include the `BypassPolicyLockoutSafetyCheck` parameter. When the parameter is not used, the default value is `false`. This slightly weaker policy statement can be overridden in the rare case that a bypass is necessary.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "kms:PutKeyPolicy",
    "Resource": "*",
    "Condition": {
      "Null": {
        "kms:BypassPolicyLockoutSafetyCheck": true
      }
    }
  }
}
```

See Also

- [kms:KeyOrigin](#) (p. 72)

kms:CallerAccount

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
kms:CallerAccount	String	The kms:CallerAccount condition key exists for all AWS KMS operations <i>except</i> for these: CreateKey, GenerateRandom, ListAliases, ListKeys, ListRetirableGrants, RetireGrant.	Key policies only

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. 75).

```
{
  "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:

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
kms:EncryptionContext	String	CreateGrant Encrypt Decrypt GenerateDataKey GenerateDataKeyWithoutPlaintext ReEncrypt	IAM and key policies

You can use the `kms:EncryptionContext:` condition key prefix to control access to a CMK based on the encryption context in a request for a cryptographic operation. Use this condition key prefix to evaluate both the key and the value in the encryption context pair. To evaluate only the encryption context keys, use the [kms:EncryptionContextKeys](#) (p. 66) condition key.

An [encryption context](#) (p. 248) is a set of nonsecret key–value pairs that you can include in a request for any AWS KMS cryptographic operation ([Encrypt](#), [Decrypt](#), [GenerateDataKey](#), [GenerateDataKeyWithoutPlaintext](#), and [ReEncrypt](#)) and the [CreateGrant](#) operation. When you specify an encryption context in an encryption operation, you must specify the same encryption context in the decryption operation. Otherwise, the decryption request fails.

To use the `kms:EncryptionContext:` condition key prefix, replace the `encryption_context_key` placeholder with the encryption context key. Replace the `encryption_context_value` placeholder with the encryption context value.

`"kms:EncryptionContext:encryption_context_key": "encryption_context_value"`

For example, the following condition key specifies an encryption context in which the key is `AppName` and the value is `ExampleApp`.

```
"kms:EncryptionContext:AppName": "ExampleApp"
```

The following example key policy statement uses this condition key. This policy allows the principal to use the CMK in a `GenerateDataKey` request only when the encryption context in the request is `"AppName": "ExampleApp"`.

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

To require more than one encryption context pair, you can include multiple instances of the `kms:EncryptionContext: condition`. For example, the following example policy statement requires both of the following encryption context pairs. The order in which the pairs are specified does not matter.

- `"AppName": "ExampleApp"`
- `"FilePath": "/var/opt/secrets/"`

```
{
  "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/"
    }
  }
}
```

The encryption context that is specified in a decryption operation must be an exact, case-sensitive match for the encryption context that is specified in the encryption operation. Only the order of pairs in an encryption context with multiple pair can vary.

However, in policy conditions, the condition key is not case sensitive. The case sensitivity of the condition value is determined by the [policy condition operator](#) that you use, such as `StringEquals` or `StringEqualsIgnoreCase`.

As such, the condition key, which consists of the `kms:EncryptionContext:` prefix and the `encryption_context_key` replacement, is not case sensitive. A policy that uses this condition does not check the case of either element of the condition key. The case sensitivity of the value, that is, the `encryption_context_value` replacement, is determined by the policy condition operator.

For example, the following policy statement allows the operation when the encryption context includes an Appname key, regardless of its capitalization. The `StringEquals` condition requires that `ExampleApp` be capitalized as it is specified.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:Decrypt",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContext:Appname": "ExampleApp"
    }
  }
}
```

To require a case-sensitive encryption context key, use the [kms:EncryptionContextKeys](#) (p. 66) policy condition with a case-sensitive condition operator, such as `StringEquals`. In this policy condition, because the encryption context key is the policy condition value, its case sensitivity is determined by the condition operator.


```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/RoleForExampleApp"
  },
  "Action": "kms:GenerateDataKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:EncryptionContextKey": "AppName"
    }
  }
}
```

To require a case-sensitive evaluation of both the encryption context key and value, use the `kms:EncryptionContextKey` and `kms:EncryptionContext` policy conditions together in the same policy statement. For example, in the following example policy statement, because the `StringEquals` operator is case sensitive, both the encryption context key and the encryption context value are case sensitive.

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

See Also

- [the section called “kms:EncryptionContextKeys” \(p. 66\)](#)
- [the section called “kms:GrantConstraintType” \(p. 69\)](#)

kms:EncryptionContextKeys

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
<code>kms:EncryptionContextKeys</code>	Strings (list)	CreateGrant Encrypt Decrypt GenerateDataKey GenerateDataKeyWithoutPlaintext ReEncrypt	IAM and key policies

You can use the `kms:EncryptionContextKeys` condition key to control access to a CMK based on the encryption context in a request for a cryptographic operation. Use this condition key prefix to evaluate only the key in each encryption context pair. To evaluate both the key and the value, use the [kms:EncryptionContext: \(p. 64\)](#) condition key prefix.

You can use this condition key to control access based on the [encryption context \(p. 248\)](#) in the AWS KMS API request. Encryption context is a set of key-value pairs that you can include in AWS KMS cryptographic operations ([Encrypt](#), [Decrypt](#), [GenerateDataKey](#), [GenerateDataKeyWithoutPlaintext](#), and [ReEncrypt](#)) and the [CreateGrant](#) operation.

The following example policy statement uses the `kms:EncryptionContextKeys` condition key to allow use of a CMK for the specified operations only when the encryption context in the request includes the `AppName` key, regardless of its value.

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

Because the [StringEquals](#) condition operation is case sensitive, the previous policy statement requires the spelling and case of the encryption context key. But you can use a condition operator that ignores the case of the key, such as `StringEqualsIgnoreCase`.

You can specify multiple encryption context keys in each condition. For example, the following policy statement allows the specified operations only when the encryption context in the request includes both the `AppName` and `FilePath` keys, regardless of their values. The order of keys in the encryption context does not matter.

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

You can also use the `kms:EncryptionContextKeys` condition key to require an encryption context in cryptographic operations that use the CMK.

The following example key policy statement uses the `kms:EncryptionContextKeys` condition key with the [Null condition operator](#) to allow access to CMK 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 the encryption context exists.

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

See Also

- [kms:EncryptionContext](#): (p. 64)
- [kms:GrantConstraintType](#) (p. 69)

kms:ExpirationModel

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
<code>kms:ExpirationModel</code>	String	<code>ImportKeyMaterial</code>	IAM and key policies

The `kms:ExpirationModel` condition key controls access to the [ImportKeyMaterial](#) operation based on the value of the [ExpirationModel](#) parameter in the request.

`ExpirationModel` is an optional parameter that determines whether the imported key material expires. Valid values are `KEY_MATERIAL_EXPIRES` and `KEY_MATERIAL_DOES_NOT_EXPIRE`. `KEY_MATERIAL_EXPIRES` is the default value.

The expiration date and time is determined by the value of the [ValidTo](#) parameter. The `ValidTo` parameter is required unless the value of the `ExpirationModel` parameter is `KEY_MATERIAL_DOES_NOT_EXPIRE`. You can also use the [kms:ValidTo](#) (p. 74) condition key to require a particular expiration date as a condition for access.

The following example policy statement uses the `kms:ExpirationModel` condition key to allow a user to import key material into a CMK only when the request includes the `ExpirationModel` parameter and its value is `KEY_MATERIAL_DOES_NOT_EXPIRE`.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": "kms:ImportKeyMaterial",
  "Resource": "*"
}
```

```
"Condition": {
  "StringEquals": {
    "kms:ExpirationModel": "KEY_MATERIAL_DOES_NOT_EXPIRE"
  }
}
```

You can also use the `kms:ExpirationModel` condition key to allow a user to import key material only when the key material expires, without [specifying an expiration date \(p. 74\)](#) in the condition. The following example policy statement uses the `kms:ExpirationModel` condition key with the [Null condition operator](#) to allow a user to import key material only when the request does not have an `ExpirationModel` parameter.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": "kms:ImportKeyMaterial",
  "Resource": "*",
  "Condition": {
    "Null": {
      "kms:ExpirationModel": true
    }
  }
}
```

See Also

- [kms:ValidTo \(p. 74\)](#)
- [kms:WrappingAlgorithm \(p. 77\)](#)
- [kms:WrappingKeySpec \(p. 77\)](#)

kms:GrantConstraintType

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
<code>kms:GrantConstraintType</code>	String	<code>CreateGrant</code>	IAM and key policies

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 that the grant permit 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.

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 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"
    }
  }
}

```

See Also

- [kms:EncryptionContext](#) (p. 64)
- [kms:EncryptionContextKeys](#) (p. 66)
- [kms:GrantIsForAWSResource](#) (p. 70)
- [kms:GrantOperations](#) (p. 71)
- [kms:GranteePrincipal](#) (p. 71)
- [kms:RetiringPrincipal](#) (p. 73)

kms:GrantIsForAWSResource

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
kms:GrantIsForAWSResource	Boolean	CreateGrant	IAM and key policies

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 \(p. 127\)](#). 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 \(p. 139\)](#).
- Amazon Relational Database Service (Amazon RDS) – For more information, see [How Amazon Relational Database Service \(Amazon RDS\) Uses AWS KMS \(p. 150\)](#).
- Amazon Redshift – For more information, see [How Amazon Redshift Uses AWS KMS \(p. 149\)](#).
- 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 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
    }
  }
}

```

See Also

- [kms:GrantConstraintType](#) (p. 69)
- [kms:GrantOperations](#) (p. 71)
- [kms:GranteePrincipal](#) (p. 71)
- [kms:RetiringPrincipal](#) (p. 73)

kms:GrantOperations

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
kms:GrantOperations	String	CreateGrant	IAM and key policies

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 reencrypt when this CMK is the destination CMK. The 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"
      ]
    }
  }
}
```

See Also

- [kms:GrantConstraintType](#) (p. 69)
- [kms:GrantIsForAWSResource](#) (p. 70)
- [kms:GranteePrincipal](#) (p. 71)
- [kms:RetiringPrincipal](#) (p. 73)

kms:GranteePrincipal

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
kms:GranteePrincipal	String	CreateGrant	IAM and key policies

You can use this condition key to control access to the [CreateGrant](#) operation based on the value of the [GranteePrincipal](#) parameter in the request. For example, you can allow a user to create grants to use a CMK only when the grantee principal in the `CreateGrant` request matches the principal specified in the condition statement.

The following example policy statement uses the `kms:GranteePrincipal` condition key to allow a user to create grants for a CMK only when the grantee principal in the grant is the `LimitedAdminRole`.

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

See Also

- [kms:GrantConstraintType](#) (p. 69)
- [kms:GrantsForAWSResource](#) (p. 70)
- [kms:GrantOperations](#) (p. 71)
- [kms:RetiringPrincipal](#) (p. 73)

kms:KeyOrigin

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
<code>kms:KeyOrigin</code>	String	<code>CreateKey</code>	IAM policies

You can use this condition key to control access to the [CreateKey](#) operation based on the value of the [Origin](#) parameter in the request. For example, you can allow a user to create a CMK only when KMS generates the key material, or only when the [key material is imported](#) (p. 93) from an external source. Valid values for `Origin` are `AWS_KMS` and `EXTERNAL`.

The following example policy statement uses the `kms:KeyOrigin` condition key to allow a user to create a CMK only when the key origin is `EXTERNAL`, that is, the key material is imported.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": "kms:CreateKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:KeyOrigin": "EXTERNAL"
    }
  }
}
```

See Also

- [kms:BypassPolicyLockoutSafetyCheck](#) (p. 62)

kms:ReEncryptOnSameKey

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
kms:ReEncryptOnSameKey	Boolean	ReEncrypt	IAM and key policies

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 reencrypt only when the destination CMK is the same one used for the original encryption. The example shows a policy statement in a key policy.

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

kms:RetiringPrincipal

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
kms:RetiringPrincipal	String (list)	CreateGrant	IAM and key policies

You can use this condition key to control access to the [CreateGrant](#) operation based on the value of the [RetiringPrincipal](#) parameter in the request. For example, you can allow a user to create grants to use a CMK only when the `RetiringPrincipal` in the `CreateGrant` request matches the `RetiringPrincipal` in the condition statement.

The following example policy statement allows a user to create grants for the CMK. The `kms:RetiringPrincipal` condition key restricts the permission to `CreateGrant` requests where the retiring principal in the grant is either the `LimitedAdminRole` or the `OpsAdmin` user.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": "kms:CreateGrant",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:RetiringPrincipal": "arn:aws:kms:us-east-1:111122223333:key/ExampleKey"
    }
  }
}
```



```

"Condition": {
  "ForAnyValue:StringEquals": {
    "kms:RetiringPrincipal": [
      "arn:aws:iam::111122223333:role/LimitedAdminRole",
      "arn:aws:iam::111122223333:user/OpsAdmin"
    ]
  }
}

```

See Also

- [kms:GrantConstraintType](#) (p. 69)
- [kms:GrantsForAWSResource](#) (p. 70)
- [kms:GrantOperations](#) (p. 71)
- [kms:GranteePrincipal](#) (p. 71)

kms:ValidTo

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
kms:ValidTo	Timestamp	ImportKeyMaterial	IAM and key policies

The `kms:ValidTo` condition key controls access to the [ImportKeyMaterial](#) operation based on the value of the [ValidTo](#) parameter in the request, which determines when the imported key material expires. The value is expressed in [Unix time](#).

By default, the `ValidTo` parameter is required in an `ImportKeyMaterial` request. However, if the value of the [ExpirationModel](#) parameter is `KEY_MATERIAL_DOES_NOT_EXPIRE`, the `ValidTo` parameter is invalid. You can also use the [kms:ExpirationModel](#) (p. 68) condition key to require the `ExpirationModel` parameter or a specific parameter value.

The following example policy statement allows a user to import key material into a CMK. The `kms:ValidTo` condition key limits the permission to `ImportKeyMaterial` requests where the `ValidTo` value is less than or equal to 1546257599.0 (December 31, 2018 11:59:59 PM).

```

{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": "kms:ImportKeyMaterial",
  "Resource": "*",
  "Condition": {
    "NumericLessThanEquals": {
      "kms:ValidTo": "1546257599.0"
    }
  }
}

```

See Also

- [kms:ExpirationModel](#) (p. 68)
- [kms:WrappingAlgorithm](#) (p. 77)
- [kms:WrappingKeySpec](#) (p. 77)

kms:ViaService

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
kms:ViaService	String	The kms:ViaService condition key is valid for all AWS KMS operations <i>except</i> : CreateKey, GenerateRandom, ListAliases, ListKeys, ListRetirableGrants, RetireGrant.	IAM and key policies

The kms:ViaService condition key limits use of a [customer-managed CMK \(p. 2\)](#) to requests from particular AWS services. (AWS managed CMKs in your account, such as aws/s3, are always restricted to the AWS service that created them.)

For example, you can use kms:ViaService to allow a user to use a customer managed CMK only for requests that Amazon S3 makes on their behalf. Or you can use it to deny the user permission to a CMK when a request on their behalf comes from AWS Lambda.

The kms:ViaService condition key is valid in IAM and key policy statements. The services that you specify must be integrated with AWS KMS, support customer managed CMKs, and support the kms:ViaService condition key. You can specify one or more services in each kms:ViaService condition key.

Important

When you use the kms:ViaService condition key, verify that the principals have the following permissions:

- Permission to use the CMK. The principal needs to grant these permissions to the integrated service so the service can use the customer managed CMK on behalf of the principal. For more information, see [How AWS Services use AWS KMS \(p. 127\)](#).
- Permission to use the integrated service. For details about giving users access to an AWS service that integrates with AWS KMS, consult the documentation for the integrated service.

The following table shows the KMS ViaService name for each service that supports customer managed CMKs and the kms:ViaService condition key. The services in this table might not be available in all regions.

Services that support the kms:ViaService condition key

Service Name	KMS ViaService Name
Amazon Connect	connect.AWS_region.amazonaws.com
AWS Database Migration Service (AWS DMS)	dms.AWS_region.amazonaws.com
Amazon EC2 Systems Manager	ssm.AWS_region.amazonaws.com
Amazon Elastic Block Store (Amazon EBS)	ec2.AWS_region.amazonaws.com (EBS only)
Amazon Elastic File System	elasticfilesystem.AWS_region.amazonaws.com

Service Name	KMS ViaService Name
Amazon Elasticsearch Service	es. <i>AWS_region</i> .amazonaws.com
Amazon Kinesis	kinesis. <i>AWS_region</i> .amazonaws.com
Amazon Kinesis Video Streams	kinesisvideo. <i>AWS_region</i> .amazonaws.com
AWS Lambda	lambda. <i>AWS_region</i> .amazonaws.com
Amazon Lex	lex. <i>AWS_region</i> .amazonaws.com
Amazon Redshift	redshift. <i>AWS_region</i> .amazonaws.com
Amazon Relational Database Service (Amazon RDS)	rds. <i>AWS_region</i> .amazonaws.com
AWS Secrets Manager (Secrets Manager)	secretsmanager. <i>AWS_region</i> .amazonaws.com
Amazon Simple Email Service (Amazon SES)	ses. <i>AWS_region</i> .amazonaws.com
Amazon Simple Storage Service (Amazon S3)	s3. <i>AWS_region</i> .amazonaws.com
AWS Snowball	importexport. <i>AWS_region</i> .amazonaws.com
Amazon SQS	sqs. <i>AWS_region</i> .amazonaws.com
Amazon WorkMail	workmail. <i>AWS_region</i> .amazonaws.com
Amazon WorkSpaces	workspaces. <i>AWS_region</i> .amazonaws.com

The following example shows a policy statement from a key policy for an AWS managed CMK. The policy statement uses the `kms:ViaService` condition key to allow the CMK to be used for the specified actions only when the request comes from Amazon EBS or Amazon RDS in the US West (Oregon) region.

```
{
  "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"
      ]
    }
  }
}
```

kms:WrappingAlgorithm

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
kms:WrappingAlgorithm	String	GetParametersForImport	IAM and key policies

This condition key controls access to the [GetParametersForImport](#) operation based on the value of the [WrappingAlgorithm](#) parameter in the request. You can use this condition to require principals to use a particular algorithm to encrypt key material during the import process. Requests for the required public key and import token fail when they specify a different wrapping algorithm.

The following example policy statement uses the `kms:WrappingAlgorithm` condition key to fail if the `WrappingAlgorithm` in the request is `RSAES_OAEP_SHA_1`. The operation succeeds, and returns a public key and import token, for any other `WrappingAlgorithm` value.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": "kms:GetParametersForImport",
  "Resource": "*",
  "Condition": {
    "StringNotEquals": {
      "kms:WrappingAlgorithm": "RSAES_OAEP_SHA_1"
    }
  }
}
```

See Also

- [kms:ExpirationModel](#) (p. 68)
- [kms:ValidTo](#) (p. 74)
- [kms:WrappingKeySpec](#) (p. 77)

kms:WrappingKeySpec

AWS KMS Condition Keys	Condition Type	API Operations	Policy Type
kms:WrappingKeySpec	String	GetParametersForImport	IAM and key policies

This condition key controls access to the [GetParametersForImport](#) operation based on the value of the [WrappingKeySpec](#) parameter in the request. You can use this condition to require principals to use a particular type of public key during the import process. If the request specifies a different key type, it fails.

Because the only valid value for the `WrappingKeySpec` parameter value is `RSA_2048`, preventing users from using this value effectively prevents them from using the `GetParametersForImport` operation.

The following example policy statement uses the `kms:WrappingAlgorithm` condition key to require that the `WrappingKeySpec` in the request is `RSA_2048`.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": "kms:GetParametersForImport",
  "Resource": "*",
  "Condition": {
    "StringNotEquals": {
      "kms:WrappingKeySpec": "RSA_2048"
    }
  }
}
```

See Also

- [kms:ExpirationModel](#) (p. 68)
- [kms:ValidTo](#) (p. 74)
- [kms:WrappingAlgorithm](#) (p. 77)

Using Grants

AWS KMS supports two resource-based access control mechanisms: [key policies](#) (p. 33) and *grants*. With grants you can programmatically delegate the use of KMS customer master keys (CMKs) to other AWS principals. You can use them to allow access, but not deny it. Grants are typically used to provide temporary permissions or more granular permissions.

You can also use key policies to allow other principals to access a CMK, but key policies work best for relatively static permission assignments. Also, key policies use the standard permissions model for AWS policies in which users either have or do not have permission to perform an action with a resource. For example, 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.

For Java code examples that demonstrate how to work with grants, see [Working with Grants](#) (p. 230).

Create a Grant

To create a grant, call the [CreateGrant](#) API operation. Specify a CMK, the grantee principal that the grant allows to use the CMK, and a list of allowed operations. The `CreateGrant` operation returns a grant ID that you can use to identify the grant in subsequent operations. To customize the grant, use optional `Constraints` parameters to define [grant constraints](#).

Grants can be revoked (canceled) by any user who has the [kms:RevokeGrant](#) permission on the CMK. Grants can be retired by any of the following:

- The AWS account (root user) in which the grant was created
- The retiring principal in the grant, if any
- The grantee principal, if the grant includes [kms:RetireGrant](#) permission

Grant Constraints

[Grant constraints](#) set conditions on the permissions that the grantee principal can perform. Grants have two supported constraints, both of which involve the [encryption context](#) (p. 6) in a request for a cryptographic operation:

- `EncryptionContextEquals` specifies that the grant applies only when the encryption context pairs in the request are an exact, case-sensitive match for the encryption context pairs in the grant constraint. The pairs can appear in any order, but the keys and values in each pair cannot vary.
- `EncryptionContextSubset` specifies that the grant applies only when the encryption context in the request includes the encryption context specified in the grant constraint. The encryption context in the request must be an exact, case-sensitive match of the encryption context in the constraint, but it can include additional encryption context pairs. The pairs can appear in any order, but the keys and values in each included pair cannot vary.

For example, consider a grant that allows [GenerateDataKey](#) and [Decrypt](#) operations. It includes an `EncryptionContextSubset` constraint with the following values.

```
{"Department": "Finance", "Classification": "Public"}
```

In this example, any of the following encryption context values would satisfy the `EncryptionContextSubset` constraint.

- `{"Department": "Finance", "Classification": "Public"}`
- `{"Classification": "Public", "Department": "Finance"}`
- `{"Customer": "12345", "Department": "Finance", "Classification": "Public", "Purpose": "Test"}`

However, the following encryption context values would not satisfy the constraint, either because they are incomplete or do not include an exact, case-sensitive match of the specified pairs.

- `{"Department": "Finance"}`
- `{"department": "finance", "classification": "public"}`
- `{"Classification": "Public", "Customer": "12345"}`

Authorizing CreateGrant in a Key Policy

When you create a key policy to control access to the [CreateGrant](#) API operation, you can use one or more policy conditions to limit the permission. AWS KMS supports all of the following grant-related condition keys. For detailed information about these condition keys, see [AWS KMS Condition Keys](#) (p. 61).

- [kms:GrantConstraintType](#) (p. 69)
- [kms:GrantsForAWSResource](#) (p. 70)
- [kms:GrantOperations](#) (p. 71)
- [kms:GranteePrincipal](#) (p. 71)
- [kms:RetiringPrincipal](#) (p. 73)

Granting CreateGrant Permission

When a grant includes permission to call the `CreateGrant` operation, the grant only allows the grantee principal to create grants that are equally restrictive or more restrictive.

For example, consider a grant that allows the grantee principal to call the `GenerateDataKey`, `Decrypt`, and `CreateGrant` operations. The grantee principal can use this permission to create a grant that includes any subset of the operations specified in the parent grant, such as `GenerateDataKey` and `Decrypt`. But it cannot include other operations, such as `ScheduleKeyDeletion` or `ReEncrypt`.

Also, the [grant constraints](#) in child grants must be equally restrictive or more restrictive than those in the parent grant. For example, the child grant can add pairs to an `EncryptionContextSubset`

constraint in the parent grant, but it cannot remove them. The child grant can change an `EncryptionContextSubset` constraint to an `EncryptionContextEquals` constraint, but not the reverse.

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 \(p. 78\)](#) 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 \(p. 80\)](#)
- [Examining the Key Policy \(p. 81\)](#)
- [Examining IAM Policies \(p. 84\)](#)
- [Examining Grants \(p. 85\)](#)

Understanding Policy Evaluation

When authorizing access to a CMK, AWS KMS evaluates the following:

- The key policy that is attached to the CMK
- All grants that apply to the CMK
- All IAM policies that are 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 \(p. 34\)](#).
- 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 \(p. 34\)](#). Be sure to [examine IAM policies \(p. 84\)](#) 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 \(Console\) \(p. 81\)](#)
- [Examining the Key Policy Document \(KMS API\) \(p. 82\)](#)

Examining the Key Policy (Console)

Authorized users can view and change the policy document for a customer master key (CMK) in the **Key Policy** section of the AWS Management Console.

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To examine a key policy (new console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. In the list of CMK, choose the alias or key ID of the CMK that you want to examine.
5. Under **Key policy**, the **Key administrators** section displays the list of IAM users and roles that can manage the CMK. The **Key users** section lists the users, roles, and AWS accounts that can use this CMK in cryptographic operations.

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 \(p. 84\)](#) in this account to determine whether they allow access to this CMK.

6. (Optional) To view the key policy document, choose **Switch to policy view**.

To examine a key policy (original console)

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
3. In the list of CMKs, choose the alias of the CMK 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 \(p. 84\)](#) 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 (KMS API)

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).
- To get the key policy document, use the [GetKeyPolicy](#) operation in the AWS KMS API.

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 \(p. 34\)](#) 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. It is also present when you create a new CMK programmatically but do not provide a key policy.

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 \(p. 34\)](#). 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 \(p. 84\)](#) 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 \(p. 127\)](#), specifically the services that use [grants \(p. 78\)](#). 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 \(p. 33\)](#).

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 \(p. 80\)](#).

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 \(p. 84\)](#)
- [Examining IAM Policies with the IAM API \(p. 85\)](#)

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, choose **Simulation Settings** and then type the Amazon Resource Name (ARN) of the KMS CMK to simulate.
6. Choose **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 includes the name of the specific AWS KMS API operation that is allowed. It also includes the ARN of the CMK that was used in the evaluation, if any.

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. 78).

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 that is associated with the EC2 instance `i-5d476fab`. The EC2 infrastructure uses this role 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 that is protected by this CMK. You 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.

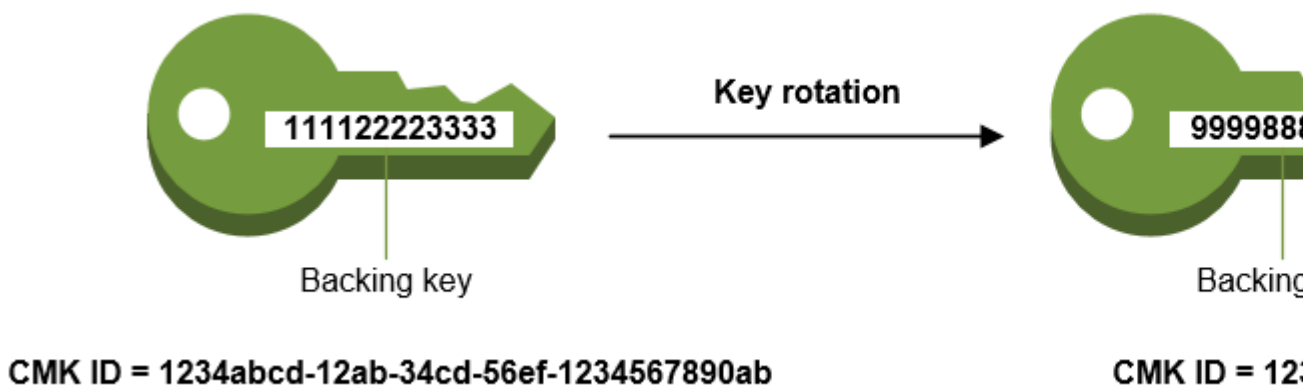
```
{
  "Grants": [
    {
      "Operations": ["Encrypt"],
      "KeyId": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "Name": "",
      "GranteePrincipal": "arn:aws:iam::444455556666:root",
      "GrantId": "f271e8328717f8bde5d03f4981f06a6b3fc18bcae2da12ac38bd9186e7925d11",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

Rotating Customer Master Keys

Cryptographic best practices discourage extensive reuse of encryption keys. To create new cryptographic material for your AWS Key Management Service (AWS KMS) customer master keys (CMKs), you can create new CMKs, and then change your applications or aliases to use the new CMKs. Or, you can enable automatic key rotation for an existing CMK.

When you enable *automatic key rotation* for a customer managed CMK, AWS KMS generates new cryptographic material for the CMK every year. AWS KMS also saves the CMK's older cryptographic material so it can be used to decrypt data that it encrypted.

Key rotation changes only the CMK's *backing key*, which is the cryptographic material that is used in encryption operations. The CMK is the same logical resource, regardless of whether or how many times its backing key changes. The properties of the CMK do not change, as shown in the following image.



Automatic key rotation has the following benefits:

- The properties of the CMK, including its key ID, key ARN, region, policies, and permissions, do not change when the key is rotated.
- You do not need to change applications or aliases that refer to the CMK ID or ARN.
- After you enable key rotation, AWS KMS rotates the CMK automatically every year. You don't need to remember or schedule the update.

However, automatic key rotation has no effect on the data that the CMK protects. It does not rotate the data keys that the CMK generated or re-encrypt any data protected by the CMK, and it will not mitigate the effect of a compromised data key.

You might decide to create a new CMK and use it in place of the original CMK. This has the same effect as rotating the key material in an existing CMK, so it's often thought of as [manually rotating the key \(p. 90\)](#). Manual rotation is a good choice when you want to control the key rotation schedule. It also provides a way to rotate CMKs with imported key material.

More Information About Key Rotation

Rotating customer managed CMKs might result in extra monthly charges. For details, see [AWS Key Management Service Pricing](#). For more detailed information about backing keys and rotation, see the [KMS Cryptographic Details](#) whitepaper.

Topics

- [How Automatic Key Rotation Works \(p. 88\)](#)
- [How to Enable and Disable Automatic Key Rotation \(p. 88\)](#)
- [Rotating Keys Manually \(p. 90\)](#)

How Automatic Key Rotation Works

Key rotation in AWS KMS is a cryptographic best practice that is designed to be transparent and easy to use.

- **Backing key management.** AWS KMS retains all backing keys for a CMK, even if key rotation is disabled. The backing keys are deleted only when the CMK is deleted. When you use a CMK to encrypt, AWS KMS uses the current backing key. When you use the CMK to decrypt, AWS KMS uses the backing key that was used to encrypt.
- **Enable and disable key rotation.** Automatic key rotation is disabled by default on customer managed CMKs. When you enable (or re-enable) key rotation, AWS KMS automatically rotates the CMK 365 days after the enable date and every 365 days thereafter.
- **Disabled CMKs.** While a CMK is disabled, AWS KMS does not rotate it. However, the underlying key rotation status does not change, and you cannot change it while the CMK is disabled. When the CMK is re-enabled, if the backing key is more than 365 days old, AWS KMS rotates it immediately and every 365 days thereafter. If the backing key is less than 365 days old, AWS KMS resumes the original key rotation schedule.
- **CMKs pending deletion.** While a CMK is pending deletion, AWS KMS does not rotate it. The key rotation status is set to `false` and you cannot change it while deletion is pending. If deletion is canceled, the previous key rotation status is restored. If the backing key is more than 365 days old, AWS KMS rotates it immediately and every 365 days thereafter. If the backing key is less than 365 days old, AWS KMS resumes the original key rotation schedule.
- **Customer managed CMKs.** Automatic key rotation is available for all customer managed CMKs with KMS-generated key material. It is not available for CMKs that have [imported key material \(p. 93\)](#) (the value of the **Origin** field is **External**), but you can [rotate these CMKs manually \(p. 90\)](#).
- **AWS managed CMKs.** You cannot manage key rotation for AWS managed CMKs. AWS KMS automatically rotates AWS managed keys every three years (1095 days).
- **Logging key rotation.** When AWS KMS rotates a CMK, it writes the **KMS CMK Rotation** event to [Amazon CloudWatch Events](#). You can use this event to verify that the CMK was rotated.

How to Enable and Disable Automatic Key Rotation

You can use the AWS KMS console or the AWS KMS API to enable and disable automatic key rotation, and view the rotation status of any customer managed CMK.

When you enable automatic key rotation, AWS KMS rotates the CMK 365 days after the enable date and every 365 days thereafter.

Topics

- [Enabling and Disabling Key Rotation \(Console\) \(p. 89\)](#)
- [Enabling and Disabling Key Rotation \(KMS API\) \(p. 90\)](#)

Enabling and Disabling Key Rotation (Console)

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

Enable and Disable Key Rotation (new console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**. (You cannot enable or disable rotation of AWS managed keys. They are automatically rotated every three years.)
4. Choose the alias or key ID of a CMK.
5. Under **General configuration**, choose the **Key rotation** tab.

If the CMK was created without key material (its **Origin** is **EXTERNAL**), there is no **Key rotation** tab. You cannot automatically rotate these CMK, but you can [rotate them manually \(p. 90\)](#).

6. Select or clear the **Automatically rotate this CMK every year** check box.

Note

If a CMK is disabled or pending deletion, the **Automatically rotate this CMK every year** check box is cleared, and you cannot change it. The key rotation status is restored when you enable the CMK or cancel deletion. For details, see [How Automatic Key Rotation Works \(p. 88\)](#) and [How Key State Affects Use of a Customer Master Key \(p. 123\)](#).

7. Choose **Save**.

Enable and Disable Key Rotation (original console)

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
3. Choose the alias of the CMK whose details you want to see.

Note

You cannot edit AWS managed CMKs, which are identified by the orange AWS icon.

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

Note

If a CMK is disabled or pending deletion, the **Key Rotation** check box is cleared, and you cannot change it. This reminds you that AWS KMS does not rotate CMKs while they are disabled or pending deletion. The key rotation status is restored when you re-enable the CMK or cancel deletion. For details, see [How Automatic Key Rotation Works \(p. 88\)](#) and [How Key State Affects Use of a Customer Master Key \(p. 123\)](#).

▼ Key Rotation

☒ Rotate this key every year. [Learn more.](#)

Save Changes

Enabling and Disabling Key Rotation (KMS API)

You can use the [AWS Key Management Service \(AWS KMS\) API](#) to enable and disable automatic key rotation, and view the current rotation status of any customer managed CMK. These examples use the [AWS Command Line Interface \(AWS CLI\)](#), but you can use any supported programming language.

The [EnableKeyRotation](#) operation enables automatic key rotation for the specified CMK. The [DisableKeyRotation](#) operation disables it. To identify the CMK, use its key ID, key ARN, alias name, or alias ARN. By default, key rotation is disabled for customer managed CMKs.

The following example enables key rotation on the specified CMK and uses the [GetKeyRotationStatus](#) operation to see the result. Then, it disables key rotation and, again, uses **GetKeyRotationStatus** to see the change.

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

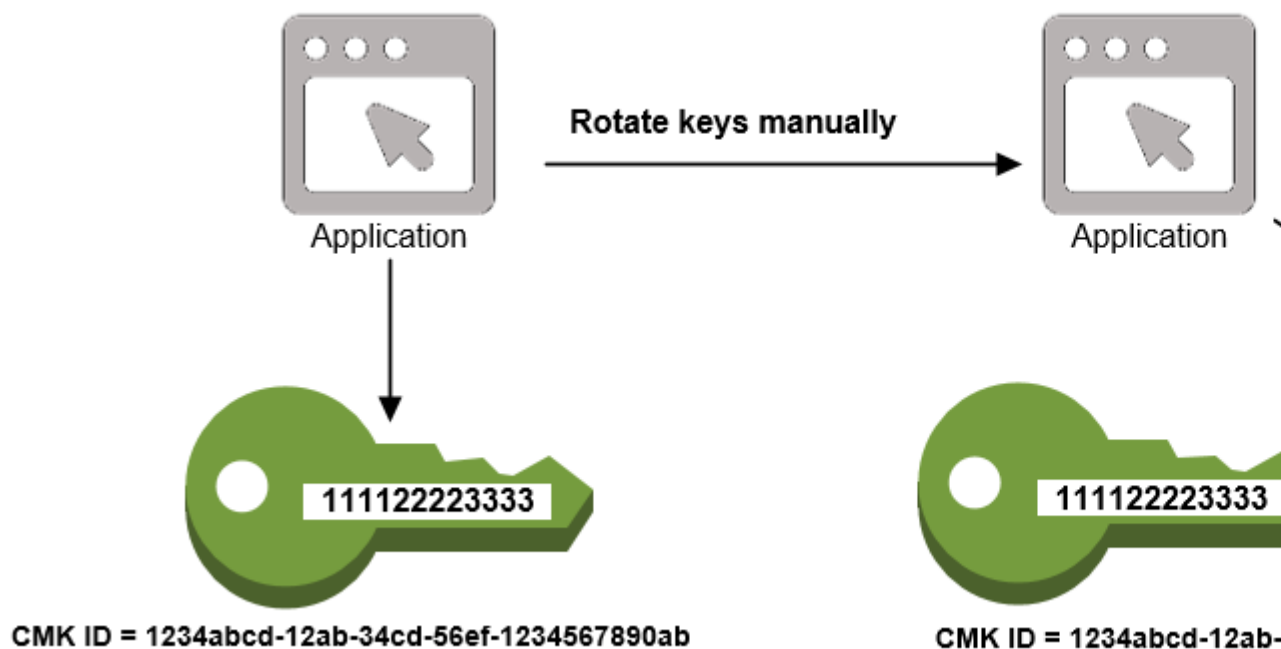
$ aws kms get-key-rotation-status --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
  "KeyRotationEnabled": true
}

$ aws kms disable-key-rotation --key-id 1234abcd-12ab-34cd-56ef-1234567890ab

$ aws kms get-key-rotation-status --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
  "KeyRotationEnabled": false
}
```

Rotating Keys Manually

You might want to create a new CMK and use it in place of a current CMK instead of enabling automatic key rotation. When the new CMK has different cryptographic material than the current CMK, using the new CMK has the same effect as changing the backing key in an existing CMK. The process of replacing one CMK with another is known as *manual key rotation*.



You might prefer to rotate keys manually so you can control the rotation frequency. It's also a good solution for CMKs that are not eligible for automatic key rotation, such as CMKs with [imported key material](#) (p. 93).

Note

When you begin using the new CMK, be sure to keep the original CMK enabled so that AWS KMS can decrypt data that the original CMK encrypted. When decrypting data, KMS identifies the CMK that was used to encrypt the data, and it uses the same CMK to decrypt the data. As long as you keep both the original and new CMKs enabled, AWS KMS can decrypt any data that was encrypted by either CMK.

Because the new CMK is a different resource from the current CMK, it has a different key ID and ARN. When you change CMKs, you need to update references to the CMK ID or ARN in your applications. Aliases, which associate a friendly name with a CMK, make this process easier. Use an alias to refer to a CMK in your applications. Then, when you want to change the CMK that the application uses, change the target CMK of the alias.

To update the target CMK of an alias, use [UpdateAlias](#) operation in the AWS KMS API. For example, this command updates the TestCMK alias to point to a new CMK. Because the operation does not return any output, the example uses the [ListAliases](#) operation to show that the alias is now associated with a different CMK.

```
$ aws kms list-aliases
{
  "Aliases": [
    {
      "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/TestCMK",
      "AliasName": "alias/TestCMK",
      "TargetKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    }
  ]
}

$ aws kms update-alias --alias-name alias/TestCMK --target-key-id 0987dcb-a09fe-87dc-65ba-ab0987654321
```

```
$ aws kms list-aliases
{
  "Aliases": [
    {
      "AliasArn": "arn:aws:kms:us-west-2:111122223333:alias/TestCMK",
      "AliasName": "alias/TestCMK",
      "TargetKeyId": "0987dcba-09fe-87dc-65ba-ab0987654321"
    },
  ]
}
```

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 \(p. 123\)](#), a CMK contains the *key material* used to encrypt and decrypt data. When you [create a CMK \(p. 8\)](#), by default AWS KMS generates the key material for that CMK. But you can 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 entropy 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 set an expiration time for the key material in AWS and to [manually delete it \(p. 107\)](#), but to also make it available again in the future. In contrast, [scheduling key deletion \(p. 110\)](#) 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 [About Imported Key Material \(p. 93\)](#).

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

Topics

- [About Imported Key Material \(p. 93\)](#)
- [How To Import Key Material \(p. 94\)](#)
- [How to Reimport Key Material \(p. 94\)](#)
- [How to Identify CMKs with Imported Key Material \(p. 95\)](#)

About Imported Key Material

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

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 can [reimport the same key material \(p. 94\)](#), but you cannot import different key material into

that CMK. Also, you cannot [enable automatic key rotation \(p. 87\)](#) for a CMK with imported key material. However, you can [manually rotate a CMK \(p. 90\)](#) with imported key material.

One CMK per ciphertext

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

Availability and durability

You are 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 \(p. 107\)](#), AWS KMS deletes the key material but does not delete the CMK or its metadata. In contrast, [scheduling key deletion \(p. 110\)](#) 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 restore the key material after events like these, you must retain a copy of the key material in a system that you control. Then, you can reimport it into the CMK.

How To Import Key Material

The following overview explains how 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 \(p. 96\)](#) – 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 \(p. 99\)](#) – 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 \(p. 104\)](#) – 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 \(p. 105\)](#) – Upload the encrypted key material that you created in step 3 and the import token that you downloaded in step 2.

How to Reimport Key Material

If you manage a CMK with imported key material, you might need to reimport the key material, either because the key material expired, or because the key material was accidentally deleted or lost.

You must reimport the same key material that was originally imported into the CMK. You cannot import different key material into a CMK. Also, AWS KMS cannot create key material for a CMK that is created without key material.

To reimport key material, use the same procedure that you used to [import the key material \(p. 94\)](#) the first time, with the following exceptions.

- Use an existing CMK, instead of creating a new CMK. You can skip [Step 1 \(p. 96\)](#) of the import procedure.
- If the CMK contains key material, you must [delete the existing key material \(p. 107\)](#) before you reimport the key material.

Each time you import key material to a CMK, you need to [download and use a new wrapping key and import token \(p. 99\)](#) for the CMK. The wrapping procedure does not affect the content of the key material, so you can use different wrapping keys (and different import tokens) to import the same key material.

How to Identify CMKs with Imported Key Material

When you create a CMK with no key material, the value of the `Origin` property of the CMK is `EXTERNAL`, and it cannot be changed. You cannot convert a key that is designed to use imported key material to one that uses the key material that AWS KMS provides.

You can identify CMKs that require imported key material in the AWS KMS console or by using the AWS KMS API.

To identify the value of the `Origin` property of CMKs (Console)

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To identify the value of the `Origin` property of CMKs (new console)

1. Open the AWS KMS console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. Use either of the following techniques to view the `Origin` property of your CMKs.
 - To add an **Origin** column to your CMK table, in the upper right corner, choose the **Settings** icon. Choose **Origin** and choose **Confirm**. The **Origin** column makes it easy to identify CMKs with an `EXTERNAL` origin property value.
 - To find the value of the `Origin` property of a particular CMK, choose the key ID or alias of the CMK. The `Origin` property value appears in the **General configuration** section.

To identify the value of the `Origin` property of CMKs (original console)

1. Open the **Encryption Keys** section of the AWS Identity and Access Management (IAM) console at <https://console.aws.amazon.com/iam/home#encryptionKeys>.

2. For **Region**, choose the appropriate AWS region. Do not use the region selector in the navigation bar (top right corner).
3. Use either of the following techniques to view the `Origin` property of your CMKs.
 - To add an **Origin** column to your CMK table, in the upper right corner, choose the **Settings** icon. Choose **Origin** and choose **Close**. The **Origin** column makes it easy to identify CMKs with an `EXTERNAL` origin property value.
 - To find the value of the `Origin` property of a particular CMK, choose the alias for the CMK. If the origin is external, the `Origin` property value appears in the **Key Material** section.

To identify the value of the `Origin` property of CMKs (KMS API)

Use the [DescribeKey](#) operation. The response includes the `Origin` property of the CMK, as shown in the following example.

```
$ aws kms describe-key --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
  "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
  "Origin": "EXTERNAL",
  "KeyManager": "CUSTOMER",
  "ValidTo": 1549224000.0,
  "Enabled": true,
  "AWSAccountId": "111122223333",
  "Arn": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
  "CreationDate": 1517867689.949,
  "KeyUsage": "ENCRYPT_DECRYPT",
  "Description": "example-key",
  "KeyState": "Enabled",
  "ExpirationModel": "KEY_MATERIAL_EXPIRES"
}
```

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. 123).

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

- [Creating a CMK with No Key Material \(Console\)](#) (p. 97)
- [Creating a CMK with No Key Material \(KMS API\)](#) (p. 99)

Creating a CMK with No Key Material (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 the **Origin** column in the list of CMKs. Imported keys have an **Origin** value of **External**.

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 \(p. 99\)](#).

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

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

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose **Create key**.
5. Type an alias and (optionally) a description for the CMK.

Choose **Next**.

6. Choose **Advanced options**.
7. 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 **security, availability, and durability implications**.

Choose **Next**.

8. (Optional). On the **Add tags** page, add tags that identify or categorize your CMK.

Choose **Next**.

9. In the **Key administrators** section, select the IAM users and roles who can manage the CMK. For more information, see [Allows Key Administrators to Administer the CMK \(p. 35\)](#).

Note

IAM policies can give other IAM users and roles permission to manage the CMK.

10. (Optional) To prevent the selected IAM users and roles from deleting this CMK, in the **Key deletion** section at the bottom of the page, clear the **Allow key administrators to delete this key** check box.

Choose **Next**.

11. In the **This account** section, select the IAM users and roles in this AWS account who can use the CMK in cryptographic operations. For more information, see [Allows Key Users to Use the CMK \(p. 37\)](#).

Note

IAM policies can give other IAM users and roles permission to use the CMK.

12. (Optional) You can allow other AWS accounts to use this CMK for cryptographic operations. To do so, in the **Other AWS accounts** section at the bottom of the page, choose **Add another AWS account** and enter the AWS account identification number of an external account. To add multiple external accounts, repeat this step.

Note

To allow principals in the external accounts to use the CMK, Administrators of the external account must create IAM policies that provide these permissions. For more information, see [Allowing External AWS Accounts to Access a CMK \(p. 47\)](#).

Choose **Next**.

13. On the **Review and edit key policy** page, review and edit the policy document for the new CMK. When you're done, choose **Finish**.

If the operation succeeds, you have created a CMK with no key material. Its status is **Pending import**. To continue the process now, see [Downloading the Public Key and Import Token \(Console\) \(p. 101\)](#). To continue the process later, choose **Cancel**.

Next: [Step 2: Download the Public Key and Import Token \(p. 99\)](#).

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

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation 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 **security, availability, and durability implications**.

Choose **Next Step**.

7. (Optional). On the **Add Tags** page, add tags that identify or categorize your CMK.

Choose **Next Step**.

8. Select which IAM users and roles can administer the CMK. For more information, see [Allows Key Administrators to Administer the CMK \(p. 35\)](#).

Note

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

Choose **Next Step**.

9. 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 \(p. 37\)](#).

Note

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

10. (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 \(p. 47\)](#).

Choose **Next Step**.

11. 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 [Downloading the Public Key and Import Token \(Console\) \(p. 101\)](#).

Otherwise, choose **Skip and do this later**. 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 \(p. 99\)](#).

Creating a CMK with No Key Material (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-2:11112223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "AWSAccountId": "11112223333"
  }
}
```

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 \(p. 99\)](#).

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

After you [create a customer master key \(CMK\) with no key material \(p. 96\)](#), you download a public key and an import token for that CMK. You need these items to import your key material. You can download both items in one step by 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 [manually rotate the key material \(p. 90\)](#), 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 that 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.

Select a Wrapping Algorithm

To protect your key material during import, you encrypt it using a wrapping key and wrapping algorithm. Typically, you choose an algorithm that is supported by 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.

Note

If you plan to try the [Encrypt Key Material with OpenSSL \(p. 104\)](#) proof-of-concept example in [Step 3 \(p. 104\)](#), use `RSAES_OAEP_SHA_1`.

If your HSM or key management system supports it, we recommend 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

- [Downloading the Public Key and Import Token \(Console\) \(p. 101\)](#)
- [Downloading the Public Key and Import Token \(KMS API\) \(p. 103\)](#)

Downloading the Public Key and Import Token (Console)

You can use the AWS Management Console to download the public key and import token.

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.


The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

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

1. If you just completed the steps to [create a CMK with no key material \(p. 97\)](#) and you are on the **Download wrapping key and import token** page, skip to [Step 7](#).
2. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
3. To change the AWS Region, use the Region selector in the upper-right corner of the page.
4. In the navigation pane, choose **Customer managed keys**.

Tip

You can import key material into a CMK only when its **Origin** is **EXTERNAL**. This indicates that the CMK was created with no key material. To add the **Origin** column to your table, in

the upper-right corner of the page, choose the settings icon (). Turn on **Origin**, and then choose **Confirm**.

5. Choose the alias or key ID of the CMK that is pending import.
6. Under **Key material**, choose **Download wrapping key and import token**.

The **Key material** section appears only when the CMK was created with no key material. These CMKs have an **Origin** value of **EXTERNAL**. You cannot import key material into a CMK with any other **Origin** value. For information about creating CMKs with imported key material, see [Importing Key Material in AWS Key Management Service \(AWS KMS\) \(p. 93\)](#).

7. For **Select wrapping algorithm**, choose the option that you will use to encrypt your key material. For more information about the options, see [Select a Wrapping Algorithm](#) in the preceding section.

If you plan to try the [Encrypt Key Material with OpenSSL \(p. 104\)](#) proof-of-concept example in [Step 3 \(p. 104\)](#), choose `RSAES_OAEP_SHA_1`.

8. Choose **Download wrapping key and import token**, and then save the file.

If you have a **Next** option, to continue the process now, choose **Next**; to continue later, choose **Cancel**. Otherwise, to close the window, choose **Cancel** or click the **X**.

9. 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.
10. To continue the process, see [encrypt your key material \(p. 104\)](#).


To download the public key and import token (original 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 \(p. 97\)](#), skip to [Step 6](#).

1. If you just completed the steps to [create a CMK with no key material \(p. 97\)](#), skip to [Step 6](#).
2. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
3. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
4. Choose the alias of the CMK for which you are downloading the public key and import token.

Tip

You can import key material into a CMK only when its **Origin** is **EXTERNAL**. This indicates that the CMK was created with no key material. To add the **Origin** column to your table, in

the upper-right corner of the page, choose the settings icon ().

5. In the **Key Material** section of the page, choose **Download wrapping key and import token**.

The **Key material** section appears only when the CMK was created with no key material. These CMKs have an **Origin** value of **EXTERNAL**. You cannot import key material into a CMK with any other **Origin** value. For information about creating CMKs with imported key material, see [Importing Key Material in AWS Key Management Service \(AWS KMS\) \(p. 93\)](#).

6. 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.

If you plan to try the [Encrypt Key Material with OpenSSL \(p. 104\)](#) proof-of-concept example in [Step 3 \(p. 104\)](#), choose `RSAES_OAEP_SHA_1`.

7. Choose **Download wrapping key and import token**, and then save the file.
8. 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 \(p. 104\)](#).

9. (Optional) To continue the process now, [encrypt your key material \(p. 104\)](#). 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 \(p. 104\)](#).

Downloading the Public Key and Import Token (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.

Note

If you plan to try the [Encrypt Key Material with OpenSSL \(p. 104\)](#) proof-of-concept example in [Step 3 \(p. 104\)](#), specify `RSAES_OAEP_SHA_1`.

```
$ 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 \(p. 104\)](#).

Importing Key Material Step 3: Encrypt the Key Material

After you [download the public key and import token \(p. 99\)](#), you use the public key to encrypt your key material. The key material must be in binary format.

Typically, you encrypt your key material when you export it from your hardware security module (HSM) or key management system. For information about how to export key material in binary format, 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 encryption scheme with the padding option that you specified when you [downloaded the public key and import token \(p. 99\)](#) (RSAES_OAEP_SHA_256, RSAES_OAEP_SHA_1, or RSAES_PKCS1_V1_5).

Example: 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.

The **RSAES_OAEP_SHA_1** encryption algorithm works best with this example. Before running the example, make sure that you used **RSAES_OAEP_SHA_1** for the wrapping algorithm in [Step 2 \(p. 99\)](#). If necessary, repeat the step to download and import the public key and token.

To use OpenSSL to generate binary 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 [Downloading the Public Key and Import Token \(KMS API\) \(p. 103\)](#)) 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 \(p. 105\)](#).

Importing Key Material Step 4: Import the Key Material

After you [encrypt your key material \(p. 104\)](#), 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 \(p. 104\)](#) and the import token that you downloaded at [Step 2: Download the Public Key and Import Token \(p. 99\)](#). 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 \(Console\) \(p. 105\)](#)
- [Import Key Material \(KMS API\) \(p. 106\)](#)

Import Key Material (Console)

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

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To import key material (new console)

1. If you are on the **Download wrapping key and import token** page, skip to [Step 7](#).
2. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
3. To change the AWS Region, use the Region selector in the upper-right corner of the page.
4. In the navigation pane, choose **Customer managed keys**.
5. Choose the key ID or alias of the CMK for which you downloaded the public key and import token.
6. In the **Key Material** section, choose **Upload key material**.

The **Key material** section appears only when the CMK was created with no key material. These CMKs have an **Origin** value of **EXTERNAL**. You cannot import key material into a CMK with any other **Origin** value. For information about creating CMKs with imported key material, see [Importing Key Material in AWS Key Management Service \(AWS KMS\) \(p. 93\)](#).

7. Under **Encrypted key material**, choose **Upload file**. Then upload the file that contains your wrapped (encrypted) key material.

8. Under **Import token**, choose **Upload file**. Upload the file that contains the import token that you [downloaded \(p. 101\)](#).
9. Under **Choose an expiration option**, you determine whether the key material expires. To set an expiration date and time, choose **Key material expires**, and use the calendar to select a date and time.
10. Choose **Finish** or **Upload key material**.

To import key material (original console)

1. If you just completed the optional final step of [downloading the public key and import token with the console \(p. 101\)](#), skip to [Step 6](#).
2. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
3. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
4. Choose the alias of the CMK for which you downloaded the public key and import token.
5. In the **Key Material** section, choose **Upload key material**.

The **Key material** section appears only when the CMK was created with no key material. These CMKs have an **Origin** value of **EXTERNAL**. You cannot import key material into a CMK with any other **Origin** value. For information about creating CMKs with imported key material, see [Importing Key Material in AWS Key Management Service \(AWS KMS\) \(p. 93\)](#).

6. 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 \(p. 101\)](#).
7. 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.
8. Choose **Upload key material**.

To close the window, choose **Cancel**.

Import Key Material (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.

To use this example:

1. Replace `1234abcd-12ab-34cd-56ef-1234567890ab` with the key ID of the CMK that you used when you downloaded the public key and import token. To identify the CMK, use its key ID or ARN. You cannot use an alias for this operation.
2. Replace `EncryptedKeyMaterial.bin` with the name of the file that contains the encrypted key material.
3. Replace `ImportToken.bin` with the name of the file that contains the import token.

```
$ 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 can specify an expiration date. 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 \(p. 123\)](#) 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 immediately, but you can reverse the deletion of key material by reimporting the same key material into the CMK. In contrast, deleting a CMK is irreversible. If you [schedule key deletion \(p. 110\)](#) and the required waiting period expires, AWS KMS deletes the key material and all metadata associated with the CMK.

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 \(p. 107\)](#)
- [Delete Key Material \(Console\) \(p. 108\)](#)
- [Delete Key Material \(KMS API\) \(p. 108\)](#)

How Deleting Key Material Affects AWS Services Integrated With AWS KMS

When you delete key material, the CMK becomes unusable right away. However, any [data keys \(p. 3\)](#) that AWS services are using are not immediately affected. This means that deleting key material might not immediately affect all of the data and AWS resources that are protected under the CMK, though they are affected eventually.

Several AWS services integrate with AWS KMS to protect your data. Some of these services, such as [Amazon EBS](#) and [Amazon Redshift](#), use a [customer master key \(p. 2\)](#) (CMK) in AWS KMS to generate a [data key \(p. 3\)](#), and then use the data key to encrypt your data. These plaintext data keys persist in memory as long as the data they are protecting is actively in use.

For example, consider this scenario:

1. You create an encrypted EBS volume and specify a CMK with imported key material. Amazon EBS asks AWS KMS to use your CMK to [generate an encrypted data key](#) for the volume. Amazon EBS stores the encrypted data key with the volume.
2. When you attach the EBS volume to an EC2 instance, Amazon EC2 asks AWS KMS to use your CMK to decrypt the EBS volume's encrypted data key. Amazon EC2 stores the plaintext data key in hypervisor memory and uses it 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.
3. You delete the imported key material from the CMK, which makes it unusable. This has no immediate effect on the EC2 instance or the EBS volume. The reason is that 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.
4. However, when the encrypted EBS volume is detached from the EC2 instance, Amazon EBS removes the plaintext key from memory. The next time the encrypted EBS volume is attached to an EC2

instance, the attachment fails, because Amazon EBS cannot use the CMK to decrypt the volume's encrypted data key. To use the EBS volume again, you must reimport the same key material into the CMK.

Delete Key Material (Console)

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

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To delete key material (new console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Do one of the following:
 - Select the check box for a CMK with imported key material. Choose **Key actions, Delete key material**.
 - Choose the alias or key ID of a CMK with imported key material. In the **Key Material** section of the page, choose **Delete key material**.
5. Confirm that you want to delete the key material and then choose **Delete key material**. The CMK's status, which corresponds to its [key state \(p. 123\)](#), changes to **Pending import**.

To delete key material (original console)

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation 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 \(p. 123\)](#) changes to **PendingImport**.

Delete Key Material (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 CMK, 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 \(p. 26\)](#) instead of deleting it. You can reenable 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 get 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 \(p. 120\)](#).

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 \(p. 255\)](#)

Note

If you [close or delete your AWS account](#), your CMKs become inaccessible and you are no longer billed for them. You do not need to schedule deletion of your CMKs separate from closing the account.

Topics

- [How Deleting Customer Master Keys Works \(p. 110\)](#)
- [Scheduling and Canceling Key Deletion \(p. 111\)](#)
- [Adding Permission to Schedule and Cancel Key Deletion \(p. 114\)](#)
- [Creating an Amazon CloudWatch Alarm to Detect Usage of a Customer Master Key that is Pending Deletion \(p. 117\)](#)
- [Determining Past Usage of a Customer Master Key \(p. 120\)](#)

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 status and key state is *Pending deletion*.

- A CMK that is pending deletion cannot be used in any cryptographic operations.
- AWS KMS does not [rotate the backing keys \(p. 88\)](#) of CMKs that are pending deletion.

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, suppose you schedule key deletion and specify a waiting period of 7 days. In that case, 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 \(p. 117\)](#) 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](#), use a [customer master key \(p. 2\)](#) (CMK) in AWS KMS to generate a [data key \(p. 3\)](#) and then use the data key to encrypt your data. These plaintext data keys persist in memory as long as the data they are protecting is actively in use.

Scheduling a CMK for deletion makes it unusable, but it does not prevent the AWS service from using data keys in memory to encrypt and decrypt your data. The service is not affected until it needs to use the CMK that is pending deletion or deleted.

For example, consider this scenario:

1. You create an encrypted EBS volume and specify a CMK. Amazon EBS asks AWS KMS to use your CMK to [generate an encrypted data key](#) for the volume. Amazon EBS stores the encrypted data key with the volume.
2. When you attach the EBS volume to an EC2 instance, Amazon EC2 asks AWS KMS to use your CMK to decrypt the EBS volume's encrypted data key. Amazon EC2 stores the plaintext data key in hypervisor memory and uses it 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.
3. You schedule the CMK for deletion, which makes it unusable. 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 disk I/O to the EBS volume.

Even when the scheduled time elapses and AWS KMS deletes the CMK, there is no immediate effect on the EC2 instance or the EBS volume, because Amazon EC2 is using the plaintext data key, not the CMK.

4. However, when the encrypted EBS volume is detached from the EC2 instance, Amazon EBS removes the plaintext key from memory. The next time the encrypted EBS volume is attached to an EC2 instance, the attachment fails, because Amazon EBS cannot use the CMK to decrypt the volume's encrypted data key.

Scheduling and Canceling Key Deletion

The following procedures describe how to schedule key deletion and cancel key deletion in AWS KMS using the AWS Management Console, the AWS CLI, and 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 \(p. 26\)](#) 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. 114).

Ways to schedule and cancel key deletion

- [Scheduling and Canceling Key Deletion \(Console\)](#) (p. 112)
- [Scheduling and Canceling Key Deletion \(AWS CLI\)](#) (p. 113)
- [Scheduling and Canceling Key Deletion \(AWS SDK for Java\)](#) (p. 114)

Scheduling and Canceling Key Deletion (Console)

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

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To schedule and cancel key deletion (new console)

To schedule key deletion

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Select the check box next to the CMK that you want to delete.
5. Choose **Key actions, Schedule key deletion**.
6. For **Waiting period (in days)**, enter a number of days between 7 and 30.
7. Select the check box next to **Confirm you want to schedule this key for deletion in *<number of days>* days**.
8. Choose **Schedule deletion**.

The CMK status changes to **Pending deletion**.

To cancel key deletion

1. Open the AWS KMS console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Select the check box next to the CMK that you want to recover.
5. Choose **Key actions, Cancel key deletion**.

The CMK status changes from **Pending deletion** to **Disabled**. To use the CMK, you must [enable it](#) (p. 26).

To schedule and cancel key deletion (original console)

To schedule key deletion

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
3. Select the check box next to the CMK 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 CMK status changes to **Pending Deletion**.

To cancel key deletion

1. Go to the original AWS KMS console at <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
3. Select the check box next to the CMK that you want to recover.
4. Choose **Key Actions, Cancel key deletion**.

The CMK status changes from **Pending Deletion** to **Disabled**. To use the CMK, you must [enable it \(p. 26\)](#).

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 AWS CLI returns output like the output shown in the following example:

```
{
  "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 AWS CLI returns output like the output shown in the following example:

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

The status of the CMK changes from **Pending Deletion** to **Disabled**. To use the CMK, you must [enable it \(p. 26\)](#).

Scheduling and Canceling Key Deletion (AWS SDK for Java)

The following example demonstrates how to schedule a CMK 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);
```

The status of the CMK changes from **Pending Deletion** to **Disabled**. To use the CMK, you must [enable it](#) (p. 26).

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

- [Adding Permission to Schedule and Cancel Key Deletion \(Console\)](#) (p. 114)
- [Adding Permission to Schedule and Cancel Key Deletion \(AWS CLI\)](#) (p. 116)

Adding Permission to Schedule and Cancel Key Deletion (Console)

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

Note

AWS KMS recently introduced a new console that makes it easier for you to organize and manage your KMS resources. It is available in all AWS Regions that AWS KMS supports except for AWS GovCloud (US). We encourage you to try the new AWS KMS console at <https://console.aws.amazon.com/kms>.

The original console will remain available for a brief period to give you time to familiarize yourself with the new one. To use the original console, choose **Encryption Keys** in the IAM console or go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>. Please share your feedback by choosing **Feedback** in either console or in the lower-right corner of this page.

To add permission to schedule and cancel key deletion (new console)

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose the alias or key ID of the CMK whose permissions you want to change.
5. 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 usually means that you have changed 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**.

To add permission to schedule and cancel key deletion (original console)

1. Sign in to the AWS Management Console and go to <https://console.aws.amazon.com/iam/home?#/encryptionKeys>.
2. For **Region**, choose the appropriate AWS Region. Do not use the region selector in the navigation bar (top right corner).
3. Choose the alias of the CMK 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"). Then 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 CMK.

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

You can combine the features of AWS CloudTrail, Amazon CloudWatch Logs, and Amazon Simple Notification Service (Amazon SNS) that notify you when someone in your account tries to use a CMK that is pending deletion in a cryptographic operation. If you receive this notification, you might want to cancel deletion of the CMK and reconsider your decision 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. This error message indicates that a person or application tried to use the CMK in a cryptographic operation (Encrypt, Decrypt, GenerateDataKey, GenerateDataKeyWithoutPlaintext, and ReEncrypt). Because the notification is linked to the error message, it is not triggered when you use API operations that are permitted on CMKs that are pending deletion, such as ListKeys, CancelKeyDeletion, and PutKeyPolicy. To see a list of the AWS KMS API operations that return this error message, see [How Key State Affects Use of a Customer Master Key \(p. 123\)](#).

The notification email that you receive does not list the CMK or the cryptographic operation. You can find that information in [your CloudTrail log \(p. 180\)](#). Instead, the email reports that the alarm state changed from **OK** to **Alarm**. For more information about CloudWatch Alarms and state changes, see [Creating Amazon CloudWatch Alarms](#) in the *Amazon CloudWatch User Guide*.

Topics

- [Requirements for a CloudWatch Alarm \(p. 117\)](#)
- [Create the CloudWatch Alarm \(p. 118\)](#)

Requirements for a CloudWatch Alarm

Before you create a CloudWatch alarm, you must create an AWS CloudTrail trail and configure CloudTrail to deliver CloudTrail log files to Amazon CloudWatch Logs.

1. [Create a CloudTrail trail](#).

CloudTrail is automatically enabled on your AWS account when you create the account. However, for an ongoing record of events in your account, including events for AWS KMS, create a trail.

2. [Configure CloudTrail to deliver your log files CloudWatch Logs](#).

Configure delivery of your CloudTrail log files to CloudWatch Logs. This allows CloudWatch Logs to monitor the logs for AWS KMS API requests that attempt to use a CMK that is pending deletion.

Create the CloudWatch Alarm

To receive a notification when AWS KMS API requests attempt to use a CMK that is pending deletion in a cryptographic operation, create a CloudWatch alarm and configure notifications.

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. Use the Region selector on the upper right to choose the AWS Region you want to monitor.
3. In the left navigation pane, choose **Logs**.
4. In the list of **Log Groups**, choose the option button next to your log group. 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 for **Metric Value**, type **1**, if this is not the current value.
 - d. Choose **Create Filter**.
7. In the filter box, choose **Create Alarm**.
8. In the **Create Alarm** window, do the following:
 - a. In the **Alarm Threshold** section, for **Name**, type **KMSKeyPendingDeletionErrorAlarm**. You can also add an optional description.
 - b. Following **Whenever**, for **is**, choose **>=** and then type **1**.
 - c. For **1 out of *n* datapoints**, if necessary, type **1**.
 - d. In the **Additional settings** section, for **Treat missing data as**, choose **good (not breaching threshold)**.
 - e. In the **Actions** section, 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, such as **KMSAlert**. 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.
 - f. Choose **Create Alarm**.

Create Alarm

1. Select Metric
2. Define Alarm

Alarm Threshold

Provide the details and threshold for your alarm. Use the graph on the right to help set the appropriate threshold.

Name:

Description:

Whenever: KMSKeyPendingDeletionErrorCount

is:
for: 1 1 datapoints

Additional settings

Provide additional configuration for your alarm.

Treat missing data as:

Actions

Define what actions are taken when your alarm changes state.

Notification

Whenever this alarm:

Send notification to:
New list Enter list

Email list:

+ Notification
+ AutoScaling Action
+ EC2 Action

Alarm Preview

This alarm will trigger when the blue line goes up to or above the red line for 1 datapoints within 5 minutes

Namespace: CloudTrailLogMetrics

Metric Name:

Period:

Statistic: ☒ Standard ☐ Custom

Cancel Previous Next Create Alarm

- 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 \(p. 111\)](#) 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 \(p. 120\)](#)
- [Examining AWS CloudTrail Logs to Determine Actual Usage \(p. 120\)](#)

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 \(p. 80\)](#).

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 \(SSE-KMS\) \(p. 161\)](#). 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 with AWS CloudTrail \(p. 180\)](#). For more information about CloudTrail go to the [AWS CloudTrail User Guide](#).













































How Key State Affects Use of a Customer Master Key

A customer master key (CMK) is always in one of the following states: **Enabled**, **Disabled**, **Pending import**, or **Pending deletion**. In the AWS Management Console, the key state appears in the **Status** field. To find the key state of a CMK by using the AWS KMS API, use the [DescribeKey](#) operation.

The following table shows whether AWS KMS API operations that run on a CMK in each state can be expected to succeed (✓), fail (✗), or succeed only under certain conditions (?). The result often differs for CMKs with imported key material.

The `CreateKey` and `GenerateRandom` API operations have an not-applicable (N/A) result because they do not use an existing CMK.

API	Enabled	Disabled	Pending import	Pending deletion
CancelKeyDeletion	 [4]	 [4]	 [4]	✓
CreateAlias	✓	✓	✓	 [3]
CreateGrant	✓	 [1]	 [5]	 [2] or [3]
CreateKey	N/A	N/A	N/A	N/A
Decrypt	✓	 [1]	 [5]	 [2] or [3]
DeleteAlias	✓	✓	✓	✓
DeleteImportedKeyMaterial	 [9]	 [9]	✓ (No effect)	 [9]
DescribeKey	✓	✓	✓	✓
DisableKey	✓	✓	 	

API	Enabled	Disabled	Pending import	Pending deletion
			[5]	[3]
DisableKeyRotation	 [7]	 [1] or [7]	 [6]	 [3] or [7]
EnableKey	 [7]	 [1] or [7]	 [5]	 [3]
EnableKeyRotation	 [7]	 [1] or [7]	 [6]	 [3] or [7]
Encrypt	 [7]	 [1]	 [5]	 [2] or [3]
GenerateDataKey	 [7]	 [1]	 [5]	 [2] or [3]
GenerateDataKeyWithoutPlaintext	 [7]	 [1]	 [5]	 [2] or [3]
GenerateRandom	N/A	N/A	N/A	N/A
GetKeyPolicy	 [7]	 [7]	 [6]	 [7]
GetKeyRotationStatus	 [7]	 [7]	 [6]	 [7]
GetParametersForImport	 [9]	 [9]	 [8] or [9]	 [8] or [9]
ImportKeyMaterial	 [9]	 [9]	 [8] or [9]	 [8] or [9]
ListAliases	 [7]	 [7]	 [6]	 [7]









API	Enabled	Disabled	Pending import	Pending deletion
ListGrants	✓	✓	✓	✓
ListKeyPolicies	✓	✓	✓	✓
ListKeys	✓	✓	✓	✓
ListResourceTags	✓	✓	✓	✓
ListRetirableGrants	✓	✓	✓	✓
PutKeyPolicy	✓	✓	✓	✓
ReEncrypt	✓	 [1]	 [5]	 [2] or [3]
RetireGrant	✓	✓	✓	✓
RevokeGrant	✓	✓	✓	✓
ScheduleKeyDeletion	✓	✓	✓	 [3]
TagResource	✓	✓	✓	 [3]
UntagResource	✓	✓	✓	 [3]
UpdateAlias	✓	✓	✓	 [10]
UpdateKeyDescription	✓	✓	✓	 [3]

Table Details

- [1] DisabledException: **<CMK ARN>** is disabled.
- [2] DisabledException: **<CMK ARN>** is pending deletion.
- [3] KMSInvalidStateException: **<CMK ARN>** is pending deletion.
- [4] KMSInvalidStateException: **<CMK ARN>** is not pending deletion.
- [5] KMSInvalidStateException: **<CMK ARN>** is pending import.
- [6] UnsupportedOperationException: **<CMK ARN>** origin is EXTERNAL which is not valid for this operation.
- [7] If the CMK has imported key material: UnsupportedOperationException.
- [8] If the CMK has imported key material: KMSInvalidStateException
- [9] If the CMK does not have imported key material: UnsupportedOperationException.
- [10] If the source CMK is pending deletion, the command succeeds. If the destination CMK is pending deletion, the command fails with error: KMSInvalidStateException : **<CMK ARN>** is pending deletion.

How AWS Services use AWS KMS

Many AWS services use AWS KMS to support encryption of your data. When an AWS service is integrated with AWS KMS, you can use the customer master keys (CMKs) in your account to protect the data that the service receives, stores, or manages for you. For the complete list of AWS services that are integrated with AWS KMS, see [AWS Service Integration](#).

The following topics discuss in detail how particular services use AWS KMS, including the CMKs they support, how they manage data keys, the permissions they require, and how to track each service's use of the CMKs in your account.

Topics

- [How AWS CloudTrail Uses AWS KMS \(p. 127\)](#)
- [How Amazon DynamoDB Uses AWS KMS \(p. 132\)](#)
- [How Amazon Elastic Block Store \(Amazon EBS\) Uses AWS KMS \(p. 139\)](#)
- [How Amazon Elastic Transcoder Uses AWS KMS \(p. 141\)](#)
- [How Amazon EMR Uses AWS KMS \(p. 145\)](#)
- [How Amazon Redshift Uses AWS KMS \(p. 149\)](#)
- [How Amazon Relational Database Service \(Amazon RDS\) Uses AWS KMS \(p. 150\)](#)
- [How AWS Secrets Manager Uses AWS KMS \(p. 151\)](#)
- [How Amazon Simple Email Service \(Amazon SES\) Uses AWS KMS \(p. 158\)](#)
- [How Amazon Simple Storage Service \(Amazon S3\) Uses AWS KMS \(p. 161\)](#)
- [How AWS Systems Manager Parameter Store Uses AWS KMS \(p. 162\)](#)
- [How Amazon WorkMail Uses AWS KMS \(p. 167\)](#)
- [How Amazon WorkSpaces Uses AWS KMS \(p. 169\)](#)

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 \(p. 127\)](#)
- [Understanding How Often Your CMK is Used \(p. 131\)](#)

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 \(p. 161\)](#) 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 \(CMK\) \(p. 128\)](#)
- [CloudTrail Puts a Log File into Your S3 Bucket \(p. 129\)](#)
- [You Get an Encrypted Log File from Your S3 Bucket \(p. 130\)](#)

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 \(p. 248\)](#):

- 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", 1
    "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. 248):

- 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", 6
    "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. 248):

- 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",
      "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",
    }
  },
  "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",
    "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. 129) 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 \(p. 130\)](#) it results in an additional AWS KMS API request.

How Amazon DynamoDB Uses AWS KMS

[Amazon DynamoDB](#) is a fully managed, scalable NoSQL database service. DynamoDB integrates with AWS Key Management Service (AWS KMS) to support an optional [encryption at rest](#) server-side encryption feature.

With *encryption at rest*, DynamoDB transparently encrypts all customer data in an encrypted table, including its primary key and local and global [secondary indexes](#), whenever the table is persisted to disk. (If your table has a sort key, some of the sort keys that mark range boundaries are stored in plaintext in the table metadata.) When you access an encrypted table, DynamoDB decrypts the table data transparently. You do not need to change your applications to use or manage encrypted tables.

You enable encryption at rest when you create a table. Each table is encrypted independently, so you can enable encryption on selected tables, while leaving others unencrypted.

Encryption at rest uses AWS KMS customer master keys in your AWS to protect your DynamoDB tables on disk. This integration also enables you to audit access to your DynamoDB tables by examining the DynamoDB API calls to AWS KMS in audit logs and trails.

Client-Side Encryption for DynamoDB

In addition to encryption at rest, which is a *server-side encryption* feature, AWS provides the [Amazon DynamoDB Encryption Client](#). This *client-side encryption* library enables you to protect your data before submitting to the DynamoDB. With server-side encryption, your data is encrypted in transit over an HTTPS connection, decrypted at the DynamoDB endpoint, and then re-encrypted before being stored in DynamoDB. Client-side encryption provides end-to-end protection for your data from its source to storage in DynamoDB.

You can use encryption at rest, the DynamoDB Encryption Client, or both. To help you decide which method is right for your DynamoDB data, see [Client-Side or Server-Side Encryption?](#) in the *Amazon DynamoDB Encryption Client Developer Guide*.

Topics

- [Using CMKs and Data Keys \(p. 132\)](#)
- [Authorizing Use of the Service Default Key \(p. 134\)](#)
- [DynamoDB Encryption Context \(p. 135\)](#)
- [Monitoring DynamoDB Interaction with AWS KMS \(p. 136\)](#)

Using CMKs and Data Keys

The DynamoDB encryption at rest feature uses an AWS KMS CMK and a hierarchy of data keys to protect your table data.

Service Default Key

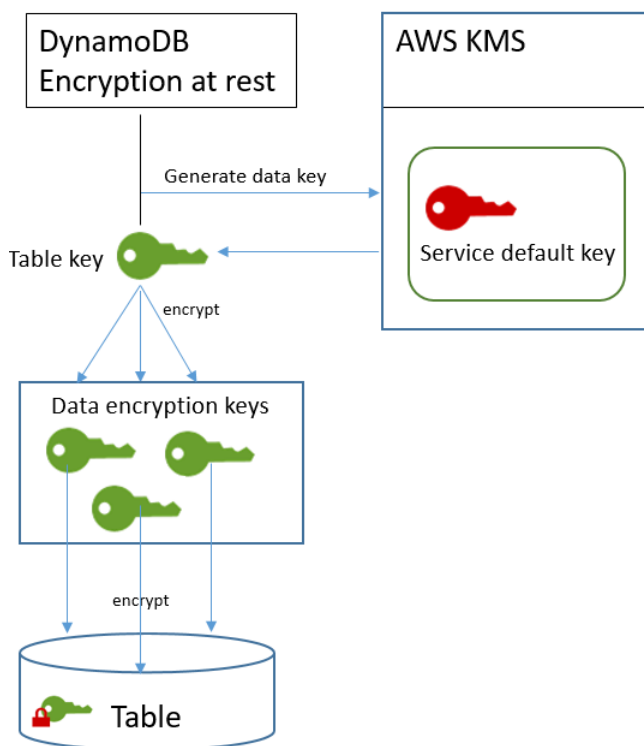
When you create an encrypted table, DynamoDB creates a unique AWS managed [customer master key \(p. 2\)](#) (CMK) in each region of your AWS account, if one does not already exist. This CMK, `aws/dynamodb`, is known as the *service default key*. Like all CMKs, the service default key never leaves AWS KMS unencrypted. The encryption at rest feature does not support the use of customer managed CMKs.

The service default key in each region protects the table keys for tables in that region.

Table Keys

DynamoDB uses the service default key in AWS KMS to generate and encrypt a unique [data key \(p. 3\)](#) for each table, known as the *table key*. The table key persists for the lifetime of the encrypted table.

The table key is used as a master key. DynamoDB generates data encryption keys and uses them to encrypt table data. Then, it uses the table key to protect the data encryption keys. DynamoDB generates a unique data encryption key for each underlying structure in a table, but multiple table items might be protected by the same data encryption key.



When you access an encrypted table, DynamoDB sends a request to AWS KMS to use the service default key in AWS KMS to decrypt the table key. Then, it uses the plaintext table key to decrypt the data encryption keys, and uses the plaintext data encryption keys to decrypt table data.

DynamoDB generates, uses, and stores the table key and data encryption keys outside of AWS KMS. It protects all keys with [Advanced Encryption Standard \(AES\)](#) encryption and 256-bit encryption keys. It removes plaintext keys as soon as possible after using them. Then, it stores the encrypted keys with the encrypted data so they are available to decrypt the table data on demand.

Table Key Caching

To avoid calling AWS KMS for every DynamoDB operation, DynamoDB caches the plaintext table keys for each principal in memory. If DynamoDB gets a request for the cached table key after five

minutes of inactivity, it sends a new request to AWS KMS to decrypt the table key. This call will capture any changes to the authorization context of the service default key.

DynamoDB caches the table key in memory for up to 12 hours. This enables DynamoDB to maintain your access to your table data in the extraordinary circumstance that AWS KMS becomes unresponsive for an extended period of time.

Authorizing Use of the Service Default Key

The authorization context on the DynamoDB [service default key \(p. 132\)](#) includes its key policy and grants that delegate the permissions to use it.

Service Default Key Policy

When DynamoDB uses the service default key in cryptographic operations, it does so on behalf of the user who is accessing the [DynamoDB resource](#). The key policy on the service default key gives users in the account permission to use the service default key for specified operations. But permission is granted only when DynamoDB makes the request on the user's behalf. The key policy does not allow any user to use the service default key directly.

This key policy, like the policies of all [AWS managed keys \(p. 2\)](#), is established by the service. You cannot change it, but you can view it at any time. To get the key policy for the DynamoDB service default key in your account, use the [GetKeyPolicy](#) operation.

The policy statements in the key policy have the following effect:

- Allow users in the account to use the service default key for cryptographic operations when the request comes from DynamoDB on their behalf. The policy also allows users to [create grants \(p. 135\)](#) for the CMK.
- Allows the AWS account root user to view the CMK properties and to [revoke the grant](#) that allows DynamoDB to use the service default key. DynamoDB uses [grants \(p. 135\)](#) for ongoing maintenance operations.
- Allows DynamoDB to perform read-only operations to get the service default key in your account.

```
{
  "Version" : "2012-10-17",
  "Id" : "auto-dynamodb-1",
  "Statement" : [ {
    "Sid" : "Allow access through Amazon DynamoDB for all principals in the account that
are authorized to use Amazon DynamoDB",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "*"
    },
    "Action" : [ "kms:Encrypt", "kms:Decrypt", "kms:ReEncrypt*", "kms:GenerateDataKey*",
"kms>CreateGrant", "kms:DescribeKey" ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:CallerAccount" : "111122223333",
        "kms:ViaService" : "dynamodb.us-west-2.amazonaws.com"
      }
    }
  }, {
    "Sid" : "Allow direct access to key metadata to the account",
    "Effect" : "Allow",
    "Principal" : {
```

```
    "AWS" : "arn:aws:iam::111122223333:root"
  },
  "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*", "kms:RevokeGrant" ],
  "Resource" : "*"
}, {
  "Sid" : "Allow DynamoDB Service with service principal name dynamodb.amazonaws.com to describe the key directly",
  "Effect" : "Allow",
  "Principal" : {
    "Service" : "dynamodb.amazonaws.com"
  },
  "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*" ],
  "Resource" : "*"
} ]
}
```

Using Grants to Authorize DynamoDB

In addition to key policies, DynamoDB uses grants to set permissions on the DynamoDB [service default key](#) (p. 132). To view the grants on the DynamoDB service default key in your account, use the [ListGrants](#) operation.

DynamoDB uses the grant permissions when it performs background system maintenance and continuous data protection tasks. It also uses grants to generate [table keys](#) (p. 132).

Each grant is specific to a table. If the account includes multiple encrypted tables, there is a grant of each type for each table. The grant is constrained by the [DynamoDB encryption context](#) (p. 135), which includes the table name and the AWS account ID, and it includes permission to the [revoke the grant](#) if it is no longer needed.

To create the grants, DynamoDB calls `CreateGrant` on behalf of the user who created the encrypted table. Permission to create the grant comes from the [key policy](#) (p. 134), which allows account users to call `CreateGrant` on the service default key only when DynamoDB makes the request on an authorized user's behalf.

The key policy also allows the account root to [revoke the grant](#) on the service default key. However, if you revoke the grant on an active encrypted table, DynamoDB will not be able to protect and maintain the table.

DynamoDB Encryption Context

An [encryption context](#) (p. 6) is a set of key–value pairs that contain arbitrary nonsecret data. When you include an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

DynamoDB uses the same encryption context in all KMS cryptographic operations. You can use the encryption context to identify a cryptographic operation in audit records and logs. It also appears in plaintext in logs, such as [AWS CloudTrail](#) and [Amazon CloudWatch Logs](#).

The encryption context can also be used as a condition for authorization in policies and grants. DynamoDB uses it to constrain the [grants](#) (p. 135) that allow access to the service default key.

In its requests to AWS KMS, DynamoDB uses an encryption context with two key–value pairs.

```
"encryptionContextSubset": {
  "aws:dynamodb:tableName": "Books"
  "aws:dynamodb:subscriberId": "111122223333"
```

```
}
```

- **Table** – The first key–value pair identifies the table that DynamoDB is encrypting. The key is `aws:dynamodb:tableName`. The value is the name of the table.

```
"aws:dynamodb:tableName": "<table-name>"
```

For example:

```
"aws:dynamodb:tableName": "Books"
```

- **Account** – The second key–value pair identifies the AWS account. The key is `aws:dynamodb:subscriberId`. The value is the account ID.

```
"aws:dynamodb:subscriberId": "<account-id>"
```

For example:

```
"aws:dynamodb:subscriberId": "111122223333"
```

Monitoring DynamoDB Interaction with AWS KMS

You can use AWS CloudTrail and Amazon CloudWatch Logs to track the requests that DynamoDB sends to AWS KMS on your behalf.

The `GenerateDataKey`, `Decrypt`, and `CreateGrant` requests are discussed in this section. In addition, DynamoDB uses a [DescribeKey](#) operation to determine whether a DynamoDB service default key exists in the account and region. It also uses a [RetireGrant](#) operation to remove a grant when you delete a table.

GenerateDataKey

When you enable encryption at rest on a table, DynamoDB creates a unique table key. It sends a [GenerateDataKey](#) request to AWS KMS that specifies the `aws/dynamodb` service default key in your account.

The event that records the `GenerateDataKey` operation is similar to the following example event. The user is DynamoDB. The parameters include the Amazon Resource Name (ARN) of the service default key, a key specifier that requires a 256-bit key, and the [encryption context](#) (p. 135) that identifies the table and the AWS account.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AWSService",
    "invokedBy": "dynamodb.amazonaws.com"
  },
  "eventTime": "2018-02-14T00:15:17Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "GenerateDataKey",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "dynamodb.amazonaws.com",
  "userAgent": "dynamodb.amazonaws.com",
  "requestParameters": {
    "encryptionContext": {
      "aws:dynamodb:tableName": "Services",
```

```
        "aws:dynamodb:subscriberId": "111122223333"
      },
      "keySpec": "AES_256",
      "keyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "responseElements": null,
    "requestID": "229386c1-111c-11e8-9e21-c11ed5a52190",
    "eventID": "e3c436e9-ebca-494e-9457-8123a1f5e979",
    "readOnly": true,
    "resources": [
      {
        "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "accountId": "111122223333",
        "type": "AWS::KMS::Key"
      }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "sharedEventID": "bf915fa6-6ceb-4659-8912-e36b69846aad"
  }
}
```

Decrypt

When you access an encrypted DynamoDB table, DynamoDB needs to decrypt the table key so that it can decrypt the keys below it in the hierarchy. It then decrypts the data in the table. To decrypt the table key, DynamoDB sends a [Decrypt](#) request to AWS KMS that specifies the `aws/dynamodb` service default key in your account.

The event that records the `Decrypt` operation is similar to the following example event. The user is the principal in your AWS account who is accessing the table. The parameters include the encrypted table key (as a ciphertext blob) and the [encryption context](#) (p. 135) that identifies the table and the AWS account. AWS KMS derives the ID of the service default key from the ciphertext.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAIQDTESTANDEXAMPLE:user01",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2018-02-14T16:42:15Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAIQDT3HGFQZX4RY6RU",
        "arn": "arn:aws:iam::111122223333:role/Admin",
        "accountId": "111122223333",
        "userName": "Admin"
      }
    }
  },
  "invokedBy": "dynamodb.amazonaws.com",
  "eventTime": "2018-02-14T16:42:39Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "dynamodb.amazonaws.com",
  "userAgent": "dynamodb.amazonaws.com",
```



```
"requestParameters":
{
  "encryptionContext":
  {
    "aws:dynamodb:tableName": "Books",
    "aws:dynamodb:subscriberId": "111122223333"
  }
},
"responseElements": null,
"requestID": "11cab293-11a6-11e8-8386-13160d3e5db5",
"eventID": "b7d16574-e887-4b5b-a064-bf92f8ec9ad3",
"readOnly": true,
"resources": [
  {
    "ARN": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "accountId": "111122223333",
    "type": "AWS::KMS::Key"
  }
],
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333"
}
```

CreateGrant

DynamoDB uses [grants \(p. 135\)](#) to allow the service to perform continuous data protection and maintenance and durability tasks.

The grants that DynamoDB creates are specific to a table. The principal in the [CreateGrant](#) request is the user who created the table.

The event that records the [CreateGrant](#) operation is similar to the following example event. The parameters include the Amazon Resource Name (ARN) of the service default key, the grantee principal and retiring principal (the DynamoDB service), and the operations that the grant covers. It also includes a constraint that requires all encryption operation use the specified [encryption context \(p. 135\)](#).

```
{
  "eventVersion": "1.05",
  "userIdentity":
  {
    "type": "AssumedRole",
    "principalId": "AROAGDTESTANDEXAMPLE:user01",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2018-02-14T00:12:02Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAGDTESTANDEXAMPLE",
        "arn": "arn:aws:iam::111122223333:role/Admin",
        "accountId": "111122223333",
        "userName": "Admin"
      }
    },
    "invokedBy": "dynamodb.amazonaws.com"
  },
  "eventTime": "2018-02-14T00:15:15Z",
```

```
{
  "eventSource": "kms.amazonaws.com",
  "eventName": "CreateGrant",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "dynamodb.amazonaws.com",
  "userAgent": "dynamodb.amazonaws.com",
  "requestParameters": {
    "keyId": "arn:aws:kms:us-west-2:11112223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "retiringPrincipal": "dynamodb.us-west-2.amazonaws.com",
    "constraints": {
      "encryptionContextSubset": {
        "aws:dynamodb:tableName": "Books",
        "aws:dynamodb:subscriberId": "11112223333"
      }
    },
    "granteePrincipal": "dynamodb.us-west-2.amazonaws.com",
    "operations": [
      "DescribeKey",
      "GenerateDataKey",
      "Decrypt",
      "Encrypt",
      "ReEncryptFrom",
      "ReEncryptTo",
      "RetireGrant"
    ]
  },
  "responseElements": {
    "grantId": "5c5cd4a3d68e65e77795f5ccc2516dff057308172b0cd107c85b5215c6e48bde"
  },
  "requestID": "2192b82a-111c-11e8-a528-f398979205d8",
  "eventID": "a03d65c3-9fee-4111-9816-8bf96b73df01",
  "readOnly": false,
  "resources": [
    {
      "ARN": "arn:aws:kms:us-west-2:11112223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "accountId": "11112223333",
      "type": "AWS::KMS::Key"
    }
  ],
  "eventType": "AwsApiCall",
  "recipientAccountId": "11112223333"
}
```

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

This topic discusses in detail how [Amazon Elastic Block Store \(Amazon EBS\)](#) uses AWS KMS to encrypt volumes and snapshots. For basic instructions about encrypting Amazon EBS volumes, see [Amazon EBS Encryption](#).

Topics

- [Amazon EBS Encryption \(p. 140\)](#)
- [Using CMKs and Data Keys \(p. 140\)](#)
- [Amazon EBS Encryption Context \(p. 140\)](#)
- [Detecting Amazon EBS Failures \(p. 141\)](#)
- [Using AWS CloudFormation to Create Encrypted Amazon EBS Volumes \(p. 141\)](#)

Amazon EBS Encryption

When you attach an encrypted Amazon EBS volume 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.

The encryption status of an EBS volume is determined when you create the volume. You cannot change the encryption status of an existing volume. However, you can [migrate data](#) between encrypted and unencrypted volumes and apply a new encryption status while copying a snapshot.

Using CMKs and Data Keys

When you [create an encrypted Amazon EBS volume](#), you specify an AWS KMS customer master key (CMK). By default, Amazon EBS uses the [AWS managed CMK \(p. 2\)](#) for Amazon EBS in your account. However, you can specify a [customer master key \(CMK\) \(p. 2\)](#).

Amazon EBS uses the CMK that you specify to generate a unique data key for each volume. It stores an encrypted copy of the data key with the volume. Then, when you attach the volume to an Amazon EC2 instance, Amazon EBS uses the data key to encrypt all disk I/O to the volume.

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 plaintext data key persists in memory as long as the EBS volume is attached to the EC2 instance.

Amazon EBS Encryption Context

In its [GenerateDataKeyWithoutPlaintext](#) and [Decrypt](#) requests to AWS KMS, Amazon EBS uses an encryption context with a name-value pair that identifies the volume or snapshot in the request. The name in the encryption context does not vary.

An [encryption context](#) is a set of key-value pairs that contain arbitrary nonsecret data. When you include an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

For all volumes and for encrypted snapshots created with the Amazon EBS [CreateSnapshot](#) operation, Amazon EBS uses the volume ID as encryption context value. In the `requestParameters` field of a CloudTrail log entry, the encryption context looks similar to the following:

```
"encryptionContext": {
```

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

For encrypted snapshots created with the Amazon EC2 [CopySnapshot](#) operation, Amazon EBS uses the snapshot ID as encryption context value. 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 [key state \(p. 123\)](#) is not `Enabled`—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 \(p. 11\)](#) 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 \(p. 26\)](#).
- If the CMK's key state is pending import, [import key material \(p. 94\)](#).
- If the CMK's key state is pending deletion, [cancel key deletion \(p. 111\)](#).

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 \(p. 142\)](#)
- [Decrypting the input file \(p. 142\)](#)

- [Encrypting the output file \(p. 143\)](#)
- [HLS Content Protection \(p. 144\)](#)
- [Elastic Transcoder Encryption Context \(p. 145\)](#)

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 AES client-side encryption or after uploading by using Amazon S3 server-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 with the specified CMK, and then sends both the plaintext data key and the encrypted data key to Amazon S3.
3. Amazon S3 uses the plaintext data key to encrypt the media file and then 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

Encryption mode	AWS KMS key	Meaning
		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

Encryption mode	AWS KMS key	Meaning
		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

An [encryption context](#) (p. 248) is a set of key–value pairs that contain arbitrary nonsecret data. When you include an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

Elastic Transcoder uses the same encryption context in all AWS KMS API requests to generate data keys, encrypt, and decrypt.

```
"service" : "elastictranscoder.amazonaws.com"
```

The encryption context is written to CloudTrail logs to help you understand how a given AWS KMS CMK was used. 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* 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. 146)
- [Encrypting Data on the Storage Volumes of Cluster Nodes](#) (p. 148)
- [Encryption Context](#) (p. 148)

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. 147).
- [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. 147).

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. 5) workflow. 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. 147)
- [Process for Encrypting Data on EMRFS with CSE-KMS](#) (p. 147)

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 \(p. 148\)](#).
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 \(p. 148\)](#).
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 \(p. 148\)](#).
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 \(p. 148\)](#).

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. 248).

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_cmek_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](#) (p. 149)
- [Encryption Context](#) (p. 150)

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 \(p. 248\)](#).

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 RDS 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 \(p. 139\)](#).

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 \(p. 248\)](#). 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 AWS Secrets Manager Uses AWS KMS

[AWS Secrets Manager](#) is an AWS service that encrypts and stores your secrets, and transparently decrypts and returns them to you in plaintext. It's designed especially to store application secrets, such as login credentials, that change periodically and should not be hard-coded or stored in plaintext in the application. In place of hard-coded credentials or table lookups, your application calls Secrets Manager.

Secrets Manager also supports features that periodically rotate the secrets associated with commonly used databases. It always encrypts newly rotated secrets before they are stored.

Secrets Manager integrates with AWS Key Management Service (AWS KMS) to encrypt every version of every secret with a unique [data encryption key \(p. 3\)](#) that is protected by an AWS KMS [customer master key \(p. 2\)](#) (CMK). This integration protects your secrets under encryption keys that never leave AWS KMS unencrypted. It also enables you to set custom permissions on the master key and audit the operations that generate, encrypt, and decrypt the data keys that protect your secrets.

Topics

- [Protecting the Secret Value \(p. 151\)](#)
- [Encrypting and Decrypting Secrets \(p. 151\)](#)
- [Using Your AWS KMS CMK \(p. 153\)](#)
- [Authorizing Use of the CMK \(p. 154\)](#)
- [Secrets Manager Encryption Context \(p. 155\)](#)
- [Monitoring Secrets Manager Interaction with AWS KMS \(p. 156\)](#)

Protecting the Secret Value

To protect a secret, Secrets Manager encrypts the *secret value* in a secret.

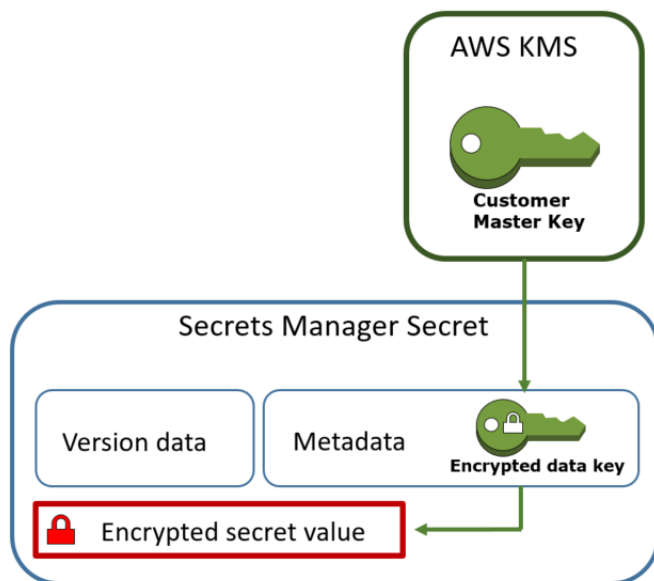
In Secrets Manager, a [secret](#) consists of a *secret value*, also known as *protected secret text* or *encrypted secret data*, and related metadata and version information. The secret value can be any string or binary data of up to 4096 bytes, but it is typically a collection of name-value pairs that comprise the login information for a server or database.

Secrets Manager always encrypts the entire secret value before it stores the secret. It decrypts the secret value transparently whenever you get or change the secret value. There is no option to enable or disable encryption. To encrypt and decrypt the secret value, Secrets Manager uses AWS KMS.

Encrypting and Decrypting Secrets

To protect secrets, Secrets Manager uses [envelope encryption \(p. 5\)](#) with AWS KMS [customer master keys \(p. 2\)](#) (CMKs) and [data keys \(p. 3\)](#).

Secrets Manager uses a unique data key to protect each secret value. Whenever the secret value in a secret changes, Secrets Manager generates a new data key to protect it. The data key is encrypted under an AWS KMS CMK and stored in the metadata of the secret, as shown in the following image. To decrypt the secret, Secrets Manager must first decrypt the encrypted data key using the CMK in AWS KMS.



An AWS KMS CMK for Each Secret

Each secret is associated with an AWS managed or customer managed [customer master key \(p. 2\)](#) (CMK). Customer managed CMKs allow authorized users to [control access \(p. 29\)](#) to the CMK through policies and grants, manage [automatic rotation \(p. 87\)](#), and use [imported key material \(p. 93\)](#).

When you create a new secret, you can specify any customer managed CMK in the account and region, or the AWS managed CMK for Secrets Manager, `aws/secretsmanager`. If you do not specify a CMK, or you select the console default value, **DefaultEncryptionKey**, Secrets Manager creates the `aws/secretsmanager` CMK, if it does not exist, and associates it with the secret. You can use the same CMK or different CMKs for each secret in your account.

You can change the CMK for a secret at any time, either in the Secrets Manager console, or by using the [UpdateSecret](#) operation. When you change the CMK, Secrets Manager does not re-encrypt the existing secret value under the new CMK. However, the next time that the secret value changes, Secrets Manager encrypts it under the new CMK.

To find the CMK that is associated with a secret, use the [ListSecrets](#) or [DescribeSecret](#) operations. When the secret is associated with the AWS managed CMK for Secrets Manager (`aws/secretsmanager`), these operations do not return a CMK identifier.

Secrets Manager does not use the CMK to encrypt the secret value directly. Instead, it uses the CMK to generate and encrypt a unique data key, and it uses the data key to encrypt the secret value.

A Unique Data Key for Each Secret Value

Every time that you create or change the secret value in a secret, Secrets Manager uses the CMK that is associated with the secret to generate and encrypt a unique 256-bit Advanced Encryption Standard (AES) symmetric [data key \(p. 3\)](#). Secrets Manager uses the plaintext data key to encrypt the secret value outside of AWS KMS, and then removes it from memory. It stores the encrypted copy of the data key in the metadata of the secret.

The secret value is ultimately protected by the CMK, which never leaves AWS KMS unencrypted. Before Secrets Manager can decrypt the secret, it must ask AWS KMS to decrypt the encrypted data key.

Encrypting a Secret Value

To encrypt the secret value in a secret, Secrets Manager uses the following process.

1. Secrets Manager calls the AWS KMS [GenerateDataKey](#) operation with the ID of the CMK for the secret and a request for a 256-bit AES symmetric key. AWS KMS returns a plaintext data key and a copy of that data key encrypted under the CMK.
2. Secrets Manager uses the plaintext data key and the Advanced Encryption Standard (AES) algorithm to encrypt the secret value outside of AWS KMS. It removes the plaintext key from memory as soon as possible after using it.
3. Secrets Manager stores the encrypted data key in the metadata of the secret so it is available to decrypt the secret value. However, none of the Secrets Manager APIs return the encrypted secret or the encrypted data key.

Decrypting a Secret Value

To decrypt an encrypted secret value, Secrets Manager must first decrypt the encrypted data key. Because the data key is encrypted under the CMK for the secret in AWS KMS, Secrets Manager must make a request to AWS KMS.

To decrypt an encrypted secret value:

1. Secrets Manager calls the AWS KMS [Decrypt](#) operation and passes in the encrypted data key.
2. AWS KMS uses the CMK for the secret to decrypt the data key. It returns the plaintext data key.
3. Secrets Manager uses the plaintext data key to decrypt the secret value. Then it removes the data key from memory as soon as possible.

Using Your AWS KMS CMK

Secrets Manager uses the [customer master key \(p. 2\)](#) (CMK) that is associated with a secret to generate a data key for each secret value. It also uses the CMK to decrypt that data key when it needs to decrypt the encrypted secret value. You can track the requests and responses in AWS CloudTrail events, [Amazon CloudWatch Logs \(p. 156\)](#), and audit trails.

The following Secrets Manager operations trigger a request to use your AWS KMS CMK.

GenerateDataKey

Secrets Manager calls the AWS KMS [GenerateDataKey](#) operation in response to the following Secrets Manager operations.

- [CreateSecret](#) – If the new secret includes a secret value, Secrets Manager requests a new data key to encrypt it.
- [PutSecretValue](#)– Secrets Manager requests a new data key to encrypt the specified secret value.
- [UpdateSecret](#) – If the update changes the secret value, Secrets Manager requests a new data key to encrypt the new secret value.

Note

The [RotateSecret](#) operation does not call [GenerateDataKey](#), because it does not change the secret value. However, if the Lambda function that [RotateSecret](#) invokes changes the secret value, its call to the [PutSecretValue](#) operation triggers a [GenerateDataKey](#) request.

Decrypt

To decrypt an encrypted secret value, Secrets Manager calls the AWS KMS [Decrypt](#) operation to decrypt the encrypted data key in the secret. Then, it uses the plaintext data key to decrypt the encrypted secret value.

Secrets Manager calls the [Decrypt](#) operation in response to the following Secrets Manager operations.

- [GetSecretValue](#) – Secrets Manager decrypts the secret value before returning it to the caller.
- [PutSecretValue](#) and [UpdateSecret](#) – Most `PutSecretValue` and `UpdateSecret` requests do not trigger a `Decrypt` operation. However, when a `PutSecretValue` or `UpdateSecret` request attempts to change the secret value in an existing version of a secret, Secrets Manager decrypts the existing secret value and compares it to the secret value in the request to confirm that they are the same. This action ensures that Secrets Manager operations are idempotent.

Validating Access to the CMK

When you establish or change the CMK that is associated with secret, Secrets Manager calls the `GenerateDataKey` and `Decrypt` operations with the specified CMK. These calls confirm that the caller has permission to use the CMK for these operation. Secrets Manager discards the results of these operations; it does not use them in any cryptographic operation.

You can identify these validation calls because the value of the `SecretVersionId` key [encryption context](#) (p. 155) in these requests is `RequestToValidateKeyAccess`.

Note

In the past, Secrets Manager validation calls did not include an encryption context. You might find calls with no encryption context in older AWS CloudTrail logs.

Authorizing Use of the CMK

When Secrets Manager uses a [customer master key](#) (p. 2) (CMK) in cryptographic operations, it acts on behalf of the user who is creating or changing the secret value in the secret.

To use the AWS KMS customer master key (CMK) for a secret on your behalf, the user must have the following permissions. You can specify these required permissions in an IAM policy or key policy.

- `kms:GenerateDataKey`
- `kms:Decrypt`

To allow the CMK to be used only for requests that originate in Secrets Manager, you can use the [kms:ViaService condition key](#) (p. 75) with the `secretsmanager.<region>.amazonaws.com` value.

You can also use the keys or values in the [encryption context](#) (p. 155) as a condition for using the CMK for cryptographic operations. For example, you can use a [string condition operator](#) in an IAM or key policy document, or use a [grant constraint](#) in a grant.

Key Policy of the AWS Customer Master Key

The key policy for the AWS managed CMK for Secrets Manager gives users permission to use the CMK for specified operations only when Secrets Manager makes the request on the user's behalf. The key policy does not allow any user to use the CMK directly.

This key policy, like the policies of all [AWS managed keys](#) (p. 2), is established by the service. You cannot change it, but you can view it at any time. To get the key policy for the Secrets Manager CMK in your account, use the [GetKeyPolicy](#) operation.

The policy statements in the key policy have the following effect:

- Allow users in the account to use the CMK for cryptographic operations only when the request comes from Secrets Manager on their behalf. The `kms:ViaService` condition key enforces this restriction.
- Allows the AWS account to create IAM policies that allow users to view CMK properties and revoke grants.
- Although Secrets Manager does not use grants to gain access to the CMK, the policy also allows Secrets Manager to [create grants \(p. 78\)](#) for the CMK on the user's behalf and allows the account to [revoke any grant](#) that allows Secrets Manager to use the CMK. These are standard elements of policy document for an AWS managed CMK.

The following is a key policy for an example AWS managed CMK for Secrets Manager.

```
{
  "Version" : "2012-10-17",
  "Id" : "auto-secretsmanager-1",
  "Statement" : [ {
    "Sid" : "Allow access through AWS Secrets Manager for all principals in the account
    that are authorized to use AWS S
    ecrets Manager",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "*"
    },
    "Action" : [ "kms:Encrypt", "kms:Decrypt", "kms:ReEncrypt*", "kms:GenerateDataKey*",
    "kms:CreateGrant", "kms:Describ
    eKey" ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:ViaService" : "secretsmanager.us-west-2.amazonaws.com",
        "kms:CallerAccount" : "111122223333"
      }
    }
  }, {
    "Sid" : "Allow direct access to key metadata to the account",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : [ "kms:Describe*", "kms:Get*", "kms:List*", "kms:RevokeGrant" ],
    "Resource" : "*"
  } ]
}
```

Secrets Manager Encryption Context

An [encryption context \(p. 6\)](#) is a set of key–value pairs that contain arbitrary nonsecret data. When you include an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

In its [GenerateDataKey](#) and [Decrypt](#) requests to AWS KMS, Secrets Manager uses an encryption context with two name–value pairs that identify the secret and its version, as shown in the following example. The names do not vary, but combined encryption context values will be different for each secret value.

```
"encryptionContext": {
  "SecretARN": "arn:aws:secretsmanager:us-west-2:111122223333:secret:test-secret-a1b2c3",
  "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987SECRET1"
```

```
}
```

You can use the encryption context to identify these cryptographic operation in audit records and logs, such as [AWS CloudTrail](#) and Amazon CloudWatch Logs, and as a condition for authorization in policies and grants.

The Secrets Manager encryption context consists of two name-value pairs.

- **SecretARN** – The first name-value pair identifies the secret. The key is `SecretARN`. The value is the Amazon Resource Name (ARN) of the secret.

```
"SecretARN": "ARN of an Secrets Manager secret"
```

For example, if the ARN of the secret is `arn:aws:secretsmanager:us-west-2:111122223333:secret:test-secret-a1b2c3`, the encryption context would include the following pair.

```
"SecretARN": "arn:aws:secretsmanager:us-west-2:111122223333:secret:test-secret-a1b2c3"
```

- **SecretVersionId** – The second name-value pair identifies the version of the secret. The key is `SecretVersionId`. The value is the version ID.

```
"SecretVersionId": "<version-id>"
```

For example, if the version ID of the secret is `EXAMPLE1-90ab-cdef-fedc-ba987SECRET1`, the encryption context would include the following pair.

```
"SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987SECRET1"
```

When you establish or change the CMK for a secret, Secrets Manager sends [GenerateDataKey](#) and [Decrypt](#) requests to AWS KMS to validate that the caller has permission to use the CMK for these operations. It discards the responses; it does not use them on the secret value.

In these validation requests, the value of the `SecretARN` is the actual ARN of the secret, but the `SecretVersionId` value is `RequestToValidateKeyAccess`, as shown in the following example encryption context. This special value helps you to identify validation requests in logs and audit trails.

```
"encryptionContext": {  
  "SecretARN": "arn:aws:secretsmanager:us-west-2:111122223333:secret:test-secret-a1b2c3",  
  "SecretVersionId": "RequestToValidateKeyAccess"  
}
```

Note

In the past, Secrets Manager validation requests did not include an encryption context. You might find calls with no encryption context in older AWS CloudTrail logs.

Monitoring Secrets Manager Interaction with AWS KMS

You can use AWS CloudTrail and Amazon CloudWatch Logs to track the requests that Secrets Manager sends to AWS KMS on your behalf.

GenerateDataKey

When you [create or change \(p. 153\)](#) the secret value in a secret, Secrets Manager sends a [GenerateDataKey](#) request to AWS KMS that specifies the CMK for the secret.

The event that records the GenerateDataKey operation is similar to the following example event. The request is invoked by `secretsmanager.amazonaws.com`. The parameters include the Amazon Resource Name (ARN) of the CMK for the secret, a key specifier that requires a 256-bit key, and the [encryption context \(p. 155\)](#) that identifies the secret and version.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AROAIQDTESTANDEXAMPLE:user01",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2018-05-31T23:23:41Z"
      }
    }
  },
  "invokedBy": "secretsmanager.amazonaws.com",
  "eventTime": "2018-05-31T23:23:41Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "GenerateDataKey",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "secretsmanager.amazonaws.com",
  "userAgent": "secretsmanager.amazonaws.com",
  "requestParameters": {
    "keyId": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "keySpec": "AES_256",
    "encryptionContext": {
      "SecretARN": "arn:aws:secretsmanager:us-west-2:111122223333:secret:test-secret-a1b2c3",
      "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987SECRET1"
    }
  },
  "responseElements": null,
  "requestID": "a7d4dd6f-6529-11e8-9881-67744a270888",
  "eventID": "af7476b6-62d7-42c2-bc02-5ce86c21ed36",
  "readOnly": true,
  "resources": [
    {
      "ARN": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "accountId": "111122223333",
      "type": "AWS::KMS::Key"
    }
  ],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

Decrypt

Whenever you [get or change \(p. 153\)](#) the secret value of a secret, Secrets Manager sends a [Decrypt](#) request to AWS KMS to decrypt the encrypted data key.

The event that records the `Decrypt` operation is similar to the following example event. The user is the principal in your AWS account who is accessing the table. The parameters include the encrypted table key (as a ciphertext blob) and the [encryption context](#) (p. 155) that identifies the table and the AWS account. AWS KMS derives the ID of the CMK from the ciphertext.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AROAIQDTESTANDEXAMPLE:user01",
    "arn": "arn:aws:sts::111122223333:assumed-role/Admin/user01",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2018-05-31T23:36:09Z"
      }
    }
  },
  "invokedBy": "secretsmanager.amazonaws.com"
},
{
  "eventTime": "2018-05-31T23:36:09Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "Decrypt",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "secretsmanager.amazonaws.com",
  "userAgent": "secretsmanager.amazonaws.com",
  "requestParameters": {
    "encryptionContext": {
      "SecretARN": "arn:aws:secretsmanager:us-west-2:111122223333:secret:test-secret-alb2c3",
      "SecretVersionId": "EXAMPLE1-90ab-cdef-fedc-ba987SECRET1"
    }
  },
  "responseElements": null,
  "requestID": "658c6a08-652b-11e8-a6d4-ffee2046048a",
  "eventID": "f333ec5c-7fc1-46b1-b985-cbda13719611",
  "readOnly": true,
  "resources": [
    {
      "ARN": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "accountId": "111122223333",
      "type": "AWS::KMS::Key"
    }
  ],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

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](#) (p. 159)
- [Amazon SES Encryption Context](#) (p. 159)
- [Giving Amazon SES Permission to Use Your AWS KMS Customer Master Key \(CMK\)](#) (p. 160)
- [Retrieving and Decrypting Email Messages](#) (p. 160)

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 [Step 3](#) (p. 159) through [Step 6](#) (p. 159), 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. 160).

Amazon SES Encryption Context

When Amazon SES requests a data key to encrypt your received email messages ([Step 3](#) (p. 159) in the [Overview of Amazon SES Encryption Using AWS KMS](#) (p. 159)), 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": "d6iitobk75ur44p8kdnp7g2n800"
}
```

For more information about encryption context, go to [Encryption Context \(p. 248\)](#).

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 \(p. 33\)](#):

```
{
  "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 conditions:

- 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 \(p. 159\)](#). For general information about encryption context, go to [Encryption Context \(p. 248\)](#).

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*.
- **AWS SDK for .NET** – See [AmazonS3EncryptionClient](#) in the *AWS SDK for .NET API Reference*.
- **AWS SDK for Go** – See [s3crypto](#) in the *AWS SDK for Go 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 \(p. 159\)](#) in the [Overview of Amazon SES Encryption Using AWS KMS \(p. 159\)](#)). The Amazon S3 encryption client handles this, and much of the other work, for you.

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

- [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 Developer 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 \(p. 161\)](#)
- [Using the Amazon S3 Encryption Client \(p. 162\)](#)
- [Encryption Context \(p. 162\)](#)

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 \(p. 248\)](#).

How AWS Systems Manager Parameter Store Uses AWS KMS

With AWS Systems Manager Parameter Store, you can create [Secure String parameters](#), which are parameters that have a plaintext parameter name and an encrypted parameter value. Parameter Store uses AWS KMS to encrypt and decrypt the parameter values of Secure String parameters.

With [Parameter Store](#) you can create, store, and manage data as parameters with values. You can create a parameter in Parameter Store and use it in multiple applications and services subject to policies and permissions that you design. When you need to change a parameter value, you change one instance, rather than managing an error-prone change to numerous sources. Parameter Store supports a hierarchical structure for parameter names, so you can qualify a parameter for specific uses.

To manage sensitive data, you can create Secure String parameters. Parameter Store uses AWS KMS customer master keys (CMKs) to encrypt the parameter values of Secure String parameters when you create or change them. It also uses CMKs to decrypt the parameter values when you access them. You can use the default CMK that Parameter Store creates for your account or specify your own customer managed CMK.

Topics

- [Encrypting and Decrypting Secure String Parameters](#) (p. 163)
- [Setting Permissions to Encrypt and Decrypt Parameter Values](#) (p. 164)
- [Parameter Store Encryption Context](#) (p. 165)
- [Troubleshooting CMK Issues in Parameter Store](#) (p. 166)

Encrypting and Decrypting Secure String Parameters

Parameter Store does not perform any cryptographic operations. Instead, it relies on AWS KMS to encrypt and decrypt Secure String parameter values. When you create or change a Secure String parameter value, Parameter Store calls the AWS KMS [Encrypt](#) API operation. This operation uses an AWS KMS CMK directly to encrypt the parameter value instead of using the CMK to generate a [data key](#) (p. 3).

You can select the CMK that Parameter Store uses to encrypt the parameter value. If you do not specify a CMK, Parameter Store uses the default `aws/ssm` CMK that Systems Manager automatically creates in your account.

To view the default `aws/ssm` CMK for your account, use the [DescribeKey](#) operation in the AWS KMS API. The following example uses the `describe-key` command in the AWS Command Line Interface (AWS CLI) with the `aws/ssm` alias name.

```
aws kms describe-key --key-id alias/aws/ssm
```

To create a Secure String parameter, use the [PutParameter](#) operation in the Systems Manager API. Include a `Type` parameter with a value of `SecureString`. To specify an AWS KMS CMK, use the `KeyId` parameter. The default is the default `aws/ssm` CMK for your account.

Parameter Store then calls the AWS KMS [Encrypt](#) API with the CMK and the plaintext parameter value. AWS KMS returns the encrypted parameter value, which Parameter Store stores with the parameter name.

The following example uses the Systems Manager [put-parameter](#) command and its `--type` parameter in the AWS CLI to create a Secure String parameter. Because the command omits the optional `--key-id` parameter, Parameter Store uses the default `aws/ssm` CMK to encrypt the parameter value.

```
aws ssm put-parameter --name MyParameter --value "secret_value" --type SecureString
```

The following similar example uses the `--key-id` parameter to specify a customer managed CMK. The example uses a CMK ID to identify the CMK, but you can use any valid CMK identifier.

```
aws ssm put-parameter --name param1 --value "secret" --type SecureString --key-id  
1234abcd-12ab-34cd-56ef-1234567890ab
```

When you get a Secure String parameter from Parameter Store, its value is encrypted. To get a parameter, use the [GetParameter](#) operation in the Systems Manager API.

The following example uses the Systems Manager [get-parameter](#) command in the AWS CLI to get the `MyParameter` parameter from Parameter Store without decrypting its value.

```
$ aws ssm get-parameter --name MyParameter

{
  "Parameter": {
    "Type": "SecureString",
    "Name": "MyParameter",
    "Value":
    "AQECAHgnOkMROh5LaLXkA4j0+vYi6tmM17Lg/9E464VRo68cvwAAAG8wbQYJKoZIhvcNAQcGoGAWXgIBADBZBgkqhkiG9w0BBwEwH
  }
}
```

To decrypt the parameter value before returning it, set the `WithDecryption` parameter of `GetParameter` to `true`. When you use `WithDecryption`, Parameter Store calls the AWS KMS [Decrypt](#) API operation on your behalf to decrypt the parameter value. As a result, the `GetParameter` request returns the parameter with a plaintext parameter value, as shown in the following example.

```
$ aws ssm get-parameter --name MyParameter --with-decryption

{
  "Parameter": {
    "Type": "SecureString",
    "Name": "MyParameter",
    "Value": "secret_value"
  }
}
```

The following workflow shows how Parameter Store uses an AWS KMS CMK.

Encrypt in Put Parameter:

1. When you create a Secure String parameter, Parameter Store sends an `Encrypt` request to AWS KMS that includes the plaintext parameter value and the CMK that you chose. During transmission to AWS KMS, the plaintext value in the Secure String parameter is protected by Transport Layer Security (TLS).
2. AWS KMS encrypts the parameter value with the specified CMK and returns the ciphertext to Parameter Store, which stores the parameter name and its encrypted value.

Decrypt in Get Parameter

1. When you include the `WithDecryption` parameter in a request to get a Secure String parameter, Parameter Store sends a `Decrypt` request to AWS KMS with the encrypted Secure String parameter value.
2. AWS KMS uses the same CMK to decrypt the encrypted value. It returns the plaintext (decrypted) parameter value to Parameter Store. During transmission, the plaintext data is protected by TLS.
3. Parameter Store returns the plaintext parameter value to you in the `GetParameter` response.

Setting Permissions to Encrypt and Decrypt Parameter Values

You can use IAM policies to allow or deny permission for a user to call the Systems Manager `PutParameter` and `GetParameter` operations.

Also, if you are using customer managed CMKs, you can use IAM policies and key policies to manage encrypt and decrypt permissions. However, you cannot establish access control policies for the default `aws/ssm` CMK. For detailed information about controlling access to customer managed CMKs, see [Authentication and Access Control for AWS KMS \(p. 29\)](#).

The following example shows an IAM policy that allows the user to call the Systems Manager `GetParameter` operation on all parameters in the `/ReadableParameters` path. The policy also allows the user to call the AWS KMS `Decrypt` operation on the specified customer managed CMK.

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

Parameter Store Encryption Context

An *encryption context* is a set of key-value pairs that contain arbitrary nonsecret data. When you include an encryption context in a request to encrypt data, AWS KMS cryptographically binds the encryption context to the encrypted data. To decrypt the data, you must pass in the same encryption context.

You can also use the encryption context to identify a cryptographic operation in audit records and logs. The encryption context appears in plaintext in logs, such as [AWS CloudTrail](#) logs.

Parameter Store uses the following encryption context in its cryptographic operations:

- Key: `PARAMETER_ARN`
- Value: The Amazon Resource Name (ARN) of the parameter that is being encrypted.

The format of the encryption context is as follows:

```
"PARAMETER_ARN": "arn:aws:ssm:<REGION_NAME>:<ACCOUNT_ID>:parameter/<parameter-name>"
```

For example, Parameter Store includes this encryption context in calls to encrypt and decrypt the `MyParameter` parameter in an example AWS account and region.

```
"PARAMETER_ARN": "arn:aws:ssm:us-west-2:111122223333:parameter/MyParameter"
```

If the parameter is in a Parameter Store hierarchical path, the path and name are included in the encryption context. For example, this encryption context is used when encrypting and decrypting the `MyParameter` parameter in the `/ReadableParameters` path in an example AWS account and region.

```
"PARAMETER_ARN": "arn:aws:ssm:us-west-2:111122223333:parameter/ReadableParameters/MyParameter"
```

You can decrypt an encrypted Secure String parameter value by calling the AWS KMS `Decrypt` operation with the correct encryption context and the encrypted parameter value that the Systems Manager `GetParameter` operation returns. However, we encourage you to use the `GetParameter` operation with the `WithDecryption` parameter to decrypt Parameter Store parameter values.

You can also include the encryption context in an IAM policy. For example, you can permit a user to decrypt only one particular parameter value or set of parameter values.

The following example IAM policy statement allows the user to the get value of the `MyParameter` parameter and to decrypt its value using the specified CMK. However the permissions apply only when the encryption context matches specified string. These permissions do not apply to any other parameter or CMK, and the call to `GetParameter` fails if the encryption context does not match the string.

Before using a policy statement like this one, replace the example AWS account, region, and parameter name with valid values.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameter*"
      ],
      "Resource": "arn:aws:ssm:us-west-2:111122223333:parameter/MyParameter",
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "Condition": {
        "StringEquals": {
          "kms:EncryptionContext:PARAMETER_ARN": "arn:aws:ssm:us-west-2:111122223333:parameter/MyParameter"
        }
      }
    }
  ]
}
```

Troubleshooting CMK Issues in Parameter Store

To perform any operation on a Secure String parameter, Parameter Store must be able to use the AWS KMS CMK that you specify for your intended operation. Most of the Parameter Store failures related to CMKs are caused by the following problems:

- The credentials that an application is using do not have permission to perform the specified action on the CMK.

To fix this error, run the application with different credentials or revise the IAM or key policy that is preventing the operation. For help with AWS KMS IAM and key policies, see [Authentication and Access Control for AWS KMS \(p. 29\)](#).

- The CMK is not found.

This typically happens when you use an incorrect identifier for the CMK. [Find the correct identifiers \(p. 16\)](#) for the CMK and try the command again.

- The CMK is not enabled. When this occurs, Parameter Store returns an `InvalidKeyId` exception with a detailed error message from AWS KMS. If the CMK state is `Disabled`, [enable it \(p. 26\)](#). If it is `Pending Import`, complete the [import procedure \(p. 93\)](#). If the key state is `Pending Deletion`, [cancel the key deletion \(p. 111\)](#) or use a different CMK.

To find the [key state \(p. 123\)](#) of a CMK in the AWS KMS console, on the **Customer managed keys** or **AWS managed keys** page, see the [Status column \(p. 11\)](#). To find the status of a CMK using the AWS KMS API, use the [DescribeKey](#) operation.

How Amazon WorkMail Uses AWS KMS

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

Topics

- [Amazon WorkMail Overview \(p. 167\)](#)
- [Amazon WorkMail Encryption \(p. 167\)](#)
- [Amazon WorkMail Encryption Context \(p. 169\)](#)

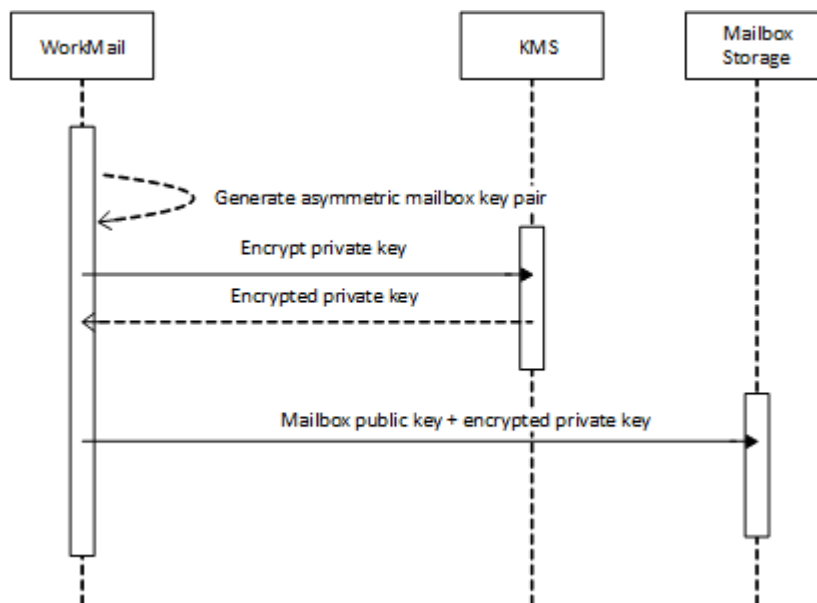
Amazon WorkMail Overview

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.

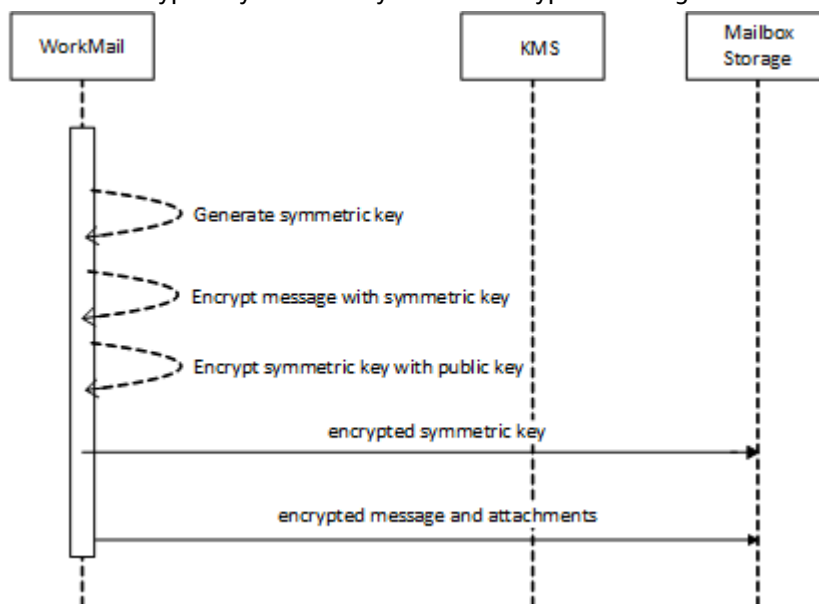
Using the Amazon WorkMail console, you can create an Amazon WorkMail organization and optionally assign to it one or more email domains that you own. Then you can create new email users and email distribution groups. Users can then send and receive messages. The messages are encrypted and stored until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

Amazon WorkMail Encryption

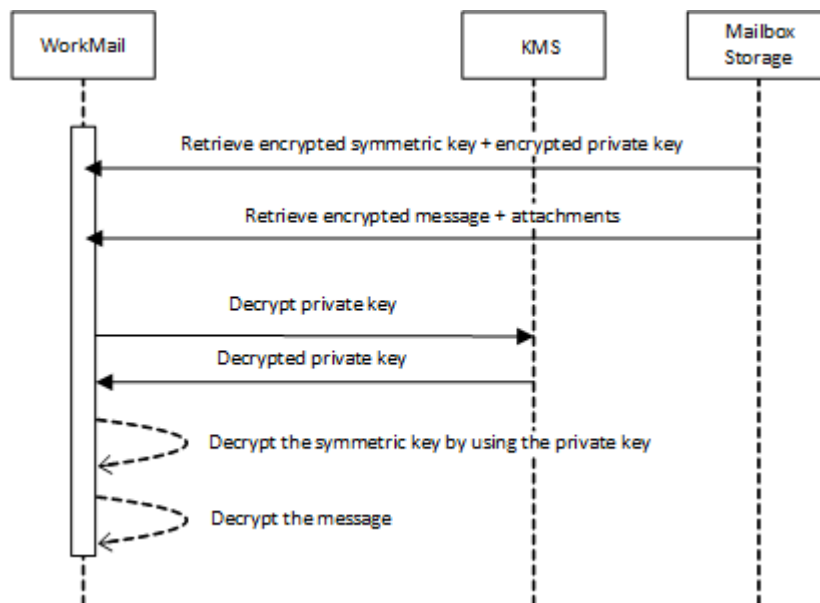
Each end user you create is associated with one mailbox. Amazon WorkMail creates an asymmetric key pair for each mailbox and sends the private key portion of the key pair to AWS KMS to be encrypted under a customer master key (CMK). The CMK can be a custom key that you choose for your organization or 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 the ARN of an organization in the `us-east-2` region.

```
arn:aws:workmail:us-east-2:111122223333:organization/m-68755160c4cb4e29a2b2f8fb58f359d7
```

For more information about the encryption context, see [Encryption Context \(p. 248\)](#).

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 the AWS managed CMK for Amazon WorkSpaces (**aws/workspaces**) or a customer managed CMK.

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 \(p. 170\)](#)
- [Amazon WorkSpaces Encryption Context \(p. 170\)](#)
- [Giving Amazon WorkSpaces Permission to Use A CMK On Your Behalf \(p. 171\)](#)

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 \(p. 139\)](#) 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 \(p. 78\)](#) 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 \(p. 170\)](#)). This action creates a [grant \(p. 78\)](#) 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 \(p. 170\)](#)) 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 \(p. 170\)](#) in the [Overview of Amazon WorkSpaces Encryption Using AWS KMS \(p. 170\)](#)) and when it requests a plaintext copy

of that data key ([Step 5 \(p. 170\)](#)), 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 \(p. 248\)](#).

Giving Amazon WorkSpaces Permission to Use A CMK On Your Behalf

You can protect your workspace data under the AWS managed CMK for Amazon WorkSpaces (**aws/workspaces**) or a customer managed CMK. If you use a customer managed CMK, you need to give Amazon WorkSpaces permission to use the CMK on behalf of the Amazon WorkSpaces administrators in your account. The AWS managed CMK for Amazon WorkSpaces has the required permissions by default.

To prepare your customer managed CMK for use with Amazon WorkSpaces, use the following procedure.

1. [Add the WorkSpaces administrators to the list of key users in the CMK's key policy \(p. 171\)](#)
2. [Give the WorkSpaces administrators additional permissions with an IAM policy \(p. 172\)](#)

Amazon 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 give Amazon WorkSpaces administrators the permissions that they require, you can use the AWS Management Console or the AWS KMS API.

Topics

- [To add WorkSpaces administrators as key users for a CMK \(Console\) \(p. 171\)](#)
- [To add WorkSpaces administrators as key users for a CMK \(KMS API\) \(p. 172\)](#)

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

1. Sign in to the AWS Management Console and open the AWS Key Management Service (AWS KMS) console at <https://console.aws.amazon.com/kms>.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. In the navigation pane, choose **Customer managed keys**.
4. Choose the key ID or alias of your preferred customer managed CMK.

5. In the **Key policy** section, under **Key users**, choose **Add**.
6. 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 (KMS API)

1. Use the [GetKeyPolicy](#) operation to get 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. 37). Then save the file.
3. Use the [PutKeyPolicy](#) operation to apply the key policy to the CMK.

Part 2: Giving WorkSpaces Administrators Extra Permissions

If you are using a customer managed CMK to protect your Amazon WorkSpaces data, in addition to the permissions in the key users section of the [default key policy](#) (p. 34), WorkSpaces administrators need permission to create [grants](#) (p. 78) on the CMK. Also, if they use the [AWS Management Console](#) to create WorkSpaces with encrypted volumes, WorkSpaces administrators need permission to list aliases and list keys. 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*.

To give these permissions to your WorkSpaces administrators, use an IAM policy. Add an policy statement similar to the following example to the IAM policy for each WorkSpaces administrator. Replace the example CMK ARN ([arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab](#)) with a valid one. If your WorkSpaces administrators use only the Amazon WorkSpaces API (not the console), you can omit the second policy 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. 173) 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 change 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. 93) 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 Amazon EC2 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 \(p. 174\)](#).

- **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 \(p. 177\)](#) 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.

You can [customize \(p. 11\)](#) the [AWS KMS console dashboard](#) to display 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 \(p. 175\)](#)
- [Creating CloudWatch Alarms to Monitor AWS KMS Metrics \(p. 176\)](#)
- [AWS KMS Events \(p. 177\)](#)

AWS KMS Metrics and Dimensions

When you [import key material into a CMK \(p. 93\)](#) 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 \(p. 176\)](#).

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 \(p. 176\)](#)
- [Create a CloudWatch Alarm to Monitor Usage of CMKs that are Pending Deletion \(p. 177\)](#)

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

When you [import key material into a CMK \(p. 93\)](#), 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. 110) 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. 117).

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 Data 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](#).

The following topics describe the CloudWatch Events that AWS KMS creates.

Topics

- [KMS CMK Rotation \(p. 178\)](#)
- [KMS Imported Key Material Expiration \(p. 178\)](#)
- [KMS CMK Deletion \(p. 179\)](#)

KMS CMK Rotation

When you enable [annual rotation of a CMK's key material \(p. 87\)](#), 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 \(p. 93\)](#), 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. 110) 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 with AWS CloudTrail

AWS KMS is integrated with AWS CloudTrail, a service that provides a record of actions performed by a user, role, or an AWS service in AWS KMS. CloudTrail captures all API calls for AWS KMS as events, including calls from the AWS KMS console and from code calls to the AWS KMS APIs. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for AWS KMS. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to AWS KMS, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#). To learn about other ways to monitor the use of your CMKs, see [Monitoring Customer Master Keys \(p. 173\)](#).

AWS KMS Information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS KMS, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your AWS account, including events for AWS KMS, create a trail. A trail enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all regions. The trail logs events from all regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#)

CloudTrail logs all AWS KMS operations, including read-only operations, such as `ListAliases` and `GetKeyPolicy`, operations that manage CMKs, such as `CreateKey` and `PutKeyPolicy`, and cryptographic operations, such as `GenerateDataKey`, `Encrypt`, and `Decrypt`. Every operation generates an entry in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the [CloudTrail `userIdentity` Element](#).

Understanding AWS KMS Log File Entries

A *trail* is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An *event* represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

For examples of what these CloudTrail log entries look like, see the following topics.

Topics

- [CreateAlias \(p. 181\)](#)
- [CreateGrant \(p. 182\)](#)
- [CreateKey \(p. 183\)](#)
- [Decrypt \(p. 184\)](#)
- [DeleteAlias \(p. 184\)](#)
- [DescribeKey \(p. 185\)](#)
- [DisableKey \(p. 187\)](#)
- [EnableKey \(p. 187\)](#)
- [Encrypt \(p. 188\)](#)
- [GenerateDataKey \(p. 189\)](#)
- [GenerateDataKeyWithoutPlaintext \(p. 189\)](#)
- [GenerateRandom \(p. 190\)](#)
- [GetKeyPolicy \(p. 190\)](#)
- [ListAliases \(p. 191\)](#)
- [ListGrants \(p. 192\)](#)
- [ReEncrypt \(p. 192\)](#)
- [Amazon EC2 Example One \(p. 193\)](#)
- [Amazon EC2 Example Two \(p. 195\)](#)

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-
bdfd-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-
bdfd-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    {\n      \"Effect\": \"Allow\",\n      \"Principal\": {\n        \"AWS\": \"arn:aws:iam::123456789012:user/Alice\"\n      },\n      \"Action\": \"kms:*\",\n      \"Resource\": \"*\"\n    },\n    {\n      \"Effect\": \"Allow\",\n      \"Principal\": {\n        \"AWS\": \"arn:aws:iam::012345678901:user/Bob\"\n      },\n      \"Action\": \"kms:CreateGrant\",\n      \"Resource\": \"*\"\n    },\n    {\n      \"Effect\": \"Allow\",\n      \"Principal\": {\n        \"AWS\": \"arn:aws:iam::012345678901:user/Charlie\"\n      },\n      \"Action\": \"kms:Encrypt\",\n      \"Resource\": \"*\"\n    }\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 that records multiple calls to [DescribeKey](#). These calls were the result of [viewing keys \(p. 11\)](#) in the AWS KMS 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": "939fcebfb-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":
"eyJiIjoiaWxpcXhMvZTU0Y2MxOTMtYTMwNC00YzEwLTliZWItYTJjZjA3NjA2OTJhIiwiaWYsI6ImFsaWZlL2U1NGNjMTkzLWEzMDQtNj"
      },

```



```

"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"
  }
]
```

Connecting to AWS KMS Through a VPC Endpoint

You can connect directly to AWS KMS through a private endpoint in your VPC instead of connecting over the internet. When you use a VPC endpoint, communication between your VPC and AWS KMS is conducted entirely within the AWS network.

AWS KMS supports [Amazon Virtual Private Cloud](#) (Amazon VPC) [interface endpoints](#) that are powered by [AWS PrivateLink](#). Each VPC endpoint is represented by one or more [Elastic Network Interfaces](#) (ENIs) with private IP addresses in your VPC subnets.

The VPC interface endpoint connects your VPC directly to AWS KMS without an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection. The instances in your VPC do not need public IP addresses to communicate with AWS KMS.

You can specify the VPC endpoint in [AWS KMS API operations](#) and [AWS CLI commands](#). For example, the following command uses the **endpoint-url** parameter to specify a VPC endpoint in an AWS CLI command to AWS KMS.

```
$ aws kms list-keys --endpoint-url https://vpce-0295a3caf8414c94a-dfm9tr04.kms.us-east-1.vpce.amazonaws.com
```

If you use the default domain name servers (**AmazonProvidedDNS**) and enable [private DNS hostnames](#) for your VPC endpoint, you do not need to specify the endpoint URL. AWS populates your VPC name server with private zone data, so the public KMS endpoint (`https://kms.<region>.amazonaws.com`) resolves to your private VPC endpoint. To enable this feature when using your own name servers, forward requests for the KMS domain to the VPC name server.

You can also use AWS CloudTrail logs to audit your use of KMS keys through the VPC endpoint. And you can use the conditions in IAM and key policies to deny access to any request that does not come from a specified VPC or VPC endpoint.

Note

Use caution when creating IAM and key policies based on your VPC endpoint. If a policy statement requires that requests come from a particular VPC or VPC endpoint, requests from integrated AWS services that use the CMK on your behalf might fail. For help, see [Using VPC Endpoint Conditions in Policies with AWS KMS Permissions](#) (p. 61).

Supported AWS Regions

AWS KMS supports VPC endpoints in all AWS regions where both [Amazon VPC](#) and [AWS KMS](#) are available, except for AWS GovCloud (US).

Topics

- [Create an AWS KMS VPC Endpoint](#) (p. 201)
- [Connecting to an AWS KMS VPC Endpoint](#) (p. 203)
- [Using a VPC Endpoint in a Policy Statement](#) (p. 204)
- [Audit the CMK Use for your VPC](#) (p. 206)

Create an AWS KMS VPC Endpoint

You [create an interface endpoint](#) in your VPC by using the KMS VPC endpoint service in each region. You can create a VPC endpoint in the AWS Management Console, or by using the [AWS CLI](#) or [Amazon EC2 API](#).

Topics

- [Creating an AWS KMS VPC Endpoint \(VPC Console\)](#) (p. 201)
- [Creating an AWS KMS VPC Endpoint \(AWS CLI\)](#) (p. 202)

Creating an AWS KMS VPC Endpoint (VPC Console)

1. Sign in to the AWS Management Console and open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. On the navigation bar, use the region selector to choose your region.
3. In the navigation pane, choose **Endpoints**. In the main pane, **Create Endpoint**.
4. For **Service category**, choose **AWS services**.
5. In the **Service Name** list, choose the entry for AWS KMS interface endpoint in the region. For example, in the US East (N.Virginia) Region, the entry name is `com.amazonaws.us-east-1.kms`.
6. For **VPC**, select a VPC. The endpoint is created in the VPC that you select.
7. For **Subnets**, choose a subnet from each Availability Zone that you want to include.

The VPC endpoint can span multiple Availability Zones. An elastic network interface (ENI) for the VPC endpoint is created in each subnet that you choose. Each ENI has a DNS hostname and a private IP address.

8. In this step, you can enable a private DNS hostname for your VPC endpoint. If you select the **Enable Private DNS Name** option, the standard AWS KMS DNS hostname (`https://kms.<region>.amazonaws.com`) resolves to your VPC endpoint.

This option makes it easier to use the VPC endpoint. The AWS KMS CLI and SDKs use the standard AWS KMS DNS hostname by default, so you do not need to specify the VPC endpoint URL in applications and commands.

This feature works only when the `enableDnsHostnames` and `enableDnsSupport` attributes of your VPC are set to `true`. To set these attributes, [update DNS support for your VPC](#).

To enable a private DNS hostname, for **Enable Private DNS Name**, select **Enable for this endpoint**.

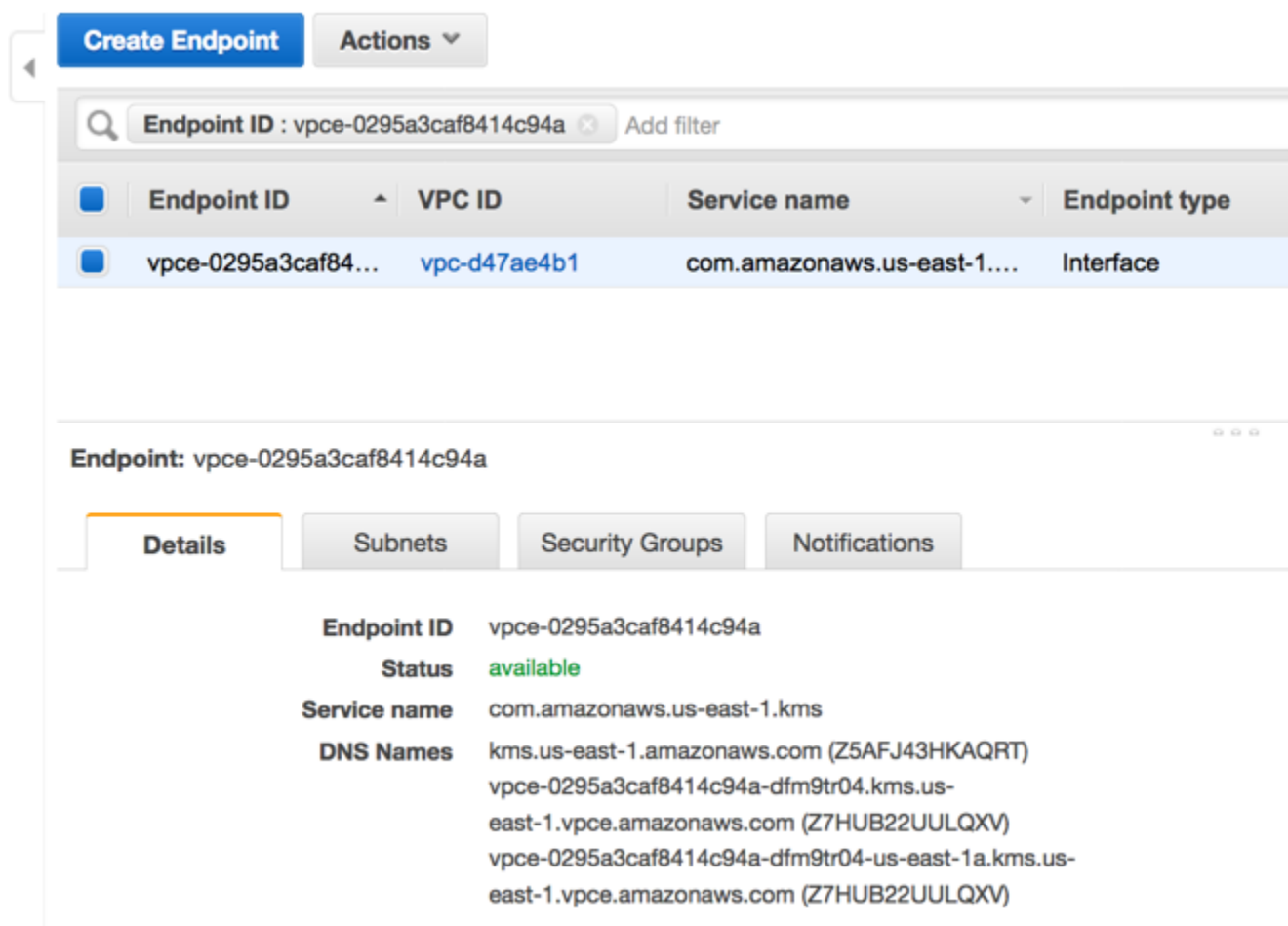
9. For **Security group**, select or create a security group.

You can use [security groups](#) to control access to your endpoint, much like you would use a firewall.

10. Choose **Create endpoint**.

The results show the VPC endpoint, including the VPC endpoint ID and the DNS names that you use to [connect to your VPC endpoint](#) (p. 203).

You can also use the Amazon VPC tools to view and manage your endpoint, including creating a notification for an endpoint, changing properties of the endpoint, and deleting the endpoint. For instructions, see [Interface VPC Endpoints](#).



Creating an AWS KMS VPC Endpoint (AWS CLI)

You can use the `create-vpc-endpoint` command in the AWS CLI to create a VPC endpoint that connects to AWS KMS.

Be sure to use `interface` as the VPC endpoint type and a service name value that includes `kms` and the region where your VPC is located.

The command does not include the `PrivateDnsNames` parameter because its default value is `true`. To disable this option, you can include the parameter with a value of `false`. Private DNS names are available only when the `enableDnsHostnames` and `enableDnsSupport` attributes of your VPC are set to `true`. To set these attributes, use the [ModifyVpcAttribute](#) API.

The following diagram shows the syntax of the command.

```
aws ec2 create-vpc-endpoint --vpc-id <vpc id> \
  --vpc-endpoint-type Interface \
  --service-name com.amazonaws.<region>.kms \
  --subnet-ids <subnet id> \
  --security-group-id <security group id>
```

For example, this command creates a VPC endpoint in the VPC with VPC ID `vpc-1a2b3c4d`, which is in the `us-east-1` region. It specifies just one subnet ID to represent the Availability Zones, but you can specify many. The security group ID is also required.

The output includes the VPC endpoint ID and DNS names that you use to connect to your new VPC endpoint.

```
$ aws ec2 create-vpc-endpoint --vpc-id vpc-1a2b3c4d \
                             --vpc-endpoint-type Interface \
                             --service-name com.amazonaws.us-west-1.kms \
                             --subnet-ids subnet-a6b10bd1 \
                             --security-group-id sg-1a2b3c4d

{
  "VpcEndpoint": {
    "PolicyDocument": "{\n  \"Statement\": [\n    {\n      \"Action\": \"*\",\n      \"Effect\": \"Allow\", \n      \"Principal\": \"*\", \n      \"Resource\": \"*\"\n    }\n  ]\n}",
    "VpcId": "vpc-1a2b3c4d",
    "NetworkInterfaceIds": [
      "eni-bf8aa46b"
    ],
    "SubnetIds": [
      "subnet-a6b10bd1"
    ],
    "PrivateDnsEnabled": true,
    "State": "pending",
    "ServiceName": "com.amazonaws.us-east-1.kms",
    "RouteTableIds": [],
    "Groups": [
      {
        "GroupName": "default",
        "GroupId": "sg-1a2b3c4d"
      }
    ],
    "VpcEndpointId": "vpce-0295a3caf8414c94a",
    "VpcEndpointType": "Interface",
    "CreationTimestamp": "2017-09-05T20:14:41.240Z",
    "DnsEntries": [
      {
        "HostedZoneId": "Z7HUB22UULQXV",
        "DnsName": "vpce-0295a3caf8414c94a-dfm9tr04.kms.us-east-1.vpce.amazonaws.com"
      },
      {
        "HostedZoneId": "Z7HUB22UULQXV",
        "DnsName": "vpce-0295a3caf8414c94a-dfm9tr04-us-east-1a.kms.us-east-1.vpce.amazonaws.com"
      },
      {
        "HostedZoneId": "Z1K56Z6FNPJRR",
        "DnsName": "kms.us-east-1.amazonaws.com"
      }
    ]
  }
}
```

Connecting to an AWS KMS VPC Endpoint

You can connect to AWS KMS through the VPC endpoint by using the AWS CLI or an AWS SDK. To specify the VPC endpoint, use its DNS name.

For example, this [list-keys](#) command uses the `endpoint-url` parameter to specify the VPC endpoint. To use a command like this, replace the example VPC endpoint ID with one in your account.

```
aws kms list-keys --endpoint-url https://vpce-0295a3caf8414c94a-dfm9tr04.kms.us-east-1.vpce.amazonaws.com
```

If you enabled private hostnames when you created your VPC endpoint, you do not need to specify the VPC endpoint URL in your CLI commands or application configuration. The standard AWS KMS DNS hostname (<https://kms.<region>.amazonaws.com>) resolves to your VPC endpoint. The AWS CLI and SDKs use this hostname by default, so you can begin using the VPC endpoint without changing anything in your scripts and application.

To use private hostnames, the `enableDnsHostnames` and `enableDnsSupport` attributes of your VPC must be set to true. To set these attributes, use the [ModifyVpcAttribute](#) API.

Using a VPC Endpoint in a Policy Statement

You can use IAM policies and AWS KMS key policies to control access to your AWS KMS customer master keys (CMKs). You can also use [global condition keys](#) to restrict these policies based on VPC endpoint or VPC in the request.

- Use the `aws:sourceVpce` condition key to grant or restrict access to an AWS KMS CMK based on the VPC endpoint.
- Use the `aws:sourceVpc` condition key to grant or restrict access to an AWS KMS CMK based on the VPC that hosts the private endpoint.

Note

Use caution when creating IAM and key policies based on your VPC endpoint. If a policy statement requires that requests come from a particular VPC or VPC endpoint, requests from integrated AWS services that use the CMK on your behalf might fail. For help, see [Using VPC Endpoint Conditions in Policies with AWS KMS Permissions \(p. 61\)](#).

Also, the `aws:sourceIP` condition key is not effective when the request comes from an [Amazon VPC endpoint](#). To restrict requests to a VPC endpoint, use the `aws:sourceVpce` or `aws:sourceVpc` condition keys. For more information, see [VPC Endpoints - Controlling the Use of Endpoints](#) in the *Amazon VPC User Guide*.

For example, the following sample key policy allows a user to perform encryption operations with a CMK only when the request comes through the specified VPC endpoint.

When a user makes a request to AWS KMS, the VPC endpoint ID in the request is compared to the `aws:sourceVpce` condition key value in the policy. If they do not match, then the request is denied.

To use a policy like this one, replace the placeholder AWS account ID and VPC endpoint IDs with valid values for your account.

```
{
  "Id": "example-key-1",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Enable IAM user permissions",
      "Effect": "Allow",
      "Principal": {"AWS":["111122223333"]},
      "Action": ["kms:*"],
      "Resource": "*"
    },
    {
      "Sid": "Restrict usage to my VPC endpoint",
      "Effect": "Deny",
```

```
    "Principal": "*",
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt",
      "kms:ReEncrypt*",
      "kms:GenerateDataKey*"
    ],
    "Resource": "*",
    "Condition": {
      "StringNotEquals": {
        "aws:sourceVpc": "vpce-0295a3caf8414c94a"
      }
    }
  }
}
```

You can also use the `aws:sourceVpc` condition key to restrict access to your CMKs based on the VPC in which VPC endpoint resides.

The following sample key policy allows commands that manage the CMK only when they come from `vpc-12345678`. In addition, it allows commands that use the CMK for cryptographic operations only when they come from `vpc-2b2b2b2b`. You might use a policy like this one if an application is running in one VPC, but you use a second, isolated VPC for management functions.

To use a policy like this one, replace the placeholder AWS account ID and VPC endpoint IDs with valid values for your account.

```
{
  "Id": "example-key-2",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow administrative actions from vpc-12345678",
      "Effect": "Allow",
      "Principal": {"AWS": "111122223333"},
      "Action": [
        "kms:Create*", "kms:Enable*", "kms:Put*", "kms:Update*",
        "kms:Revoke*", "kms:Disable*", "kms:Delete*",
        "kms:TagResource", "kms:UntagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:sourceVpc": "vpc-12345678"
        }
      }
    },
    {
      "Sid": "Allow key usage from vpc-2b2b2b2b",
      "Effect": "Allow",
      "Principal": {"AWS": "111122223333"},
      "Action": [
        "kms:Encrypt", "kms:Decrypt", "kms:GenerateDataKey*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:sourceVpc": "vpc-2b2b2b2b"
        }
      }
    }
  ]
}
```

```
        "Sid": "Allow read actions from everywhere",
        "Effect": "Allow",
        "Principal": {"AWS": "111122223333"},
        "Action": [
            "kms:Describe*", "kms:List*", "kms:Get*"
        ],
        "Resource": "*",
    }
}
```

Audit the CMK Use for your VPC

When a request to AWS KMS uses a VPC endpoint, the VPC endpoint ID appears in the [AWS CloudTrail log \(p. 180\)](#) entry that records the request. You can use the endpoint ID to audit the use of your AWS KMS VPC endpoint.

For example, this sample log entry records a `GenerateDataKey` request that used the VPC endpoint. The `vpcEndpointId` field appears at the end of the log entry.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "EX_PRINCIPAL_ID",
    "arn": "arn:aws:iam::111122223333:user/Alice",
    "accessKeyId": "EXAMPLE_KEY_ID",
    "accountId": "111122223333",
    "userName": "Alice"
  },
  "eventTime": "2018-01-16T05:46:57Z",
  "eventSource": "kms.amazonaws.com",
  "eventName": "GenerateDataKey",
  "awsRegion": "eu-west-1",
  "sourceIPAddress": "172.01.01.001",
  "userAgent": "aws-cli/1.14.23 Python/2.7.12 Linux/4.9.75-25.55.amzn1.x86_64
botocore/1.8.27",
  "requestParameters": {
    "keyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "numberOfBytes": 128
  },
  "responseElements": null,
  "requestID": "a9fff0bf-fa80-11e7-a13c-afcabff2f04c",
  "eventID": "77274901-88bc-4e3f-9bb6-acf1c16f6a7c",
  "readOnly": true,
  "resources": [{
    "ARN": "arn:aws:kms:eu-west-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "accountId": "111122223333",
    "type": "AWS::KMS::Key"
  }],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333",
  "vpcEndpointId": "vpce-0295a3caf8414c94a"
}
```

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.

The sample code in the following topics show how to use the AWS SDKs to call the AWS KMS API.

Topics

- [Creating a Client \(p. 207\)](#)
- [Working With Keys \(p. 208\)](#)
- [Encrypting and Decrypting Data Keys \(p. 217\)](#)
- [Working with Key Policies \(p. 223\)](#)
- [Working with Grants \(p. 230\)](#)
- [Working with Aliases \(p. 237\)](#)

Creating a Client

To use the [AWS SDK for Java](#), the [AWS SDK for .NET](#), the [AWS SDK for Python \(Boto 3\)](#), the [AWS SDK for Ruby](#), the [AWS SDK for PHP](#), or the [AWS SDK for JavaScript in Node.js](#) to write code that uses the [AWS Key Management Service \(AWS KMS\) API](#), start by creating an AWS KMS client.

The client object that you create is used in the example code in the topics that follow.

Java

To create an AWS KMS client in Java, use the client builder.

```
AWSKMS kmsClient = AWSKMSClientBuilder.defaultClient();
```

For more information about using the Java client builder, see the following resources.

- [Fluent Client Builders](#) on the AWS Developer Blog
- [Creating Service Clients](#) in the *AWS SDK for Java Developer Guide*
- [AWSKMSClientBuilder](#) in the *AWS SDK for Java API Reference*

C#

```
AmazonKeyManagementServiceClient kmsClient = new AmazonKeyManagementServiceClient();
```

Python

```
kms_client = boto3.client('kms')
```

Ruby

```
require 'aws-sdk-kms' # in v2: require 'aws-sdk'

kmsClient = Aws::KMS::Client.new
```

PHP

To create an AWS KMS client in PHP, use an AWS KMS client object, and specify version 2014-11-01. For more information see the [KMSSClient class](#) in the AWS SDK for PHP API Reference.

```
// Create a KMSSClient
$KmsClient = new Aws\Kms\KmsClient([
    'profile' => 'default',
    'version' => '2014-11-01',
    'region'  => 'us-east-1'
]);
```

Node.js

```
const kmsClient = new AWS.KMS();
```

Working With Keys

The examples in this topic use the AWS KMS API to create, view, enable, and disable AWS KMS customer master keys, and to generate data keys.

Topics

- [Creating a Customer Master Key \(p. 208\)](#)
- [Generating a Data Key \(p. 210\)](#)
- [Viewing a Custom Master Key \(p. 212\)](#)
- [Getting Key IDs and Key ARNs of Customer Master Keys \(p. 213\)](#)
- [Enabling Customer Master Keys \(p. 214\)](#)
- [Disabling Customer Master Keys \(p. 216\)](#)

Creating a Customer Master Key

To create a [customer master key \(p. 2\)](#), use the [CreateKey](#) operation.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details, see the [createKey method](#) in the *AWS SDK for Java API Reference*.

```
// Create a CMK
//
String desc = "Key for protecting critical data";
```

```
CreateKeyRequest req = new CreateKeyRequest().withDescription(desc);  
CreateKeyResult result = kmsClient.createKey(req);
```

C#

For details, see the [CreateKey method](#) in the *AWS SDK for .NET*.

```
// Create a CMK  
//  
String desc = "Key for protecting critical data";  
  
CreateKeyRequest req = new CreateKeyRequest()  
{  
    Description = desc  
};  
CreateKeyResponse response = kmsClient.CreateKey(req);
```

Python

For details, see the [create_key method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Create a CMK  
  
desc = 'Key for protecting critical data'  
  
response = kms_client.create_key(  
    Description=desc  
)
```

Ruby

For details, see the [create_key](#) instance method in the *AWS SDK for Ruby*.

```
# Create a CMK  
  
desc = 'Key for protecting critical data'  
  
response = kmsClient.create_key({  
    description: desc  
})
```

PHP

For details, see the [CreateKey method](#) in the *AWS SDK for PHP*.

```
// Create a CMK  
//  
$desc = "Key for protecting critical data";  
  
$result = $KmsClient->createKey([  
    'Description' => $desc  
]);
```

Node.js

For details, see the [createKey property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// Create a CMK  
//  
const Description = 'Key for protecting critical data';
```



```
kmsClient.createKey({ Description }, (err, data) => {  
    ...  
});
```

Generating a Data Key

To generate a data key, use the [GenerateDataKey](#) operation. This operation returns plaintext and encrypted copies of the data key that it creates.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details, see the [generateDataKey](#) method in the *AWS SDK for Java API Reference*.

```
// Generate a data key  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
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_256");  
  
GenerateDataKeyResult dataKeyResult = kmsClient.generateDataKey(dataKeyRequest);  
  
ByteBuffer plaintextKey = dataKeyResult.getPlaintext();  
  
ByteBuffer encryptedKey = dataKeyResult.getCiphertextBlob();
```

C#

For details, see the [GenerateDataKey](#) method in the *AWS SDK for .NET*.

```
// Generate a data key  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
String keyId = "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
GenerateDataKeyRequest dataKeyRequest = new GenerateDataKeyRequest()  
{  
    KeyId = keyId,  
    KeySpec = DataKeySpec.AES_256  
};  
  
GenerateDataKeyResponse dataKeyResponse = kmsClient.GenerateDataKey(dataKeyRequest);  
  
MemoryStream plaintextKey = dataKeyResponse.Plaintext;  
  
MemoryStream encryptedKey = dataKeyResponse.CiphertextBlob;
```

Python

For details, see the [generate_date_key](#) method in the AWS SDK for Python (Boto 3).

```
# Generate a data key  
  
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
```

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

response = kms_client.generate_data_key(
    KeyId=key_id,
    KeySpec='AES_256'
)

plaintext_key = response['Plaintext']

encrypted_key = response['CiphertextBlob']
```

Ruby

For details, see the [generate_data_key](#) instance method in the [AWS SDK for Ruby](#).

```
# Generate a data key

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
key_id = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.generate_data_key({
  key_id: key_id,
  key_spec: 'AES_256'
})

plaintextKey = response.plaintext

encryptedKey = response.ciphertext_blob
```

PHP

For details, see the [GenerateDataKey](#) method in the [AWS SDK for PHP](#).

```
// Generate a data key
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$keySpec = 'AES_256';

$result = $KmsClient->generateDataKey([
    'KeyId' => $keyId,
    'KeySpec' => $keySpec,
]);

$plaintextKey = $result['Plaintext'];

$encryptedKey = $result['CiphertextBlob'];
```

Node.js

For details, see the [generateDataKey](#) property in the [AWS SDK for JavaScript in Node.js](#).

```
// Generate a data key
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
const KeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const KeySpec = 'AES_256';
kmsClient.generateDataKey({ KeyId, KeySpec }, (err, data) => {
  if (err) console.log(err, err.stack);
  else {
    const { CiphertextBlob, Plaintext } = data;
```

```
    }  
    ...  
  });
```

Viewing a Custom Master Key

To get detailed information about a customer master key (CMK), including the CMK ARN and [key state](#) (p. 123), use the [DescribeKey](#) operation.

`DescribeKey` does not get aliases. To get aliases, use the [ListAliases](#) operation.

This example uses the `kmsClient` client object that you created in [Creating a Client](#) (p. 207).

Java

For details, see the [describeKey method](#) in the *AWS SDK for Java API Reference*.

```
// Describe a CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
String keyId = "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
  
DescribeKeyRequest req = new DescribeKeyRequest().withKeyId(keyId);  
DescribeKeyResult result = kmsClient.describeKey(req);
```

C#

For details, see the [DescribeKey method](#) in the *AWS SDK for .NET*.

```
// Describe a CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
String keyId = "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
  
DescribeKeyRequest describeKeyRequest = new DescribeKeyRequest()  
{  
    KeyId = keyId  
};  
  
DescribeKeyResponse describeKeyResponse = kmsClient.DescribeKey(describeKeyRequest);
```

Python

For details, see the [describe_key method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Describe a CMK  
  
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
key_id = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'  
  
response = kms_client.describe_key(  
    KeyId=key_id  
)
```

Ruby

For details, see the [describe_key](#) instance method in the *AWS SDK for Ruby*.

```
# Describe a CMK

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.describe_key({
    key_id: keyId
})
```

PHP

For details, see the [DescribeKey method](#) in the *AWS SDK for PHP*.

```
// Describe a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

$result = $KmsClient->describeKey([
    'KeyId' => $keyId,
]);
```

Node.js

For details, see the [describeKey property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// Describe a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
const KeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
kmsClient.describeKey({ KeyId }, (err, data) => {
    ...
});
```

Getting Key IDs and Key ARNs of Customer Master Keys

To get the IDs and ARNs of the customer master keys, use the [ListKeys](#) operation.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details, see the [listKeys method](#) in the *AWS SDK for Java API Reference*.

```
// List CMKs in this account
//
Integer limit = 10;

ListKeysRequest req = new ListKeysRequest().withLimit(limit);
ListKeysResult result = kmsClient.listKeys(req);
```

C#

For details, see the [ListKeys method](#) in the *AWS SDK for .NET*.

```
// List CMKs in this account
//
int limit = 10;

ListKeysRequest listKeysRequest = new ListKeysRequest()
{
    Limit = limit
};
ListKeysResponse listKeysResponse = kmsClient.ListKeys(listKeysRequest);
```

Python

For details, see the [list_keys method](#) in the AWS SDK for Python (Boto 3).

```
# List CMKs in this account

response = kms_client.list_keys(
    Limit=10
)
```

Ruby

For details, see the [list_keys](#) instance method in the [AWS SDK for Ruby](#).

```
# List CMKS in this account

response = kmsClient.list_keys({
  limit: 10
})
```

PHP

For details, see the [ListKeys method](#) in the *AWS SDK for PHP*.

```
// List CMKs in this account
//
$limit = 10;

$result = $KmsClient->listKeys([
    'Limit' => $limit,
]);
```

Node.js

For details, see the [listKeys property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// List CMKs in this account
//
const Limit = 10;
kmsClient.listKeys({ Limit }, (err, data) => {
    ...
});
```

Enabling Customer Master Keys

To enable a disabled customer master key (CMK), use the [EnableKey](#) operation.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details about the Java implementation, see the [enableKey method](#) in the *AWS SDK for Java API Reference*.

```
// Enable a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
String keyId = "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

EnableKeyRequest req = new EnableKeyRequest().withKeyId(keyId);
kmsClient.enableKey(req);
```

C#

For details, see the [EnableKey method](#) in the *AWS SDK for .NET*.

```
// Enable a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
String keyId = "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

EnableKeyRequest enableKeyRequest = new EnableKeyRequest()
{
    KeyId = keyId
};
kmsClient.EnableKey(enableKeyRequest);
```

Python

For details, see the [enable_key method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Enable a CMK

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
key_id = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kms_client.enable_key(
    KeyId=key_id
)
```

Ruby

For details, see the [enable_key](#) instance method in the *AWS SDK for Ruby*.

```
# Enable a CMK

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.enable_key({
  key_id: keyId
})
```

PHP

For details, see the [EnableKey method](#) in the *AWS SDK for PHP*.

```
// Enable a CMK
```

```
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';  
  
$result = $KmsClient->enableKey([  
    'KeyId' => $keyId,  
]);
```

Node.js

For details, see the [enableKey property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// Enable a CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
const KeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';  
kmsClient.enableKey({ KeyId }, (err, data) => {  
    ...  
});
```

Disabling Customer Master Keys

To disable a CMK, use the [DisableKey](#) operation. Disabling a CMK prevents it from being used.

This example uses the `kmsClient` client object that you created in [Creating a Client](#) (p. 207).

Java

For details, see the [disableKey method](#) in the *AWS SDK for Java API Reference*.

```
// Disable a CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
String keyId = "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
  
DisableKeyRequest req = new DisableKeyRequest().withKeyId(keyId);  
kmsClient.disableKey(req);
```

C#

For details, see the [DisableKey method](#) in the *AWS SDK for .NET*.

```
// Disable a CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
String keyId = "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
  
DisableKeyRequest disableKeyRequest = new DisableKeyRequest()  
{  
    KeyId = keyId  
};  
kmsClient.DisableKey(disableKeyRequest);
```

Python

For details, see the [disable_key method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Disable a CMK

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
key_id = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kms_client.disable_key(
    KeyId=key_id
)
```

Ruby

For details, see the [disable_key](#) instance method in the [AWS SDK for Ruby](#).

```
# Disable a CMK

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.disable_key({
  key_id: keyId
})
```

PHP

For details, see the [DisableKey](#) method in the [AWS SDK for PHP](#).

```
// Disable a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

$result = $KmsClient->disableKey([
    'KeyId' => $keyId,
]);
```

Node.js

For details, see the [disableKey](#) property in the [AWS SDK for JavaScript in Node.js](#).

```
// Disable a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
const KeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
kmsClient.disableKey({ KeyId }, (err, data) => {
    ...
});
```

Encrypting and Decrypting Data Keys

The examples in this topic use the [Encrypt](#), [Decrypt](#), and [ReEncrypt](#) operations in the AWS KMS API.

These operations are designed to encrypt and decrypt [data keys \(p. 3\)](#). They use an AWS KMS [customer master key \(p. 2\)](#) (CMK) in the encryption operations and they cannot accept more than 4 KB (4096 bytes) of data. Although you might use them to encrypt small amounts of data, such as a password or RSA key, they are not designed to encrypt application data.

To encrypt application data, use the server-side encryption features of an AWS service, or a client-side encryption library, such as the [AWS Encryption SDK](#) or the [Amazon S3 encryption client](#).

Topics

- [Encrypting a Data Key](#) (p. 218)
- [Decrypting a Data Key](#) (p. 220)
- [Re-Encrypting a Data Key Under a Different Customer Master Key](#) (p. 221)

Encrypting a Data Key

The [Encrypt](#) operation is designed to encrypt data keys, but it is not frequently used. The [GenerateDataKey](#) and [GenerateDataKeyWithoutPlaintext](#) operations return encrypted data keys. You might use this method when you are moving encrypted data to a new region and want to encrypt its data key with a CMK in the new region.

This example uses the `kmsClient` client object that you created in [Creating a Client](#) (p. 207).

Java

For details, see the [encrypt method](#) in the *AWS SDK for Java API Reference*.

```
// Encrypt a data key
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
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 = kmsClient.encrypt(req).getCiphertextBlob();
```

C#

For details, see the [Encrypt method](#) in the *AWS SDK for .NET*.

```
// Encrypt a data key
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
MemoryStream plaintext = new MemoryStream();
plaintext.Write(new byte[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 }, 0, 10);

EncryptRequest encryptRequest = new EncryptRequest()
{
    KeyId = keyId,
    Plaintext = plaintext
};
MemoryStream ciphertext = kmsClient.Encrypt(encryptRequest).CiphertextBlob;
```

Python

For details, see the [encrypt method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Encrypt a data key

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
```

```
key_id = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
plaintext = b'\x01\x02\x03\x04\x05\x06\x07\x08\x09\x00'

response = kms_client.encrypt(
    KeyId=key_id,
    Plaintext=plaintext
)

ciphertext = response['CiphertextBlob']
```

Ruby

For details, see the [encrypt](#) instance method in the [AWS SDK for Ruby](#).

```
# Encrypt a data key

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
plaintext = "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x00"

response = kmsClient.encrypt({
  key_id: keyId,
  plaintext: plaintext
})

ciphertext = response.ciphertext_blob
```

PHP

For details, see the [Encrypt method](#) in the [AWS SDK for PHP](#).

```
// Encrypt a data key
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$message = pack('C*', 1, 2, 3, 4, 5, 6, 7, 8, 9, 0);

$result = $KmsClient->encrypt([
    'KeyId' => $keyId,
    'Plaintext' => $message,
]);

$ciphertext = $result['CiphertextBlob'];
```

Node.js

For details, see the [encrypt property](#) in the [AWS SDK for JavaScript in Node.js](#).

```
// Encrypt a data key
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
const KeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const Plaintext = Buffer.from([1, 2, 3, 4, 5, 6, 7, 8, 9, 0]);
kmsClient.encrypt({ KeyId, Plaintext }, (err, data) => {
  if (err) console.log(err, err.stack); // an error occurred
  else {
    const { CiphertextBlob } = data;
    ...
  }
});
```

Decrypting a Data Key

To decrypt a data key, use the [Decrypt](#) operation.

The `ciphertextBlob` that you specify must be the value of the `CiphertextBlob` field from a [GenerateDataKey](#), [GenerateDataKeyWithoutPlaintext](#), or [Encrypt](#) response.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details, see the [decrypt method](#) in the *AWS SDK for Java API Reference*.

```
// Decrypt a data key
//

ByteBuffer ciphertextBlob = Place your ciphertext here;

DecryptRequest req = new DecryptRequest().withCiphertextBlob(ciphertextBlob);
ByteBuffer plainText = kmsClient.decrypt(req).getPlaintext();
```

C#

For details, see the [Decrypt method](#) in the *AWS SDK for .NET*.

```
// Decrypt a data key
//

MemoryStream ciphertextBlob = new MemoryStream();
// Write ciphertext to memory stream

DecryptRequest decryptRequest = new DecryptRequest()
{
    CiphertextBlob = ciphertextBlob
};
MemoryStream plainText = kmsClient.Decrypt(decryptRequest).Plaintext;
```

Python

For details, see the [decrypt method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Decrypt a data key

ciphertext = 'Place your ciphertext here'

response = kms_client.decrypt(
    CiphertextBlob=ciphertext
)

plaintext = response['Plaintext']
```

Ruby

For details, see the [decrypt](#) instance method in the *AWS SDK for Ruby*.

```
# Decrypt a data key

ciphertext = 'Place your ciphertext here'
ciphertext_packed = [ciphertext].pack("H*")
```

```
response = kmsClient.decrypt({
  ciphertext_blob: ciphertext_packed
})

plaintext = response.plaintext
```

PHP

For details, see the [Decrypt method](#) in the *AWS SDK for PHP*.

```
// Decrypt a data key
//
$ciphertext = 'Place your cipher text blob here';

$result = $KmsClient->decrypt([
  'CiphertextBlob' => $ciphertext
]);

$plaintext = $result['Plaintext'];
```

Node.js

For details, see the [decrypt property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// Decrypt a data key
//
const CiphertextBlob = 'Place your cipher text blob here';
kmsClient.decrypt({ CiphertextBlob }, (err, data) => {
  if (err) console.log(err, err.stack); // an error occurred
  else {
    const { Plaintext } = data;
    ...
  }
});
```

Re-Encrypting a Data Key Under a Different Customer Master Key

To decrypt an encrypted data key, and then immediately re-encrypt the data key under a different customer master key (CMK), use the [ReEncrypt](#) operation. The operations are performed entirely on the server side within AWS KMS, so they never expose your plaintext outside of AWS KMS.

The `ciphertextBlob` that you specify must be the value of the `CiphertextBlob` field from a [GenerateDataKey](#), [GenerateDataKeyWithoutPlaintext](#), or [Encrypt](#) response.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details, see the [reEncrypt method](#) in the *AWS SDK for Java API Reference*.

```
// Re-encrypt a data key

ByteBuffer sourceCiphertextBlob = Place your ciphertext here;

// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
```

```
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 = kmsClient.reEncrypt(req).getCiphertextBlob();
```

C#

For details, see the [ReEncrypt method](#) in the *AWS SDK for .NET*.

```
// Re-encrypt a data key  
  
MemoryStream sourceCiphertextBlob = new MemoryStream();  
// Write ciphertext to memory stream  
  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
String destinationKeyId = "arn:aws:kms:us-  
west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";  
  
ReEncryptRequest reEncryptRequest = new ReEncryptRequest()  
{  
    CiphertextBlob = sourceCiphertextBlob,  
    DestinationKeyId = destinationKeyId  
};  
MemoryStream destinationCipherTextBlob =  
    kmsClient.ReEncrypt(reEncryptRequest).CiphertextBlob;
```

Python

For details, see the [re_encrypt method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Re-encrypt a data key  
ciphertext = 'Place your ciphertext here'  
  
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
key_id = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321'  
  
response = kms_client.re_encrypt(  
    CiphertextBlob=ciphertext,  
    DestinationKeyId=key_id  
)  
  
destination_ciphertext_blob = response['CiphertextBlob']
```

Ruby

For details, see the [re_encrypt](#) instance method in the *AWS SDK for Ruby*.

```
# Re-encrypt a data key  
  
ciphertext = 'Place your ciphertext here'  
ciphertext_packed = [ciphertext].pack("H*")  
  
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
keyId = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321'  
  
response = kmsClient.re_encrypt({  
    ciphertext_blob: ciphertext_packed,  
    destination_key_id: keyId
```

```
    })  
  
    destination_ciphertext_blob = response.ciphertext_blob.unpack('H*')
```

PHP

For details, see the [ReEncrypt method](#) in the *AWS SDK for PHP*.

```
// Re-encrypt a data key  
  
$ciphertextBlob = 'Place your ciphertext here';  
  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321';  
  
$result = $KmsClient->reEncrypt([  
    'CiphertextBlob' => $ciphertextBlob,  
    'DestinationKeyId' => $keyId,  
]);
```

Node.js

For details, see the [reEncrypt property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// Re-encrypt a data key  
const CiphertextBlob = 'Place your cipher text blob here';  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
const DestinationKeyId = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321';  
kmsClient.reEncrypt({ CiphertextBlob, DestinationKeyId }, (err, data) => {  
    ...  
});
```

Working with Key Policies

The examples in this topic use the AWS KMS API to view and change the key policies of AWS KMS customer master keys (CMKs). For details about how to use key policies and IAM policies to manage access to your CMKs, see [Authentication and Access Control for AWS KMS \(p. 29\)](#).

Topics

- [Listing Key Policy Names \(p. 223\)](#)
- [Getting a Key Policy \(p. 225\)](#)
- [Setting a Key Policy \(p. 227\)](#)

Listing Key Policy Names

To get the names of key policies for a customer master key, use the [ListKeyPolicies](#) operation. The only key policy name it returns is **default**.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details about the Java implementation, see the [listKeyPolicies method](#) in the *AWS SDK for Java API Reference*.

```
// List key policies
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

ListKeyPoliciesRequest req = new ListKeyPoliciesRequest().withKeyId(keyId);
ListKeyPoliciesResult result = kmsClient.listKeyPolicies(req);
```

C#

For details, see the [ListKeyPolicies method](#) in the *AWS SDK for .NET*.

```
// List key policies
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
String keyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

ListKeyPoliciesRequest listKeyPoliciesRequest = new ListKeyPoliciesRequest()
{
    KeyId = keyId
};
ListKeyPoliciesResponse listKeyPoliciesResponse =
    kmsClient.ListKeyPolicies(listKeyPoliciesRequest);
```

Python

For details, see the [list_key_policies method](#) in the *AWS SDK for Python (Boto 3)*.

```
# List key policies

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
key_id = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kms_client.list_key_policies(
    KeyId=key_id
)
```

Ruby

For details, see the [list_key_policies](#) instance method in the *AWS SDK for Ruby*.

```
# List key policies

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.list_key_policies({
  key_id: keyId
})
```

PHP

For details, see the [ListKeyPolicies method](#) in the *AWS SDK for PHP*.

```
// List key policies
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
```

```
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';  
  
$result = $KmsClient->listKeyPolicies([  
    'KeyId' => $keyId  
]);
```

Node.js

For details, see the [listKeyPolicies property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// List key policies  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
const KeyId = 'arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';  
  
kmsClient.listKeyPolicies({ KeyId }, (err, data) => {  
    ...  
});
```

Getting a Key Policy

To get the key policy for a customer master key, use the [GetKeyPolicy](#) operation.

[GetKeyPolicy](#) requires a policy name. The only valid policy name is **default**.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details, see the [getKeyPolicy method](#) in the *AWS SDK for Java API Reference*.

```
// Get the policy for a CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
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 = kmsClient.getKeyPolicy(req);
```

C#

For details, see the [GetKeyPolicy method](#) in the *AWS SDK for .NET*.

```
// Get the policy for a CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
String keyId = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
String policyName = "default";  
  
GetKeyPolicyRequest getKeyPolicyRequest = new GetKeyPolicyRequest()  
{  
    KeyId = keyId,  
    PolicyName = policyName  
};
```



```
GetKeyPolicyResponse getKeyPolicyResponse =  
    kmsClient.GetKeyPolicy(getKeyPolicyRequest);
```

Python

For details, see the [get_key_policy method](#) in the AWS SDK for Python (Boto 3).

```
# Get the policy for a CMK  
  
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
key_id = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'  
policy_name = 'default'  
  
response = kms_client.get_key_policy(  
    KeyId=key_id,  
    PolicyName=policy_name  
)
```

Ruby

For details, see the [get_key_policy](#) instance method in the [AWS SDK for Ruby](#).

```
# Get the policy for a CMK  
  
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'  
policyName = 'default'  
  
response = kmsClient.get_key_policy({  
  key_id: keyId,  
  policy_name: policyName  
})
```

PHP

For details, see the [GetKeyPolicy method](#) in the *AWS SDK for PHP*.

```
// Get the policy for a CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';  
$policyName = "default";  
  
$result = $KmsClient->getKeyPolicy([  
    'KeyId' => $keyId,  
    'PolicyName' => $policyName  
]);
```

Node.js

For details, see the [getKeyPolicy property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// Get the policy for a CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
const KeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';  
const PolicyName = 'default';  
kmsClient.getKeyPolicy({ KeyId, PolicyName }, (err, data) => {  
    ...  
});
```

Setting a Key Policy

To establish or change a key policy for a CMK, use the [PutKeyPolicy](#) operation.

`PutKeyPolicy` requires a policy name. The only valid policy name is **default**.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details, see the [putKeyPolicy method](#) in the *AWS SDK for Java API Reference*.

```
// Set a key policy for a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
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);
kmsClient.putKeyPolicy(req);
```

C#

For details, see the [PutKeyPolicy method](#) in the *AWS SDK for .NET*.

```
// Set a key policy for a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
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 putKeyPolicyRequest = new PutKeyPolicyRequest()
{
    KeyId = keyId,
    Policy = policy,
    PolicyName = policyName
};
kmsClient.PutKeyPolicy(putKeyPolicyRequest);
```

Python

For details, see the [put_key_policy method](#) in the AWS SDK for Python (Boto 3).

```
# Set a key policy for a CMK

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
key_id = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
policy_name = 'default'
policy = """
{
    "Version": "2012-10-17",
    "Statement": [{
        "Sid": "Allow access for ExampleUser",
        "Effect": "Allow",
        "Principal": {"AWS": "arn:aws:iam::111122223333:user/ExampleUser"},
        "Action": [
            "kms:Encrypt",
            "kms:GenerateDataKey*",
            "kms:Decrypt",
            "kms:DescribeKey",
            "kms:ReEncrypt*"
        ],
        "Resource": "*"
    }]
}"""

response = kms_client.put_key_policy(
    KeyId=key_id,
    Policy=policy,
    PolicyName=policy_name
)
```

Ruby

For details, see the [put_key_policy](#) instance method in the [AWS SDK for Ruby](#).

```
# Set a key policy for a CMK

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
policyName = 'default'
policy = "{\n
  \"Version\": \"2012-10-17\",
  \"Statement\": [{
    \"Sid\": \"Allow access for ExampleUser\",
    \"Effect\": \"Allow\",
    \"Principal\": {\"AWS\": \"arn:aws:iam::111122223333:user/ExampleUser\"},
```

```
\n        "Action": [\n\n            "kms:Encrypt",\n\n            "kms:GenerateDataKey*",\n\n            "kms:Decrypt",\n\n            "kms:DescribeKey",\n\n            "kms:ReEncrypt*",\n\n        ],\n\n        "Resource": "*" \n    }\n]\n\n\nresponse = kmsClient.put_key_policy({\n    key_id: keyId,\n    policy: policy,\n    policy_name: policyName\n})
```

PHP

For details, see the [PutKeyPolicy method](#) in the *AWS SDK for PHP*.

```
// Set a key policy for a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$policyName = "default";

$result = $KmsClient->putKeyPolicy([
    'KeyId' => $keyId,
    'PolicyName' => $policyName,
    'Policy' => '{
        "Version": "2012-10-17",
        "Id": "custom-policy-2016-12-07",
        "Statement": [
            { "Sid": "Enable IAM User Permissions",
              "Effect": "Allow",
              "Principal":
                { "AWS": "arn:aws:iam::111122223333:user/root" },
              "Action": [ "kms:*" ],
              "Resource": "*" },
            { "Sid": "Enable IAM User Permissions",
              "Effect": "Allow",
              "Principal":
                { "AWS": "arn:aws:iam::111122223333:user/ExampleUser" },
              "Action": [
                  "kms:Encrypt*",
                  "kms:GenerateDataKey*",
                  "kms:Decrypt*",
                  "kms:DescribeKey*",
                  "kms:ReEncrypt*"
                ],
              "Resource": "*" }
        ],
        "Resource": "*" }
    ]
} ,
]);
```

Node.js

For details, see the [putKeyPolicy property](#) in the *AWS SDK for Node.js*.

```
// Set a key policy for a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
```

```
const KeyId = 'arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';  
const PolicyName = 'default';  
const Policy = `{  
  "Version": "2012-10-17",  
  "Id": "custom-policy-2016-12-07",  
  "Statement": [  
    {  
      "Sid": "Enable IAM User Permissions",  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": "arn:aws:iam::111122223333:root"  
      },  
      "Action": "kms:*",  
      "Resource": "*"   
    },  
    {  
      "Sid": "Enable IAM User Permissions",  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": "arn:aws:iam::111122223333:user/ExampleUser"  
      },  
      "Action": [  
        "kms:Encrypt*",  
        "kms:GenerateDataKey*",  
        "kms:Decrypt*",  
        "kms:DescribeKey*",  
        "kms:ReEncrypt*"   
      ],  
      "Resource": "*"   
    }  
  ]  
}; // The key policy document  
  
kmsClient.putKeyPolicy({ KeyId, Policy, PolicyName }, (err, data) => {  
  ...  
});
```

Working with Grants

The examples in this topic use the AWS KMS API to create, view, retire, and revoke grants on AWS KMS customer master keys (CMKs).

Topics

- [Creating a Grant \(p. 230\)](#)
- [Viewing a Grant \(p. 232\)](#)
- [Retiring a Grant \(p. 234\)](#)
- [Revoking a Grant \(p. 235\)](#)

Creating a Grant

To create a grant for an AWS KMS customer master key, use the [CreateGrant](#) operation.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details, see the [createGrant method](#) in the *AWS SDK for Java API Reference*.

```
// Create a grant
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
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 = kmsClient.createGrant(req);
```

C#

For details, see the [CreateGrant method](#) in the *AWS SDK for .NET*.

```
// Create a grant
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
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 createGrantRequest = new CreateGrantRequest()
{
    KeyId = keyId,
    GranteePrincipal = granteePrincipal,
    Operations = new List<string>() { operation }
};

CreateGrantResponse createGrantResult = kmsClient.CreateGrant(createGrantRequest);
```

Python

For details, see the [create_grant method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Create a grant

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
key_id = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
grantee_principal = 'arn:aws:iam::111122223333:user/Alice'
operation = 'Encrypt'

response = kms_client.create_grant(
    KeyId=key_id,
    GranteePrincipal=grantee_principal,
    Operations=operation
)
```

Ruby

For details, see the [create_grant](#) instance method in the *AWS SDK for Ruby*.

```
# Create a grant

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
granteePrincipal = 'arn:aws:iam::111122223333:user/Alice'
```

```
operation = ['Encrypt']

response = kmsClient.create_grant({
    key_id: keyId,
    grantee_principal: granteePrincipal,
    operations: operation
})
```

PHP

For details, see the [CreateGrant method](#) in the *AWS SDK for PHP*.

```
// Create a grant
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$granteePrincipal = "arn:aws:iam::111122223333:user/Alice";
$operation = ['Encrypt', 'Decrypt']

$result = $KmsClient->createGrant([
    'GranteePrincipal' => $granteePrincipal,
    'KeyId' => $keyId,
    'Operations' => $operation
]);
```

Node.js

For details, see the [createGrant property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// Create a grant
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
const KeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const GranteePrincipal = 'arn:aws:iam::111122223333:user/Alice';
const Operations = ["Encrypt", "Decrypt"];
kmsClient.createGrant({ KeyId, GranteePrincipal, Operations }, (err, data) => {
    ...
});
```

Viewing a Grant

To get detailed information about the grants on an AWS KMS customer master key, use the [ListGrants](#) operation.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details about the Java implementation, see the [listGrants method](#) in the *AWS SDK for Java API Reference*.

```
// Listing grants on a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
String keyId = "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
Integer limit = 10;
```

```
ListGrantsRequest req = new ListGrantsRequest().withKeyId(keyId).withLimit(limit);  
ListGrantsResult result = kmsClient.listGrants(req);
```

C#

For details, see the [ListGrants method](#) in the *AWS SDK for .NET*.

```
// Listing grants on a CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
String keyId = "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
int limit = 10;  
  
ListGrantsRequest listGrantsRequest = new ListGrantsRequest()  
{  
    KeyId = keyId,  
    Limit = limit  
};  
ListGrantsResponse listGrantsResponse = kmsClient.ListGrants(listGrantsRequest);
```

Python

For details, see the [list_grants method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Listing grants on a CMK  
  
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
key_id = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'  
  
response = kms_client.list_grants(  
    KeyId=key_id,  
    Limit=10  
)
```

Ruby

For details, see the [list_grants](#) instance method in the *AWS SDK for Ruby*.

```
# Listing grants on a CMK  
  
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'  
  
response = kmsClient.list_grants({  
    key_id: keyId,  
    limit: 10  
})
```

PHP

For details, see the [ListGrants method](#) in the *AWS SDK for PHP*.

```
// Listing grants on a CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';  
$limit = 10;  
  
$result = $KmsClient->listGrants([  
    'KeyId' => $keyId,
```



```
'Limit' => $limit,  
]);
```

Node.js

For details, see the [listGrants property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// Listing grants on a CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
const KeyId = 'arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';  
const Limit = 10;  
kmsClient.listGrants({ KeyId, Limit }, (err, data) => {  
    ...  
});
```

Retiring a Grant

To retire a grant for an AWS KMS customer master key, use the [RetireGrant](#) operation. You should retire a grant to clean up after you are done using it.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details, see the [retireGrant method](#) in the *AWS SDK for Java API Reference*.

```
// Retire a grant  
//  
String grantToken = Place your grant token here;  
  
RetireGrantRequest req = new RetireGrantRequest().withGrantToken(grantToken);  
kmsClient.retireGrant(req);
```

C#

For details, see the [RetireGrant method](#) in the *AWS SDK for .NET*.

```
// Retire a grant  
//  
String grantToken = "Place your grant token here";  
  
RetireGrantRequest retireGrantRequest = new RetireGrantRequest()  
{  
    GrantToken = grantToken  
};  
kmsClient.RetireGrant(retireGrantRequest);
```

Python

For details, see the [retire_grant method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Retire a grant  
  
grant_token = Place your grant token here  
  
response = kms_client.retire_grant(  
    GrantToken=grant_token
```

```
)
```

Ruby

For details, see the [retire_grant](#) instance method in the [AWS SDK for Ruby](#).

```
# Retire a grant

grantToken = Place your grant token here

response = kmsClient.retire_grant({
  grant_token: grantToken
})
```

PHP

For details, see the [RetireGrant](#) method in the [AWS SDK for PHP](#).

```
// Retire a grant
//
$grantToken = 'Place your grant token here';

$result = $KmsClient->retireGrant([
    'GrantToken' => $grantToken,
]);
```

Node.js

For details, see the [retireGrant](#) property in the [AWS SDK for JavaScript in Node.js](#).

```
// Retire a grant
//
const GrantToken = 'Place your grant token here';
kmsClient.retireGrant({ GrantToken }, (err, data) => {
  ...
});
```

Revoking a Grant

To revoke a grant to an AWS KMS customer master key, use the [RevokeGrant](#) operation. You can revoke a grant to explicitly deny operations that depend on it.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details, see the [revokeGrant](#) method in the [AWS SDK for Java API Reference](#).

```
// Revoke a grant on a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
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);
kmsClient.revokeGrant(req);
```

C#

For details, see the [RevokeGrant method](#) in the *AWS SDK for .NET*.

```
// Revoke a grant on a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
String keyId = "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";
String grantId = "grant1";

RevokeGrantRequest revokeGrantRequest = new RevokeGrantRequest()
{
    KeyId = keyId,
    GrantId = grantId
};
kmsClient.RevokeGrant(revokeGrantRequest);
```

Python

For details, see the [revoke_grant method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Revoke a grant on a CMK

// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
key_id = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
grant_id = 'grant1'

response = kms_client.revoke_grant(
    KeyId=key_id,
    GrantId=grant_id
)
```

Ruby

For details, see the [revoke_grant](#) instance method in the *AWS SDK for Ruby*.

```
# Revoke a grant on a CMK

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
grantId = 'grant1'

response = kmsClient.revoke_grant({
  key_id: keyId,
  grant_id: grantId
})
```

PHP

For details, see the [RevokeGrant method](#) in the *AWS SDK for PHP*.

```
// Revoke a grant on a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
$grantId = "grant1";

$result = $KmsClient->revokeGrant([
    'KeyId' => $keyId,
    'GrantId' => $grantId,
```

```
});
```

Node.js

For details, see the [revokeGrant property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// Revoke a grant on a CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
const KeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
const GrantId = 'grant1';
kmsClient.revokeGrant({ GrantId, KeyId }, (err, data) => {
    ...
});
```

Working with Aliases

The examples in this topic use the AWS KMS API to create, view, update, and delete aliases.

An *alias* is an optional display name for a [customer master key \(CMK\)](#) (p. 2). Each CMK can have multiple aliases, but each alias points to only one CMK. The alias name must be unique in the AWS account and region. To simplify code that runs in multiple regions, you can use the same alias name but point it to a different CMK in each region.

You can use AWS KMS API operations to list, create, and delete aliases. You can also update an alias, which associates an existing alias with a different CMK. There is no operation to edit or change an alias name. If you create an alias for a CMK that already has an alias, the operation creates another alias for the same CMK. To change an alias name, delete the current alias and then create a new alias for the CMK.

Because an alias is not a property of a CMK, it can be associated with and disassociated from an existing CMK without changing the properties of the CMK. Deleting an alias does not delete the underlying CMK.

You can use an alias as the value of the `KeyId` parameter only in the following operations:

- `DescribeKey`
- `Encrypt`
- `GenerateDataKey`
- `GenerateDataKeyWithoutPlaintext`
- `ReEncrypt`

Aliases are created in an AWS account and are known only to the account in which you create them. You cannot use an alias name or alias ARN to identify a CMK in a different AWS account.

To specify an alias, use the alias name or alias ARN, as shown in the following example. In either case, be sure to prepend `"alias/"` to the alias name.

```
// Fully specified ARN
arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias

// Fully specified ARN
arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias
```

Topics

- [Creating an Alias \(p. 238\)](#)

- [Listing Aliases \(p. 239\)](#)
- [Updating an Alias \(p. 242\)](#)
- [Deleting an Alias \(p. 244\)](#)

Creating an Alias

To create an alias, use the [CreateAlias](#) operation. The alias must be unique in the account and region. If you create an alias for a CMK that already has an alias, [CreateAlias](#) creates another alias to the same CMK. It does not replace the existing alias.

You cannot create an alias that begins with `aws/`. The `aws/` prefix is reserved by Amazon Web Services for [AWS managed CMKs \(p. 2\)](#).

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details, see the [createAlias method](#) in the *AWS SDK for Java API Reference*.

```
// Create an alias for a CMK
//
String aliasName = "alias/projectKey1";
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
String targetKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

CreateAliasRequest req = new
    CreateAliasRequest().withAliasName(aliasName).withTargetKeyId(targetKeyId);
kmsClient.createAlias(req);
```

C#

For details, see the [CreateAlias method](#) in the *AWS SDK for .NET*.

```
// Create an alias for a CMK
//
String aliasName = "alias/projectKey1";
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
String targetKeyId = "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";

CreateAliasRequest createAliasRequest = new CreateAliasRequest()
{
    AliasName = aliasName,
    TargetKeyId = targetKeyId
};
kmsClient.CreateAlias(createAliasRequest);
```

Python

For details, see the [create_alias method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Create an alias for a CMK

alias_name = 'alias/projectKey1'
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
target_key_id = 'arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'
```

```
response = kms_client.create_alias(  
    AliasName=alias_name,  
    TargetKeyId=key_id  
)
```

Ruby

For details, see the [create_alias](#) instance method in the [AWS SDK for Ruby](#).

```
# Create an alias for a CMK  
  
aliasName = 'alias/projectKey1'  
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
targetKeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'  
  
response = kmsClient.create_alias({  
    alias_name: aliasName,  
    target_key_id: targetKeyId  
})
```

PHP

For details, see the [CreateAlias method](#) in the [AWS SDK for PHP](#).

```
// Create an alias for a CMK  
//  
$aliasName = "alias/projectKey1";  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';  
  
$result = $KmsClient->createAlias([  
    'AliasName' => $aliasName,  
    'TargetKeyId' => $keyId,  
]);
```

Node.js

For details, see the [createAlias property](#) in the [AWS SDK for JavaScript in Node.js](#).

```
// Create an alias for a CMK  
//  
const AliasName = 'alias/projectKey1';  
  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
const TargetKeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';  
kmsClient.createAlias({ AliasName, TargetKeyId }, (err, data) => {  
    ...  
});
```

Listing Aliases

To list aliases in the account and region, use the [ListAliases](#) operation.

By default, the **ListAliases** command returns all aliases in the account and region. This includes aliases that you created and associated with your [customer-managed CMKs](#) (p. 2), and aliases that AWS created and associated with your [AWS managed CMKs](#) (p. 2). The response might also include aliases that have

no `TargetKeyId` field. These are predefined aliases that AWS has created but has not yet associated with a CMK.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details about the Java implementation, see the [listAliases method](#) in the *AWS SDK for Java API Reference*.

```
// List the aliases in this AWS account
//
Integer limit = 10;

ListAliasesRequest req = new ListAliasesRequest().withLimit(limit);
ListAliasesResult result = kmsClient.listAliases(req);
```

C#

For details, see the [ListAliases method](#) in the *AWS SDK for .NET*.

```
// List the aliases in this AWS account
//
int limit = 10;

ListAliasesRequest listAliasesRequest = new ListAliasesRequest()
{
    Limit = limit
};
ListAliasesResponse listAliasesResponse = kmsClient.ListAliases(listAliasesRequest);
```

Python

For details, see the [list_aliases method](#) in the AWS SDK for Python (Boto 3).

```
# List the aliases in this AWS account

response = kms_client.list_aliases(
    Limit=10
)
```

Ruby

For details, see the [list_aliases](#) instance method in the *AWS SDK for Ruby*.

```
# List the aliases in this AWS account

response = kmsClient.list_aliases({
  limit: 10
})
```

PHP

For details, see the [List Aliases method](#) in the *AWS SDK for PHP*.

```
// List the aliases in this AWS account
//
$limit = 10;

$result = $KmsClient->listAliases([
```

```
    'Limit' => $limit,  
  ]);
```

Node.js

For details, see the [listAliases property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// List the aliases in this AWS account  
//  
const Limit = 10;  
kmsClient.listAliases({ Limit }, (err, data) => {  
  ...  
});
```

To list only the aliases that are associated with a particular CMK, use the `KeyId` parameter. Its value can be the ID or Amazon Resource Name (ARN) of any CMK in the region. You cannot specify an alias name or alias ARN.

Java

For details about the Java implementation, see the [listAliases method](#) in the *AWS SDK for Java API Reference*.

```
// List the aliases for one CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
String keyId = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
  
ListAliasesRequest req = new ListAliasesRequest().withKeyId(keyId);  
ListAliasesResult result = kmsClient.listAliases(req);
```

C#

For details, see the [ListAliases method](#) in the *AWS SDK for .NET*.

```
// List the aliases for one CMK  
//  
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
String keyId = "arn:aws:kms:us-  
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab";  
  
ListAliasesRequest listAliasesRequest = new ListAliasesRequest()  
{  
    KeyId = keyId  
};  
ListAliasesResponse listAliasesResponse = kmsClient.ListAliases(listAliasesRequest);
```

Python

For details, see the [list_aliases method](#) in the *AWS SDK for Python (Boto 3)*.

```
# List the aliases for one CMK  
  
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN  
key_id = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'  
  
response = kms_client.list_aliases(  
    KeyId=key_id
```



```
)
```

Ruby

For details, see the [list_aliases](#) instance method in the [AWS SDK for Ruby](#).

```
# List the aliases for one CMK

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab'

response = kmsClient.list_aliases({
  key_id: keyId
})
```

PHP

For details, see the [List Aliases method](#) in the *AWS SDK for PHP*.

```
// List the aliases for one CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';

$result = $KmsClient->listAliases([
    'KeyId' => $keyId,
]);
```

Node.js

For details, see the [listAliases property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// List the aliases for one CMK
//
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
const KeyId = 'arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab';
kmsClient.listAliases({ KeyId }, (err, data) => {
    ...
});
```

Updating an Alias

To associate an existing alias with a different CMK, use the [UpdateAlias](#) operation.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details about the Java implementation, see the [updateAlias method](#) in the *AWS SDK for Java API Reference*.

```
// Updating an alias
//
String aliasName = "alias/projectKey1";
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
String targetKeyId = "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";
```

```
UpdateAliasRequest req = new UpdateAliasRequest()
    .withAliasName(aliasName)
    .withTargetKeyId(targetKeyId);

kmsClient.updateAlias(req);
```

C#

For details, see the [UpdateAlias method](#) in the *AWS SDK for .NET*.

```
// Updating an alias
//
String aliasName = "alias/projectKey1";
// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
String targetKeyId = "arn:aws:kms:us-west-2:11112223333:key/0987dcba-09fe-87dc-65ba-ab0987654321";

UpdateAliasRequest updateAliasRequest = new UpdateAliasRequest()
{
    AliasName = aliasName,
    TargetKeyId = targetKeyId
};

kmsClient.UpdateAlias(updateAliasRequest);
```

Python

For details, see the [update_alias method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Updating an alias

alias_name = 'alias/projectKey1'
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
key_id = 'arn:aws:kms:us-west-2:11112223333:key/0987dcba-09fe-87dc-65ba-ab0987654321'

response = kms_client.update_alias(
    AliasName=alias_name,
    TargetKeyId=key_id
)
```

Ruby

For details, see the [update_alias](#) instance method in the *AWS SDK for Ruby*.

```
# Updating an alias

aliasName = 'alias/projectKey1'
# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
keyId = 'arn:aws:kms:us-west-2:11112223333:key/0987dcba-09fe-87dc-65ba-ab0987654321'

response = kmsClient.update_alias({
  alias_name: aliasName,
  target_key_id: keyId
})
```

PHP

For details, see the [UpdateAlias method](#) in the *AWS SDK for PHP*.

```
// Updating an alias
//
```

```
$aliasName = "alias/projectKey1";

// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
$keyId = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321';

$result = $KmsClient->updateAlias([
    'AliasName' => $aliasName,
    'TargetKeyId' => $keyId,
]);
```

Node.js

For details, see the [updateAlias property](#) in the *AWS SDK for JavaScript in Node.js*.

```
// Updating an alias
//
const AliasName = 'alias/projectKey1';

// Replace the following fictitious CMK ARN with a valid CMK ID or ARN
const TargetKeyId = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321';
kmsClient.updateAlias({ AliasName, TargetKeyId }, (err, data) => {
    ...
});
```

Deleting an Alias

To delete an alias, use the [DeleteAlias](#) operation. Deleting an alias has no effect on the underlying CMK.

This example uses the `kmsClient` client object that you created in [Creating a Client \(p. 207\)](#).

Java

For details, see the [deleteAlias method](#) in the *AWS SDK for Java API Reference*.

```
// Delete an alias for a CMK
//
String aliasName = "alias/projectKey1";

DeleteAliasRequest req = new DeleteAliasRequest().withAliasName(aliasName);
kmsClient.deleteAlias(req);
```

C#

For details, see the [DeleteAlias method](#) in the *AWS SDK for .NET*.

```
// Delete an alias for a CMK
//
String aliasName = "alias/projectKey1";

DeleteAliasRequest deleteAliasRequest = new DeleteAliasRequest()
{
    AliasName = aliasName
};
kmsClient.DeleteAlias(deleteAliasRequest);
```

Python

For details, see the [delete_alias method](#) in the *AWS SDK for Python (Boto 3)*.

```
# Delete an alias for a CMK

alias_name = 'alias/projectKey1'

response = kms_client.delete_alias(
    AliasName=alias_name
)
```

Ruby

For details, see the [delete_alias](#) instance method in the [AWS SDK for Ruby](#).

```
# Delete an alias for a CMK

aliasName = 'alias/projectKey1'

response = kmsClient.delete_alias({
  alias_name: aliasName
})
```

PHP

For details, see the [DeleteAlias](#) method in the *AWS SDK for PHP*.

```
// Delete an alias for a CMK
//
$aliasName = "alias/projectKey1";

$result = $KmsClient->deleteAlias([
    'AliasName' => $aliasName,
]);
```

Node.js

For details, see the [deleteAlias](#) property in the *AWS SDK for JavaScript in Node.js*.

```
// Delete an alias for a CMK
//
const AliasName = 'alias/projectKey1';
kmsClient.deleteAlias({ AliasName }, (err, data) => {
    ...
});
```

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.

Topics

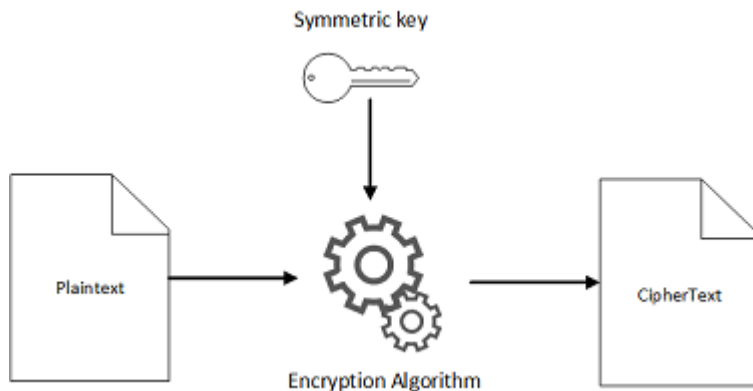
- [How Symmetric Key Cryptography Works](#) (p. 246)
- [Authenticated Encryption](#) (p. 247)
- [Encryption Context](#) (p. 248)
- [Reference: AWS KMS and Cryptography Terminology](#) (p. 249)

How Symmetric Key Cryptography Works

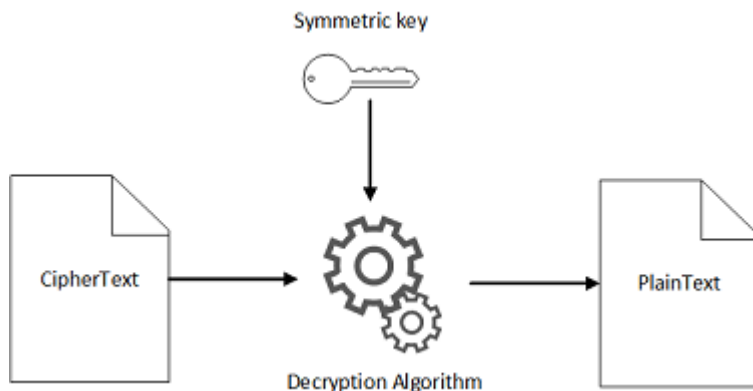
This topic provides a high-level introduction to how symmetric key cryptography uses algorithms to encrypt and decrypt data, the difference between block and stream ciphers, and how block ciphers use encryption modes to expand the effectiveness of the generic encryption schemes.

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. 248) 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. 248) 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 non-secret key-value pairs that you can pass to AWS KMS when you call the [Encrypt](#), [Decrypt](#), [ReEncrypt](#), [GenerateDataKey](#), or [GenerateDataKeyWithoutPlaintext](#) APIs.

When an encryption context is provided in an encryption request, it is cryptographically bound to the ciphertext such that the same encryption context required to decrypt (or decrypt and re-encrypt) the data. If the encryption context provided in the decryption request is not an exact, case-sensitive match, the decrypt request fails. Only the order of encryption context pairs can vary.

The encryption context is never included in the ciphertext that AWS KMS returns for any encryption operation. However, it 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 secret, not encrypted, and appears in CloudTrail logs, your encryption context should not include sensitive information. We 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.

Encryption Context in Grants and Key Policies

In addition to its primary use in verifying integrity and authenticity, you can also use the encryption context as a condition for authorizing use of customer master keys (CMKs) in IAM and key policies, and grants. This element can limit the permissions to very specific types of data or data from a limited set of sources.

- In key policies and IAM policies that control access to AWS KMS CMKs, you can include condition keys that limit the permission to requests that include particular [encryption context keys \(p. 66\)](#) or [key-value pairs \(p. 64\)](#).
- When you [create a grant \(p. 78\)](#), you can include [grant constraints](#) that allow access only when a request includes a particular encryption context or encryption context keys.

For example, when an Amazon EBS volume is attached to an Amazon EC2 instance, a grant is created that allows only that instance to decrypt only that volume. This is accomplished by including the volume ID in the encryption context, then adding a [grant constraint](#) that requires an encryption context with that volume ID. If the grant did not include the encryption context constraint, the Amazon EC2 instance could decrypt any volume that was 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. 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

To simplify use of any encryption context when you call the [Decrypt](#) (or [ReEncrypt](#)) API, you can store the encryption context alongside the encrypted data. We recommend that you store only enough of the encryption context to help you create the full encryption context when you need it for encryption or decryption.

For example, if the encryption context is the fully qualified path to a file, store only part of that path with the encrypted file contents. Then, when you need the full encryption context, reconstruct it from the stored fragment. If someone tampers with the file, such as renaming it or moving it to a different location, the encryption context value changes and the decryption request fails.

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

This topic describes significant updates to the *AWS Key Management Service Developer Guide*.

Topics

- [Recent Updates \(p. 251\)](#)
- [Earlier Updates \(p. 251\)](#)

Recent Updates

The following table describes significant changes to this documentation since January 2018. In addition to major changes listed here, we also update the documentation frequently to improve the descriptions and examples, and to address the feedback that you send to us. To be notified about significant changes, use the link in the upper right corner to subscribe to the RSS feed.

Current API version: 2014-11-01

update-history-change	update-history-description	update-history-date
New console	Explains how to use the new AWS KMS console, which is independent of the IAM console. The original console, and instructions for using it, will remain available for a brief period to give you time to familiarize yourself with the new console.	November 7, 2018
Limit change	Changed the shared requests-per-second limit on customer master keys.	August 21, 2018
New content	Explains how AWS Secrets Manager uses AWS KMS customer master keys to encrypt the secret value in a secret.	July 13, 2018
New content	Explains how DynamoDB uses AWS KMS customer master keys to support its server-side encryption option.	May 23, 2018
New feature	Explains how to use a private endpoint in your VPC to connect directly to AWS KMS, instead of connecting over the internet.	January 22, 2018

Earlier Updates

The following table describes the important changes to the AWS Key Management Service Developer Guide prior to 2018.

Change	Description	Date
New content	Added documentation about Tagging Keys (p. 24).	February 15, 2017
New content	Added documentation about Monitoring Customer Master Keys (p. 173) and Monitoring with Amazon CloudWatch (p. 174).	August 31, 2016
New content	Added documentation about Importing Key Material (p. 93).	August 11, 2016
New content	Added the following documentation: Overview of Managing Access (p. 30), Using IAM Policies (p. 50), AWS KMS API Permissions Reference (p. 54), and Using Policy Conditions (p. 59).	July 5, 2016
Update	Updated portions of the documentation in the Authentication and Access Control (p. 29) chapter.	July 5, 2016
Update	Updated the Limits (p. 255) page to reflect new default limits.	May 31, 2016
Update	Updated the Limits (p. 255) page to reflect new default limits, and updated the Grant Tokens (p. 7) documentation to improve clarity and accuracy.	April 11, 2016
New content	Added documentation about Allowing Multiple IAM Users to Access a CMK (p. 46) and Using the IP Address Condition (p. 60).	February 17, 2016
Update	Updated the Using Key Policies in AWS KMS (p. 33) and Changing a Key Policy (p. 43) pages to improve clarity and accuracy.	February 17, 2016
Update	Updated the Getting Started (p. 8) topic pages to improve clarity.	January 5, 2016
New content	Added documentation about How AWS CloudTrail Uses AWS KMS (p. 127).	November 18, 2015
New content	Added instructions for Changing a Key Policy (p. 43).	November 18, 2015

Change	Description	Date
Update	Updated the documentation about How Amazon Relational Database Service (Amazon RDS) Uses AWS KMS (p. 150).	November 18, 2015
New content	Added documentation about How Amazon WorkSpaces Uses AWS KMS (p. 169).	November 6, 2015
Update	Updated the Using Key Policies in AWS KMS (p. 33) page to improve clarity.	October 22, 2015
New content	Added documentation about Deleting Customer Master Keys (p. 110), including supporting documentation about Creating an Amazon CloudWatch Alarm (p. 117) and Determining Past Usage of a Customer Master Key (p. 120).	October 15, 2015
New content	Added documentation about Determining Access to an AWS KMS Customer Master Key (p. 80).	October 15, 2015
New content	Added documentation about How Key State Affects Use of a Customer Master Key (p. 123).	October 15, 2015
New content	Added documentation about How Amazon Simple Email Service (Amazon SES) Uses AWS KMS (p. 158).	October 1, 2015
Update	Updated the Limits (p. 255) 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. 1).	August 14, 2015
New content	Added requests per second to the AWS KMS Limits (p. 255).	June 11, 2015
New content	Added a new Java code sample demonstrating use of the UpdateAlias API. See Updating an Alias (p. 242).	June 1, 2015
Update	Moved the AWS Key Management Service regions table to the <i>AWS General Reference</i> .	May 29, 2015

Change	Description	Date
New content	Added documentation about How Amazon EMR Uses AWS KMS (p. 145).	January 28, 2015
New content	Added documentation about How Amazon WorkMail Uses AWS KMS (p. 167).	January 28, 2015
New content	Added documentation about How Amazon Relational Database Service (Amazon RDS) Uses AWS KMS (p. 150).	January 6, 2015
New content	Added documentation about How Amazon Elastic Transcoder Uses AWS KMS (p. 141).	November 24, 2014
New guide	Introduced the <i>AWS Key Management Service Developer Guide</i> .	November 12, 2014

Limits

AWS KMS resources have limits that apply to each region and each AWS account. Some limits apply to all resources. Others apply only to resources that you create, but not to resources that AWS services create in your account. Resources that you use, but that aren't in your AWS account, such as [AWS owned CMKs \(p. 3\)](#), do not count against these limits.

Important

If you need to exceed these limits, please visit the [AWS Support Center](#) and create a case.

Resource	Default Limit	Applies To
Customer Master Keys (CMKs) (p. 255)	1000	Customer-managed CMKs
Aliases (p. 255)	1100	Customer-created aliases
Key policy document size (p. 256)	32 KB (32,768 bytes)	Customer managed CMKs AWS managed CMKs
Grants per CMK (p. 256)	2500	Customer-managed CMKs
Grants for a given principal per CMK (p. 256)	500	Customer managed CMKs AWS managed CMKs
Requests per second (p. 256)	Varies by API operation; see table (p. 257) .	Customer managed CMKs AWS managed CMKs

Note

If you are exceeding the [requests per second \(p. 256\)](#) limit, consider using the [data key caching](#) feature of the AWS Encryption SDK. Reusing data keys, rather than requesting a new data key for every encryption operation, might reduce the frequency of your requests to AWS KMS.

Customer Master Keys (CMKs): 1000

You can have up to 1000 [customer managed CMKs \(p. 3\)](#) in each Region of your AWS account. This limit applies to all customer managed CMKs regardless of their [key state \(p. 123\)](#). [AWS managed CMKs \(p. 3\)](#) and [AWS owned CMKs \(p. 3\)](#) do not count against this limit.

You can create a [support case](#) to 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

You can create up to 1100 aliases in each region of your account. Aliases that AWS creates in your account, such as `aws/<service-name>`, do not count against this limit.

An *alias* is a display name that you can map to a CMK. Each alias is mapped to exactly one CMK and multiple aliases can map to the same CMK.

If you use a [support case](#) to increase your CMK limit, you might also need to request an increase in the number of aliases.

Key policy document size: 32 KB

The maximum length of each key policy document is 32 KB (32,768 bytes). If the document exceeds this length, operations that use the key policy document to set or change the key policy fail. If you must exceed this limit, create a [support case](#).

A [key policy document](#) (p. 33) is a collection of policy statements in JSON format. The statements in the key policy document determine who has permission to use the CMK and how they can use it. You may also use IAM policies and grants to control access to the CMK, but every CMK must have a key policy document.

You can create a key policy document by using the [default view](#) (p. 43) or [policy view](#) (p. 45) in the AWS Management Console, or by using the [PutKeyPolicy](#) API operation. All of these techniques involve an underlying key policy document.

Grants per CMK: 2500

Each [customer managed CMK](#) (p. 3) can have up to 2500 grants, including the grants created by [AWS services that are integrated with AWS KMS](#). This limit does not apply to [AWS managed CMKs](#) (p. 3).

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. 139) that use the same CMK.

[Grants](#) (p. 78) are an alternative to [key policy](#) (p. 33). They are advanced mechanisms for specifying permissions.

You or an AWS service integrated with AWS KMS can use a grant to limit how and when the grantee can use a CMK. Grants are attached to a CMK, and each grant includes the principal who receives permission to use the CMK, the ID of the CMK, and a list of operations that the grantee can perform.

Grants for a given principal per CMK: 500

For a given CMK, no more than 500 grants can specify the same grantee principal. This limit applies to all CMKs, including [AWS managed CMKs](#) (p. 3).

For example, you might want to encrypt multiple Amazon EBS volumes and attach them to a single Amazon Elastic Compute Cloud (Amazon EC2) instance. 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. For each CMK, you can have up to 500 grants that specify the same EC2 instance as the grantee principal. This effectively means that you can have no more than 500 encrypted EBS volumes per EC2 instance for a given CMK.

Requests per second: varies

AWS KMS throttles API requests at different limits depending on the API operation. Throttling means that AWS KMS rejects an otherwise valid request because the request exceeds the limit for the number of

requests per second. When a request is throttled, AWS KMS returns a `ThrottlingException` error. [The following table \(p. 257\)](#) lists each API operation and the point at which AWS KMS throttles requests for that operation.

This limit applies to all CMKs, including [AWS managed CMKs \(p. 3\)](#).

Note

If you need to exceed these limits, please visit the [AWS Support Center](#) and create a case. If you are exceeding the requests per second limit for the `GenerateDataKey` API operation, consider using the [data key caching](#) feature of the AWS Encryption SDK. Reusing data keys might reduce the frequency of your requests to AWS KMS.

Shared limit

The API operations in the first row of the following table share a limit of 5500 (or 10,000) requests per second. For example, with a shared limit of 5500 requests per second, when you make 3000 `GenerateDataKey` requests per second and 1000 `Decrypt` requests per second, AWS KMS doesn't throttle your requests. However, when you make 5000 `GenerateDataKey` and 1000 `Encrypt` and requests per second, AWS KMS throttles your requests because you are making more than 5500 requests per second for operations with the shared limit.

The remaining API operations have a unique limit for requests per second, which means the limit is not shared.

API requests made on your behalf

You can make API requests directly or by using an integrated AWS service that makes API requests to AWS KMS on your behalf. The limit applies to both kinds of requests.

For example, you might store data in Amazon S3 using server-side encryption with AWS KMS (SSE-KMS). 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) request to AWS KMS on your behalf. These requests count toward your limit, so AWS KMS throttles the requests if you exceed a combined total of 5500 (or 10,000) uploads or downloads per second of S3 objects encrypted with SSE-KMS.

Cross-account requests

When an application in one AWS account uses a CMK owned by a different account, that's known as a cross-account request. For cross-account requests, AWS KMS throttles the account that makes the requests, not the account that owns the CMK. For example, you might have applications in accounts A and B that both use a CMK in account C. In this scenario, the limit for requests per second applies separately to accounts A and B, not to account C.

Requests per second limit for each AWS KMS API operation

API operation	Requests per second limit
<code>Decrypt</code> <code>Encrypt</code> <code>GenerateDataKey</code> <code>GenerateDataKeyWithoutPlaintext</code> <code>GenerateRandom</code> <code>ReEncrypt</code>	5500 (shared) 10,000 (shared) only in the following regions: <ul style="list-style-type: none">US East (N. Virginia), us-east-1US West (Oregon), us-west-2EU (Ireland), eu-west-1
<code>CancelKeyDeletion</code>	5

API operation	Requests per second limit
CreateAlias	5
CreateGrant	50
CreateKey	5
DeleteAlias	5
DeleteImportedKeyMaterial	5
DescribeKey	30
DisableKey	5
DisableKeyRotation	5
EnableKey	5
EnableKeyRotation	5
GetKeyPolicy	30
GetKeyRotationStatus	30
GetParametersForImport	0.25 (AWS KMS throttles requests when the rate is more than 1 per 4 seconds)
ImportKeyMaterial	5
ListAliases	5
ListGrants	5
ListKeyPolicies	5
ListKeys	5
ListResourceTags	5
ListRetirableGrants	5
PutKeyPolicy	5
RetireGrant	15
RevokeGrant	15
ScheduleKeyDeletion	5
TagResource	5
UntagResource	5
UpdateAlias	5
UpdateKeyDescription	5