AWS Key Management Service

Developer Guide

AWS Key Management Service: Developer Guide

Copyright © 2020 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) ... 3
  - Data keys ... 5
  - Data key pairs ... 6
  - Cryptographic operations ... 10
  - Key identifiers (KeyId) ... 11
  - Key material origin ... 17
  - Key spec ... 17
  - Key usage ... 18
  - Envelope encryption ... 18
  - Encryption context ... 19
  - Key policies ... 22
  - Grants ... 23
  - Grant tokens ... 23
  - Auditing CMK usage ... 23
  - Key management infrastructure ... 23

Getting started ... 24
- Creating keys ... 24
  - Creating symmetric CMKs ... 25
  - Creating asymmetric CMKs ... 27
- Viewing keys ... 29
  - Viewing CMKs in the console ... 30
  - Viewing CMKs with the API ... 36
  - Finding the key ID and ARN ... 40
  - Finding the alias name and alias ARN ... 41
  - Identifying symmetric and asymmetric CMKs ... 43
- Editing keys ... 46
  - Editing CMKs (console) ... 46
  - Editing CMKs (AWS KMS API) ... 48
- Tagging keys ... 49
  - Managing CMK tags (console) ... 50
  - Managing CMK tags (AWS KMS API) ... 50
- Enabling and disabling keys ... 51
  - Enabling and disabling CMKs (console) ... 52
  - Enabling and disabling CMKs (AWS KMS API) ... 52
- Downloading public keys ... 53
  - Special considerations for downloading public keys ... 53
  - Downloading a public key (console) ... 54
  - Downloading a public key (AWS KMS API) ... 54

Authentication and access control ... 56
- Authentication ... 56
- Access control ... 57
- Overview of managing access ... 57
  - AWS KMS resources and operations ... 58
  - Managing access to AWS KMS CMKs ... 58
  - Specifying permissions in a policy ... 59
  - Specifying conditions in a policy ... 59
- Using key policies ... 59
  - Overview of key policies ... 60
  - Default key policy ... 61
  - Example key policy ... 68
  - Viewing a key policy ... 71
  - Changing a key policy ... 74

Keeping key policies up to date ... 76
Using IAM policies ... 77
Overview of IAM policies ... 78
Specifying CMKs in IAM policy statements ... 78
Permissions required to use the AWS KMS console ... 79
AWS managed (predefined) policies for AWS KMS ... 80
Customer managed policy examples ... 80
Allowing cross-account access to a CMK ... 82
Step 1: Add a key policy statement in the local account ... 83
Step 2: Add IAM policies in the external account ... 85
Creating CMKs that other accounts can use ... 85
Using external CMKs with AWS services ... 87
AWS KMS API permissions reference ... 87
Using policy conditions ... 98
AWS global condition keys ... 98
AWS KMS condition keys ... 99
Using grants ... 126
Create a grant ... 127
Grants for symmetric and asymmetric CMKs ... 128
Grant constraints ... 128
Authorizing CreateGrant in a key policy ... 129
Granting CreateGrant permission ... 129
Using service-linked roles ... 129
Service-linked role permissions for AWS KMS custom key stores ... 130
Determining access ... 130
Examining the key policy ... 130
Examining IAM policies ... 133
Examining grants ... 134
Troubleshooting key access ... 135
Using symmetric and asymmetric keys ... 141
About symmetric and asymmetric CMKs ... 142
Symmetric customer master keys ... 142
Asymmetric customer master keys ... 142
How to choose your CMK configuration ... 143
Selecting the key usage ... 144
Selecting the key spec ... 145
Viewing the cryptographic configuration of CMKs ... 149
Comparing symmetric and asymmetric CMKs ... 150
Rotating keys ... 154
How automatic key rotation works ... 155
How to enable and disable automatic key rotation ... 155
Enabling and disabling key rotation (console) ... 156
Enabling and disabling key rotation (AWS KMS API) ... 156
Rotating keys manually ... 157
Importing key material ... 159
About imported key material ... 159
How to import key material ... 160
How to reimport key material ... 160
How to identify CMKs with imported key material ... 161
To identify the value of the `Origin` property of CMKs (console) ... 161
To identify the value of the `Origin` property of CMKs (AWS KMS API) ... 161
Step 1: Create a CMK with no key material ... 162
Creating a CMK with no key material (console) ... 162
Creating a CMK with no key material (AWS KMS API) ... 163
Step 2: Download the public key and import token ... 164
Downloading the public key and import token (console) ... 165
Downloading the public key and import token (AWS KMS API) ... 166

Step 3: Encrypt the key material ... 167
Example: Encrypt key material with OpenSSL ... 167
Step 4: Import the key material ... 168
Import key material (console) ... 168
Import key material (AWS KMS API) ... 169
Deleting key material ... 169
How deleting key material affects AWS services integrated with AWS KMS ... 170
Delete key material (console) ... 170
Delete key material (AWS KMS API) ... 171
Deleting customer master keys ... 172
How deleting CMKs works ... 172
Deleting asymmetric CMKs ... 173
How deleting CMKs affects integrated AWS services ... 173
Scheduling and canceling key deletion ... 174
Using the AWS Management Console ... 174
Using the AWS CLI ... 175
Using the AWS SDK for Java ... 175
Adding permission to schedule and cancel key deletion ... 176
Using the AWS Management Console ... 176
Using the AWS CLI ... 177
Creating an Amazon CloudWatch alarm ... 177
Requirements for a CloudWatch alarm ... 178
Create the CloudWatch alarm ... 178
Determining past usage of a CMK ... 181
Examining CMK permissions to determine the scope of potential usage ... 181
Examining AWS CloudTrail logs to determine actual usage ... 181
Using a custom key store ... 184
What is a custom key store? ... 186
AWS KMS custom key store ... 186
AWS CloudHSM cluster ... 187
`kmsuser` Crypto user ... 187
CMKs in a custom key store ... 188
Controlling access to your custom key store ... 188
Authorizing custom key store managers and users ... 188
Authorizing AWS KMS to manage AWS CloudHSM and Amazon EC2 resources ... 189
Creating a custom key store ... 190
Assemble the prerequisites ... 191
Create a custom key store (console) ... 192
Create a custom key store (API) ... 193
Managing a custom key store ... 194
Viewing a custom key store ... 194
Editing custom key store settings ... 196
Connecting and disconnecting a custom key store ... 198
Deleting a custom key store ... 202
Managing CMKs in a custom key store ... 204
Creating CMKs in a custom key store ... 204
Viewing CMKs in a custom key store ... 208
Using CMKs in a custom key store ... 210
Finding CMKs and key material ... 210
Scheduling deletion of CMKs from a custom key store ... 214
Troubleshooting a custom key store ... 215
How to fix unavailable CMKs ... 215
How to fix a failing CMK ... 216
How to fix a connection failure ... 216
How to fix invalid `kmsuser` credentials ... 217
How to delete orphaned key material ... 218
How to recover deleted key material for a CMK ... 219

How to log in as `kmsuser` ... 220
Using a VPC endpoint ... 223
Considerations for AWS KMS VPC endpoints ... 223
Creating a VPC endpoint for AWS KMS ... 223
Connecting to a VPC endpoint ... 224
Using a VPC endpoint in a policy statement ... 224
Auditing your VPC endpoint ... 226
Using hybrid post-quantum TLS ... 228
About post-quantum TLS ... 229
How to use it ... 229
How to configure it ... 230
How to test it ... 231
Learn more ... 231
Key state: Effect on your CMK ... 233
How AWS services use AWS KMS ... 238
AWS CloudTrail ... 238
Understanding when your CMK is used ... 238
Understanding how often your CMK is used ... 242
Amazon DynamoDB ... 243
Client-side encryption for DynamoDB ... 243
Using CMKs and data keys ... 243
Authorizing use of your CMK ... 245
DynamoDB encryption context ... 249
Monitoring DynamoDB interaction with AWS KMS ... 249
Amazon Elastic Block Store (Amazon EBS) ... 253
Amazon EBS encryption ... 253
Using CMKs and data keys ... 253
Amazon EBS encryption context ... 254
Detecting Amazon EBS failures ... 254
Using AWS CloudFormation to create encrypted Amazon EBS volumes ... 255
Amazon Elastic Transcoder ... 255
Encrypting the input file ... 255
Decrypting the input file ... 256
Encrypting the output file ... 257
HLS content protection ... 258
Elastic Transcoder encryption context ... 258
Amazon EMR ... 259
Encrypting data on the EMR file system (EMRFS) ... 259
Encrypting data on the storage volumes of cluster nodes ... 261
Encryption context ... 262
Amazon Redshift ... 262
Amazon Redshift encryption ... 262
Encryption context ... 263
Amazon Relational Database Service (Amazon RDS) ... 263
Amazon RDS encryption context ... 264
AWS Secrets Manager ... 264
Protecting the secret value ... 265
Encrypting and decrypting secrets ... 265
Using your AWS KMS CMK ... 267
Authorizing use of the CMK ... 268
Secrets Manager encryption context ... 269
Monitoring Secrets Manager interaction with AWS KMS ... 270
Amazon Simple Email Service (Amazon SES) ... 272
Overview of Amazon SES encryption using AWS KMS ... 272
Amazon SES encryption context ... 273
Giving Amazon SES permission to use your AWS KMS customer master key (CMK) ... 273
Getting and decrypting email messages ... 274

Amazon Simple Storage Service (Amazon S3) ... 274
Server-Side Encryption: Using SSE-KMS ... 275
Using the Amazon S3 encryption client ... 276
Encryption context ... 276
AWS Systems Manager Parameter Store ... 276
Protecting standard secure string parameters ... 277
Protecting advanced secure string parameters ... 279
Setting permissions to encrypt and decrypt parameter values ... 282
Parameter Store encryption context ... 283
Troubleshooting CMK issues in Parameter Store ... 285
Amazon WorkMail ... 285
Amazon WorkMail overview ... 285
Amazon WorkMail encryption ... 286
Authorizing use of the CMK ... 288
Amazon WorkMail encryption context ... 290
Monitoring Amazon WorkMail interaction with AWS KMS ... 290
Amazon WorkSpaces ... 292
Overview of Amazon WorkSpaces encryption using AWS KMS ... 292
Amazon WorkSpaces encryption context ... 293
Giving Amazon WorkSpaces permission to use a CMK on your behalf ... 293
Monitoring customer master keys ... 296
Monitoring tools ... 296
Automated tools ... 296
Manual tools ... 297
Monitoring with CloudWatch ... 297
Metrics and dimensions ... 298
Creating alarms ... 299
AWS KMS events ... 300
Logging AWS KMS API calls with AWS CloudTrail ... 303
AWS KMS information in CloudTrail ... 303
Excluding AWS KMS events from a trail ... 304
Understanding AWS KMS log file entries ... 304
CreateAlias ... 305
CreateGrant ... 306
CreateKey ... 307
Decrypt ... 308
DeleteAlias ... 308
DescribeKey ... 309
DisableKey ... 310
EnableKey ... 311
Encrypt ... 312
GenerateDataKey ... 312
GenerateDataKeyWithoutPlaintext ... 313
GenerateRandom ... 314
GetKeyPolicy ... 314
ListAliases ... 315
ListGrants ... 316
ReEncrypt ... 316
Amazon EC2 example one ... 317
Amazon EC2 example two ... 319
Programming the AWS KMS API ... 324
Creating a client ... 324
Working with keys ... 325
Creating a customer master key ... 325
Generating a data key ... 327
Viewing a custom master key ... 329
Getting key IDs and ARNs ... 331

Enabling customer master keys ... 333
Disabling customer master keys ... 334
Encrypting and decrypting data keys ... 336
Encrypting a data key ... 336
Decrypting a data key ... 339
Re-encrypting a data key under a different customer master key ... 341
Working with key policies ... 343
Listing key policy names ... 343
Getting a key policy ... 345
Setting a key policy ... 347
Working with grants ... 351
Creating a grant ... 352
Viewing a grant ... 354
Retiring a grant ... 356
Revoking a grant ... 358
Working with aliases ... 360
Creating an alias ... 360
Listing aliases ... 362
Updating an alias ... 366
Deleting an alias ... 368
Quotas ... 370
Resource quotas ... 370
Customer master keys (CMKs): 10,000 ... 371
Aliases: 10,000 ... 371
Grants per CMK: 10,000 ... 371
Grants for a given principal per CMK: 500 ... 371
Key policy document size: 32 KB ... 372
Request quotas ... 372
Applying request quotas ... 373
Shared quotas for cryptographic operations ... 373
API requests made on your behalf ... 374
Cross-account requests ... 374
Custom key store quota ... 374
Request quotas for each AWS KMS API operation ... 375
Request an AWS KMS Quota Increase ... 378
Using the Service Quotas console ... 378
Using the Service Quotas API ... 378
Document history ... 380
Recent updates ... 380
Earlier updates ... 381

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 *customer master keys* (CMKs), the encryption keys used to encrypt your data. AWS KMS CMKs are protected by hardware security modules (HSMs) that are validated by the FIPS 140-2 Cryptographic Module Validation Program except in the China (Beijing) and China (Ningxia) Regions.

AWS KMS is integrated with most other AWS services that encrypt your data. AWS KMS is also integrated with AWS CloudTrail to log use of your CMKs for auditing, regulatory, and compliance needs.

You can perform the following basic management actions on your AWS KMS customer master keys (CMKs):

- Create (p. 24), and view (p. 29) symmetric and asymmetric CMKs (p. 141), and edit (p. 46) their properties.
- Enable and disable (p. 51) CMKs
- Create and view access control policies (p. 56) and grants (p. 126) for your CMKs
- Enable and disable automatic rotation (p. 154) of the cryptographic material in a CMK
- Tag your CMKs (p. 49) for easier identification, categorizing, and tracking use and costs
- Create, delete, list, and update *aliases* (p. 360), which are friendly names for your CMKs
- Delete CMKs (p. 172) to complete the key lifecycle

You can also perform the following cryptographic operations (p. 10) with your CMKs. For examples, see Programming the AWS KMS API (p. 324).

- Encrypt, decrypt, and re-encrypt data with symmetric or asymmetric CMKs
- Sign and verify messages with asymmetric CMKs
- Generate exportable symmetric data keys and asymmetric data key pairs
- Generate random numbers suitable for cryptographic applications

You can also use the advanced features of AWS KMS

- Import cryptographic material (p. 159) into a CMK
- Create CMKs in your own custom key store (p. 184) backed by a AWS CloudHSM cluster.
- Connect directly to AWS KMS through a private endpoint in your VPC (p. 223)
- Use hybrid post-quantum TLS (p. 228) to provide forward-looking encryption in transit for the data that you send AWS KMS

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 customer 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

- To learn about the terms and concepts used in AWS KMS, see AWS KMS Concepts (p. 2).
- For information about the AWS KMS API, see the AWS Key Management Service API Reference. For examples in different programming languages, see Programming the AWS KMS API (p. 324).
- For detailed technical information about how AWS KMS uses cryptography and secures master keys, see the AWS Key Management Service Cryptographic Details whitepaper. This whitepaper does not describe how AWS KMS works in the China (Beijing) and China (Ningxia) Regions.
- For help with questions about AWS KMS, see the AWS Key Management Service Discussion Forum.

AWS KMS in AWS Regions

The AWS Regions in which AWS KMS is supported are listed in AWS Key Management Service Endpoints and Quotas. If an AWS KMS feature is not supported in an AWS Region that AWS KMS supports, the regional difference is described in the topic about the feature.

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.

Service level agreement

AWS Key Management Service is backed by a service level agreement that defines our service availability policy.

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. 3)
- Data keys (p. 5)
- Data key pairs (p. 6)
- Cryptographic operations (p. 10)
- Key identifiers (KeyId) (p. 11)
- Key material origin (p. 17)
- Key spec (p. 17)
- Key usage (p. 18)
- Envelope encryption (p. 18)
- Encryption context (p. 19)
- Key policies (p. 22)
- Grants (p. 23)
- Grant tokens (p. 23)
- Auditing CMK usage (p. 23)

- Key management infrastructure (p. 23)

Customer master keys (CMKs)

Customer master keys are the primary resources in AWS KMS.

A *customer master key* (CMK) is a logical representation of a master key. The CMK includes metadata, such as the key ID, creation date, description, and key state. The CMK also contains the key material used to encrypt and decrypt data.

AWS KMS supports symmetric and asymmetric CMKs. A *symmetric CMK* represents a 256-bit key that is used for encryption and decryption. An *asymmetric CMK* represents an RSA key pair that is used for encryption and decryption or signing and verification (but not both), or an elliptic curve (ECC) key pair that is used for signing and verification. For detailed information about symmetric and asymmetric CMKs, see Using symmetric and asymmetric keys (p. 141).

CMKs are created in AWS KMS. Symmetric CMKs and the private keys of asymmetric CMKs never leave AWS KMS unencrypted. To manage your CMK, you can use the AWS Management Console or the AWS KMS API. To use a CMK in cryptographic operations (p. 10), you must use the AWS KMS API. This strategy differs from data keys (p. 5). AWS KMS does not store, manage, or track your data keys. You must use them outside of AWS KMS.

By default, AWS KMS creates the key material for a CMK. You cannot extract, export, view, or manage this key material. Also, you cannot delete this key material; you must delete the CMK (p. 172). However, you can import your own key material (p. 159) into a CMK or create the key material for a CMK in the AWS CloudHSM cluster associated with an AWS KMS custom key store (p. 184).

For information about creating and managing CMKs, see *Getting started* (p. 24). For information about using CMKs, see the AWS Key Management Service API Reference.

AWS KMS supports three types of CMKs: customer managed CMKs, AWS managed CMKs, and AWS owned CMKs.

Type of CMK	Can view CMK metadata	Can manage CMK	Used only for my AWS account	Automatic rotation (p. 154)	
Customer managed CMK (p. 4)	Yes	Yes	Yes	Optional. Every 365 days (1 year).	
AWS managed CMK (p. 4)	Yes	No	Yes	Required. Every 1095 days (3 years).	
AWS owned CMK (p. 4)	No	No	No	Varies	

To distinguish customer managed CMKs from AWS managed CMKs, use the `KeyManager` field in the DescribeKey operation response. For customer managed CMKs, the `KeyManager` value is `Customer`. For AWS managed CMKs, the `KeyManager` value is `AWS`.

AWS services that integrate with AWS KMS (p. 238) differ in their support for CMKs. Some AWS services encrypt your data by default with an AWS owned CMK or an AWS managed CMK. Other AWS services offer to encrypt your data under a customer managed CMK that you choose. And other AWS services 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. For detailed information about the encryption options

that an AWS service offers, see the *Encryption at Rest* topic in the user guide or the developer guide for the service.

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. 56), enabling and disabling (p. 51) them, rotating their cryptographic material (p. 154), adding tags (p. 49), creating aliases (p. 360) that refer to the CMK, and scheduling the CMKs for deletion (p. 172).

Customer managed CMKs appear on the **Customer managed keys** page of the AWS Management Console for AWS KMS. To definitively identify a customer managed CMK, use the DescribeKey operation. For customer managed CMKs, the value of the `KeyManager` field of the `DescribeKey` response is `CUSTOMER`.

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. 238) 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 quotas (p. 370) for your account. For details, see AWS Key Management Service Pricing and Quotas (p. 370).

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 is integrated with AWS KMS. Some AWS services support only an AWS managed CMK

You can view the AWS managed CMKs (p. 29) in your account, view their key policies (p. 71), and audit their use (p. 303) in AWS CloudTrail logs. However, you cannot manage these CMKs, rotate them, or change their key policies. And, you cannot use AWS managed CMKs in cryptographic operations directly; the service that creates them uses them on your behalf.

AWS managed CMKs appear on the **AWS managed keys** page of the AWS Management Console for AWS KMS. You can also identify most AWS managed CMKs by their aliases, which have the format `aws/`*`service-name`*, such as `aws/redshift`. To definitively identify an AWS managed CMK, use the DescribeKey operation. For AWS managed CMKs, the value of the `KeyManager` field of the `DescribeKey` response is `AWS`.

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 at Rest* topic in the user guide or developer guide for the service. AWS managed CMKs do not count against resource quotas on the number of CMKs in each Region of your account. But when they are used on behalf of a principal in your account, these CMKs count against request quotas. For details, see AWS Key Management Service Pricing and Quotas (p. 370).

AWS owned CMKs

AWS owned CMKs are a collection of CMKs that an AWS service owns and manages for use in multiple AWS accounts. Although AWS owned CMKs are not in your AWS account, an AWS service can use its AWS owned CMKs to protect the resources in your account.

You do not need to create or manage the AWS owned CMKs. However, you cannot view, use, track, or audit them. You are not charged a monthly fee or usage fee for AWS owned CMKs and they do not count against the AWS KMS quotas (p. 370) for your account.

The key rotation (p. 154) strategy for an AWS owned CMK is determined by the AWS service that creates and manages the CMK. For information about the types of CMKs that an AWS service supports,

including AWS owned CMKs, see the *Encryption at Rest* topic in the user guide or developer guide for the service.

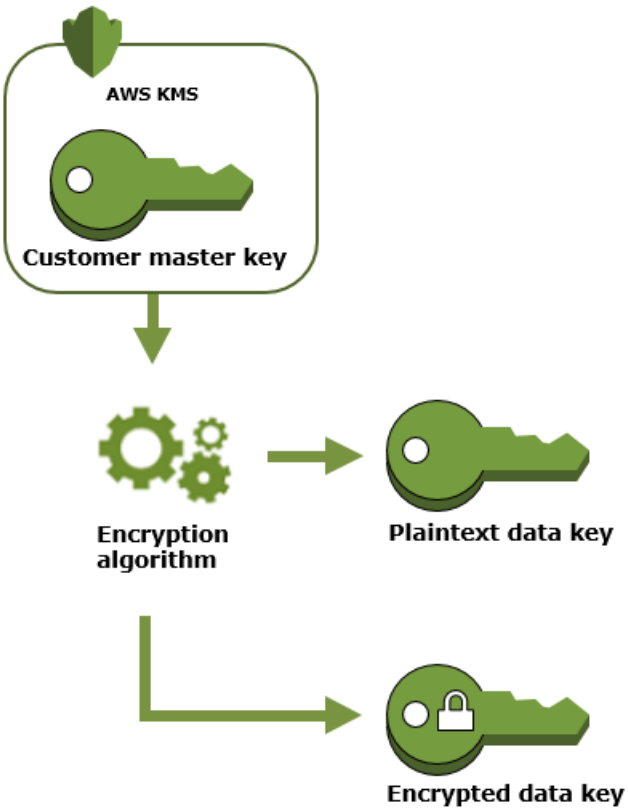
Data keys

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. 3) (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. The following image shows this operation.

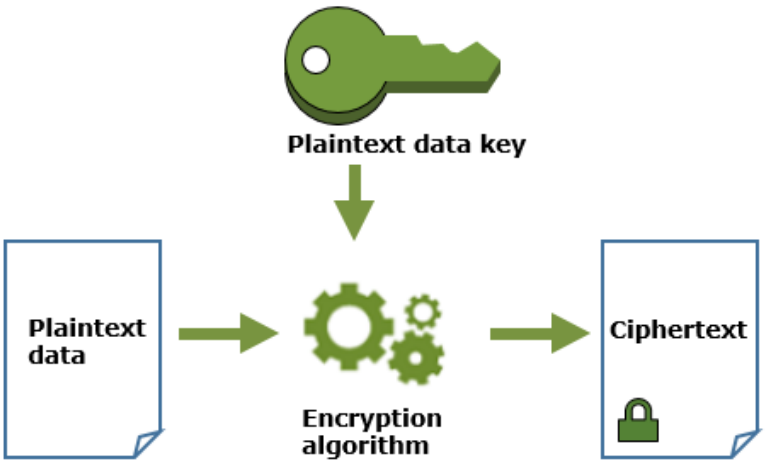


AWS KMS also supports the GenerateDataKeyWithoutPlaintext operation, which returns only an encrypted data key. When you need to use the data key, ask AWS KMS to decrypt it.

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.

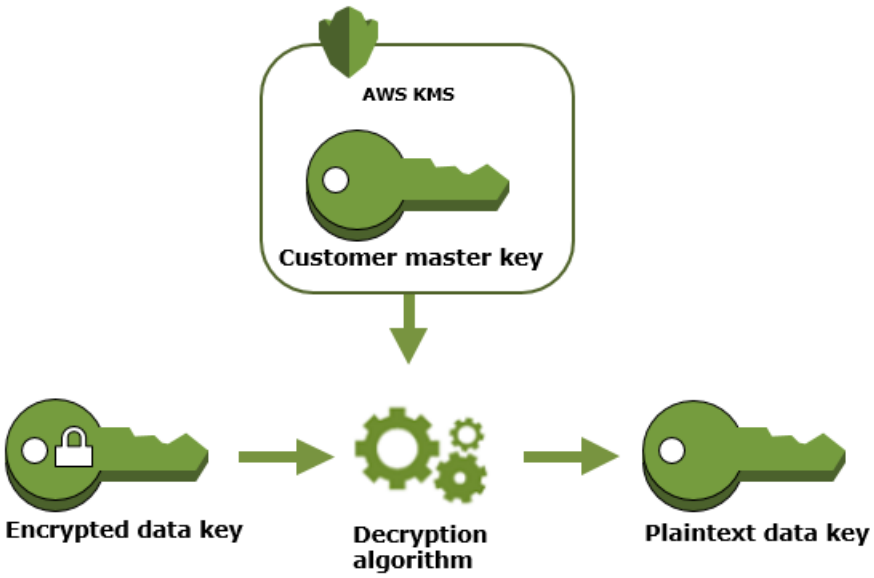
After using the plaintext data key to encrypt data, remove it from memory as soon as possible. You can safely store the encrypted data key with the encrypted data so it is available to decrypt the data.



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.



Data key pairs

Data key pairs are asymmetric data keys that consist of a mathematically-related public key and private key. They are designed to be used for client-side encryption and decryption or signing and verification outside of AWS KMS.

Unlike the data key pairs that tools like OpenSSL generate, AWS KMS protects the private key in each data key pair under a symmetric CMK in AWS KMS that you specify. However, AWS KMS does not store, manage, or track your data key pairs, or perform cryptographic operations with data key pairs. You must use and manage data key pairs outside of AWS KMS.

AWS KMS supports the following types of data key pairs:

- RSA key pairs: RSA_2048, RSA_3072, and RSA_4096
- Elliptic curve key pairs, ECC_NIST_P256, ECC_NIST_P384, ECC_NIST_P521, and ECC_SECG_P256K1

The type of data key pair that you select usually depends on your use case or regulatory requirements. Most certificates require RSA keys. Elliptic curve keys are often used for digital signatures. ECC_SECG_P256K1 keys are commonly used for cryptocurrencies.

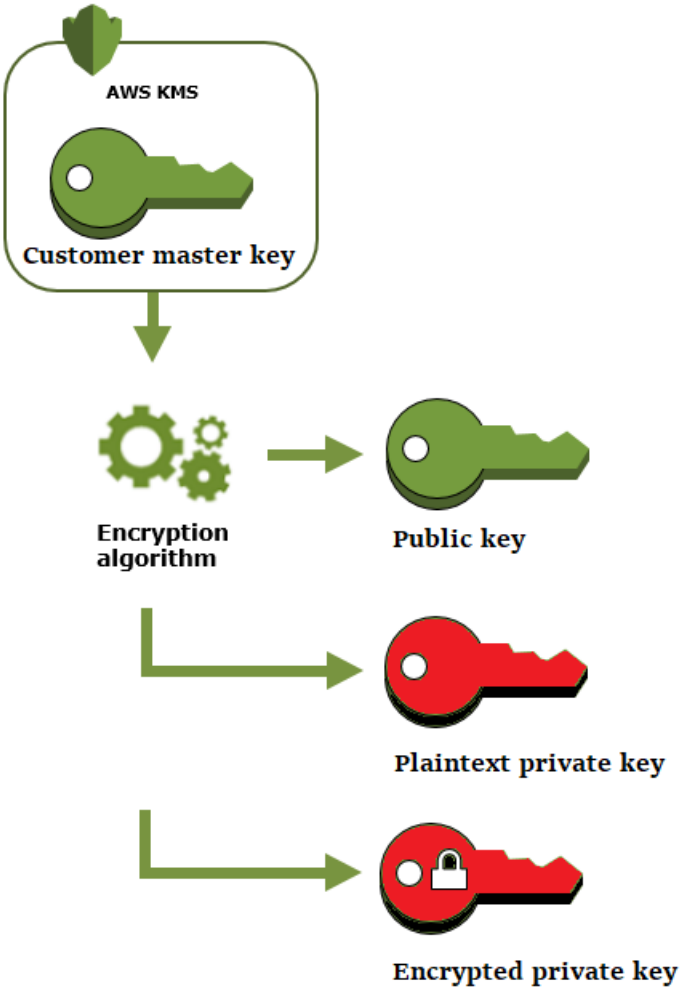
Create a data key pair

To create a data key pair, call the GenerateDataKeyPair or GenerateDataKeyPairWithoutPlaintext operations. Specify the symmetric CMK you want to use to encrypt the private key.

`GenerateDataKeyPair` returns a plaintext public key, a plaintext private key, and an encrypted private key. Use this operation when you need a plaintext private key immediately, such as to generate a digital signature.

`GenerateDataKeyPairWithoutPlaintext` returns a plaintext public key and an encrypted private key, but not a plaintext private key. Use this operation when you don't need a plaintext private key immediately, such as when you're encrypting with a public key. Later, when you need a plaintext private key to decrypt the data, you can call the Decrypt operation.

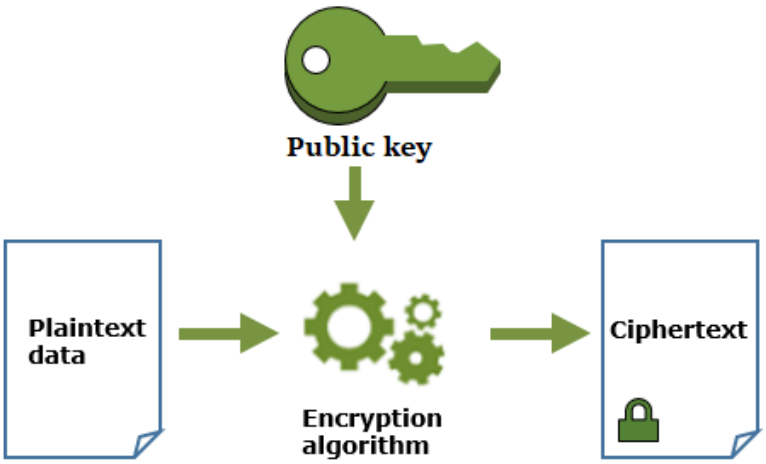
The following image shows the `GenerateDataKeyPair` operation. The `GenerateDataKeyWithoutPlaintext` operation omits the plaintext private key.



Encrypt data with a data key pair

When you encrypt with a data key pair, you use the public key of the pair to encrypt the data and the private key of the same pair to decrypt the data. Typically, data key pairs are used when many parties need to encrypt data that only the party that holds the private key can decrypt.

The parties with the public key use that key to encrypt data, as shown in the following diagram.

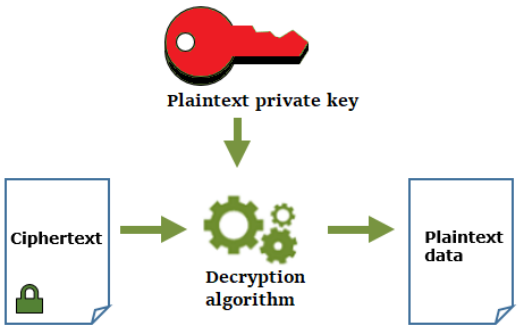


Decrypt data with a data key pair

To decrypt your data, use the private key in the data key pair. For the operation to succeed, the public and private keys must be from the same data key pair, and you must use the same encryption algorithm.

To decrypt the encrypted private key, pass it to the Decrypt operation. Use the plaintext private key to decrypt the data. Then remove the plaintext private key from memory as soon as possible.

The following diagram shows how to use the private key in a data key pair to decrypt ciphertext.



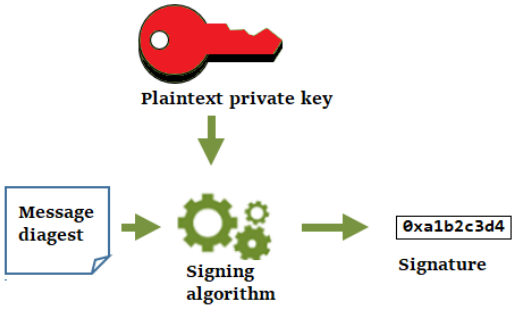
Sign messages with a data key pair

To generate a cryptographic signature for a message, use the private key in the data key pair. Anyone with the public key can use it to verify that the message was signed with your private key and that it has not changed since it was signed.

If your private key is encrypted, pass the encrypted private key to the Decrypt operation. AWS KMS uses your CMK to decrypt the data key and then it returns the plaintext private key. Use the plaintext private key to generate the signature. Then remove the plaintext private key from memory as soon as possible.

To sign a message, create a message digest using a cryptographic hash function, such as the dgst command in OpenSSL. Then, pass your plaintext private key to the signing algorithm. The result is a signature that represents the contents of the message.

The following diagram shows how to use the private key in a data key pair to sign a message.

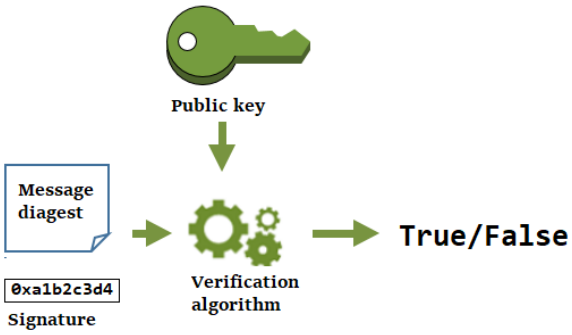


Verify a signature with a data key pair

Anyone who has the public key in your data key pair can use it to verify the signature that you generated with your private key. Verification confirms that an authorized user signed the message with the specified private key and signing algorithm, and the message hasn't changed since it was signed.

To be successful, the party verifying the signature must generate the same type of digest, use the same algorithm, and use the public key that corresponds to the private key used to sign the message.

The following diagram shows how to use the public key in a data key pair to verify a message signature.



Cryptographic operations

In AWS KMS, *cryptographic operations* are API operations that use CMKs to protect data. Because CMKs remain within AWS KMS, you must call AWS KMS to use a CMK in a cryptographic operation.

To perform cryptographic operations with CMKs, use the AWS SDKs, AWS Command Line Interface (AWS CLI), or the AWS Tools for PowerShell. You cannot perform cryptographic operations in the AWS KMS console. For examples of calling the cryptographic operations in several programming languages, see Programming the AWS KMS API (p. 324).

The following table lists the AWS KMS cryptographic operations. It also shows the key type and key usage (p. 18) requirements for CMKs used in the operation.

Operation	CMK key type	CMK key usage
Decrypt	Any	`ENCRYPT_DECRYPT`

Operation	CMK key type	CMK key usage
Encrypt	Any	`ENCRYPT_DECRYPT`
GenerateDataKey	Symmetric	`ENCRYPT_DECRYPT`
GenerateDataKeyPair	Symmetric [1]	`ENCRYPT_DECRYPT`
GenerateDataKeyPairWithoutPlaintext	Symmetric [1]	`ENCRYPT_DECRYPT`
GenerateDataKeyWithoutPlaintext	Symmetric	`ENCRYPT_DECRYPT`
GenerateRandom	N/A. This operation doesn't use a CMK.	N/A
ReEncrypt	Any	`ENCRYPT_DECRYPT`
Sign	Asymmetric	`SIGN_VERIFY`
Verify	Asymmetric	`SIGN_VERIFY`

[1] `GenerateDataKeyPair` and `GenerateDataKeyPairWithoutPlaintext` generate an asymmetric data key pair that is protected by a symmetric CMK.

For information about the permissions for cryptographic operations, see the the section called “AWS KMS API permissions reference” (p. 87).

To make AWS KMS responsive and performant for all users, AWS KMS establishes quotas on number of cryptographic operations that can be called in each second. For details, see the section called “Shared quotas for cryptographic operations” (p. 373).

Key identifiers (KeyId)

Key identifiers act as names for your AWS KMS customer master keys (CMKs). They help you to recognize your CMKs in the console. You use them to indicate which CMKs you want to use in AWS KMS API operations, IAM policies, and grants.

AWS KMS defines several key identifiers. When you create a CMK, AWS KMS generates a key ARN and key ID, which are properties of the CMK. When you create an alias, AWS KMS generates an alias ARN based on the alias name that you define. You can view the key and alias identifiers in the AWS Management Console and in the AWS KMS API.

In the AWS KMS console, you can view and filter CMKs by their key ARN, key ID, or alias name, and sort by key ID and alias name. For help finding the key identifiers in the console, see the section called “Finding the key ID and ARN” (p. 40).

In the AWS KMS API, the parameters that you use to identify a CMK are named `KeyId` or a variation, such as `TargetKeyId` or `DestinationKeyId`. However, the values of those parameters are not limited to key IDs. Some can take any valid key identifier. For information about the values for each parameter, see the parameter description in the AWS Key Management Service API Reference.

> **Note**
> When using the AWS KMS API, be careful about the key identifier that you use. Different APIs require different key identifiers. In general, use the most complete key identifier that is practical for your task.

AWS KMS supports the following key identifiers.

Key ARN

The key ARN is the Amazon Resource Name (ARN) of a CMK. It is a unique, fully qualified identifier for the CMK. A key ARN includes the AWS account, Region, and the key ID. For help finding the key ARN of a CMK, see the section called "Finding the key ID and ARN" (p. 40).

The format of a key ARN is as follows:

```
arn:<partition>:kms:<region>:<account-id>:key/<key-id>
```

The following is an example key ARN.

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

Key ID

The key ID uniquely identifies a CMK within an account and Region. For help finding the key ID of a CMK, see the section called "Finding the key ID and ARN" (p. 40).

The following is an example key ID.

```
1234abcd-12ab-34cd-56ef-1234567890ab
```

Alias ARN

The alias ARN is the Amazon Resource Name (ARN) of an AWS KMS alias. It is a unique, fully qualified identifier for the alias, and for the CMK it represents. An alias ARN includes the AWS account, Region, and the alias name.

At any given time, an alias ARN identifies one particular CMK. However, because you can change the CMK associated with the alias, the alias ARN can identify different CMKs at different times. For help finding the alias ARN of a CMK, see

To identify an AWS KMS customer master key (p. 3) (CMK) in a cryptographic operation (p. 10), you can use its key ID (p. 12), key ARN (p. 12), alias name (p. 14), or alias ARN (p. 12). In other AWS KMS operations, only the key ID or key ARN are valid.

For detailed information about the CMK identifiers that AWS KMS supports, see Key identifiers (KeyId) (p. 11).

Topics

- To find the alias name (console) (p. 41)
- To derive the alias ARN (console) (p. 42)
- To find the alias name and alias ARN (AWS KMS API) (p. 41)

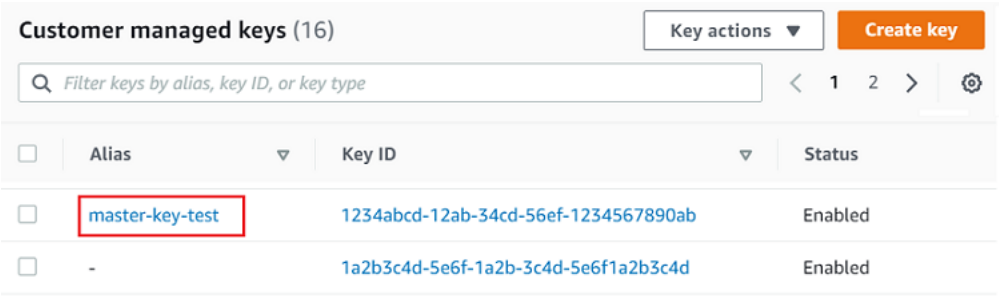
To find the alias name (console)

The AWS KMS console displays one alias name (p. 14) associated with the CMK. If the CMK was created in the console, the console displays the alias that was assigned to the CMK when it was created. To find other aliases associated with the CMK, if any, you need to use the AWS KMS API.

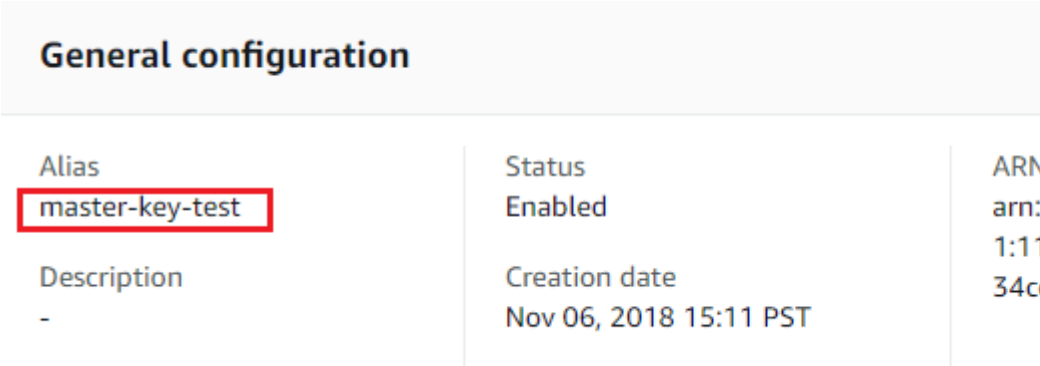
1. Open the AWS KMS console at https://console.aws.amazon.com/kms.

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 alias name for a CMK, see the **Alias** column in the row for each CMK. If a CMK does not have an alias, a dash (**-**) appears in the **Alias** column.



5. You can also find the alias on the details page for a CMK. To open the details page, choose the key ID or alias. The alias appears in the **General Configuration** section.



To derive the alias ARN (console)

The alias ARN (p. 12) is not displayed in the AWS KMS console. The **ARN** field displays the key ARN (p. 12). However, you can derive the alias ARN for this alias by using the values in the **Alias** and **ARN** fields.

To derive the alias ARN, begin with the key ARN.

```
# Key ARN format
arn:<partition>:kms:<region>:<account>:key/<key-id>

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

Replace `key` with `alias`. Replace the key ID with the alias name.

```
# Alias ARN format
arn:<partition>:kms:<region>:<account>:alias/<alias-name>
```

```
# Example alias ARN
arn:aws:kms:us-west-2:111122223333:alias/master-key-test
```

To find the alias name and alias ARN (AWS KMS API)

To find the alias name (p. 14) and alias ARN (p. 12) of a customer master key (CMK), use the ListAliases operation. For examples in multiple programming languages, see Listing aliases (p. 362) and Get alias names and ARNs (p. 38).

By default, the response includes the alias name and alias ARN for every alias in the account and Region. To get only the aliases for a particular CMK, use the `KeyId` parameter.

For example, the following command gets only the aliases for an example CMK with key ID `1234abcd-12ab-34cd-56ef-1234567890ab`.

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

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

(p. 41).

The format of an alias ARN is as follows:

```
arn:<partition>:kms:<region>:<account-id>:alias/<alias-name>
```


The following is the alias ARN for a fictitious `ExampleAlias`.

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

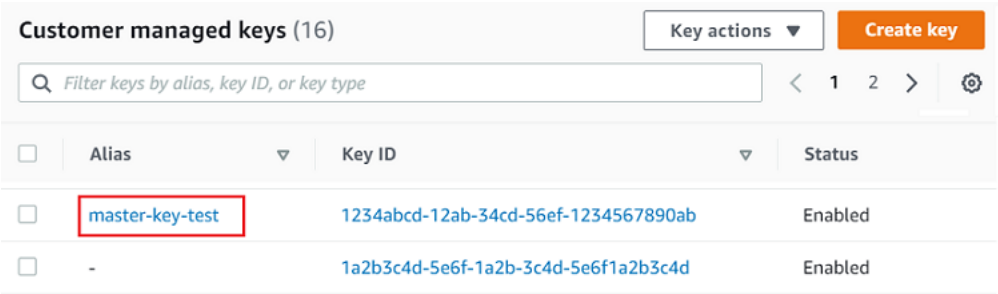
Topics

- To find the alias name (console) (p. 41)
- To derive the alias ARN (console) (p. 42)
- To find the alias name and alias ARN (AWS KMS API) (p. 41)

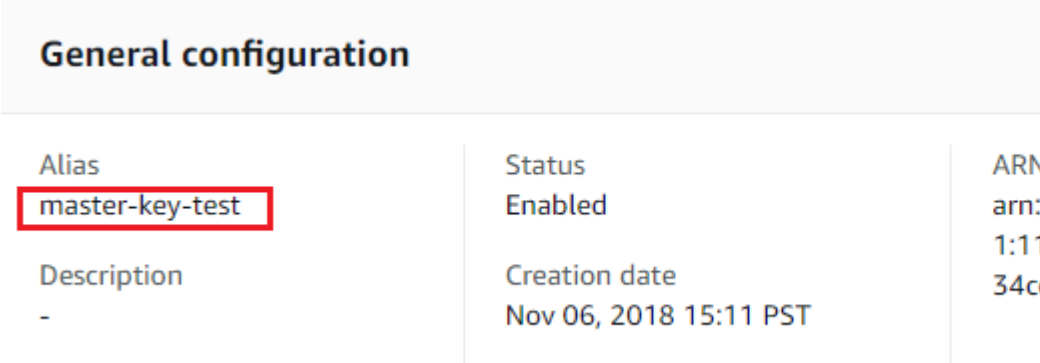
To find the alias name (console)

The AWS KMS console displays one alias name (p. 14) associated with the CMK. If the CMK was created in the console, the console displays the alias that was assigned to the CMK when it was created. To find other aliases associated with the CMK, if any, you need to use the AWS KMS API.

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 alias name for a CMK, see the **Alias** column in the row for each CMK. If a CMK does not have an alias, a dash (**-**) appears in the **Alias** column.



5. You can also find the alias on the details page for a CMK. To open the details page, choose the key ID or alias. The alias appears in the **General Configuration** section.



To derive the alias ARN (console)

The alias ARN (p. 12) is not displayed in the AWS KMS console. The **ARN** field displays the key ARN (p. 12). However, you can derive the alias ARN for this alias by using the values in the **Alias** and **ARN** fields.

To derive the alias ARN, begin with the key ARN.

```
# Key ARN format
arn:<partition>:kms:<region>:<account>:key/<key-id>

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

Replace `key` with `alias`. Replace the key ID with the alias name.

```
# Alias ARN format
arn:<partition>:kms:<region>:<account>:alias/<alias-name>

# Example alias ARN
arn:aws:kms:us-west-2:111122223333:alias/master-key-test
```

To find the alias name and alias ARN (AWS KMS API)

To find the alias name (p. 14) and alias ARN (p. 12) of a customer master key (CMK), use the ListAliases operation. For examples in multiple programming languages, see Listing aliases (p. 362) and Get alias names and ARNs (p. 38).

By default, the response includes the alias name and alias ARN for every alias in the account and Region. To get only the aliases for a particular CMK, use the `KeyId` parameter.

For example, the following command gets only the aliases for an example CMK with key ID `1234abcd-12ab-34cd-56ef-1234567890ab`.

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

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

The `aws/` prefix for an alias name is reserved for AWS managed CMKs (p. 4). You cannot create an alias with this prefix. For example, the alias name of the AWS managed CMK for Amazon Simple Storage Service (Amazon S3) is the following.

```
alias/aws/s3
```

Key material origin

Key material origin is a CMK property that identifies the source of the key material in the CMK. You choose the key material origin when you create the CMK, and you cannot change it. To find the key material origin of a CMK, use the DescribeKey operation, or see the **Origin** value in the **Cryptographic configuration** section of the detail page for a CMK in the AWS KMS console. For help, see Viewing Keys (p. 29).

CMKs can have one of the following key material origin values.

KMS (default)

API value: `AWS_KMS`

AWS KMS creates and manages the key material for the CMK in its own key store. This is the default and the recommended value for most CMKs.

For help creating keys with key material from AWS KMS, see Creating keys (p. 24).

External

API value: `EXTERNAL`

The CMK has imported key material (p. 159). When you create a CMK with an `External` key material origin, the CMK has no key material. Later, you can import key material into the CMK. When you use imported key material, you need to secure and manage that key material outside of AWS KMS, including replacing the key material if it expires. For details, see About imported key material (p. 159).

For help creating a CMK for imported key material, see Step 1: Create a CMK with no key material (p. 162).

Custom key store (CloudHSM)

API value: `AWS_CLOUDHSM`

AWS KMS created the key material for the CMK in your custom key store (p. 184).

For help creating a CMK in a custom key store, see Creating CMKs in a custom key store (p. 204)

Key spec

Key spec is a CMK property that represents cryptographic configuration of the CMK. The key spec determines whether the CMK is symmetric or asymmetric, the type of key material in the CMK, and the encryption algorithms or signing algorithms you can use with the CMK.

Typically, the key spec that you choose for your CMK is based on your use case and regulatory requirements. You choose the key spec when you create the CMK (p. 24), and you cannot change it. If you've chosen the wrong key spec, delete the CMK (p. 172), and create a new one.

For a list of key specs and help with choosing a key spec, see Selecting the key spec (p. 145). To find the key spec of a CMK, use the DescribeKey operation, or see the **Cryptographic configuration** section of the detail page for a CMK in the AWS KMS console. For help, see Viewing Keys (p. 29).

> **Note**
> In AWS KMS API operations, the key spec for CMKs is known as the `CustomerMasterKeySpec`. This distinguishes it from the key spec for data keys (`KeySpec`) and data key pairs (`KeyPairSpec`), and the key spec used when wrapping key material for import (`WrappingKeySpec`). Each key spec type has different values.

To limit the key specs that principals can use when creating CMKs, use the kms:CustomerMasterKeySpec (p. 102) condition key. You can also use the `kms:CustomerMasterKeySpec` condition key to allow principals to call AWS KMS operations for a CMK based on its key spec. For example, you can deny permission to schedule deletion of CMK with an `RSA_4096` key spec.

Key usage

Key usage is a CMK property that determines whether a CMK is used for encryption and decryption -or- signing and verification. You cannot choose both. Using a CMK for more than one type of operations makes the product of both operations more vulnerable to attack.

The key usage for symmetric CMKs is always encryption and decryption. The key usage for elliptic curve (ECC) CMKs is always signing and verification. You only need to choose a key usage for RSA CMKs. You choose the key usage when you create the CMK (p. 24), and you cannot change it. If you've chosen the wrong key usage, delete the CMK (p. 172), and create a new one.

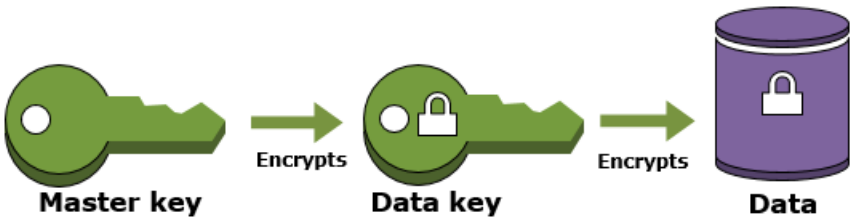
For choosing the key usage, see Selecting the key usage (p. 144). To find the key usage of a CMK, use the DescribeKey operation, or see the **Cryptographic configuration** section of the detail page for a CMK in the AWS KMS console. For help, see Viewing Keys (p. 29).

To allow principals to create CMKs only for signing and verification or only for encryption and decryption, use the kms:CustomerMasterKeyUsage (p. 104) condition key. You can also use the `kms:CustomerMasterKeyUsage` condition key to allow principals to call API operations for a CMK based on its key usage. For example, you can allow permission to disable a CMK only if its key usage is SIGN_VERIFY.

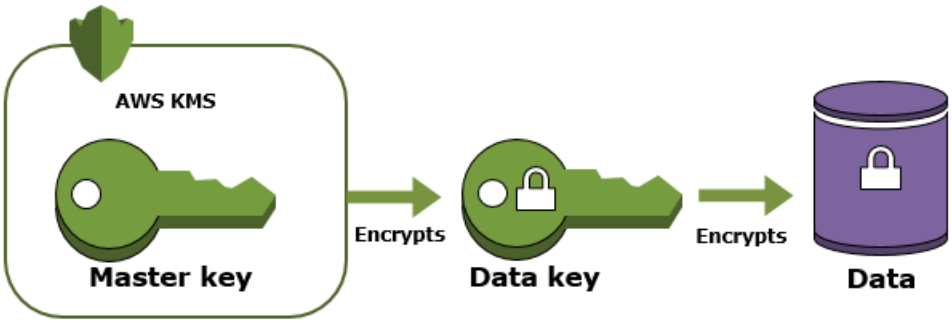
Envelope encryption

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 under 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. 3) (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 (p. 10) with symmetric CMKs accept an *encryption context*, an optional set of key–value pairs that can contain additional contextual information about the data. AWS KMS uses the encryption context as additional authenticated data (AAD) to support authenticated encryption.

You cannot specify an encryption context in a cryptographic operation with an asymmetric CMK (p. 142). The standard asymmetric encryption algorithms that AWS KMS uses do not support an encryption context.

When you include an encryption context in an encryption request, it is cryptographically bound to the ciphertext such that the same encryption context is 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 the key-value pairs in the encryption context can vary.

The encryption context is not secret. It appears in plaintext in AWS CloudTrail Logs (p. 303) so you can use it to identify and categorize your cryptographic operations.

An encryption context can consist of any keys and values. However, because it is not secret and not encrypted, your encryption context should not include sensitive information. We recommend that your encryption context describe the data being encrypted or decrypted. For example, when you encrypt a file, you might use part of the file path as encryption context.

The key and value in an encryption context pair must be simple literal strings. They cannot be integers or objects, or any type that is not fully resolved. If you use a different type, such as an integer or float, AWS KMS interprets it as a string.

```
"encryptionContext": {
    "department": "10103.0"
}
```

The encryption context key and value can include special characters, such as underscores (_), dashes (-), slashes (/, \) and colons (:).

For example, Amazon Simple Storage Service (p. 276) (Amazon S3) uses an encryption context in which the key is `aws:s3:arn`. 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. 126) and as a condition in policy statements (p. 98).

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.

More about encryption context.

Encryption context in policies

The encryption context is used primarily to verify integrity and authenticity. But you can also use the encryption context to control access to symmetric customer master keys (CMKs) in key policies and IAM policies.

The kms:EncryptionContext: (p. 107) and kms:EncryptionContextKeys (p. 107) condition keys allow (or deny) a permission only when the request includes particular encryption context keys or key–value pairs.

For example, the following key policy statement allows the `RoleForExampleApp` role to use the CMK in `Decrypt` operations. It uses the `kms:EncryptionContext:` condition key to allow this permission only when the encryption context in the request includes an `AppName:ExampleApp` encryption context pair.

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

For more information about these encryption context condition keys, see Using policy conditions with AWS KMS (p. 98).

Encryption context in grants

When you create a grant (p. 126), you can include grant constraints that allow access only when a request includes a particular encryption context or encryption context keys. For details about the `EncryptionContextEquals` and `EncryptionContextSubset` grant constraints, see Grant constraints (p. 128).

To specify an encryption context constraint in a grant for a symmetric CMK, use the `Constraints` parameter in the CreateGrant operation. This example uses the AWS Command Line Interface, but you can use any AWS SDK. The grant that this command creates gives the `exampleUser` permission to call the Decrypt operation. But that permission is effective only when the encryption context in the `Decrypt` request includes a `"Department": "IT"` encryption context pair.

```
$ aws kms create-grant \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --grantee-principal arn:aws:iam::111122223333:user/exampleUser \
    --operations Decrypt \
    --retiring-principal arn:aws:iam::111122223333:role/adminRole \
    --constraints EncryptionContextSubset={Department=IT}
```

The resulting grant looks like the following one. Notice that the permission granted to `exampleUser` is effective only when the `Decrypt` request includes the encryption context pair specified in the grant constraint. To find the grants on a CMK, use the ListGrants operation.

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

{
    "Grants": [
        {
            "Name": "",
            "IssuingAccount": "arn:aws:iam::111122223333:root",
            "GrantId": "8c94d1f12f5e69f440bae30eaec9570bb1fb7358824f9ddfa1aa5a0dab1a59b2",
            "Operations": [
                "Decrypt"
            ],
            "GranteePrincipal": "arn:aws:iam::111122223333:user/exampleUser",
            "Constraints": {
                "EncryptionContextSubset": {
                    "Department": "IT"
                }
            },
            "CreationDate": 1568565290.0,
            "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "RetiringPrincipal": "arn:aws:iam::111122223333:role/adminRole"
        }
    ]
}
```

AWS services often use encryption context constraints in the grants that give them permission to use CMKs in your AWS account. For example, Amazon DynamoDB uses a grant like the following one to get permission to use the AWS managed CMK (p. 4) for DynamoDB in your account. The `EncryptionContextSubset` grant constraint in this grant makes the permissions in the grant effective only when the encryption context in the request includes `"subscriberID": "111122223333"` and `"tableName": "Services"` pairs. This grant constraint means that the grant allows DynamoDB to use the specified CMK only for a particular table in your AWS account.

To get this output, run the ListGrants operation on the AWS managed CMK for DynamoDB in your account.

```
$ aws kms list-grants --key-id 0987dcba-09fe-87dc-65ba-ab0987654321

{
    "Grants": [
        {
            "Operations": [
                "Decrypt",
                "Encrypt",
                "GenerateDataKey",
                "ReEncryptFrom",
                "ReEncryptTo",
                "RetireGrant",
                "DescribeKey"
            ],
            "IssuingAccount": "arn:aws:iam::111122223333:root",
            "Constraints": {
                "EncryptionContextSubset": {
                    "aws:dynamodb:tableName": "Services",
                    "aws:dynamodb:subscriberId": "111122223333"
                }
            },
            "CreationDate": 1518567315.0,
            "KeyId": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
            "GranteePrincipal": "dynamodb.us-west-2.amazonaws.com",
            "RetiringPrincipal": "dynamodb.us-west-2.amazonaws.com",
            "Name": "8276b9a6-6cf0-46f1-b2f0-7993a7f8c89a",
            "GrantId": "1667b97d27cf748cf05b487217dd4179526c949d14fb3903858e25193253fe59"
        }
    ]
}
```

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` operations, 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.

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. 56).

Grants

A *grant* is another mechanism for providing permissions. It's an alternative to key policies. Because grants can be very specific, and are easy to create and revoke, they are often used to provide temporary permissions or more granular permissions.

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 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. The log files also include requests to AWS KMS that AWS services make on your behalf. You can use these log files to find important information, including when the CMK was used, the operation that was requested, the identity of the requester, and the source IP address. For more information, see Logging AWS KMS API calls with AWS CloudTrail (p. 303) 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

To get started, create an AWS KMS customer master key (p. 3) (CMK). The topics in this section explain how to create, view, edit, enable, disable, and tag your CMK. They also explain how to download the public key of an asymmetric CMK.

You can perform these AWS Key Management Service (AWS KMS) tasks in the AWS Management Console. You can access all AWS KMS features by using the AWS KMS API operations. To run the AWS KMS API operations, use any AWS SDK, the AWS Command Line Interface (AWS CLI), or AWS Tools for PowerShell.

Topics

- Creating keys (p. 24)
- Viewing keys (p. 29)
- Editing keys (p. 46)
- Tagging keys (p. 49)
- Enabling and disabling keys (p. 51)
- Downloading public keys (p. 53)

Creating keys

You can create symmetric and asymmetric customer master keys (p. 141) (CMKs) in the AWS Management Console or by using the CreateKey operation. During this process, you determine the cryptographic configuration of your CMK and the origin of its key material. You cannot change these properties after the CMK is created. You also set the key policy for the CMK, which you can change at any time.

If you are creating a CMK to encrypt data you store or manage in an AWS service, create a symmetric CMK. AWS services that are integrated with AWS KMS use symmetric CMKs to encrypt your data. These services do not support encryption with asymmetric CMKs. For help deciding which type of CMK to create, see How to choose your CMK configuration (p. 143).

Learn more:

- For help creating a CMK with imported key material (key material origin (p. 17) is External), see Create a Customer Master Key With No Key Material (p. 162).
- For help creating a CMK in a custom key store (key material origin (p. 17) is Custom Key Store (CloudHSM)), see Creating CMKs in a Custom Key Store (p. 204).
- For help determining whether an existing CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).
- To use your CMKs programmatically and in command line interface operations, you need a key ID (p. 12) or key ARN (p. 12). For detailed instructions, see Finding the key ID and ARN (p. 40).
- To use your CMKs in cryptographic operations, you can use an alias name (p. 14) or alias ARN (p. 12), as well as a key ID (p. 12) or key ARN (p. 12). For detailed instructions, see Finding the alias name and alias ARN (p. 41).

Topics

- Creating symmetric CMKs (p. 25)

- Creating asymmetric CMKs (p. 27)

Creating symmetric CMKs

You can create symmetric CMKs (p. 142) in the AWS Management Console or by using the AWS KMS API. Symmetric key encryption uses the same key to encrypt and decrypt data.

Creating symmetric CMKs (console)

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

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. To create a symmetric CMK, for **Key type** choose **Symmetric**.

   For information about how to create an asymmetric CMK in the AWS KMS console, see Creating asymmetric CMKs (console) (p. 27).
6. Choose **Next**.
7. 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.
8. (Optional) Type a description for the CMK.

   Enter a description that explains the type of data you plan to protect or the application you plan to use with the CMK. Don't use the description format that's used for AWS managed CMKs (p. 4). The *Default master key that protects my ... when no other key is defined* description format is reserved for AWS managed CMKs.

   You can add a description now or update it any time unless the key state (p. 233) is `Pending Deletion`. To add, change, or delete the description of an existing customer managed CMK, edit the CMK (p. 46) in the AWS Management Console or use the UpdateKeyDescription operation.
9. Choose **Next**.
10. (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. 49).
11. Choose **Next**.
12. 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.
13. (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.
14. Choose **Next**.

15. Select the IAM users and roles that can use the CMK for cryptographic operations (p. 10).

    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.

16. (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 users in other accounts to use a CMK (p. 82).

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

Creating symmetric CMKs (AWS KMS API)

You can use the CreateKey operation to create a new symmetric 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, 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. Also, if you are creating a CMK for imported key material (p. 159) or a CMK in a custom key store (p. 184), the `Origin` parameter is required.

The following is an example of a call to the `CreateKey` operation with no parameters. This command uses all of the default values. It creates a symmetric CMK for encrypting and decrypting with key material generated by AWS KMS.

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

If you do not specify a key policy for your new CMK, the default key policy (p. 61) 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 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. 56)

```
$ 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" : "*"
  } ]
}
```

Creating asymmetric CMKs

You can create asymmetric CMKs (p. 142) in the AWS Management Console or by using the AWS KMS API. An asymmetric CMK represents a public and private key pair that can be used for encryption or signing.

If you are creating a CMK to encrypt data that you store or manage in an AWS service, use a symmetric CMK. AWS services that integrate with AWS KMS do not support asymmetric CMKs. For help deciding whether to create a symmetric or asymmetric CMK, see How to Choose Your CMK Configuration (p. 143).

Creating asymmetric CMKs (console)

You can use the AWS Management Console to create asymmetric customer master keys (CMKs). Each asymmetric CMK represents a public and private key pair.

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. To create an asymmetric CMK, in **Key type,** choose **Asymmetric**.

   For information about how to create an symmetric CMK in the AWS KMS console, see Creating symmetric CMKs (console) (p. 25).
6. To create an asymmetric CMK for public key encryption, in **Key usage**, choose **Encrypt and decrypt**. Or, to create an asymmetric CMK for signing messages and verifying signatures, in **Key usage**, choose **Sign and verify**.

   For help choosing a key usage value, see Selecting the key usage (p. 144).
7. Select a specification (**Key spec**) for your asymmetric CMK.

   Often the key spec that you select is determined by regulatory, security, or business requrirements. It might also be influenced by the size of messages that you need to encrypt or sign. In general, longer encryption keys are more resistant to brute-force attacks.

   For help choosing a key spec, see Selecting the key spec (p. 145).
8. Choose **Next**.

9. 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.
10. (Optional) Type a description for the CMK.

    Enter a description that explains the type of data you plan to protect or the application you plan to use with the CMK. Don't use the description format that's used for AWS managed CMKs (p. 4). The *Default master key that protects my ... when no other key is defined* description format is reserved for AWS managed CMKs.

    You can add a description now or update it any time unless the key state (p. 233) is `Pending Deletion`. To add, change, or delete the description of an existing customer managed CMK, edit the CMK (p. 46) in the AWS Management Console or use the UpdateKeyDescription operation.
11. (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. 49).
12. Choose **Next**.
13. 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.
14. (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.
15. Choose **Next**.
16. Select the IAM users and roles that can use the CMK for cryptographic operations (p. 10).

    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.
17. (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 users in other accounts to use a CMK (p. 82).
18. Choose **Next**.
19. Review the key policy document that was created from your choices. You can edit it, too.
20. Choose **Finish** to create the CMK.

Creating asymmetric CMKs (AWS KMS API)

You can use the CreateKey operation to create an asymmetric customer master key (CMK). These examples use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

When you create an asymmetric CMK, you must specify the `CustomerMasterKeySpec` parameter, which determines the type of keys you create. Also, you must specify a `KeyUsage` value of ENCRYPT_DECRYPT or SIGN_VERIFY. You cannot change these properties after the CMK is created.

The following example uses the `CreateKey` operation to create an asymmetric CMK of 4096-bit RSA keys designed for public key encryption.

```
$ aws kms create-key --customer-master-key-spec RSA_4096 --key-usage ENCRYPT_DECRYPT
{
    "KeyMetadata": {
        "KeyState": "Enabled",
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "CustomerMasterKeySpec": "RSA_4096",
        "KeyManager": "CUSTOMER",
        "Description": "",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "CreationDate": 1569973196.214,
        "EncryptionAlgorithms": [
            "RSAES_OAEP_SHA_1",
            "RSAES_OAEP_SHA_256"
        ],
        "AWSAccountId": "111122223333",
        "Origin": "AWS_KMS",
        "Enabled": true
    }
}
```

The following example command creates an asymmetric CMK that represents a pair of ECDSA keys used for signing and verification. You cannot create an elliptic curve key pair for encryption and decryption.

```
$ aws kms create-key --customer-master-key-spec ECC_NIST_P521 --key-usage SIGN_VERIFY
{
    "KeyMetadata": {
        "KeyState": "Enabled",
        "KeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
        "CreationDate": 1570824817.837,
        "Origin": "AWS_KMS",
        "SigningAlgorithms": [
            "ECDSA_SHA_512"
        ],
        "Arn": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
        "AWSAccountId": "111122223333",
        "CustomerMasterKeySpec": "ECC_NIST_P521",
        "KeyManager": "CUSTOMER",
        "Description": "",
        "Enabled": true,
        "KeyUsage": "SIGN_VERIFY"
    }
}
```

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 in the console (p. 30)
- Viewing CMKs with the API (p. 36)
- Finding the key ID and ARN (p. 40)
- Finding the alias name and alias ARN (p. 41)
- Identifying symmetric and asymmetric CMKs (p. 43)

Viewing CMKs in the console

In the AWS Management Console, you can view lists of your CMKs and details about each CMK.

Topics

- Navigating to the key tables (p. 30)
- Sorting and filtering your CMKs (p. 30)
- Displaying CMK details (p. 33)
- Customizing your CMK tables (p. 35)

Navigating to the key tables

The AWS KMS customer master keys (CMKs) in each account and region are displayed in tables. There are separate tables for the CMKs that you create and the CMKs that AWS services create for you.

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**. For information about the different types of CMKs, see Customer master keys (p. 3).

   Tip
   To view AWS managed CMKs (p. 4) that are missing an alias, use the **Customer managed keys** page.
   The AWS KMS console also displays the custom key stores in the account and Region. CMKs that you create in custom key stores appear on the **Customer managed keys** page. For information about custom key stores, see Using a custom key store (p. 184).

Sorting and filtering your CMKs

To make it easier to find your CMKs in the console, you can sort and filter them.

Note
The **Key type** column is displayed optionally (p. 35) and is available only in AWS Regions where AWS KMS supports asymmetric CMKs.

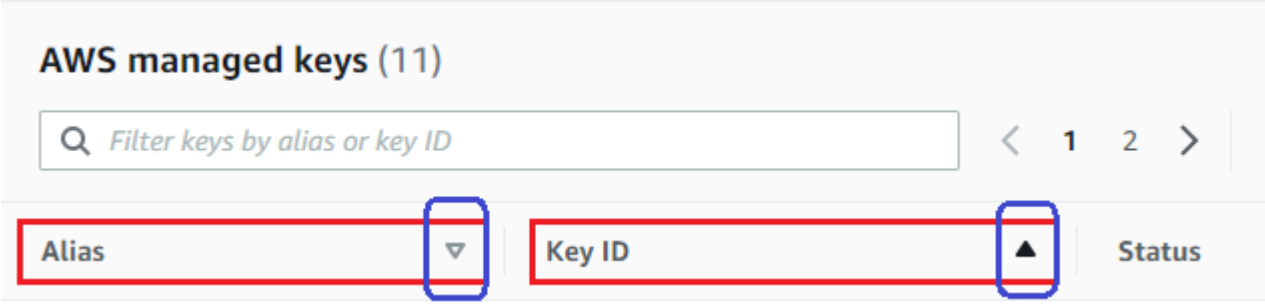
Sort

You can sort customer managed CMKs in ascending or descending order by their column values. This feature sorts all CMKs in the table, even if they don't appear on the current table page.

Sortable columns are indicated by an arrow beside the column name. On the **AWS managed keys** page, you can sort by **Alias** or **Key ID**. On the **Customer managed keys** page, you can sort by **Alias**, **Key ID**, or **Key type**.

To sort in ascending order, choose the column heading until the arrow points upward. To sort in descending order, choose the column heading until the arrow points downward. You can sort by only one column at a time.

For example, you can sort CMKs in ascending order by key ID, instead of alias, which is the default.



When your sort CMKs on the **Customer master keys** page in ascending order by **Key type**, all asymmetric keys are displayed before all symmetric keys.

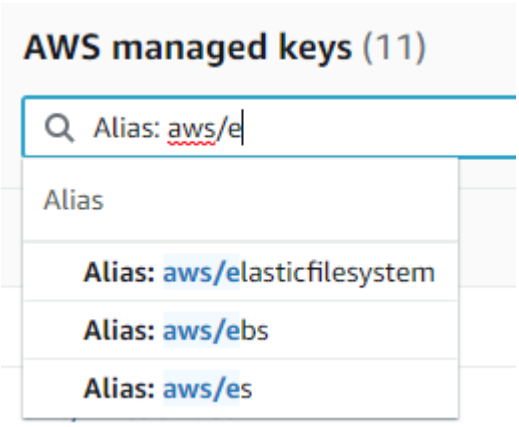
Filter

You can filter CMKs by their column values. The filter applies to all CMKs in the table, even if they don't appear on the current table page. The filter is not case-sensitive.

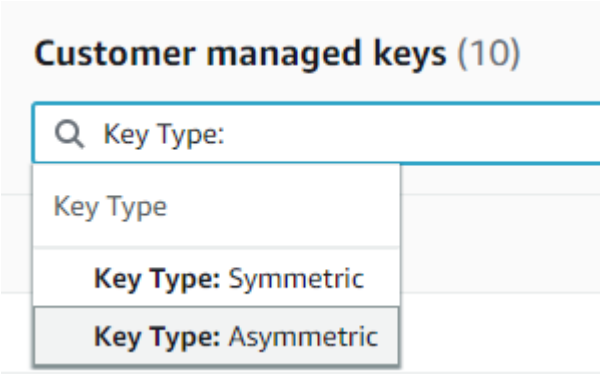
Filterable columns are listed in the filter box. On the **AWS managed keys** page, you can filter by **Alias** and **Key ID**. On the **Customer managed keys** page, you can filter by **Alias**, **Key ID**, and **Key type**.

To filter by the value in a particular column, choose the filter, choose the column name, and then choose from the list of actual column values. After choosing a column, you can also type all or part of the column value. You'll see a preview of the results before you make your choice.

For example, to display CMKs with an alias name that contains with `aws/e`, choose the filter box, choose **Alias**, type `aws/e`, and then press `Enter` or `Return` to add the filter.

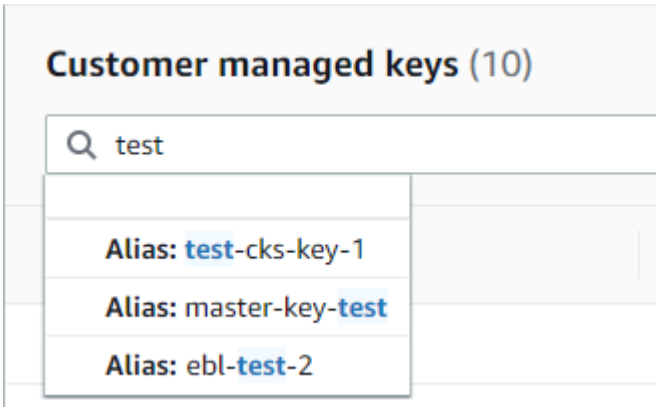


To display only asymmetric CMKs on the **Customer master keys** page, click the filter box, choose **Key type** and then choose **Key type: Asymmetric**. The **Asymmetric** option appears only when you have asymmetric CMKs in the table.

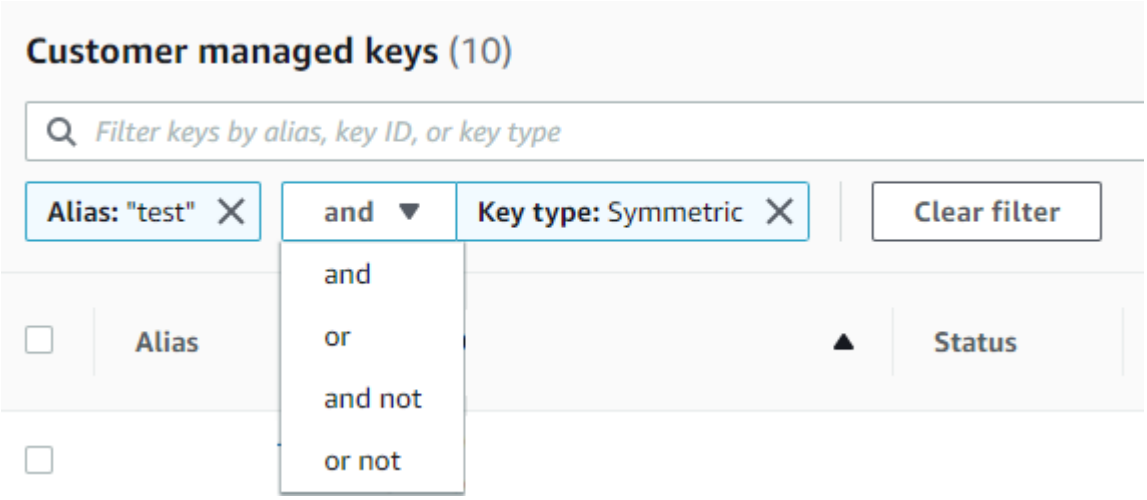


To search for text in all filterable columns, in the filter box, type all or part of a column value. You'll see a preview of the results before you make your choice.

For example, to display CMKs with `test` in any of the column values, type `test` in the filter box. The preview shows the CMKs that the filter will select. In this case, `test` appears only in the **Alias** column.



You can have multiple filters at the same time. When you add additional filters, you can also select a logical operator.

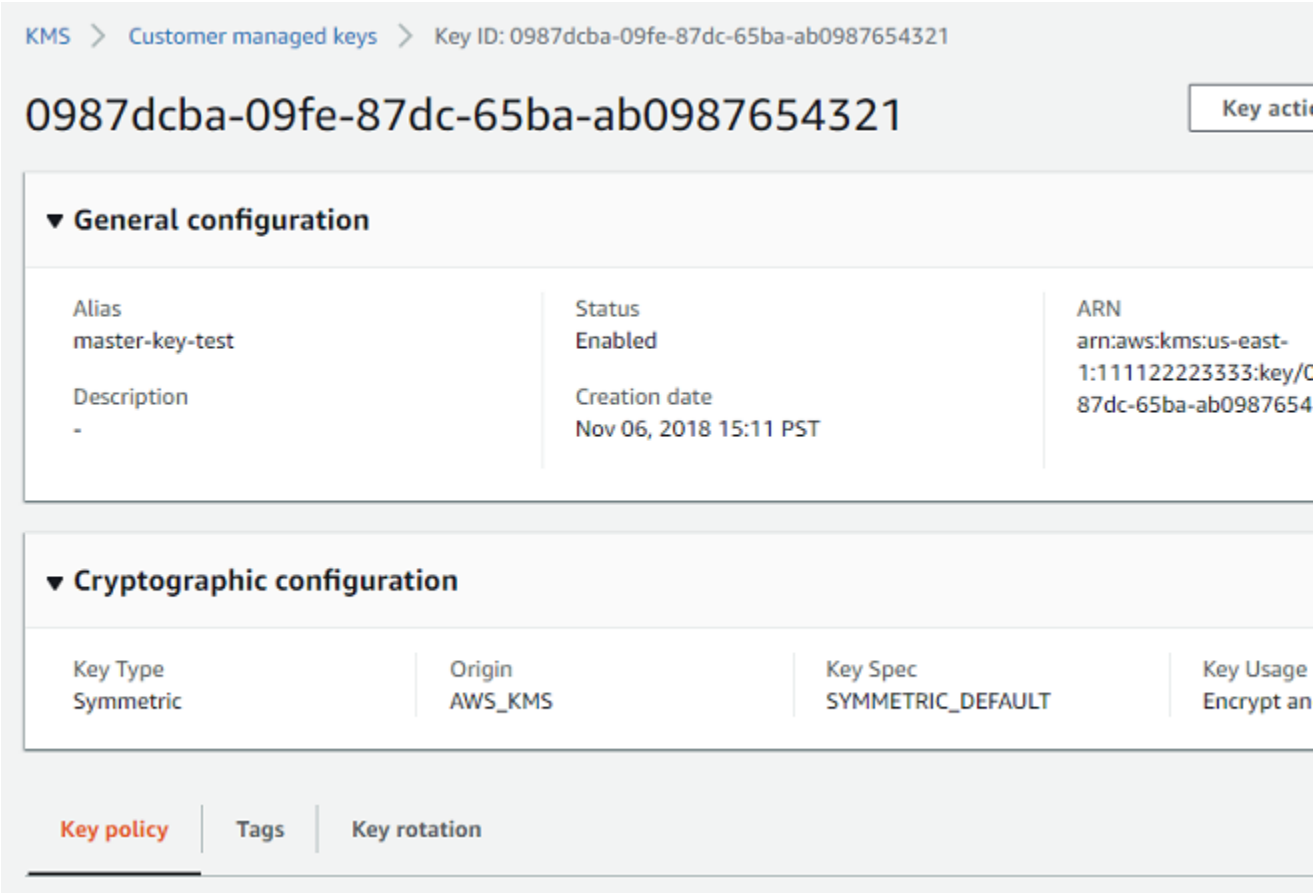


Displaying CMK details

The details page for each CMK displays the properties of the CMK. It differs slightly for the different types of CMKs.

To display detailed information about a CMK:

1. To display the details page for a CMK, on the **AWS managed keys** or **Customer managed keys** page, choose the alias or key ID of the CMK.
2. To display all details, expand the **General configuration** and **Cryptographic configuration** sections of the page. If the CMK is configured for imported key material, the page also has a **Key materials** section that you can expand.



The details page for a CMK includes a **General Configuration** section that displays the basic properties of the CMK, a **Cryptographic Configuration** section that displays the cryptographic properties of the CMK, and a tabbed display that includes the key policy, tags, key rotation (for symmetric CMKs), and public key (for asymmetric CMKs).

The following list describes the fields in the detailed display. Some of these fields are also available as columns in the table display.

Alias

A friendly name for the CMK. The **Alias** field in the console lists only one alias. To find all aliases for the CMK, use the ListAliases operation.

ARN

The Amazon Resource Name (ARN) of the CMK. This value uniquely identifies the CMK. You can use it to identify the CMK in AWS KMS API operations.

Creation date

The date and time that the CMK was created. This value is displayed in local time for the device. The time zone does not depend on the Region.

Unlike **Expiration**, the creation refers only to the CMK, not its key material.

CloudHSM cluster ID

The cluster ID of the AWS CloudHSM cluster that contains the key material for the CMK. This field appears only when the CMK is created in an AWS KMS custom key store (p. 184).

If you click the CloudHSM cluster ID, it opens the **Clusters** page in the AWS CloudHSM console.

Custom key store ID

The ID of the custom key store (p. 184) that contains the CMK. This field appears only when the CMK is created in an AWS KMS custom key store.

If you click the custom key store ID, it opens the **Custom key stores** page in the AWS KMS console.

Custom key store name

The name of the custom key store (p. 184) that contains the CMK. This field appears only when the CMK is created in an AWS KMS custom key store.

Description

A brief, optional description of the CMK. To add or update the description of a customer managed CMK, above **General Configuration**, choose **Edit**.

Encryption algorithms

Lists the encryption algorithms that can be used with the CMK in AWS KMS. This field appears only when the **Key type** is **Asymmetric** and the **Key usage** is **Encrypt and decrypt**. For information about the encryption algorithms that AWS KMS supports, see SYMMETRIC_DEFAULT key spec (p. 146) and RSA key specs for encryption and decryption (p. 147).

Expiration date

The date and time when the key material for the CMK expires. This field appears only for CMKs with imported key material (p. 159), that is, when the **Origin** is **External** and the CMK has key material that expires.

Key policy

Controls access to the CMK along with IAM policies (p. 77) and grants (p. 126). Every CMK has one key policy. It is the only mandatory authorization element. To change the key policy of a customer managed CMK, on the **Key policy** tab, choose **Edit**. For details, see the section called "Using key policies" (p. 59).

Key rotation

Enables and disables automatic key rotation (p. 154) every year.

To change the key rotation status of a customer managed CMK (p. 4), use the checkbox on the **Key rotation** tab. All AWS managed CMKs (p. 4) are automatically rotated every three years.

Key spec

The type of key material in the CMK. AWS KMS supports symmetric CMKs (SYMMETRIC_DEFAULT), CMKs for RSA keys of different lengths, and elliptic curve keys with different curves. For details, see Key spec (p. 17).

Key type

Indicates whether the CMK is **Symmetric** or **Asymmetric**.

Key usage

Indicates whether a CMK can be used for **Encrypt and decrypt** or **Sign and verify**. Only asymmetric CMKs can be used to sign and verify. For details, see Key usage (p. 18).

Origin

The source of the key material for the CMK. Valid values are **AWS_KMS** for key material that AWS KMS generates, **EXTERNAL** for imported key material (p. 159), and **AWS_CloudHSM** for CMKs in custom key stores (p. 184).

Public key

Displays the public key of an asymmetric CMK. Authorized users can use this tab to copy and download the public key (p. 53).

Signing algorithms

Lists the signing algorithms that can be used with the CMK in AWS KMS. This field appears only when the **Key type** is **Asymmetric** and the **Key usage** is **Sign and verify**. For information about the signing algorithms that AWS KMS supports, see RSA key specs for signing and verification (p. 148) and Elliptic curve key specs (p. 148).

Status

The key state of the CMK. You can use the CMK in cryptographic operations (p. 10) only when the status is **Enabled**. For a detailed description of each CMK status and its effect on the operations that you can run on the CMK, see Key state: Effect on your CMK (p. 233).

Tags

Optional key-value pairs that describe the CMK. To add or change the tags for a CMK, on the **Tags** tab, choose **Edit**.

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. 49).

Customizing your CMK tables

You can customize the tables that appear on the **AWS managed keys** and **Customer managed keys** pages in the AWS Management Console to suit your needs. You can choose the table columns, the number of customer master keys (CMKs) on each page (**Page size**), and the text wrap. The configuration you choose is saved when you confirm it and reapplied whenever you open the pages.

To customize your CMK tables

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**.

Consider using the **Page size** setting to increase the number of CMKs displayed on each page, especially if you typically use a device that's easy to scroll.

The data columns that you display might vary depending on the table, your job role, and the types of CMKs in the account and Region. The following table offers some suggested configurations. For descriptions of the columns, see Displaying CMK details (p. 33).

Suggested CMK table configurations

You can customize the columns that appear in your CMK table to display the information you need about your CMKs.

AWS managed keys

By default, the **AWS managed keys** table displays the **Alias**, **Key ID**, and **Status** columns. These columns are ideal for most use cases.

Symmetric customer managed keys

If you use only symmetric CMKs with key material generated by AWS KMS, the **Alias**, **Key ID**, **Status**, and **Creation date** columns are likely to be the most useful.

Asymmetric customer managed keys

If you use asymmetric CMKs, in addition to the **Alias**, **Key ID**, and **Status** columns, consider adding the **Key type**, **Key spec**, and **Key usage** columns. These columns will show you whether a CMK is symmetric or asymmetric, the type of key material, and whether the CMK can be used for encryption or signing.

Imported key material

If you have CMKs with imported key material (p. 159), in addition to the **Alias**, **Key ID**, and **Status** columns, consider adding the **Origin** and **Expiration date** columns. These columns will show you whether the key material in a CMK is imported or generated by AWS KMS and when the key material expires, if at all. The **Creation date** field displays the date that the CMK was created (without key material). It doesn't reflect any characteristic of the key material.

Keys in custom key stores

If you have CMKs in custom key stores (p. 184), in addition to the **Alias**, **Key ID**, and **Status** columns, consider adding the **Custom key store ID** column. A value in this column indicates that the CMK is in a custom key store, as well as showing which custom key store it's in.

Viewing CMKs with the 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. 36)
- DescribeKey: Get detailed information about a CMK (p. 37)
- GetKeyPolicy: Get the key policy attached to a CMK (p. 38)
- ListAliases: Get alias names and ARNs for CMKs (p. 38)

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.

For example, this call to the `ListKeys` operation returns the ID and ARN of each CMK in this fictitious account. For examples in multiple programming languages, see Getting key IDs and key ARNs of CMKs (p. 331).

```
$ 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/0987dcba-09fe-87dc-65ba-
ab0987654321",
      "KeyId": "0987dcba-09fe-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 (p. 12), key ARN (p. 12), alias name (p. 14), or alias ARN (p. 12).

For example, this call to `DescribeKey` returns information about a symmetric CMK. The fields in the response vary with the customer master key spec (p. 17), key state (p. 233), and the key material origin (p. 17). For examples in multiple programming languages, see Viewing a custom master key (p. 329).

```
$ 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,
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "Enabled",
        "CreationDate": 1499988169.234,
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "AWSAccountId": "111122223333"
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ]
    }
}
```

This example calls `DescribeKey` operation on an asymmetric CMK used for signing and verification. The response includes the signing algorithms that AWS KMS supports for this CMK.

```
$ aws kms describe-key --key-id 0987dcba-09fe-87dc-65ba-ab0987654321

{
    "KeyMetadata": {
        "KeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
        "Origin": "AWS_KMS",
```

```
        "Arn": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321",
        "KeyState": "Enabled",
        "KeyUsage": "SIGN_VERIFY",
        "CreationDate": 1569973196.214,
        "Description": "",
        "CustomerMasterKeySpec": "ECC_NIST_P521",
        "AWSAccountId": "111122223333",
        "Enabled": true,
        "KeyManager": "CUSTOMER"
        "SigningAlgorithms": [
            "ECDSA_SHA_512"
        ]
    }
}
```

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. 3) 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.)

For examples in multiple programming languages, see Getting a key policy (p. 345).

```
$ 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: Get alias names and ARNs for CMKs

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. 3), and aliases that AWS created and associated with AWS managed CMKs (p. 3) 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.

For examples in multiple programming languages, see Listing aliases (p. 362).

```
$ 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 key ID (p. 12) or key ARN (p. 12). You cannot specify an alias name (p. 14) or alias ARN (p. 12).

The command in the following example gets the aliases that refer to a customer managed CMK (p. 4). But you can use a command like this one to find the aliases that refer to AWS managed CMKs (p. 4), 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"
        },
    ]
}
```

To get only the aliases for AWS managed CMKs, use the features of your programming language to filter the response.

```
$ aws kms list-aliases --query 'Aliases[?starts_with(AliasName, `alias/aws/`)]'
```

Finding the key ID and ARN

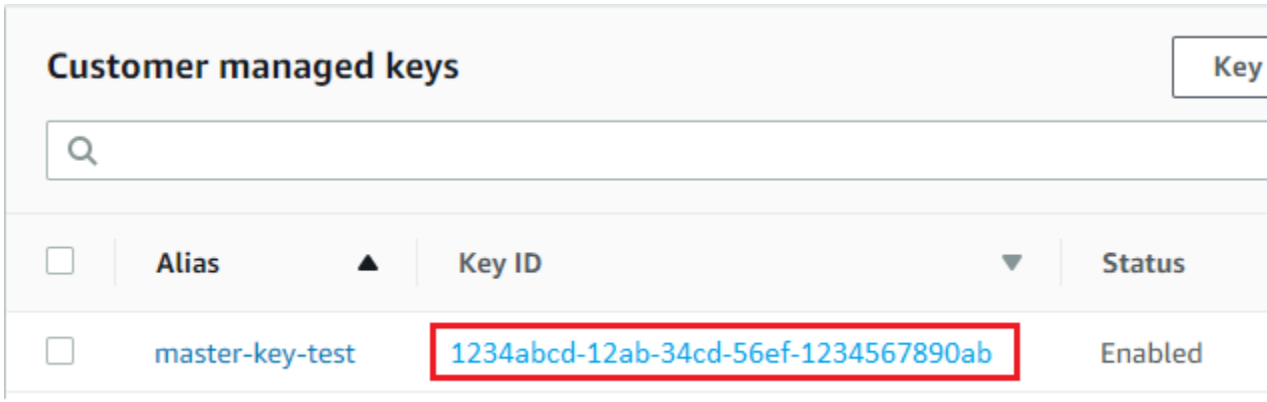
To identify an AWS KMS CMK, you can use its key ID (p. 12) or its Amazon Resource Name (key ARN (p. 12)). In cryptographic operations (p. 10), you can also use the alias name (p. 14) or alias ARN (p. 12).

For detailed information about the CMK identifiers that AWS KMS supports, see Key identifiers (KeyId) (p. 11).

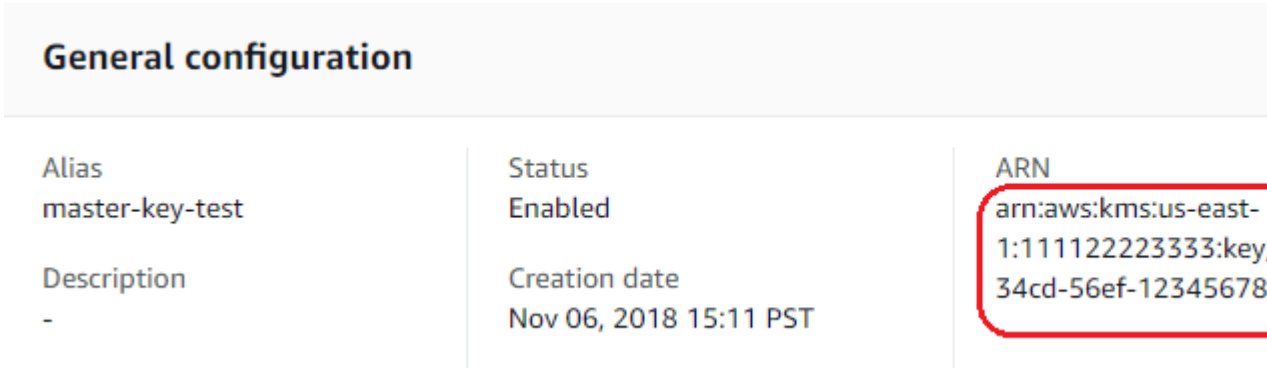
To find the key ID and ARN (console)

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 (p. 12) for a CMK, see the row that begins with the CMK alias.

   The **Key ID** column appears in the tables by default. If the Key ID column doesn't appear in your table, use the procedure described in the section called "Customizing your CMK tables" (p. 35) to restore it. You can also view the key ID of a CMK on its details page.



5. To find the Amazon Resource Name (ARN) of the CMK, choose the key ID or alias. The key ARN (p. 12) appears in the **General Configuration** section.



To find the key ID and key ARN (AWS KMS API)

To find the key ID (p. 12) and key ARN (p. 12) of a customer master key (CMK), use the ListKeys operation. For examples in multiple programming languages, see Getting key IDs and ARNs (p. 331) and Get key IDs and ARNs (p. 36).

The `ListKeys` response includes the key ID and key ARN for every CMK in the account and Region.

```
$ aws kms list-keys
{
    "Keys": [
        {
            "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
            "KeyArn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        },
        {
            "KeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
            "KeyArn": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321"
        }
    ]
}
```

Finding the alias name and alias ARN

To identify an AWS KMS customer master key (p. 3) (CMK) in a cryptographic operation (p. 10), you can use its key ID (p. 12), key ARN (p. 12), alias name (p. 14), or alias ARN (p. 12). In other AWS KMS operations, only the key ID or key ARN are valid.

For detailed information about the CMK identifiers that AWS KMS supports, see Key identifiers (KeyId) (p. 11).

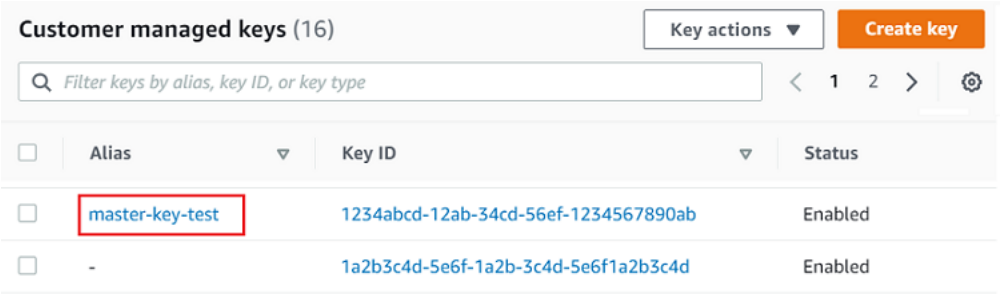
Topics

- To find the alias name (console) (p. 41)
- To derive the alias ARN (console) (p. 42)
- To find the alias name and alias ARN (AWS KMS API) (p. 41)

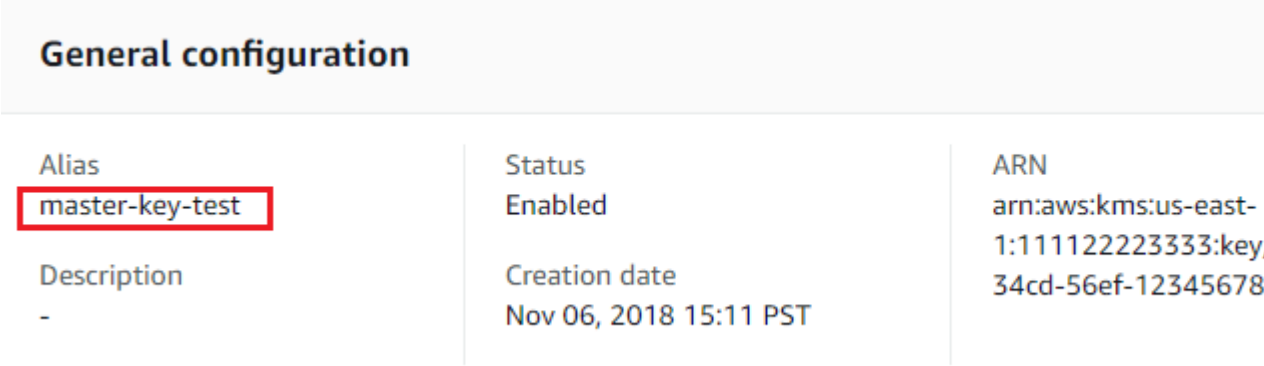
To find the alias name (console)

The AWS KMS console displays one alias name (p. 14) associated with the CMK. If the CMK was created in the console, the console displays the alias that was assigned to the CMK when it was created. To find other aliases associated with the CMK, if any, you need to use the AWS KMS API.

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 alias name for a CMK, see the **Alias** column in the row for each CMK. If a CMK does not have an alias, a dash (**-**) appears in the **Alias** column.



5. You can also find the alias on the details page for a CMK. To open the details page, choose the key ID or alias. The alias appears in the **General Configuration** section.



To derive the alias ARN (console)

The alias ARN (p. 12) is not displayed in the AWS KMS console. The **ARN** field displays the key ARN (p. 12). However, you can derive the alias ARN for this alias by using the values in the **Alias** and **ARN** fields.

To derive the alias ARN, begin with the key ARN.

```
# Key ARN format
arn:<partition>:kms:<region>:<account>:key/<key-id>

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

Replace `key` with `alias`. Replace the key ID with the alias name.

```
# Alias ARN format
arn:<partition>:kms:<region>:<account>:alias/<alias-name>

# Example alias ARN
arn:aws:kms:us-west-2:111122223333:alias/master-key-test
```

To find the alias name and alias ARN (AWS KMS API)

To find the alias name (p. 14) and alias ARN (p. 12) of a customer master key (CMK), use the ListAliases operation. For examples in multiple programming languages, see Listing aliases (p. 362) and Get alias names and ARNs (p. 38).

By default, the response includes the alias name and alias ARN for every alias in the account and Region. To get only the aliases for a particular CMK, use the `KeyId` parameter.

For example, the following command gets only the aliases for an example CMK with key ID `1234abcd-12ab-34cd-56ef-1234567890ab`.

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

Identifying symmetric and asymmetric CMKs

To determine whether a particular CMK is symmetric or asymmetric (p. 141), find its *key type* or key spec (p. 17). You can use the AWS KMS console or AWS KMS API.

Some of these methods will also show you other aspects of the cryptographic configuration of a CMK, including its key usage and the encryption or signing algorithms that the CMK supports. You can view the cryptographic configuration of an existing CMK, but you cannot change it.

For general information about viewing CMKs, including sorting, filtering, and choosing columns for your console display, see Viewing CMKs in the console (p. 30).

Topics

- Finding the key type in the CMK table (p. 43)
- Finding the key type on the details page (p. 44)
- Finding the key spec using the AWS KMS API (p. 45)

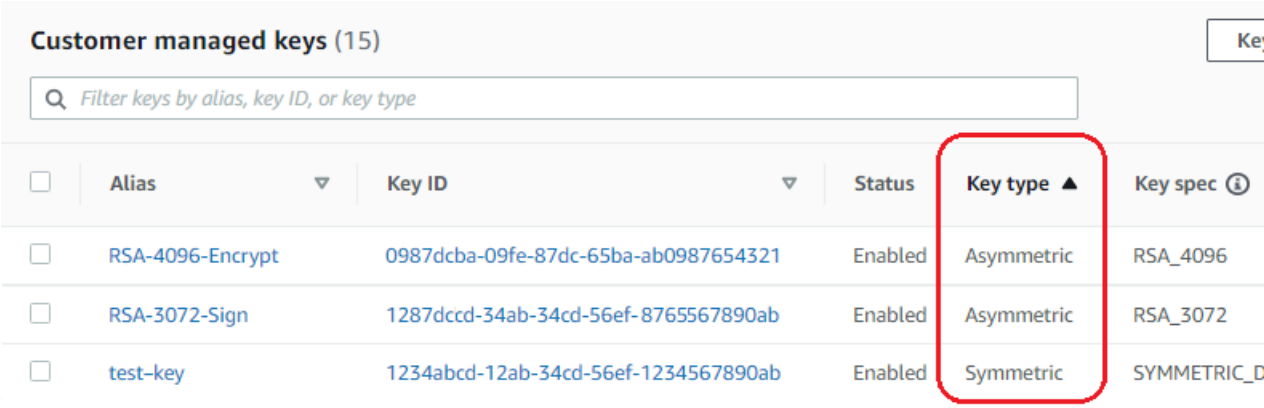
Finding the key type in the CMK table

In the AWS KMS console, the **Key type** column shows whether each CMK is symmetric or asymmetric. You can add a **Key type** column to the CMK table on the **Customer managed keys** or **AWS managed keys** pages in the console.

To identify symmetric and asymmetric CMKs in your CMK table, use the following procedure.

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. The **Key type** columns shows whether each CMK is symmetric or asymmetric. You can also sort and filter (p. 30) by the **Key type** value.

   If the **Key type** column does not appear in your CMK table, choose the gear icon in the upper right corner of the page, choose **Key type**, and then choose **Confirm**. You can also add the **Key spec** and **Key usage** columns.



Finding the key type on the details page

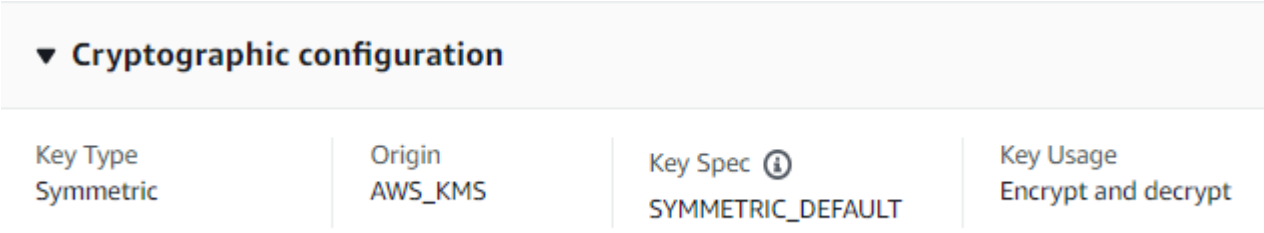
In the AWS KMS console, the details page for each CMK includes a **Cryptographic Configuration** section that displays the key type (symmetric or asymmetric) and other cryptographic details about the CMK.

To identify symmetric and asymmetric CMKs on the details page for a CMK, use the following procedure.

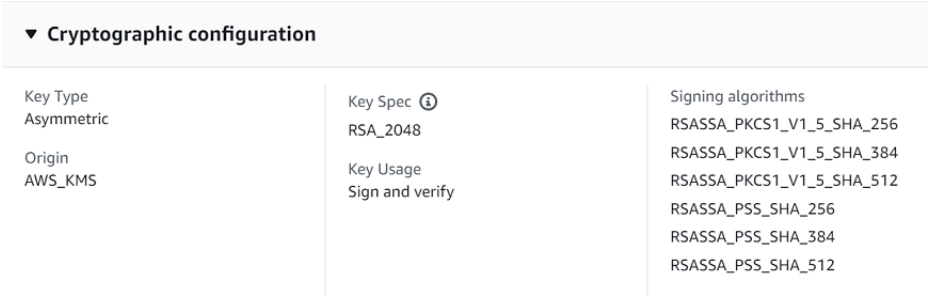
1. Open the AWS KMS console at https://console.aws.amazon.com/kms.
2. To change the AWS Region, use the Region selector in the upper-right corner of the page.
3. To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**. To view the keys in your account that AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**.
4. Choose the alias or key ID of a CMK.
5. Choose **Cryptographic configuration**.

   The **Cryptographic configuration** section includes the **Key Type**, which indicates whether it is symmetric or asymmetric. It also displays other details about the CMK, including the **Key Usage**, which tells whether a CMK can be used for encryption and decryption or signing and verification. For asymmetric CMKs, it displays the encryption algorithms or signing algorithms that the CMK supports.

   For example, the following is an example **Cryptographic configuration** section for a symmetric CMK.



   The following is an example **Cryptographic configuration** section for an asymmetric RSA CMK that's used for signing and verification.



Finding the key spec using the AWS KMS API

To determine whether a CMK is symmetric or asymmetric, use the DescribeKey operation. The `CustomerMasterKeySpec` field in the response contains the key spec (p. 17) of the CMK. For a symmetric CMK, the value of `CustomerMasterKeySpec` is `SYMMETRIC_DEFAULT`. All other values indicate an asymmetric CMK.

For example, `DescribeKey` returns the following response for a symmetric CMK. The `CustomerMasterKeySpec` value is `SYMMETRIC_DEFAULT`.

```
{
  "KeyMetadata": {
    "AWSAccountId": "111122223333",
    "KeyId": "0987dcba-09fe-87dc-65ba-ab0987654321",
    "Arn": "arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-ab0987654321",
    "CreationDate": 1496966810.831,
    "Enabled": true,
    "Description": "",
    "KeyState": "Enabled",
    "Origin": "AWS_KMS",
    "KeyManager": "CUSTOMER",
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "EncryptionAlgorithms": [
        "SYMMETRIC_DEFAULT"
    ]
  }
}
```

The `DescribeKey` response for an asymmetric RSA CMK used in signing and verification looks similar to this example. The key spec, as shown in the `CustomerMasterKeySpec` value is RSA_2048 (p. 146). The `KeyUsage` is `SIGN_VERIFY`, and the `SigningAlgorithms` element lists the valid signing algorithms for the CMK.

```
{
  "KeyMetadata": {
    "AWSAccountId": "111122223333",
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "Arn": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "CreationDate": 1571767572.317,
    "Enabled": false,
    "Description": "",
    "KeyState": "Disabled",
    "Origin": "AWS_KMS",
    "KeyManager": "CUSTOMER",
    "CustomerMasterKeySpec": "RSA_2048",
    "KeyUsage": "SIGN_VERIFY",
    "SigningAlgorithms": [
```

```
        "RSASSA_PKCS1_V1_5_SHA_256",
        "RSASSA_PKCS1_V1_5_SHA_384",
        "RSASSA_PKCS1_V1_5_SHA_512",
        "RSASSA_PSS_SHA_256",
        "RSASSA_PSS_SHA_384",
        "RSASSA_PSS_SHA_512"
    ]
  }
}
```

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. 3) (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. 3).

Topics

- Editing CMKs (console) (p. 46)
- Editing CMKs (AWS KMS API) (p. 48)

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. 29), but not edit, the properties of AWS managed CMKs. To view the key policy for an AWS managed CMK, use the GetKeyPolicy operation.

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 add, change, or delete the description of your CMK unless its key state (p. 233) 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. You can also delete an existing description.

   Enter a description that explains the type of data you plan to protect or the application you plan to use with the CMK. The *Default master key that protects my ... when no other key is defined* description format is reserved for AWS managed CMKs (p. 4).
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 (p. 10).

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. 56).

1. On the details page for CMK, 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 (p. 10), 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 an 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 users in other accounts to use a CMK (p. 82).

   - 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. 49).

- On the details page for CMK, 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. 154) of the cryptographic material in a symmetric customer managed CMK (p. 3). This feature is not supported for asymmetric CMKs or for CMKs with imported key material.

AWS managed CMKs (p. 3) are automatically rotated every three years. You cannot enable or disable this feature.

1. On the details page for CMK, 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**.

Editing CMKs (AWS KMS API)

You can use the AWS Key Management Service (AWS KMS) API to edit the properties of your customer managed CMKs (p. 3). 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. 3).

Topics

- UpdateKeyDescription: Change the description of a CMK (p. 48)
- PutKeyPolicy: Change the key policy for a CMK (p. 49)
- Enable and disable key rotation (p. 49)

Tip
For information about adding, deleting, and editing tags, see Tagging keys (p. 49).

UpdateKeyDescription: Change the description of a CMK

The UpdateKeyDescription operation replaces the description of the CMK with the one that you specify. You can use it to add, change, or delete 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"
      "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
      "EncryptionAlgorithms": [
        "SYMMETRIC_DEFAULT"
      ]
    }
}
```

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. 154) of the cryptographic material in a symmetric 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. 154).

Tagging keys

Many AWS services support resource tags. You can use tags to label, identify, and categorize your resources across AWS services.

AWS KMS supports resource tags on customer master keys (p. 3) (CMKs). You can add, change, and delete resource tags for customer managed CMKs (p. 3). AWS KMS doesn't support resource tags on aliases and you cannot create or manage the tags on AWS managed CMKs (p. 3).

Each resource 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 can attach the same tag, with the same tag key and tag value, to multiple CMKs. However, every tag on a CMK must have a different tag key. For example, you cannot have more than one "Cost Center" tag key on the same CMK. For detailed information about resource tags, see Tagging AWS Resources in the *AWS General Reference*.

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. 50)
- Managing CMK tags (AWS KMS API) (p. 50)

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. 24) and edit them at any time. You cannot edit the tags of CMKs that are pending deletion. For more information, see Creating keys (p. 24) and Editing keys (p. 46).

The following procedure shows how to manage tags using the check boxes on the **Customer managed keys** page. You can use the **Tags** tab on the detail page (p. 33) for any customer managed CMK.

To add, edit, or delete a tag for an existing CMK

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.
7. To save your changes, choose **Save changes**.

Managing CMK tags (AWS 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 (p. 4).

Topics

- CreateKey: Add tags to a new CMK (p. 50)
- TagResource: Add or change tags for a CMK (p. 51)
- ListResourceTags: Get the tags for a CMK (p. 51)
- UntagResource: Delete tags from a CMK (p. 51)

CreateKey: Add tags to a new CMK

You can add tags when you create a customer managed CMK (p. 4). To specify the tags, use the `Tags` parameter of the CreateKey operation. The value of the `Tags` parameter is a collection of tag key-value pairs.

For example, the following AWS CLI command creates a symmetric CMK with a `Project:Alpha` tag. When specifying more than one key-value pair, use a space to separate each pair.

```
$ aws kms create-key --tags TagKey=Project,TagValue=Alpha
```

When this command is successful, it returns a `KeyMetadata` object with information about the new CMK. However, the `KeyMetadata` does not include tags. To get the tags, use the ListResourceTags (p. 51) operation.

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, the following command adds **Purpose** and **Department** tags to an example CMK.

```
$ 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 an example CMK.

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

    "Truncated": false,
    "Tags": [
        {
            "TagKey": "Project",
            "TagValue": "Alpha"
        }
        {
            "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 --key-id 1234abcd-12ab-34cd-56ef-1234567890ab --tag-keys Purpose
```

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

Enabling and disabling keys

You can disable and reenable the customer master keys (p. 3) (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. 238). You cannot disable them.

You can also delete CMKs. For more information, see Deleting customer master keys (p. 172).

> **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. 155).

Topics

- Enabling and disabling CMKs (console) (p. 52)
- Enabling and disabling CMKs (AWS KMS API) (p. 52)

Enabling and disabling CMKs (console)

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

Enabling and disabling CMKs (AWS 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,
        "KeyState": "Disabled",
```

```
        "KeyUsage": "ENCRYPT_DECRYPT",
        "CreationDate": 1502910355.475,
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "AWSAccountId": "111122223333"
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ]
    }
}
```

Downloading public keys

You can view, copy, and download the public key from an asymmetric CMK pair by using the AWS Management Console or the AWS KMS API. You must have `kms:GetPublicKey` permission on the asymmetric CMK.

Each asymmetric CMK pair consists of a private key that never leaves AWS KMS unencrypted and a public key that you can download and share.

You might share a public key to let others encrypt data outside of AWS KMS that you can decrypt only with your private key. Or, to allow others to verify a digital signature outside of AWS KMS that you have generated with your private key.

When you use the public key in your asymmetric CMK within AWS KMS, you benefit from the authentication, authorization, and logging that are part of every AWS KMS operation. You also reduce of risk of encrypting data that cannot be decrypted. These features are not effective outside of AWS KMS. For details, see Special considerations for downloading public keys (p. 53).

Topics

- Special considerations for downloading public keys (p. 53)
- Downloading a public key (console) (p. 54)
- Downloading a public key (AWS KMS API) (p. 54)

Special considerations for downloading public keys

To protect your CMKs, AWS KMS provides access controls, authenticated encryption, and detailed logs of every operation. AWS KMS also allows you to prevent the use of CMKs, temporarily or permanently. Finally, AWS KMS operations are designed to minimize of risk of encrypting data that cannot be decrypted. These features are not available when you use downloaded public keys outside of AWS KMS.

Authorization

Key policies (p. 59) and IAM policies (p. 77) that control access to the CMK within AWS KMS have no effect on operations performed outside of AWS. Any user who can get the public key can use it outside of AWS KMS even if they don't have permission to encrypt data or verify signatures with the CMK.

Key usage restrictions

Key usage restrictions are not effective outside of AWS KMS. If you call the Encrypt operation with a CMK that has a `KeyUsage` of `SIGN_VERIFY`, the AWS KMS operation fails. But if you encrypt data outside of AWS KMS with a public key from an CMK with a `KeyUsage` of `SIGN_VERIFY`, the data cannot be decrypted.

Algorithm restrictions

Restrictions on the encryption and signing algorithms that AWS KMS supports are not effective outside of AWS KMS. If you encrypt data with the public key from a CMK outside of AWS KMS, and use an encryption algorithm that AWS KMS does not support, the data cannot be decrypted.

Disabling and deleting CMKs

Actions that you can take to prevent the use of CMK in a cryptographic operation within AWS KMS do not prevent anyone from using the public key outside of AWS KMS. For example, disabling a CMK, scheduling deletion of a CMK, deleting a CMK, or deleting the key material from a CMK have no effect on a public key outside of AWS KMS. If you delete an asymmetric CMK or delete or lose its key material, data that you encrypt with a public key outside of AWS KMS is unrecoverable.

Logging

AWS CloudTrail logs that record every AWS KMS operation, including the request, response, date, time, and authorized user, do not record the use of the public key outside of AWS KMS.

Downloading a public key (console)

You can use the AWS Management Console to view, copy, and download the public key from an asymmetric CMK in your AWS account. To download the public key from an asymmetric CMK in different AWS account, use the AWS KMS API.

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 an asymmetric CMK.
5. From the **Cryptographic configuration** section, record the values of the **Key spec**, **Key usage**, and **Encryption algorithms** or **Signing Algorithms** fields. You'll need to use these values to use the public key outside of AWS KMS. Be sure to share this information when you share the public key.
6. Choose the **Public key** tab.
7. To copy the public key to your clipboard, choose **Copy**. To download the public key to a file, choose **Download**.

Downloading a public key (AWS KMS API)

The GetPublicKey operation returns the public key in an asymmetric CMK. It also returns critical information that you need to use the public key correctly outside of AWS KMS, including the key usage and encryption algorithms. Be sure to save these values and share them whenever you share the public key.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

To specify a CMK, use its key ID (p. 12), key ARN (p. 12), alias name (p. 14), or alias ARN (p. 12). When using an alias name, prefix it with **alias/**. To specify a CMK in a different AWS account, you must use its key ARN or alias ARN.

Before running this command, replace the example alias name with a valid identifier for the CMK. To run this command, you must have `kms:GetPublicKey` permissions on the CMK.

```
$ aws kms get-public-key --key-id alias/example_RSA_3072
```

```
{
    "CustomerMasterKeySpec": "RSA_3072",
    "KeyId": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "EncryptionAlgorithms": [
        "RSAES_OAEP_SHA_1",
        "RSAES_OAEP_SHA_256"
    ],
    "PublicKey": "MIIBojANBgkqhkiG..."
}
```

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. 56)
- Access control (p. 57)

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, and to use the CMK for cryptographic operations (p. 10).

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

- Overview of managing access (p. 57)
- Using key policies (p. 59)
- Using IAM policies (p. 77)
- AWS KMS API permissions reference (p. 87)
- Using policy conditions (p. 98)
- Using grants (p. 126)
- Using service-linked roles (p. 129)
- Determining access (p. 130)

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. 58)
- Managing access to AWS KMS CMKs (p. 58)
- Specifying permissions in a policy (p. 59)

- Specifying conditions in a policy (p. 59)

AWS KMS resources and operations

In AWS KMS, the primary resource is a customer master key (p. 3) (CMK). AWS KMS also supports an alias (p. 360), a resource that is independent of a CMK, but provides a friendly name that you can associate with a CMK. Some AWS KMS operations allow you to use an alias to identify a CMK.

Each instance of a CMK or alias has a unique Amazon Resource Name (ARN) with a standard format. In AWS KMS resources, the AWS service name is `kms`.

- **Customer master key (CMK)**

  ARN format:

  ```
  arn:AWS partition name:AWS service name: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 partition name:AWS service name: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 more information about identifying CMKs in the AWS Management Console and AWS KMS API operations, see Key identifiers (KeyId) (p. 11). For a list of AWS KMS operations, see the AWS Key Management Service API Reference.

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. 59).

For more information about using IAM policies, see Using IAM policies (p. 77).

For more information about using grants, see Using grants (p. 126).

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. 87).

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. 58). 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 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 an AWS 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. 59) and Using IAM policies (p. 77).

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. 98).

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. 58).

Topics

- Overview of key policies (p. 60)
- Default key policy (p. 61)
- Example key policy (p. 68)

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. 61) and Changing a key policy (p. 74).

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.

  Note
  Do not set the Principal to an asterisk (*) in any key policy statement that allows permissions. An asterisk gives every identity in every AWS account permission to use the CMK, unless another policy statement explicitly denies it. Users in other AWS accounts just need corresponding IAM permissions in their own accounts to use the CMK.
- **Action** – (Required) Actions specify the API operations to allow or deny. For example, the `kms:Encrypt` action corresponds to the AWS KMS Encrypt operation. You can list more than one action in a policy statement. For more information, see AWS KMS API permissions reference (p. 87).
- **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. 98).

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. 61).

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

When you create a CMK with the AWS Management Console (p. 24), 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. 61)
- Allows key administrators to administer the CMK (p. 62)
- Allows key users to use the CMK (p. 64)
  - Allows key users to use a CMK for cryptographic operations (p. 66)
  - Allows key users to use the CMK with AWS services (p. 67)

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. 58).

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

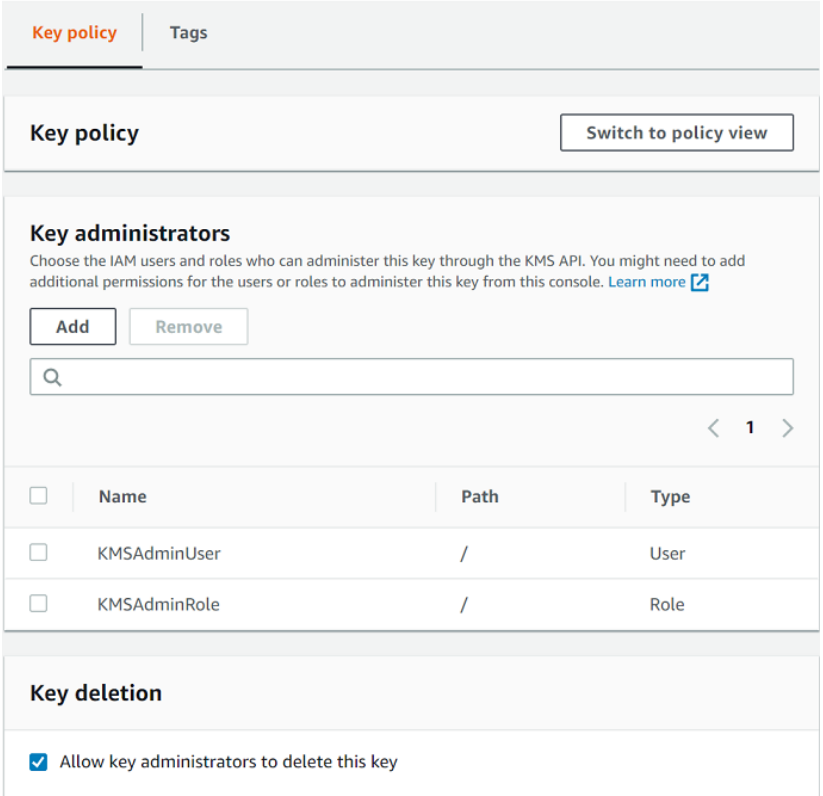
```
{
  "Sid": "Enable IAM User Permissions",
  "Effect": "Allow",
  "Principal": {"AWS": "arn:aws:iam::111122223333:root"},
  "Action": "kms:*",
  "Resource": "*"
}
```

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 in cryptographic operations (p. 10).

> **Warning**
> Even though key administrators do not have permissions to use the CMK to encrypt and decrypt data, they do have permission to change the key policy. This means they can give themselves any AWS KMS permission.

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.



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. 126) for this CMK.
- **kms:Describe*** – Allows key administrators to get 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. For symmetric CMKs, it allows key administrators to specify annual rotation of the CMK's key material (p. 154).
- **kms:List*** – Allows key administrators to get 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 change the key policy for this CMK.
- **kms:Update*** – Allows key administrators to change the target of an alias to this CMK, and to change this CMK's description.
- **kms:Revoke*** – Allows key administrators to revoke the permissions for this CMK that are allowed by a grant (p. 126).
- **kms:Disable*** – Allows key administrators to set this CMK's key state to disabled. For symmetric CMKs, it allows key administrators to disable annual rotation of this CMK's key material (p. 154).
- **kms:Get*** – Allows key administrators to get the key policy for this CMK and to determine whether this CMK's key material is rotated annually. For symmetric CMKs (p. 142) with imported key material (p. 159), it also allows key administrators to download the import token and public key that they need to import key material into the CMK. For asymmetric CMKs (p. 142), it allows key administrators to download the public key (p. 53) of the CMK.
- **kms:Delete*** – Allows key administrators to delete an alias that points to this CMK. For symmetric CMKs with imported key material (p. 159), it lets the key administrator, delete the imported key material. Note that this permission does not allow key administrators to delete the CMK (p. 172).
- **kms:ImportKeyMaterial** – Allows key administrators to import key material into the CMK. This permission is included in the key policy only when you create a CMK with no key material (p. 162).

  Note
  This permission is not shown in the preceding example policy statement.
- **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. 172).
- **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. 24). 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. 46).

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. 76).

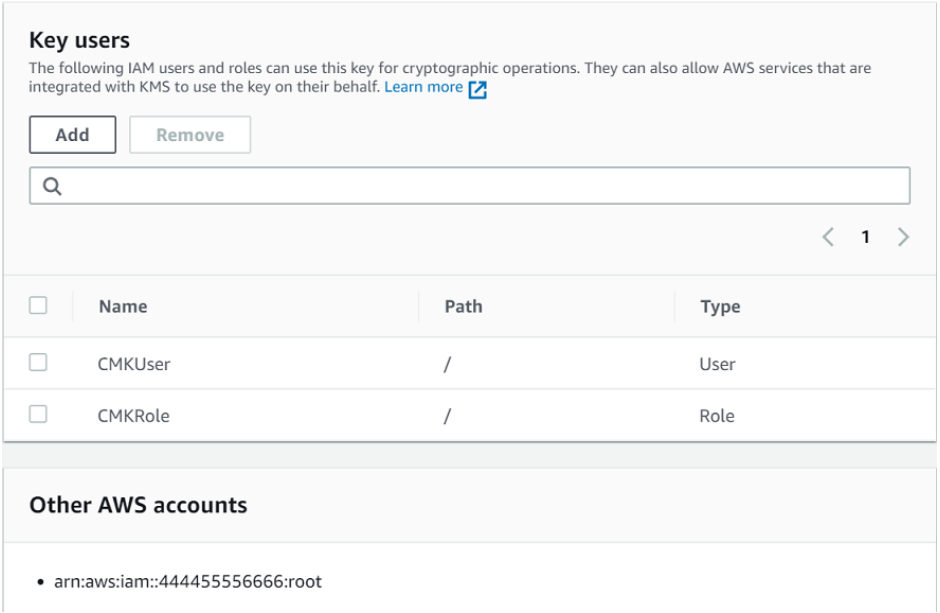
Allows key users to use the CMK

The default key policy that the console creates for symmetric CMKs allows you to choose IAM users and roles in the account, and external AWS accounts, and make them *key users*.

The console adds two policy statements to the key policy for key users.

- Use the CMK directly (p. 66) — The first key policy statement gives key users permission to use the CMK directly for all supported cryptographic operations (p. 10) for that type of CMK.
- Use the CMK with AWS services (p. 67) — The second policy statement gives key users permission to allow AWS services that are integrated with AWS KMS to use the CMK on their behalf to protect resources, such as Amazon Simple Storage Service buckets (p. 274) and Amazon DynamoDB tables (p. 243).

You can add IAM users, IAM roles, and other 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. For more information about allowing users in other AWS accounts to use the CMK, see Allowing users in other accounts to use a CMK (p. 82).



When you use the console's default view to change the list of key users, the console changes 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 for symmetric CMKs.

```
{
  "Sid": "Allow use of the key",
  "Effect": "Allow",
  "Principal": {"AWS": [
    "arn:aws:iam::111122223333:user/CMKUser",
    "arn:aws:iam::111122223333:role/CMKRole",
    "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/CMKUser",
    "arn:aws:iam::111122223333:role/CMKRole",
    "arn:aws:iam::444455556666:root"
  ]},
  "Action": [
    "kms:CreateGrant",
    "kms:ListGrants",
    "kms:RevokeGrant"
  ],
  "Resource": "*",
  "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}
}
```

Allows key users to use a CMK for cryptographic operations

Key users have permission to use the CMK directly in all cryptographic operations (p. 10) supported on the CMK. They can also use the DescribeKey operation to get detailed information about the CMK in the AWS KMS console or by using the AWS KMS API operations.

By default, the AWS KMS console adds key users statements like those in the following examples to the default key policy. Because they support different API operations, the actions in the policy statements for symmetric CMKs, asymmetric CMKs for public key encryption, and asymmetric CMKs for signing and verification are slightly different.

Symmetric CMKs

The console adds the following statement to the key policy for symmetric CMKs.

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

Asymmetric CMKs for public key encryption

The console adds the following statement to the key policy for asymmetric CMKs with a key usage of **Encrypt and decrypt**.

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

Asymmetric CMKs for signing and verification

The console adds the following statement to the key policy for asymmetric CMKs with a key usage of **Sign and verify**.

```
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
    "Principal": {"AWS": "arn:aws:iam::111122223333:user/CMKUser"},
    "Action": [
        "kms:DescribeKey",
        "kms:GetPublicKey",
        "kms:Sign",
```

```
        "kms:Verify"
    ],
    "Resource": "*"
}
```

The actions in these statements give the key users some of the following permissions.

- kms:Encrypt – Allows key users to encrypt data with this CMK.
- kms:Decrypt – Allows key users to decrypt data with this CMK.
- kms:DescribeKey – Allows key users to get detailed information about this CMK including its identifiers, creation date, and key state. It also allows the key users to display details about the CMK in the AWS KMS console.
- **kms:GenerateDataKey*** – Allows key users to request a symmetric data key or an asymmetric data key pair for client-side cryptographic operations. The console uses the * wildcard character to represent permission for the following API operations: GenerateDataKey, GenerateDataKeyWithoutPlaintext, GenerateDataKeyPair, and GenerateDataKeyPairWithoutPlaintext.
- kms:GetPublicKey – Allows key users to download the public key of the asymmetric CMK. Parties with whom you share this public key can encrypt data outside of AWS KMS. However, those ciphertexts can be decrypted only by calling the Decrypt operation in AWS KMS.
- kms:ReEncrypt* – Allows key users to re-encrypt data that was originally encrypted with this CMK, or to use this CMK to re-encrypt previously encrypted data. The ReEncrypt operation requires access to both source and destination CMKs. To accomplish this, you can allow the `kms:ReEncryptFrom` permission on the source CMK and `kms:ReEncryptTo` permission on the destination CMK. However, for simplicity, the console allows `kms:ReEncrypt*` (with the `*` wildcard character) on both CMKs.
- kms:Sign – Allows key users to sign messages with this CMK.
- kms:Verify – Allows key users to verify signatures with this CMK.

Allows key users to use the CMK with AWS services

The default key policy in the console also gives key users permission to allow AWS services that are integrated with AWS KMS (p. 238) to use the CMK, particularly services that use grants.

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. 126). These grants allow the integrated AWS service to use the CMK to protect resources in the account.

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

For example, key users can use these permissions on the CMK in the following ways.

- 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. 253).

- 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. 262).
- Use this CMK with other AWS services integrated with AWS KMS (p. 238), specifically the services that use grants, to create, manage, or use encrypted resources with those services.

The kms:GrantIsForAWSResource (p. 116) condition key allows key users to create and manage grants, but only when the grantee is an AWS service that uses grants. The permission allows key users to use *all* of the integrated services that use grants. However, you can create a custom key policy that allows particular AWS services to use the CMK on the key user's behalf. For more information, see the kms:ViaService (p. 122) condition key.

Key users need these grant permissions to use their CMK with integrated services, but these permissions are not sufficient. Key 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.

Example key policy

The following example shows a complete key policy for a symmetric CMK. This key policy combines the example policy statements from the preceding default key policy (p. 61) 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 CMKUser, IAM role CMKRole, 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/CMKUser",
        "arn:aws:iam::111122223333:role/CMKRole",
        "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/CMKUser",
        "arn:aws:iam::111122223333:role/CMKRole",
        "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 | Tags

Key policy

Switch to policy view

Key administrators

Choose the IAM users and roles who can administer this key through the KMS API. You might need to add additional permissions for the users or roles to administer this key from this console. Learn more

Add | Remove

1

Name	Path	Type
KMSAdminUser	/	User
KMSAdminRole	/	Role

Key deletion

☑ Allow key administrators to delete this key

Key users

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

1

Name	Path	Type
CMKUser	/	User
CMKRole	/	Role

Other AWS accounts

- arn:aws:iam::444455556666:root

Add other AWS accounts

Viewing a key policy

You can view the key policy for an AWS KMS customer managed CMK (p. 4) or an AWS managed CMK (p. 4) in your account by using the AWS Management Console or the GetKeyPolicy operation in the AWS KMS API. You cannot use these techniques to view the key policy of a CMK in a different AWS account.

To learn more about AWS KMS key policies, see Using key policies in AWS KMS (p. 59). To learn how to determine which users and roles have access to a CMK, see the section called "Determining access" (p. 130).

Topics

- Viewing a key policy (console) (p. 71)
- Viewing a key policy (AWS KMS API) (p. 73)

Viewing a key policy (console)

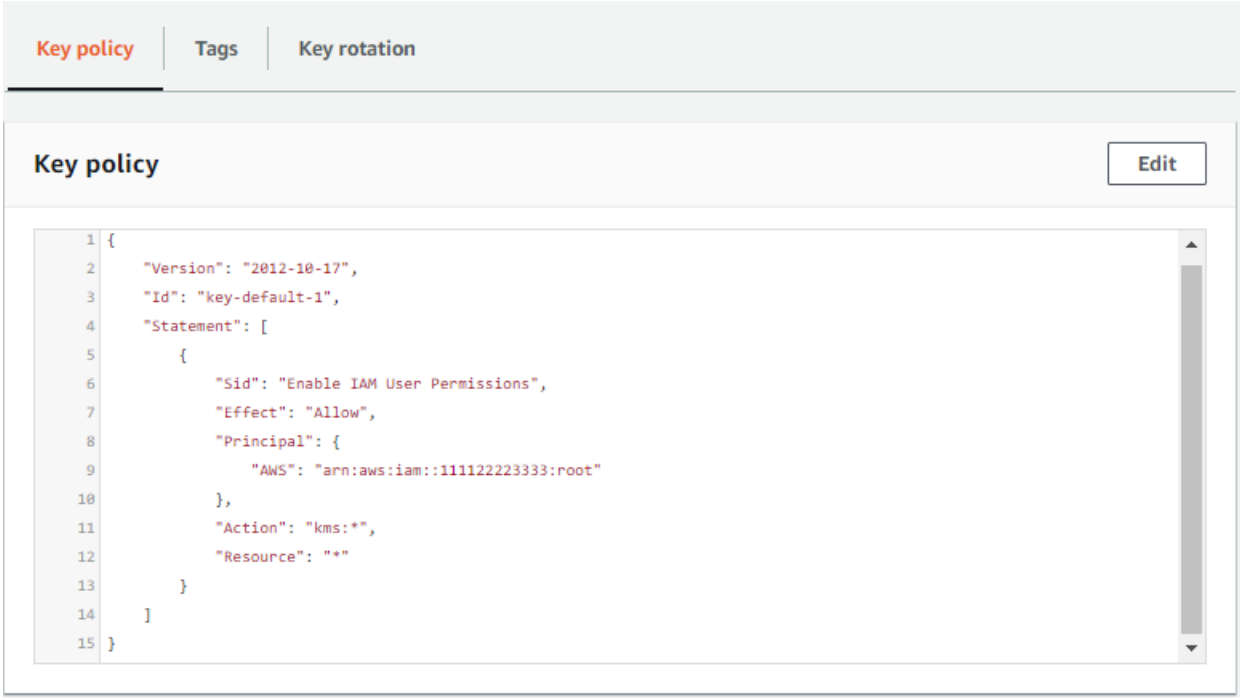
Authorized users can view the key policy for an AWS managed CMK (p. 4) or a customer managed CMK (p. 4) on the **Key policy** tab of the AWS Management Console.

To view the key policy for a CMK in the AWS Management Console, you must have kms:ListAliases, kms:DescribeKey, and kms:GetKeyPolicy permissions.

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 AWS creates and manages for you, in the navigation pane, choose **AWS managed keys**. To view the keys in your account that you create and manage, in the navigation pane choose **Customer managed keys**.
4. In the list of CMKs, choose the alias or key ID of the CMK that you want to examine.
5. Choose the **Key policy** tab.

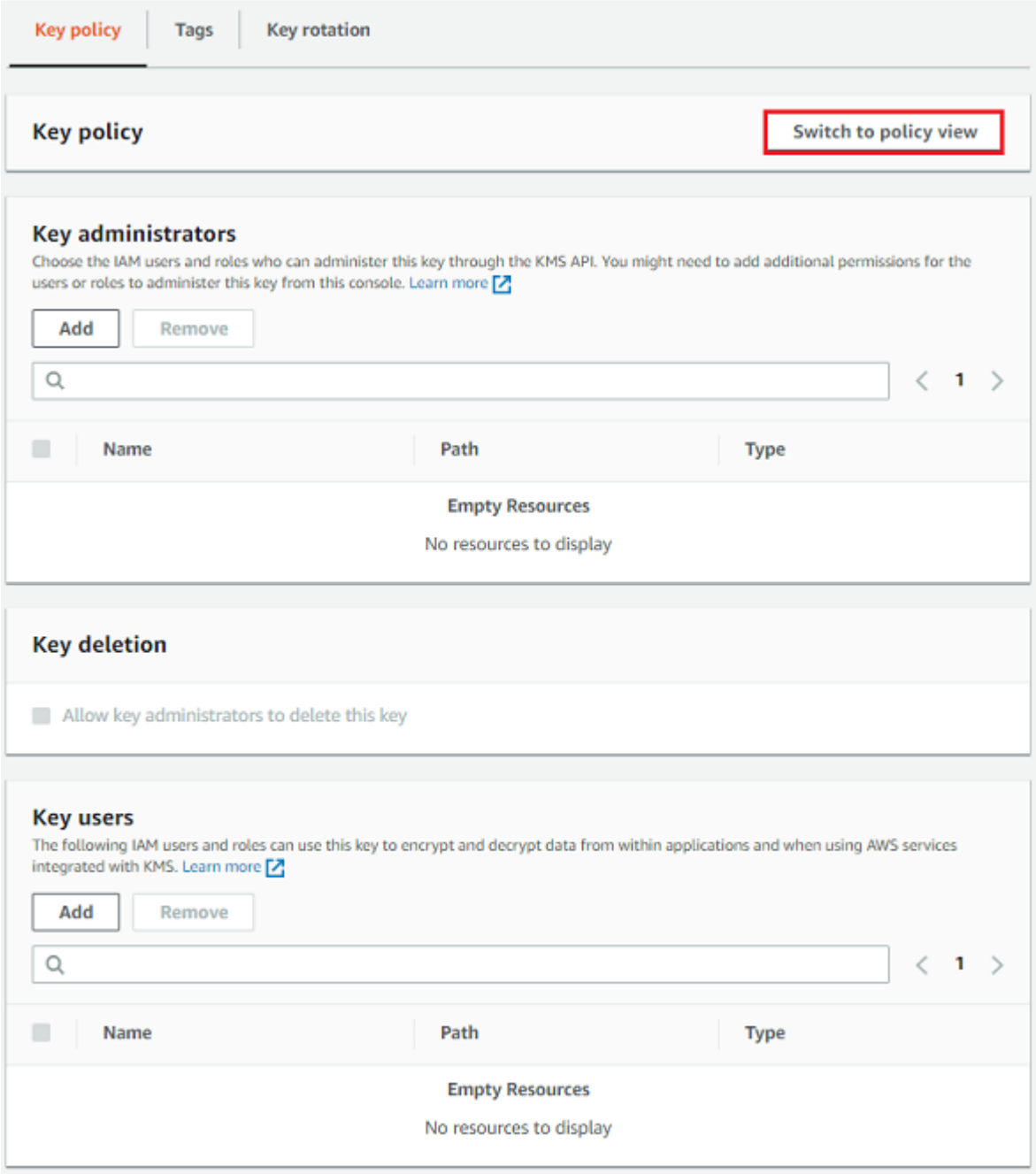
   In the **Key policy** section, you might see the key policy document. This is *policy view*. In the key policy statements, you can see the principals who have been given access to the CMK by the key policy, and you can see the actions they can perform.

   The following example shows the policy view for the default key policy (p. 61).



Or, if you created the CMK in the AWS Management Console, you will see the *default view* with sections for **Key administrators**, **Key deletion**, and **Key Users**. To see the key policy document, choose **Switch to policy view**.

The following example shows the default view for the default key policy (p. 61).



Viewing a key policy (AWS KMS API)

To get the key policy for an AWS managed CMK (p. 4) or a customer managed CMK (p. 4) in your AWS account, use the GetKeyPolicy operation in the AWS KMS API. You cannot use this operation to view a key policy in a different account.

The following example uses the get-key-policy command in the AWS Command Line Interface (AWS CLI), but you can use any AWS SDK to make this request.

Note that the `PolicyName` parameter is required even though `default` is its only valid value. Also, this command requests the output in text, rather than JSON, to make it easier to view.

Before running this command, replace the example key ID with a valid one from your account.

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

The response should be similar to the following one, which returns the default key policy (p. 61).

```
{
  "Version" : "2012-10-17",
  "Id" : "key-consolepolicy-3",
  "Statement" : [ {
    "Sid" : "Enable IAM User Permissions",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "arn:aws:iam::111122223333:root"
    },
    "Action" : "kms:*",
    "Resource" : "*"
  } ]
}
```

Changing a key policy

You can change the key policy for a customer master key (CMK) in your AWS account by using the AWS Management Console or the PutKeyPolicy operation. You cannot use these techniques to change the key policy of a CMK in a different AWS account.

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

- You can view the key policy for an AWS managed CMK (p. 4) or a customer managed CMK (p. 4), but you can only change the key policy for a customer managed CMK. The policies of AWS managed CMKs are created and managed by the AWS service that created the CMK in your account.
- 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. 59).
- 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. 76).
- 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 users in other accounts to use a CMK (p. 82).
- The resulting key policy document cannot exceed 32 KB (32,768 bytes).

Topics

- How to change a key policy (p. 74)
- Allowing multiple IAM users to access a CMK (p. 76)

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. 75)
- Using the AWS Management Console policy view (p. 75)
- Using the AWS KMS API (p. 75)

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. 75) or Using the AWS KMS API (p. 75).

1. View the key policy for a customer managed CMK as decribed in Viewing a key policy (console) (p. 71). (You cannot change the key policies of AWS managed keys.)
2. Decide what to change.
   - To add or remove key administrators (p. 62), and to allow or prevent key administrators from deleting the CMK (p. 172), 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. 154).
   - To add or remove key users (p. 64), 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 (p. 10), such as encrypting, decrypting, re-encrypting, and generating data keys.

Using the AWS Management Console policy view

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

1. View the key policy for a customer managed CMK as decribed in Viewing a key policy (console) (p. 71). (You cannot change the key policies of AWS managed keys.)
2. In the **Key Policy** section, choose **Switch to policy view**.
3. Edit the key policy document, and then choose **Save changes**.

Using the AWS KMS API

You can use the PutKeyPolicy operation to change the key policy of a CMK in your AWS account. You cannot use this API on a CMK in a different AWS account.

1. Use the GetKeyPolicy operation to get the existing key policy document, and then save the key policy document to a file. For sample code in multiple programming languages, see Getting a key policy (p. 345).
2. Open the key policy document in your preferred text editor, edit the key policy document, and then save the file.
3. Use the PutKeyPolicy operation to apply the updated key policy document to the CMK. For sample code in multiple programming languages, see Setting a key policy (p. 347).

For an example of copying a key policy from one CMK to another, see the GetKeyPolicy example in the AWS CLI Command Reference.

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. 61). 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 Troubleshooting key access (p. 135).

Keeping key policies up to date

When you use the AWS Management Console to create a customer master key (CMK) (p. 24), 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. 61) 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 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. 77). 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.

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.

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. 77). 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. 61), 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. 49).

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. 172).

> **Note**
> The `kms:ScheduleKeyDeletion` and `kms:CancelKeyDeletion` permissions are included by default when you create a CMK (p. 24) 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, along with key policies (p. 59), to control access to your customer master keys (CMKs) in AWS KMS.

All CMKs must have a key policy. IAM policies are optional. To use an IAM policy to control access to a CMK, the key policy for the CMK must give the account permission to use IAM policies. Specifically, the key policy must include the policy statement that enables IAM policies (p. 61).

IAM policies can control access to any AWS KMS operation. Unlike key policies, IAM policies can control access to multiple CMKs and provide permissions for the operations of several related AWS services. But IAM policies are particularly useful for controlling access to operations, such as CreateKey, that can't be controlled by a key policy because they don't involve any particular CMK.

> **Note**
> This section explains how to use IAM policies to control access to AWS KMS operations. For more general information about IAM, see the IAM User Guide.

Topics

- Overview of IAM policies (p. 78)
- Specifying CMKs in IAM policy statements (p. 78)
- Permissions required to use the AWS KMS console (p. 79)
- AWS managed (predefined) policies for AWS KMS (p. 80)
- Customer managed policy examples (p. 80)

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 call AWS KMS operations.
- **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 get all CMKs and aliases.

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

Like all IAM policies, this policy doesn't have a `Principal` element. When you attach an IAM policy to an IAM user or IAM role, the user or *assumed role user* gets the permissions specified in the policy.

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. 87).

Specifying CMKs in IAM policy statements

Some IAM policies control access to AWS KMS operations, such as CreateKey, that don't involve a particular customer master key (CMK). However, you can specify one or more CMKs as the resource in an IAM policy statement.

> **Note**
> To use an IAM policy to control access to a CMK, the key policy for the CMK must give the account permission to use IAM policies. Specifically, the key policy must include the policy statement that enables IAM policies (p. 61).

To specify particular CMKs in an IAM policy, use the key ARN (p. 12) of the CMK, that is, the Amazon Resource Name (ARN) of the CMK. You cannot use a key id (p. 12), alias name (p. 14), or alias ARN (p. 12) to identify a CMK in an IAM policy statement.

For example, the following IAM policy statement allows the principal to call the DescribeKey, GenerateDataKey, and Decrypt operations on the CMKs listed in the **Resource** element of the policy statement.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:DescribeKey",
      "kms:GenerateDataKey",
      "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"
    ]
  }
}
```

To specify multiple CMKs, use a wildcard character (*). For example, the following policy statement allows the principal to call the specified operations on any CMK in the example account.

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

To determine whether an AWS KMS operation involves a particular CMK, look for the **CMK** value in the **Resources** column of the table in the section called “AWS KMS API permissions reference” (p. 87).

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. 80).

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. 87).

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. 59). 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. 87).

Examples

- Allow a user read-only access to all CMKs through the AWS KMS console (p. 80)
- Allow a user to create CMKs (p. 81)
- Allow a user to encrypt and decrypt with any CMK in a specific AWS account (p. 81)
- Allow a user to encrypt and decrypt with any CMK in a specific AWS account and Region (p. 81)
- Allow a user to encrypt and decrypt with specific CMKs (p. 82)
- Prevent a user from disabling or deleting any CMKs (p. 82)

Allow a user read-only access to all CMKs through the AWS KMS console

The following IAM 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:DescribeKey",
      "kms:GetKeyPolicy",
      "kms:GetKeyRotationStatus",
      "kms:GetPublicKey",
      "kms:ListKeys",
      "kms:ListAliases",
      "kms:ListKeyPolicies",
      "iam:ListUsers",
      "iam:ListRoles"
    ],
    "Resource": "*"
  }
}
```

Allow a user to create CMKs

The following IAM policy allows a user to create CMKs. The value of the `Resource` element is `*` because the `CreateKey` operation does not use any particular AWS KMS resources (CMKs or aliases).

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

Allow a user to encrypt and decrypt with any CMK in a specific AWS account

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

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

Allow a user to encrypt and decrypt with any CMK in a specific AWS account and Region

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

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "kms:Encrypt",
      "kms:Decrypt"
```

```
    ],
    "Resource": [
      "arn:aws:kms:us-west-2:111122223333:key/*"
    ]
  }
}
```

Allow a user to encrypt and decrypt with specific CMKs

The following IAM policy allows a user to encrypt and decrypt data with the two CMKs specified in the `Resource` element. When specifying a CMK in an IAM policy statement (p. 78), use the key ARN of the CMK.

```
{
  "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 IAM 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 Troubleshooting key access (p. 135).

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

Allowing users in other accounts to use a CMK

You can allow IAM users or roles in one AWS account to use a customer master key (CMK) in a different AWS account. You can add these permissions when you create the CMK or change the permissions for an existing CMK.

To give permission to use a CMK to users and roles in another account, you must use two different types of policies:

- The **key policy** for the CMK must give the external account (or users and roles in the external account) permission to use the CMK. The key policy is in the account that owns the CMK.

- You must attach **IAM policies** to IAM users and roles in the external account. These IAM policies delegate the permissions that are specified in the key policy.

In this scenario, the key policy determines who *can* have access to the CMK. The IAM policy determines who *does* have access to the CMK. Neither the key policy nor the IAM policy alone is sufficient—you must change both.

To edit the key policy, you can use the Policy View (p. 75) in the AWS Management Console or use the CreateKey or PutKeyPolicy operations. For help setting the key policy when creating a CMK, see Creating CMKs that other accounts can use (p. 85).

For help with editing IAM policies, see Using IAM policies with AWS KMS (p. 77).

For an example that shows how the key policy and IAM policies work together to allow use of a CMK in a different account, see Example 2: User assumes role with permission to use a CMK in a different AWS account (p. 137).

Topics

- Step 1: Add a key policy statement in the local account (p. 83)
- Step 2: Add IAM policies in the external account (p. 85)
- Creating CMKs that other accounts can use (p. 85)
- Using external CMKs with AWS services (p. 87)

Step 1: Add a key policy statement in the local account

The key policy for a CMK is the primary determinant of who can access the CMK and which operations they can perform. The key policy is always in the account that owns the CMK. Unlike IAM policies, key policies do not specify a resource. The resource is the CMK that is associated with the key policy.

To give an external account permission to use the CMK, add a statement to the key policy that specifies the external account. In the `Principal` element of the key policy, enter the Amazon Resource Name (ARN) of the external account.

When you specify an external account in a key policy, IAM administrators in the external account can use IAM policies to delegate those permissions to any users and roles in the external account. They can also decide which of the actions specified in the key policy the users and roles can perform.

For example, assume that you want to allow account 444455556666 to use a symmetric CMK in account 111122223333. To do that, add a policy statement like the one in the following example to the key policy for the CMK in account 111122223333. This policy statement gives the external account, 444455556666, permission to use the CMK in cryptographic operations for symmetric CMKs.

```
{
    "Sid": "Allow an external account to use this CMK",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::444455556666:root"
        ]
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
```

```
        "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

Instead of giving permission to the external account, you can specify particular external users and roles in the key policy . However, those users and roles cannot use the CMK until IAM administrators in the external account attach the proper IAM policies to their identities. The IAM policies can give permission to all or a subset of the external users and roles that are specified in the key policy. And they can allow all or a subset of the actions specified in the key policy.

Specifying identities in a key policy restricts the permissions that IAM administrators in the external account can provide. However, it makes policy management with two accounts more complex. For example, assume that you need to add a user or role. You must add that identity to the key policy in the account that owns the CMK and create IAM policies in the identity's account.

To specify particular external users or roles in a key policy, in the `Principal` element, enter the Amazon Resource Name (ARN) of a user or role in the external account.

For example, the following example key policy statement allows `ExampleRole` and ExampleUser in account 444455556666 to use a CMK in account 111122223333. This key policy statement gives the external account, 444455556666, permission to use the CMK in cryptographic operations for symmetric CMKs.

```
{
    "Sid": "Allow an external account to use this CMK",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::444455556666:role/ExampleRole",
            "arn:aws:iam::444455556666:user/ExampleUser"
        ]
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

Note

Do not set the Principal to an asterisk (*) in any key policy statement that allows permissions. An asterisk gives every identity in every AWS account permission to use the CMK, unless another policy statement explicitly denies it. Users in other AWS accounts just need corresponding IAM permissions in their own accounts to use the CMK.

You also need to decide which permissions you want to give to the external account. For a list of permissions on CMKs, see AWS KMS API permissions: Actions and resources reference (p. 87).

You can give the external account permission to use the CMK in cryptographic operations (p. 10) and use the CMK with AWS services that are integrated with AWS KMS. To do that, use the **Key Users** section of the AWS Management Console. For details, see Creating CMKs that other accounts can use (p. 85).

To specify other permissions in key policies, edit the key policy document. For example, you might want to give users permission to decrypt but not encrypt, or permission to view the CMK but not use it. To edit the key policy document, you can use the Policy View (p. 75) in the AWS Management Console or the CreateKey or PutKeyPolicy operations.

Step 2: Add IAM policies in the external account

The key policy in the account that owns the CMK sets the valid range for permissions. But, users and roles in the external account cannot use the CMK until you attach IAM policies that delegate those permissions, or use grants to manage access to the CMK. The IAM policies are set in the external account.

If the key policy gives permission to the external account, you can attach IAM policies to any user or role in the account. But if the key policy gives permission to specified users or roles, the IAM policy can only give those permissions to all or a subset of the specified users and roles. If an IAM policy gives CMK access to other external users or roles, it has no effect.

The key policy also limits the actions in the IAM policy. The IAM policy can delegate all or a subset of the actions specified in the key policy. If the IAM policy lists actions that are not specified in the key policy, those permissions are not effective.

The following example IAM policy allows the principal to use the CMK in account 111122223333 for cryptographic operations. To give this permission to users and roles in account `444455556666`, attach the policy to the users or roles in account `444455556666`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow Use Of CMK In Account 111122223333",
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    }
  ]
}
```

Note the following details about this policy:

- Unlike key policies, IAM policy statements do not contain the `Principal` element. In IAM policies, the principal is the identity to which the policy is attached.
- The `Resource` element in the IAM policy identifies the CMK that the principal can use. To specify a CMK, add its key ARN (p. 12) to the `Resource` element.
- You can specify more than one CMK in the `Resource` element. But if you don't specify particular CMKs in the `Resource` element, you might inadvertently give access to more CMKs than you intend.
- To allow the external user to use the CMK with AWS services that integrate with AWS KMS, you might need to add permissions to the key policy or the IAM policy. For details, see Using external CMKs with AWS services (p. 87).

For more information about working with IAM policies, see Using IAM policies (p. 77).

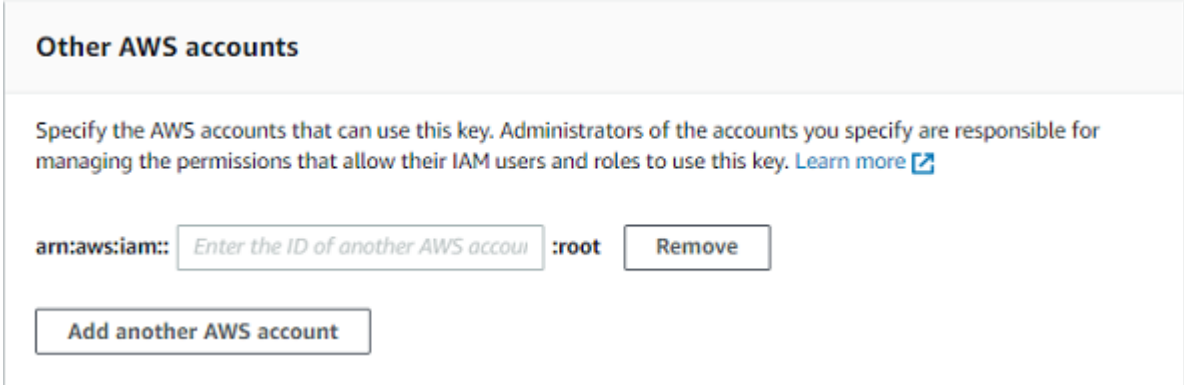
Creating CMKs that other accounts can use

When you use the CreateKey operation create a CMK, you can use its `Policy` parameter to specify a key policy (p. 83) that gives an external account, or external users and roles, permission to use the CMK.

You must also add IAM policies (p. 85) in the external account that delegate these permissions to the account's users and roles, even when users and roles are specified in the key policy. You can change the key policy at any time by using the PutKeyPolicy operation.

When you create a CMK in the AWS Management Console, you also create its key policy. When you select identities in the **Key Administrators** and **Key Users** sections, AWS KMS adds policy statements for those identities to the CMK's key policy.

The **Key Users** section also lets you add external accounts as key users.



When you enter the account ID of an external account, AWS KMS adds two statements to the key policy. This action only affects the key policy. Users and roles in the external account cannot use the CMK until you attach IAM policies (p. 85) to give them some or all of these permissions.

The first policy statement gives the external account permission to use the CMK in cryptographic operations.

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

The second policy statement allows the external account to create, view, and revoke grants on the CMK, but only when the request comes from an AWS service that is integrated with AWS KMS. These permissions allow other AWS services, such as that encrypt user data to use the CMK.

These permissions are designed for CMKs that encrypt user data in AWS services, such as Amazon WorkMail (p. 285). These services typically use grants to get the permissions they need to use the CMK on the user's behalf. For details, see Using external CMKs with AWS services (p. 87).

```
{
    "Sid": "Allow attachment of persistent resources",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::444455556666:root"
```

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

If these permissions don't meet your needs, you can edit them in the console policy view (p. 75) or by using the PutKeyPolicy operation. You can specify particular external users and role instead of giving permission to the external account. You can change the actions that the policy specifies. And you can use global and AWS KMS policy conditions to refine the permissions.

Using external CMKs with AWS services

You can give a user in a different account permission to use your CMK with a service that is integrated with AWS KMS. For example, a user in an external account can use your CMK to encrypt the objects in an Amazon S3 bucket (p. 274) or to encrypt the secrets they store in AWS Secrets Manager (p. 264).

The key policy must give the external user or the external user's account permission to use the CMK. In addition, you need to attach IAM policies to the identity that gives the user permission to use the AWS service.

Also, the service might require that users have additional permissions in the key policy. For example, it might require permission to create, list, and revoke grants on the CMK. Or it might require particular IAM policies. For details, see the documentation for the service.

Finally, the lists of CMKs displayed in the AWS Management Console for an integrated service do not include CMKs in external accounts. This is true even when the user or role has permission to use them. To use an external account's CMK, the user must enter the ID or ARN of the CMK. For details, see the service's console documentation.

AWS KMS API permissions: Actions and resources reference

The Actions and Resources Table is designed to help you define access control (p. 57) in key policies (p. 59) and IAM policies (p. 77). 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. 61), 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** lists the AWS KMS resources to which the permissions apply. AWS KMS supports two resource types: a customer master key (CMK) and an alias. In a key policy, the value of the `Resource` element is always `*`, which indicates the CMK to which the key policy is attached.

  Use the following values to represent an AWS KMS resource in an IAM policy.

CMK

When the resource is a customer master key (CMK), use its key ARN (p. 12). For help, see the section called "Finding the key ID and ARN" (p. 40).

`arn:AWS_partition_name:kms:AWS_Region:AWS_account_ID:key/key_ID`

For example:

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

Alias

When the resource is an alias, use its alias ARN (p. 12). To get the alias ARN, use the ListAliases operation.

`arn:AWS_partition_name:kms:AWS_region:AWS_account_ID:alias/alias_name`

For example:

arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias

`*` (asterisk)

When the permission doesn't apply to a particular resource (CMK or alias), use an asterisk (`*`).

In an IAM policy for an AWS KMS permission, an asterisk in the `Resource` element indicates all AWS KMS resources (CMKs and aliases). You can also use an asterisk in the `Resource` element when the AWS KMS permission doesn't apply to any particular CMKs or aliases. For example, when allowing or denying `kms:CreateKey` or `kms:ListKeys` permission, you can set the `Resource` element to `*` or to an account-specific variation, such as `arn:AWS_partition_name:kms:AWS_region:AWS_account_ID:*`.

- **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. 99). 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 (for IAM policies)	AWS KMS condition keys

CancelKeyDeletion
`kms:CancelKeyDeletion`
	Key policy	CMK	`kms:CallerAccount`
			`kms:CustomerMasterKeySpec`
			`kms:CustomerMasterKeyUsage`
			`kms:KeyOrigin`
			`kms:ViaService`

ConnectCustomKeyStore `kms:ConnectCustomKeyStore`	IAM policy	`*`	`kms:CallerAccount`
CreateAlias `kms:CreateAlias`	IAM policy (for the alias)	Alias	None (when controlling access to the alias)
	Key policy (for the CMK)	CMK	`kms:CallerAccount`

API operations and actions (permissions)	Policy type	Resources (for IAM policies)	AWS KMS condition keys

To use this operation, the caller needs `kms:CreateAlias` permission on two resources:
• The alias (in an IAM policy)
• The CMK (in a key policy)
			`kms:CustomerMasterKeySpec`
			`kms:CustomerMasterKeyUsage`
			`kms:KeyOrigin`
			`kms:ViaService`

CreateCustomKeyStore `kms:CreateCustomKeyStore`	IAM policy	*	`kms:CallerAccount`

CreateGrant
`kms:CreateGrant`
	Key policy	CMK	`kms:CallerAccount`
			`kms:CustomerMasterKeySpec`
			`kms:CustomerMasterKeyUsage`
			`kms:KeyOrigin`
			`kms:EncryptionContext:`
			`kms:EncryptionContextKeys`
			`kms:GrantConstraintType`
			`kms:GranteePrincipal`
			`kms:GrantIsForAWSResource`
			`kms:GrantOperations`
			`kms:RetiringPrincipal`
			`kms:ViaService`

CreateKey
`kms:CreateKey`
	IAM policy	*	`kms:BypassPolicyLockoutSafetyCheck`
			`kms:CustomerMasterKeySpec`
			`kms:CustomerMasterKeyUsage`
			`kms:KeyOrigin`

Decrypt
`kms:Decrypt`
	Key policy	CMK	`kms:CallerAccount`
			`kms:CustomerMasterKeySpec`
			`kms:CustomerMasterKeyUsage`
			`kms:KeyOrigin`
			`kms:EncryptionAlgorithm`
			`kms:EncryptionContext:`
			`kms:EncryptionContextKeys`
			`kms:ViaService`

API operations and actions (permissions)	Policy type	Resources (for IAM policies)	AWS KMS condition keys
DeleteAlias `kms:DeleteAlias` To use this operation, the caller needs `kms:DeleteAlias` permission on two resources: • The alias (in an IAM policy) • The CMK (in a key policy)	IAM policy (for the alias)	Alias	None (when controlling access to the alias)

	Key policy (for the CMK)	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`

DeleteCustomKeyStore `kms:DeleteCustomKeyStore`	IAM policy	*	`kms:CallerAccount`

DeleteImportedKeyMaterial
 	
`kms:DeleteImportedKeyMaterial`
	Key policy	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`

DescribeCustomKeyStores `kms:DescribeCustomKeyStores`	IAM policy	*	`kms:CallerAccount`

DescribeKey
 	
`kms:DescribeKey`
	Key policy	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`

DisableKey
 	
`kms:DisableKey`
	Key policy	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`

API operations and actions (permissions)	Policy type	Resources (for IAM policies)	AWS KMS condition keys

DisableKeyRotation
`kms:DisableKeyRotation`
	Key policy	CMK	`kms:CallerAccount`
			`kms:CustomerMasterKeySpec`
			`kms:CustomerMasterKeyUsage`
			`kms:KeyOrigin`
			`kms:ViaService`

DisconnectCustomKeyStore `kms:DisconnectCustomKeyStore`	IAM policy	*	

EnableKey
`kms:EnableKey`
	Key policy	CMK	`kms:CallerAccount`
			`kms:CustomerMasterKeySpec`
			`kms:CustomerMasterKeyUsage`
			`kms:KeyOrigin`
			`kms:ViaService`

EnableKeyRotation
`kms:EnableKeyRotation`
	Key policy	CMK (symmetric only)	`kms:CallerAccount`
			`kms:CustomerMasterKeySpec`
			`kms:CustomerMasterKeyUsage`
			`kms:KeyOrigin`
			`kms:ViaService`

Encrypt
`kms:Encrypt`
	Key policy	CMK	`kms:CallerAccount`
			`kms:CustomerMasterKeySpec`
			`kms:CustomerMasterKeyUsage`
			`kms:KeyOrigin`
			`kms:EncryptionAlgorithm`
			`kms:EncryptionContext:`
			`kms:EncryptionContextKeys`
			`kms:ViaService`

API operations and actions (permissions)	Policy type	Resources (for IAM policies)	AWS KMS condition keys

GenerateDataKey
 	
`kms:GenerateDataKey`
	Key policy	CMK (symmetric only)	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			`kms:EncryptionAlgorithm`
			
			`kms:EncryptionContext:`
			
			`kms:EncryptionContextKeys`
			
			`kms:ViaService`

GenerateDataKeyPair
 	
`kms:GenerateDataKeyPair`
	Key policy	CMK (symmetric only)
		
		`GenerateDataKeyPair` and `GenerateDataKeyPairWithoutPlaintext` generate an asymmetric data key pair that is protected by a symmetric CMK.
			`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			`kms:DataKeyPairSpec`
			
			`kms:EncryptionAlgorithm`
			
			`kms:EncryptionContext:`
			
			`kms:EncryptionContextKeys`
			
			`kms:ViaService`

GenerateDataKeyPairWithoutPlaintext
 	
`kms:GenerateDataKeyPairWithoutPlaintext`
	Key policy	CMK (symmetric only)
		
		`GenerateDataKeyPair` and `GenerateDataKeyPairWithoutPlaintext` generate an asymmetric data key pair that is protected by a symmetric CMK.
			`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			`kms:DataKeyPairSpec`
			
			`kms:EncryptionAlgorithm`
			
			`kms:EncryptionContext:`
			
			`kms:EncryptionContextKeys`
			
			`kms:ViaService`

API operations and actions (permissions)	Policy type	Resources (for IAM policies)	AWS KMS condition keys

GenerateDataKeyWithoutPlaintext
 	
`kms:GenerateDataKeyWithoutPlaintext`
	Key policy	CMK (symmetric only)	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			`kms:EncryptionAlgorithm`
			
			`kms:EncryptionContext:`
			
			`kms:EncryptionContextKeys`
			
			`kms:ViaService`

GenerateRandom `kms:GenerateRandom`	IAM policy	*	None

GetKeyPolicy
 	
`kms:GetKeyPolicy`
	Key policy	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`

GetKeyRotationStatus
 	
`kms:GetKeyRotationStatus`
	Key policy	CMK (symmetric only)	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`

GetParametersForImport
 	
`kms:GetParametersForImport`
	Key policy	CMK (symmetric only)	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`
			
			`kms:WrappingAlgorithm`
			
			`kms:WrappingKeySpec`

API operations and actions (permissions)	Policy type	Resources (for IAM policies)	AWS KMS condition keys

GetPublicKey
 	
`kms:GetPublicKey`
	Key policy	CMK (asymmetric only)	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`

ImportKeyMaterial
 	
`kms:ImportKeyMaterial`
	Key policy	CMK (symmetric only)	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ExpirationModel`
			
			`kms:ValidTo`
			
			`kms:ViaService`

ListAliases `kms:ListAliases`	IAM policy	*	None

ListGrants
 	
`kms:ListGrants`
	Key policy	CMK	`kms:CallerAccount`
			`kms:GrantIsForAWSResource`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			`kms:ViaService`

ListKeyPolicies
 	
`kms:ListKeyPolicies`
	Key policy	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`

ListKeys `kms:ListKeys`	IAM policy	*	None

API operations and actions (permissions)	Policy type	Resources (for IAM policies)	AWS KMS condition keys

ListResourceTags
 	
`kms:ListResourceTags`
	Key policy	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`

ListRetirableGrants `kms:ListRetirableGrants`	IAM policy	*	None

PutKeyPolicy
 	
`kms:PutKeyPolicy`
	Key policy	CMK	`kms:BypassPolicyLockoutSafetyCheck`
			
			`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`

ReEncrypt
 	
`kms:ReEncryptFrom`
 	
`kms:ReEncryptTo`
 	
To use this operation, the caller needs permission on two CMKs:
 	
• `kms:ReEncryptFrom` on the CMK used to decrypt
• `kms:ReEncryptTo` on the CMK used to encrypt
	Key policy	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			`kms:EncryptionAlgorithm`
			
			`kms:EncryptionContext:`
			
			`kms:EncryptionContextKeys`
			
			`kms:ReEncryptOnSameKey`
			
			`kms:ViaService`

RetireGrant Permission to retire a grant is specified in the grant. You cannot control access to this operation in a policy. For more information, see RetireGrant in the *AWS Key Management Service API Reference*.	Not applicable	Not applicable	Not applicable

API operations and actions (permissions)	Policy type	Resources (for IAM policies)	AWS KMS condition keys

RevokeGrant
 	
`kms:RevokeGrant`
	Key policy	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:GrantIsForAWSResource`
			
			`kms:ViaService`

ScheduleKeyDeletion
 	
`kms:ScheduleKeyDeletion`
	Key policy	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			`kms:ViaService`

Sign
 	
`kms:Sign`
	Key policy	CMK (asymmetric only)	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			`kms:MessageType`
			
			`kms:SigningAlgorithm`
			
			`kms:ViaService`

TagResource
 	
`kms:TagResource`
	Key policy	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`
			
			`aws:RequestTag` (AWS global condition key)
			
			`aws:TagKeys` (AWS global condition key)

API operations and actions (permissions)	Policy type	Resources (for IAM policies)	AWS KMS condition keys

UntagResource
 	
`kms:UntagResource`
	Key policy	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`
			
			`aws:RequestTag` (AWS global condition key)
			
			`aws:TagKeys` (AWS global condition key)

UpdateAlias `kms:UpdateAlias` To use this operation, the caller needs `kms:UpdateAlias` permission on three resources: • The alias • The CMK that alias currently points to • The CMK that is specified in the `UpdateAlias` request	IAM policy (for the alias)	Alias	None (when controlling access to the alias)

	Key policy (for the CMKs)	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`

UpdateCustomKeyStore `kms:UpdateCustomKeyStore`	IAM policy	*	`kms:CallerAccount`

UpdateKeyDescription
 	
`kms:UpdateKeyDescription`
	Key policy	CMK	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:ViaService`

Verify
 	
`kms:Verify`
	Key policy	CMK (asymmetric only)	`kms:CallerAccount`
			
			`kms:CustomerMasterKeySpec`
			
			`kms:CustomerMasterKeyUsage`
			
			`kms:KeyOrigin`
			
			`kms:MessageType`
			
			`kms:SigningAlgorithm`
			
			`kms:ViaService`

Using policy conditions with AWS KMS

You can specify conditions in the key policies and AWS Identity and Access Management policies (IAM policies (p. 77)) 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. 98)
- AWS KMS condition keys (p. 99)

AWS global condition keys

AWS defines global condition keys, a set of policy conditions keys for all AWS services that use IAM for access control. You can use global condition keys in AWS KMS key policies and IAM policies. For example, you can use the aws:PrincipalArn condition key to allow access to a customer master key (CMK) only when the principal in the request is represented by the Amazon Resource Name (ARN) in the condition key value.

AWS KMS supports all AWS global condition keys except for the following ones:

- aws:ResourceTag
- aws:SourceAccount
- aws:SourceArn

For information about AWS global condition keys, including the types of requests in which they are available, see AWS Global Condition Context Keys in the *IAM User Guide*. For examples of using global condition keys in IAM policies, see Controlling Access to Requests and Controlling Tag Keys in the *IAM User Guide*.

The following topics provide special guidance for using condition keys based on IP addresses and VPC endpoints.

Topics

- Using the IP address condition in policies with AWS KMS permissions (p. 98)
- Using VPC endpoint conditions in policies with AWS KMS permissions (p. 99)

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. 238). 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. 223), 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. 223) 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 global condition keys in AWS KMS key policies (p. 224) 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. 98) 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. 19) when controlling access to an AWS KMS symmetric customer master key (CMK).

Conditions for an API operation request

Many of the AWS KMS condition keys control access to a CMK based on the value of a parameter in the request for an API operation. For example, you can use the kms:CustomerMasterKeySpec (p. 102) condition key in an IAM policy to allow use of the CreateKey operation only when the value of the `CustomerMasterKeySpec` parameter in the `CreateKey` request is `RSA_4096`.

This type of condition works even when the parameter doesn't appear in the request, such as when you use the parameter's default value. For example you can use the kms:CustomerMasterKeySpec (p. 102) condition key to allow users to use the `CreateKey` operation only when the value of the `CustomerMasterKeySpec` parameter is `SYMMETRIC_DEFAULT`, which is the default value.

This condition allows requests that have the `CustomerMasterKeySpec` parameter with the `SYMMETRIC_DEFAULT` value and requests that have no `CustomerMasterKeySpec` parameter.

Conditions for CMKs used in API operations

Some of the AWS KMS condition keys control access to operations based on a property of the CMK that is used in the operation. For example, you can use the kms:KeyOrigin (p. 118) condition to allow principals to call GenerateDataKey on a CMK only when the `Origin` of the CMK is `AWS_KMS`. To find out if a condition key can be used in this way, see the description of the condition key.

The operation must be a *CMK resource operation*, that is, an operation that is authorized for a particular CMK. To identify the CMK resource operations, in the Actions and Resources Table (p. 88), look for a value of `CMK` in the `Resources` column for the operation. If you use this type of condition key with an operation that is not authorized for a particular CMK resource, like ListKeys, the permission is not effective because the condition can never be satisfied. There is no CMK resource involved in authorizing the `ListKeys` operation and no `CustomerMasterKeySpec` property.

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

Topics

- kms:BypassPolicyLockoutSafetyCheck (p. 100)
- kms:CallerAccount (p. 102)
- kms:CustomerMasterKeySpec (p. 102)
- kms:CustomerMasterKeyUsage (p. 104)
- kms:DataKeyPairSpec (p. 105)
- kms:EncryptionAlgorithm (p. 106)
- kms:EncryptionContext: (p. 107)
- kms:EncryptionContextKeys (p. 112)
- kms:ExpirationModel (p. 114)
- kms:GrantConstraintType (p. 115)
- kms:GrantIsForAWSResource (p. 116)
- kms:GrantOperations (p. 116)
- kms:GranteePrincipal (p. 117)
- kms:KeyOrigin (p. 118)
- kms:MessageType (p. 119)
- kms:ReEncryptOnSameKey (p. 120)
- kms:RetiringPrincipal (p. 120)
- kms:SigningAlgorithm (p. 121)
- kms:ValidTo (p. 122)
- kms:ViaService (p. 122)
- kms:WrappingAlgorithm (p. 125)
- kms:WrappingKeySpec (p. 126)

kms:BypassPolicyLockoutSafetyCheck

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:BypassPolicyLockoutSafetyCheck`	Boolean	`CreateKey`	IAM policies only

AWS KMS condition keys	Condition type	API operations	Policy type
		`PutKeyPolicy`	Key policies and IAM 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 overriden 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:CustomerMasterKeySpec (p. 102)
- kms:KeyOrigin (p. 118)
- kms:CustomerMasterKeyUsage (p. 104)

kms:CallerAccount

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:CallerAccount`	String	All AWS KMS operations *except* for `CreateKey`, `GenerateRandom`, `ListAliases`, `ListKeys`, `ListRetirableGrants`, and `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. 122).

```
{
  "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:CustomerMasterKeySpec

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:CustomerMasterKeySpec`	String	`CreateKey` CMK resource operations	IAM policies Key policies and IAM policies

The `kms:CustomerMasterKeySpec` condition key controls access to operations based on the value of the `CustomerMasterKeySpec` property of the CMK that is created by or used in the operation.

You can use this condition key in an IAM policy to control access to the CreateKey operation based on the value of the CustomerMasterKeySpec parameter in a `CreateKey` request. For example, you can use this condition to allow users to create only symmetric CMKs or only CMKs with RSA keys.

The following example IAM policy statement uses the `kms:CustomerMasterKeySpec` condition key to allow the principals to create a CMK only when the `CustomerMasterKeySpec` in the request is `RSA_4096`.

```
{
  "Effect": "Allow",
  "Action": "kms:CreateKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:CustomerMasterKeySpec": "RSA_4096"
    }
  }
}
```

You can also use the `kms:CustomerMasterKeySpec` condition key to control access to operations that use or manage a CMK based on the `CustomerMasterKeySpec` property of the CMK used for the operation. The operation must be a *CMK resource operation*, that is, an operation that is authorized for a particular CMK. To identify the CMK resource operations, in the Actions and Resources Table (p. 88), look for a value of `CMK` in the `Resources` column for the operation.

For example, the following IAM policy allows principals to perform the specified CMK resource operations, but only with the symmetric CMKs in the account.

```
{
  "Effect": "Allow",
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:DescribeKey"
  ],
  "Resource": {
      "arn:aws:kms:us-west-2:111122223333:key/*"
  },
  "Condition": {
    "StringEquals": {
      "kms:CustomerMasterKeySpec": "SYMMETRIC_DEFAULT"
    }
  }
}
```

See also

- kms:BypassPolicyLockoutSafetyCheck (p. 100)
- kms:CustomerMasterKeyUsage (p. 104)
- kms:DataKeyPairSpec (p. 105)
- kms:KeyOrigin (p. 118)

kms:CustomerMasterKeyUsage

AWS KMS condition keys	Condition type	API operations	Policy type
kms:CusomterMasterKeyUsage	String	CreateKey CMK resource operations	IAM policies Key policies and IAM policies

The kms:CustomerMasterKeyUsage condition key controls access to operations based on the value of the KeyUsage property of the CMK that is created by or used in the operation.

You can use this condition key to control access to the CreateKey operation based on the value of the KeyUsage parameter in the request. Valid values for KeyUsage are ENCRYPT_DECRYPT and SIGN_VERIFY.

For example, you can allow a user to create a CMK only when the KeyUsage is ENCRYPT_DECRYPT or deny a user permission when the KeyUsage is SIGN_VERIFY.

The following example IAM policy statement uses the kms:CustomerMasterKeyUsage condition key to allow a user to create a CMK only when the KeyUsage is ENCRYPT_DECRYPT.

```
{
  "Effect": "Allow",
  "Action": "kms:CreateKey",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:CustomerMasterKeyUsage": "ENCRYPT_DECRYPT"
    }
  }
}
```

You can also use the kms:CustomerMasterKeyUsage condition key to control access to operations that use or manage a CMK based on the KeyUsage property of the CMK used for the operation. The operation must be a *CMK resource operation*, that is, an operation that is authorized for a particular CMK. To identify the CMK resource operations, in the Actions and Resources Table (p. 88), look for a value of CMK in the Resources column for the operation.

For example, the following IAM policy allows principals to perform the specified CMK resource operations, but only with CMKs in the account that are used for signing and verification.

```
{
  "Effect": "Allow",
  "Action": [
    "kms:CreateGrant",
    "kms:DescribeKey",
    "kms:GetPublicKey",
    "kms:ScheduleKeyDeletion"
  ],
  "Resource": {
      "arn:aws:kms:us-west-2:111122223333:key/*"
  },
  "Condition": {
    "StringEquals": {
      "kms:CustomerMasterKeyUsage": "SIGN_VERIFY"
```

```
    }
  }
}
```

See also

- kms:BypassPolicyLockoutSafetyCheck (p. 100)
- kms:CustomerMasterKeySpec (p. 102)
- kms:KeyOrigin (p. 118)

kms:DataKeyPairSpec

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:DataKeySpec`	String	`GenerateDataKeyPair` `GenerateDataKeyPairWithoutPlaintext`	Key policies and IAM policies

You can use this condition key to control access to the GenerateDataKeyPair and GenerateDataKeyPairWithoutPlaintext operations based on the value of the `KeyPairSpec` parameter in the request. For example, you can allow a user to generate only particular types of data key pairs.

The following example key policy statement uses the `kms:DataKeyPairSpec` condition key to allow a user to use the CMK to generate only RSA data key pairs.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": [
    "kms:GenerateDataKeyPair",
    "kms:GenerateDataKeyPairWithoutPlaintext"
  ],
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "kms:DataKeyPairSpec": "RSA*"
    }
  }
}
```

See also

- kms:CustomerMasterKeySpec (p. 102)
- the section called “kms:EncryptionAlgorithm” (p. 106)
- the section called “kms:EncryptionContext:” (p. 107)
- the section called “kms:EncryptionContextKeys” (p. 112)

kms:EncryptionAlgorithm

AWS KMS condition keys	Condition type	API operations	Policy type

`kms:EncryptionAlgorithm`	String	`Decrypt`
		`Encrypt`
		`GenerateDataKey`
		`GenerateDataKeyPair`
		`GenerateDataKeyPairWithoutPlaintext`
		`GenerateDataKeyWithoutPlaintext`
		`ReEncrypt`
			Key policies and IAM policies

You can use the `kms:EncryptionAlgorithm` condition key to control access to cryptographic operations based on the encryption algorithm that is used in the operation. For the Encrypt, Decrypt, and ReEncrypt operations, it controls access based on the value of the EncryptionAlgorithm parameter in the request. For operations that generate data keys and data key pairs, it controls access based on the encryption algorithm that is used to encrypt the data key.

This condition key has no effect on operations performed outside of AWS KMS, such as encrypting with the public key in an asymmetric CMK pair outside of AWS KMS.

EncryptionAlgorithm parameter in a request

To allow users to use only a particular encryption algorithm with a CMK, use a policy statement with a `Deny` effect and a `StringNotEquals` condition operator. For example, the following example key policy statement prohibits principals who can assume the `ExampleRole` role from using this symmetric CMK in the specified cryptographic operations unless the encryption algorithm in the request is `RSAES_OAEP_SHA_256`.

Unlike a policy statement that allows a user to use a particular encryption algorithm, a policy statement with a double-negative like this one prevents other policies and grants for this CMK from allowing this role to use other encryption algorithms. The `Deny` in this policy statement takes precedence over any key policy or IAM policy with an `Allow` effect, and it takes precedence over all grants for this CMK and its principals.

```
{
  "Sid": "Allow only one encryption algorithm with this asymmetric CMK",
  "Effect": "Deny",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/ExampleRole"
  },
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*"
  ],
  "Resource": "*",
  "Condition": {
    "StringNotEquals": {
      "kms:EncryptionAlgorithm": "RSAES_OAEP_SHA_256"
    }
```

```
  }
}
```

Encryption algorithm used for the operation

You can also use the `kms:EncryptionAlgorithm` condition key to control access to the operations that generate data keys and data key pairs. These operations use only symmetric CMKs and the `SYMMETRIC_DEFAULT` algorithm.

For example, this IAM policy limits its principals to symmetric encryption. It denies access to any CMK in the example account for cryptographic operations unless the encryption algorithm specified in the request or used in the operation is SYMMETRIC_DEFAULT. The addition of GenerateDataKey, GenerateDataKeyWithoutPlaintext, GenerateDataKeyPair, and GenerateDataKeyPairWithoutPlaintext have no immediate practical effect because you can't use an asymmetric CMK or asymmetric encryption algorithm to encrypt a data key or encrypt the private key in a data key pair.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Deny",
    "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:GenerateDataKeyPair*"
  ],
    "Resource": {
      "arn:aws:kms:us-west-2:111122223333:key/*"
    },
  "Condition": {
    "StringNotEquals": {
      "kms:EncryptionAlgorithm": "SYMMETRIC_DEFAULT"
    }
  }
}
```

See also

- kms:SigningAlgorithm (p. 121)

kms:EncryptionContext:

AWS KMS condition keys	Condition type	API operations	Policy type

`kms:EncryptionContext:`	String	`CreateGrant`
		`Encrypt`
		`Decrypt`
		`GenerateDataKey`
		`GenerateDataKeyPair`
		`GenerateDataKeyPairWithoutPlaintext`
		`GenerateDataKeyWithoutPlaintext`
			Key policies and IAM policies

AWS KMS condition keys	Condition type	API operations	Policy type
		`ReEncrypt`	

You can use the `kms:EncryptionContext:` condition key prefix to control access to a symmetric CMK (p. 142) 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. 112) condition key.

An encryption context (p. 19) is a set of nonsecret key–value pairs that you can include in a request for any AWS KMS cryptographic operation that uses a symmetric CMK (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.

You cannot specify an encryption context in a cryptographic operation with an asymmetric CMK (p. 142). The standard asymmetric encryption algorithms that AWS KMS uses do not support an encryption context.

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. Because there can be multiple encryption context pairs in a request, the condition operator must include `ForAnyValue` or `ForAllValues`.

This policy allows the principal to use the CMK in a `GenerateDataKey` request only when at least one of the encryption context pairs in the request is `"AppName": "ExampleApp"`.

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

Requiring multiple encryption context pairs

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 uses the

`ForAllValues` operator to require both of the following encryption context pairs (and no others). 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": {
    "ForAllValues:StringEquals": {
      "kms:EncryptionContext:AppName": "ExampleApp",
      "kms:EncryptionContext:FilePath": "/var/opt/secrets/"
    }
  }
}
```

Case sensitivity of the encryption context condition

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": {
    "ForAnyValue:StringEquals": {
      "kms:EncryptionContext:Appname": "ExampleApp"
    }
  }
}
```

To require a case-sensitive encryption context key, use the kms:EncryptionContextKeys (p. 112) 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": {
    "ForAnyValue:StringEquals": {
      "kms:EncryptionContextKey": "AppName"
    }
  }
}
```

To require a case-sensitive evaluation of both the encryption context key and value, use the `kms:EncryptionContextKeys` 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": {
    "ForAnyValue:StringEquals": {
      "kms:EncryptionContextKeys": "AppName",
      "kms:EncryptionContext:AppName": "ExampleApp"
    }
  }
}
```

Using variables in an encryption context condition

The key and value in an encryption context pair must be simple literal strings. They cannot be integers or objects, or any type that is not fully resolved. If you use a different type, such as an integer or float, AWS KMS interprets it as a literal string.

```
"encryptionContext": {
    "department": "10103.0"
}
```

However, the value in the `kms:EncryptionContext:` condition key pair can be an IAM policy variable. These policy variables are resolved at runtime based on values in the request. For example, `aws:CurrentTime` resolves to the time of the request and `aws:username` resolves to the friendly name of the caller.

You can use these policy variables to create a policy statement with a condition that requires very specific information in an encryption context, such as the caller's user name. Because it contains a variable, you can use the same policy statement for all users who can assume the role. You don't have to write a separate policy statement for each user.

Consider a situation where you want to all users who can assume a role to use the same CMK to encrypt and decrypt their data. However, you want to allow them to decrypt only the data that they encrypted. Start by requiring that every request to AWS KMS include an encryption context where the key is `user` and the value is the caller's AWS user name, such as the following one.

```
"encryptionContext": {
    "user": "bob"
}
```

Then, to enforce this requirement, you can use a policy statement like the one in the following example. This policy statement gives the `TestTeam` role permission to encrypt and decrypt data with the CMK. However, the permission is valid only when the encryption context in the request includes a `"user": "<username>"` pair. To represent the user name, the condition uses the `aws:username` policy variable.

When the request is evaluated, the caller's user name replaces the variable in the condition. As such, the condition requires an encryption context of `"user": "bob"` for "bob" and `"user": "alice"` for "alice."

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/TestTeam"
  },
  "Action": [
    "kms:Decrypt",
    "kms:Encrypt"
  ]
  "Resource": "*",
  "Condition": {
    "ForAnyValue:StringEquals": {
       "kms:EncryptionContext:user": "${aws:username}"
    }
  }
}
```

You can use an IAM policy variable only in the value of the `kms:EncryptionContext:` condition key pair. You cannot use a variable in the key.

You can also use provider-specific context keys in variables. These context keys uniquely identify users who logged into AWS by using web identity federation.

Like all variables, these variables can be used only in the `kms:EncryptionContext:` policy condition, not in the actual encryption context. And they can be used only in the value of the condition, not in the key.

For example, the following key policy statement is similar to the previous one. However, the condition requires an encryption context where the key is `sub` and the value uniquely identifies a user logged into a Amazon Cognito user pool. For details about identifying users and roles in Amazon Cognito, see IAM Roles in the Amazon Cognito Developer Guide.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/TestTeam"
  },
  "Action": [
    "kms:Decrypt",
    "kms:Encrypt"
  ]
  "Resource": "*",
  "Condition": {
    "ForAnyValue:StringEquals": {
       "kms:EncryptionContext:sub": "${cognito-identity.amazonaws.com:sub}"
    }
  }
```

```
}
```

See also

- the section called “kms:EncryptionContextKeys” (p. 112)
- the section called “kms:GrantConstraintType” (p. 115)

kms:EncryptionContextKeys

AWS KMS condition keys	Condition type	API operations	Policy type

`kms:EncryptionContextKeys`	String (list)	`CreateGrant`
		`Decrypt`
		`Encrypt`
		`GenerateDataKey`
		`GenerateDataKeyPair`
		`GenerateDataKeyPairWithoutPlaintext`
		`GenerateDataKeyWithoutPlaintext`
		`ReEncrypt`
			Key policies and IAM policies

You can use the `kms:EncryptionContextKeys` condition key to control access to a symmetric CMK (p. 142) 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. 107) condition key prefix.

You cannot specify an encryption context in a cryptographic operation with an asymmetric CMK (p. 142). The standard asymmetric encryption algorithms that AWS KMS uses do not support an encryption context.

You can use this condition key to control access based on the encryption context (p. 19) in the AWS KMS API request. Encryption context is a set of key–value pairs that you can include in AWS KMS cryptographic operations with symmetric CMKs (Encrypt, Decrypt, GenerateDataKey, GenerateDataKeyWithoutPlaintext, and ReEncrypt) and the CreateGrant operation. Because there can be multiple encryption context pairs in a request, the condition operator must include `ForAnyValue` or `ForAllValues`.

The following example policy statement uses the `kms:EncryptionContextKeys` condition key to allow use of a CMK for the specified operations only when at least one of the encryption context pairs 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": {
    "ForAnyValue: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 uses the `ForAllValues` and `StringEquals` condition operators to allow the specified operations only when the encryption context in the request includes both the `AppName` and `FilePath` keys (and no others), 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": {
    "ForAllValues: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. 107)
- kms:GrantConstraintType (p. 115)

kms:ExpirationModel

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:ExpirationModel`	String	`ImportKeyMaterial`	Key policies and IAM 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. 122) 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. 122) 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. 122)
- kms:WrappingAlgorithm (p. 125)
- kms:WrappingKeySpec (p. 126)

kms:GrantConstraintType

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:GrantConstraintType`	String	`CreateGrant`	Key policies and IAM 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 (p. 19) 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. 107)
- kms:EncryptionContextKeys (p. 112)
- kms:GrantIsForAWSResource (p. 116)

- kms:GrantOperations (p. 116)
- kms:GranteePrincipal (p. 117)
- kms:RetiringPrincipal (p. 120)

kms:GrantIsForAWSResource

AWS KMS condition keys	Condition type	API operations	Policy type

`kms:GrantIsForAWSResource`	Boolean	`CreateGrant`
		`ListGrants`
		`RevokeGrant`
			Key policies and IAM policies

Allows or denies access to the CreateGrant, ListGrants, or RevokeGrant operations when any of the AWS services that are integrated with AWS KMS performs the grant operation on the user's behalf. This condition key does not affect the user's permissions to perform the grant operation directly.

For example, the following key policy statement uses the `kms:GrantIsForAWSResource` condition key. It allows a user to create grants on this CMK only when the grant is created on the user's behalf by any one of the integrated services. It does not allow the user to create grants directly.

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

See also

- kms:GrantConstraintType (p. 115)
- kms:GrantOperations (p. 116)
- kms:GranteePrincipal (p. 117)
- kms:RetiringPrincipal (p. 120)

kms:GrantOperations

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:GrantOperations`	String	`CreateGrant`	Key policies and IAM 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 re-encrypt 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. 115)
- kms:GrantIsForAWSResource (p. 116)
- kms:GranteePrincipal (p. 117)
- kms:RetiringPrincipal (p. 120)

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. 115)
- kms:GrantIsForAWSResource (p. 116)
- kms:GrantOperations (p. 116)
- kms:RetiringPrincipal (p. 120)

kms:KeyOrigin

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:KeyOrigin`	String	`CreateKey` CMK resource operations	IAM policies Key policies and IAM policies

The `kms:KeyOrigin` condition key controls access to operations based on the value of the `Origin` property of the CMK that is created by or used in the operation.

You can use this condition key to control access to the CreateKey operation based on the value of the Origin parameter in the request. Valid values for `Origin` are `AWS_KMS`, `AWS_CLOUDHSM`, and `EXTERNAL`.

For example, you can allow a user to create a CMK only when the key material is generated in KMS (`AWS_KMS`), only when the key material is generated in an AWS CloudHSM cluster that is associated with a custom key store (p. 184) (`AWS_CLOUDHSM`), or only when the key material is imported (p. 159) from an external source (`EXTERNAL`).

The following example policy statement uses the `kms:KeyOrigin` condition key to allow a user to create a CMK only when AWS KMS creates the key material.

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

You can also use the `kms:KeyOrigin` condition key to control access to operations that use or manage a CMK based on the `Origin` property of the CMK used for the operation. The operation must be a *CMK resource operation*, that is, an operation that is authorized for a particular CMK. To identify the CMK resource operations, in the Actions and Resources Table (p. 88), look for a value of `CMK` in the `Resources` column for the operation.

For example, the following IAM policy allows principals to perform the specified CMK resource operations, but only with CMKs in the account that were created in a custom key store.

```
{
  "Effect": "Allow",
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:GenerateDataKey",
    "kms:GenerateDataKeyWithoutPlaintext",
    "kms:GenerateDataKeyPair",
    "kms:GenerateDataKeyPairWithoutPlaintext",
    "kms:ReEncrypt*"
  ],
  "Resource": {
      "arn:aws:kms:us-west-2:111122223333:key/*"
  },
  "Condition": {
    "StringEquals": {
      "kms:KeyOrigin": "AWS_CLOUDHSM"
    }
  }
}
```

See also

- kms:BypassPolicyLockoutSafetyCheck (p. 100)
- kms:CustomerMasterKeySpec (p. 102)
- kms:CustomerMasterKeyUsage (p. 104)

kms:MessageType

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:MessageType`	String	`Sign` `Verify`	Key policies and IAM policies

The `kms:MessageType` condition key controls access to the Sign and Verify operations based on the value of the `MessageType` parameter in the request. Valid values for `MessageType` are `RAW` and `DIGEST`.

For example, the following key policy statement uses the `kms:MessageType` condition key to allow a user to use an asymmetric CMK to sign a message, but not a message digest.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": "kms:Sign",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:MessageType": "RAW"
    }
  }
```

```
}
```

See also

- the section called “kms:SigningAlgorithm” (p. 121)

kms:ReEncryptOnSameKey

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:ReEncryptOnSameKey`	Boolean	`ReEncrypt`	Key policies and IAM 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`	Key policies and IAM 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:iam::111122223333:role/LimitedAdminRole",
         "arn:aws:iam::111122223333:user/OpsAdmin"
      ]
    }
  }
}
```

See also

- kms:GrantConstraintType (p. 115)
- kms:GrantIsForAWSResource (p. 116)
- kms:GrantOperations (p. 116)
- kms:GranteePrincipal (p. 117)

kms:SigningAlgorithm

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:SigningAlgorithm`	String	`Sign` `Verify`	Key policies and IAM policies

You can use the `kms:SigningAlgorithm` condition key to control access to the Sign and Verify operations based on the value of the SigningAlgorithm parameter in the request. This condition key has no effect on operations performed outside of AWS KMS, such as verifying signatures with the public key in an asymmetric CMK pair outside of AWS KMS.

The following example key policy allows users who can assume the `testers` role to use the CMK to sign messages only when the signing algorithm used for the request is an RSASSA_PSS algorithm, such as `RSASSA_PSS_SHA512`.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:role/testers"
  },
  "Action": "kms:Sign",
  "Resource": "*",
  "Condition": {
    "StringLike": {
      "kms:SigningAlgorithm": "RSASSA_PSS*"
    }
  }
}
```

See also

- kms:EncryptionAlgorithm (p. 106)
- the section called “kms:MessageType” (p. 119)

kms:ValidTo

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:ValidTo`	Timestamp	`ImportKeyMaterial`	Key policies and IAM 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. 114) 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. 114)
- kms:WrappingAlgorithm (p. 125)
- kms:WrappingKeySpec (p. 126)

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 `except`: CreateKey, GenerateRandom, ListAliases, ListKeys, ListRetirableGrants,	Key policies and IAM policies

AWS KMS condition keys	Condition type	API operations	Policy type
		RetireGrant, and the API operations that create and manage custom key stores (p. 184).	

The `kms:ViaService` condition key limits use of an AWS KMS customer master key (p. 3) (CMK) to requests from specified AWS services. You can specify one or more services in each `kms:ViaService` condition key.

For example, the following statement from a key policy uses the `kms:ViaService` condition key to allow a customer managed CMK (p. 4) to be used for the specified actions only when the request comes from Amazon EC2 or Amazon RDS in the US West (Oregon) region on behalf of `ExampleUser`.

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

You can also use a `kms:ViaService` condition key to deny permission to use a CMK when the request comes from particular services. For example, the following policy statement from a key policy uses a `kms:ViaService` condition key to prevent a customer managed CMK from being used for `Encrypt` operations when the request comes from AWS Lambda on behalf of `ExampleUser`.

```
{
  "Effect": "Deny",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/ExampleUser"
  },
  "Action": [
    "kms:Encrypt"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": [
          "lambda.us-west-2.amazonaws.com"
      ]
```

```
        }
    }
}
```

Important
When you use the `kms:ViaService` condition key, the service makes the request on behalf of a principal in the AWS account. These principals must 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. 238).
- 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.

All AWS managed CMKs (p. 4) use a `kms:ViaService` condition key in their key policy document. This condition allows the CMK to be used only for requests that come from the service that created the CMK. To see the key policy for an AWS managed CMK, use the GetKeyPolicy operation.

The `kms:ViaService` condition key is valid in IAM and key policy statements. The services that you specify must be integrated with AWS KMS and support the `kms:ViaService` condition key.

The following table lists AWS services that are integrated with AWS KMS, support customer managed CMKs, and support the use of the `kms:ViaService` condition key in customer managed CMKs. The services in this table might not be available in all regions.

Services that support the `kms:ViaService` condition key in customer managed CMKs

Service name	KMS ViaService name
AWS Backup	backup.*AWS_region*.amazonaws.com
Amazon Connect	connect.*AWS_region*.amazonaws.com
AWS Database Migration Service (AWS DMS)	dms.*AWS_region*.amazonaws.com
AWS Directory Service	directoryservice.*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
Amazon Elasticsearch Service	es.*AWS_region*.amazonaws.com
Amazon FSx	fsx.*AWS_region*.amazonaws.com
AWS Glue	glue.*AWS_region*.amazonaws.com
Amazon Kinesis	kinesis.*AWS_region*.amazonaws.com
Amazon Kinesis Video Streams	kinesisvideo.*AWS_region*.amazonaws.com
AWS Lambda	lambda.*AWS_region*.amazonaws.com
Amazon Lex	lex.*AWS_region*.amazonaws.com
Amazon Managed Streaming for Apache Kafka	kafka.*AWS_region*.amazonaws.com
Amazon Neptune	rds.*AWS_region*.amazonaws.com

Service name	KMS ViaService name
Amazon Redshift	redshift.*AWS_region*.amazonaws.com
Amazon Relational Database Service (Amazon RDS)	rds.*AWS_region*.amazonaws.com
Amazon RDS Performance Insights	rds.*AWS_region*.amazonaws.com
AWS Secrets Manager (Secrets Manager)	secretsmanager.*AWS_region*.amazonaws.com
Amazon Simple Email Service (Amazon SES)	ses.*AWS_region*.amazonaws.com
Amazon Simple Notification Service (Amazon SNS)	sns.*AWS_region*.amazonaws.com
Amazon Simple Storage Service (Amazon S3)	s3.*AWS_region*.amazonaws.com
AWS Snowball	importexport.*AWS_region*.amazonaws.com
Amazon SQS	sqs.*AWS_region*.amazonaws.com
Amazon WorkMail	workmail.*AWS_region*.amazonaws.com
Amazon WorkSpaces	workspaces.*AWS_region*.amazonaws.com
AWS X-Ray	xray.*AWS_region*.amazonaws.com

kms:WrappingAlgorithm

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:WrappingAlgorithm`	String	`GetParametersForImport`	Key policies and IAM 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 give the example user permission to call the `GetParametersForImport` operation, but prevents them from using the `RSAES_OAEP_SHA_1` wrapping algorithm. When the `WrappingAlgorithm` in the `GetParametersForImport` request is `RSAES_OAEP_SHA_1`, the operation fails.

```
{
  "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. 114)
- kms:ValidTo (p. 122)
- kms:WrappingKeySpec (p. 126)

kms:WrappingKeySpec

AWS KMS condition keys	Condition type	API operations	Policy type
`kms:WrappingKeySpec`	String	`GetParametersForImport`	Key policies and IAM 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": {
    "StringEquals": {
      "kms:WrappingKeySpec": "RSA_2048"
    }
  }
}
```

See also

- kms:ExpirationModel (p. 114)
- kms:ValidTo (p. 122)
- kms:WrappingAlgorithm (p. 125)

Using grants

AWS KMS supports two resource-based access control mechanisms: key policies (p. 59) 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. Because grants can be very specific, and are easy to create and revoke, they are often used to provide temporary permissions or more granular permissions.

You can also use key policies to allow other AWS 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 code examples that demonstrate how to work with grants, see Working with grants (p. 351).

Create a grant

To create a grant, call the CreateGrant 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.

For example, the following `CreateGrant` command creates a grant that allows `exampleUser` to call the Decrypt operation on the specified symmetric CMK (p. 142). The grant uses the `RetiringPrincipal` parameter to designate a principal that can retire the grant. It also includes a grant constraint that allows the permission only when the encryption context (p. 19) in the request includes `"Department": "IT"`.

```
$ aws kms create-grant \
    --key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
    --grantee-principal arn:aws:iam::111122223333:user/exampleUser \
    --operations Decrypt \
    --retiring-principal arn:aws:iam::111122223333:role/adminRole \
    --constraints EncryptionContextSubset={Department=IT}
```

To view the grant, use the ListGrants operation.

> **Note**
> The `GranteePrincipal` field in the `ListGrants` response usually contains the grantee principal of the grant. However, when the grantee principal in the grant is an AWS service, the `GranteePrincipal` field contains the service principal, which might represent several different grantee principals.

```
$ aws kms list-grants --key-id 1234abcd-12ab-34cd-56ef-1234567890ab
{
    "Grants": [
        {
            "KeyId": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "CreationDate": 1572216195.0,
            "GrantId": "abcde1237f76e4ba7987489ac329fbfba6ad343d6f7075dbd1ef191f0120514",
            "Constraints": {
                "EncryptionContextSubset": {
                    "Department": "IT"
                }
            },
            "RetiringPrincipal": "arn:aws:iam::111122223333:role/adminRole",
            "Name": "",
            "IssuingAccount": "arn:aws:iam::111122223333:root",
            "GranteePrincipal": "arn:aws:iam::111122223333:user/exampleUser",
            "Operations": [
                "Decrypt"
            ]
        }
    ]
}
```

Grants can be revoked (deleted) by any user who has the kms:RevokeGrant permission on the CMK.

Grants can be retired by any of the following principals:

- 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 allows the RetireGrant operation.

Grants for symmetric and asymmetric CMKs

You can create a grant that controls access to a symmetric CMK or an asymmetric CMK. However, you cannot create a grant that allows a principal to perform an operation that is not supported by the CMK. If you try, AWS KMS returns a `ValidationError` exception.

Symmetric CMKs

Grants for symmetric CMKs cannot allow the Sign, Verify, or GetPublicKey operations. (There are limited exceptions to this rule for legacy operations, but you should not create a grant for an operation that AWS KMS does not support.)

Asymmetric CMKs

Grants for asymmetric CMKs cannot allow operations that generate data keys or data key pairs. They also cannot allow operations related to automatic key rotation (p. 154), imported key material (p. 159), or CMKs in custom key stores (p. 184).

Grants for CMKs with a key usage of `SIGN_VERIFY` cannot allow encryption operations. Grants for CMKs with a key usage of `ENCRYPT_DECRYPT` cannot allow the `Sign` or `Verify` operations.

Grant constraints

Grant constraints set conditions on the permissions that the grantee principal can perform. AWS KMS supports two supported constraints, both of which involve the encryption context (p. 19) in a request for a cryptographic operation.

> **Note**
> You cannot use encryption context grant constraints in a grant for an asymmetric CMK. The asymmetric encryption algorithms that AWS KMS uses do not support an encryption context.

- `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 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. 99).

- kms:GrantConstraintType (p. 115)
- kms:GrantIsForAWSResource (p. 116)
- kms:GrantOperations (p. 116)
- kms:GranteePrincipal (p. 117)
- kms:RetiringPrincipal (p. 120)

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.

Using service-linked roles for AWS KMS

AWS Key Management Service uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role that is linked directly to AWS KMS. Service-linked roles are defined by AWS KMS and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up AWS KMS easier because you don't have to manually add the necessary permissions. AWS KMS defines the permissions of its service-linked roles, and unless defined otherwise, only AWS KMS can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting the related resources. This protects your AWS KMS resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for AWS KMS custom key stores

AWS KMS uses a service-linked role named **AWSServiceRoleForKeyManagementServiceCustomKeyStores** to support custom key stores (p. 184). This service-linked role gives AWS KMS permission to view your AWS CloudHSM clusters and create the network infrastructure to support a connection between your custom key store and its AWS CloudHSM cluster. AWS KMS creates this role only when you create a custom key store (p. 184). You cannot create this service-linked role directly.

The **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role trusts `cks.kms.amazonaws.com` to assume the role. As a result, only AWS KMS can assume this service-linked role.

The permissions in the role are limited to the actions that AWS KMS performs to connect a custom key store to an AWS CloudHSM cluster. It does not give AWS KMS any additional permissions. For example, AWS KMS does not have permission to create, manage, or delete your AWS CloudHSM clusters, HSMs, or backups.

For more information about the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** role, including a list of permissions and instructions for how to view the role, edit the role description, delete the role, and have AWS KMS recreate it for you, see Authorizing AWS KMS to manage AWS CloudHSM and Amazon EC2 resources (p. 189).

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. 126) 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

- Examining the key policy (p. 130)
- Examining IAM policies (p. 133)
- Examining grants (p. 134)
- Troubleshooting key access (p. 135)

Examining the key policy

Key policies (p. 59) are the primary way to control access to AWS KMS customer master keys (CMKs).

When a key policy consists of or includes the default key policy (p. 61), the key policy allows IAM administrators in the account to use IAM policies to control access to the CMK. Also, if the key policy gives another AWS account (p. 82) permission to use the CMK, the IAM administrators in the external account can use IAM policies to delegate those permissions. To determine the complete list of principals that can access the CMK, examine the IAM policies (p. 133).

To view the key policy of an AWS KMS customer managed CMK (p. 4) or AWS managed CMK (p. 4) in your account, use the AWS Management Console or the GetKeyPolicy operation in the AWS KMS API. To view the key policy, you must have `kms:GetKeyPolicy` permissions for the CMK. For instructions for viewing the key policy for a CMK, see the section called "Viewing a key policy" (p. 71).

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.

> **Note**
> Do not set the Principal to an asterisk (*) in any key policy statement that allows permissions. An asterisk gives every identity in every AWS account permission to use the CMK, unless another policy statement explicitly denies it. Users in other AWS accounts just need corresponding IAM permissions in their own accounts to use the CMK.

The following examples use the policy statements found in the default key policy (p. 61) 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. 61). 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. 133) 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. 238), specifically the services that use grants (p. 126). 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. 59).

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 Troubleshooting key access (p. 135).

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. 133)
- Examining IAM policies with the IAM API (p. 133)

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` request, so you must call this operation 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` 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. 126).

To get a list of grants for a CMK, use the AWS KMS ListGrants operation. 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
```

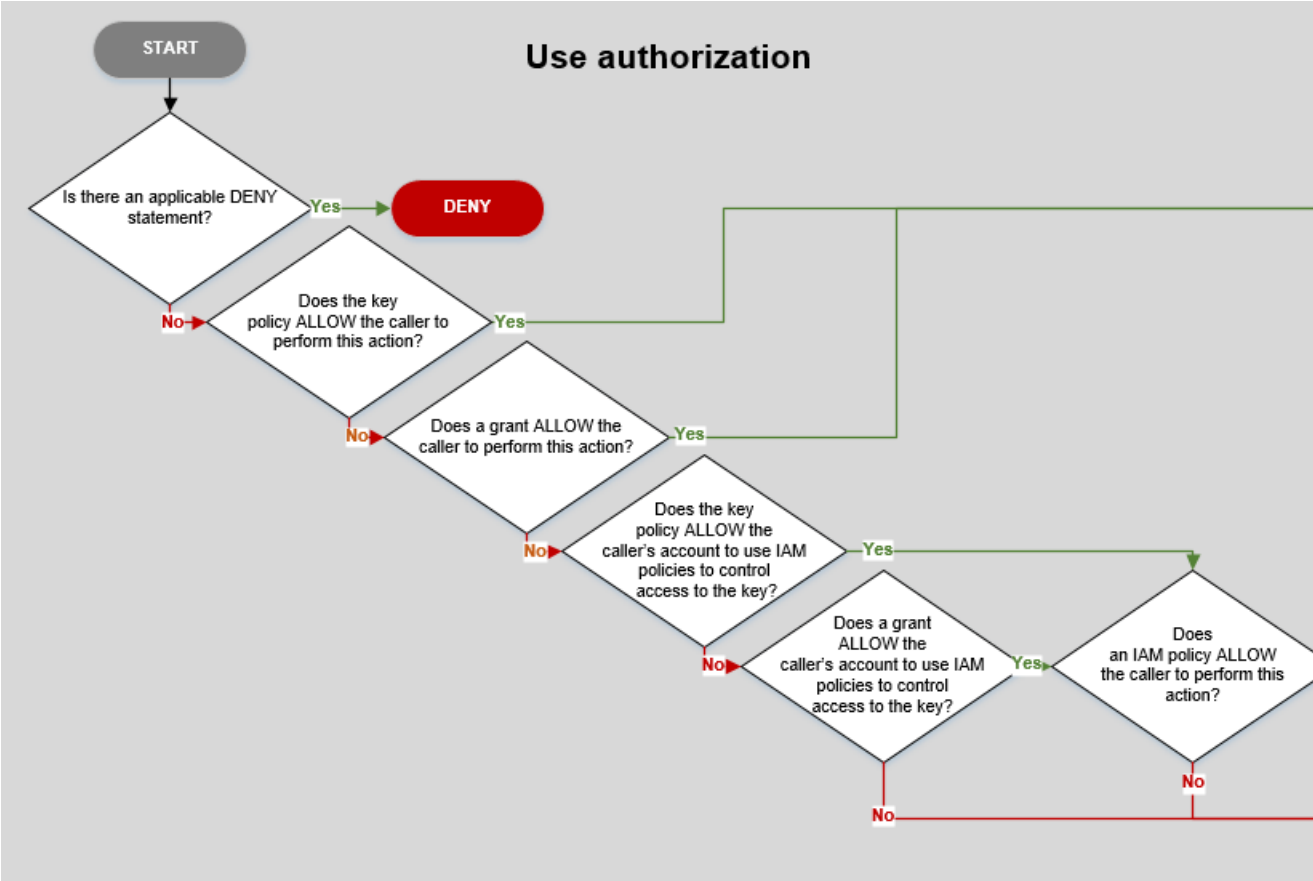
```
}]}
```

Troubleshooting key access

When authorizing access to a customer master key (CMK), AWS KMS evaluates the following:

- The key policy that is attached to the CMK. The key policy is always defined in the AWS account that owns the CMK.
- All IAM policies that are attached to the IAM user or role making the request. IAM policies that govern a principal's use of a CMK are always defined in the principal's AWS account.
- All grants that apply to the CMK.

AWS KMS evaluates the CMK's key policy (p. 130), IAM policies (p. 133), and grants (p. 134) together to determine whether access to the CMK is allowed or denied. To do this, AWS KMS uses a process similar to the one depicted in the following flowchart. The following flowchart provides a visual representation of the policy evaluation process.



This flowchart is divided into two parts. The parts appear to be sequential, but they are typically evaluated at the same time.

- *Use authorization* determines whether you are permitted to use a CMK based on its key policy, IAM policies, and grants.
- *Key trust* determines whether you should trust a CMK that you are permitted to use. In general, you trust the resources in your AWS account. But, you can also feel confident about using CMKs in a different AWS account if a grant or IAM policy in your account allows you to use the CMK.

You can use this flowchart to discover why a caller was allowed or denied permission to use a CMK. You can also use it to evaluate your policies and grants. For example, the flowchart shows that a caller can be denied access by an explicit `DENY` statement, or by the absence of an explicit `ALLOW` statement, in the key policy, IAM policy, or grant.

The flowchart can explain some common permission scenarios.

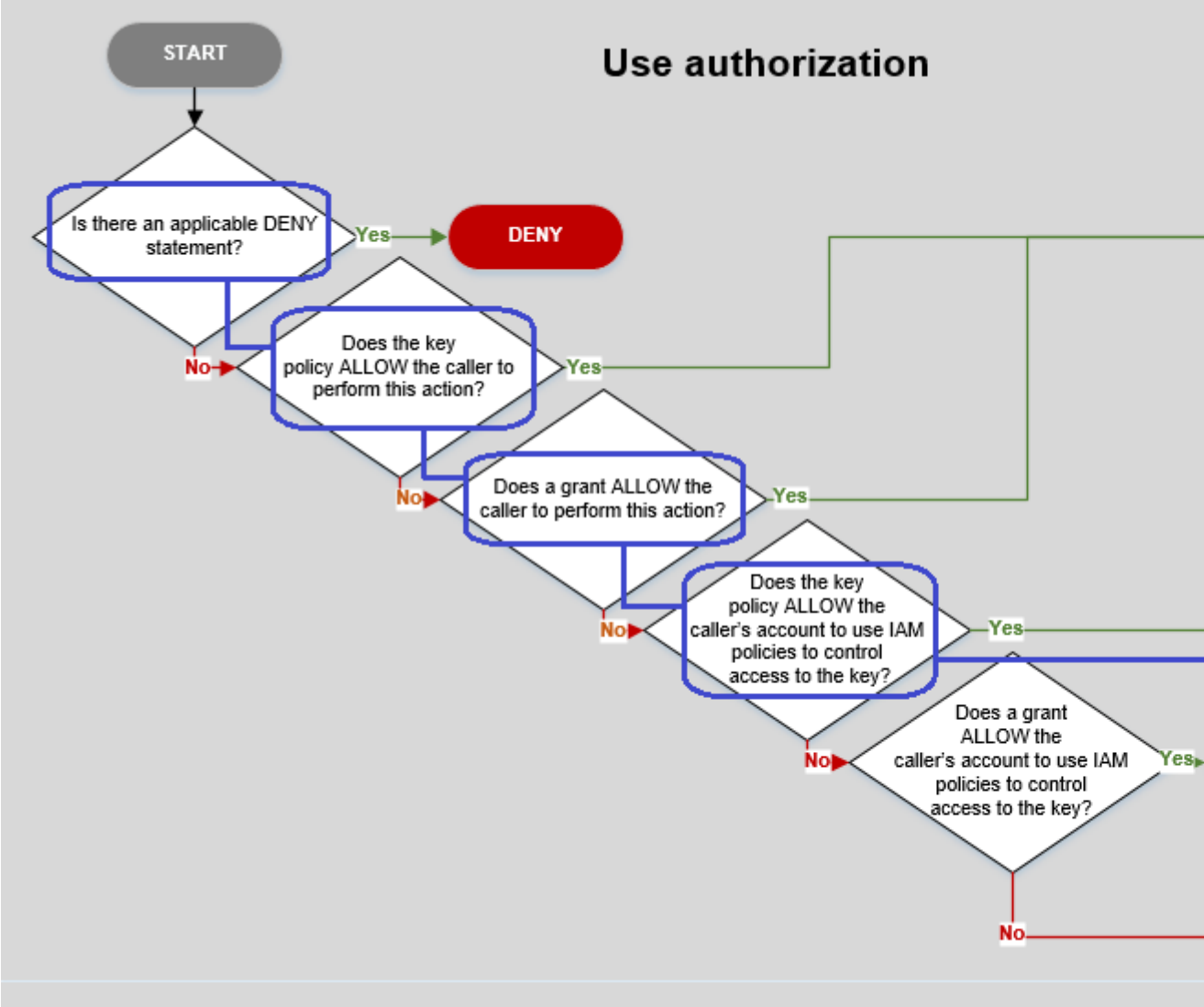
Permission Examples

- Example 1: User is denied access to a CMK in their AWS account (p. 136)
- Example 2: User assumes role with permission to use a CMK in a different AWS account (p. 137)

Example 1: User is denied access to a CMK in their AWS account

Alice is an IAM user in the 111122223333 AWS account. She was denied access to a CMK in same AWS account. Why can't Alice use the CMK?

In this case, Alice is denied access to the CMK because there is no key policy, IAM policy, or grant that gives her the required permissions. The CMK's key policy allows the AWS account to use IAM policies to control access to the CMK, but no IAM policy gives Alice permission to use the CMK.



Consider the relevant policies for this example.

- The CMK that Alice wants to use has the default key policy (p. 61). This policy allows the AWS account (p. 61) that owns the CMK to use IAM policies to control access to the CMK. This key policy satisfies the *Does the key policy ALLOW the callers account to use IAM policies to control access to the key?* condition in the flowchart.

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

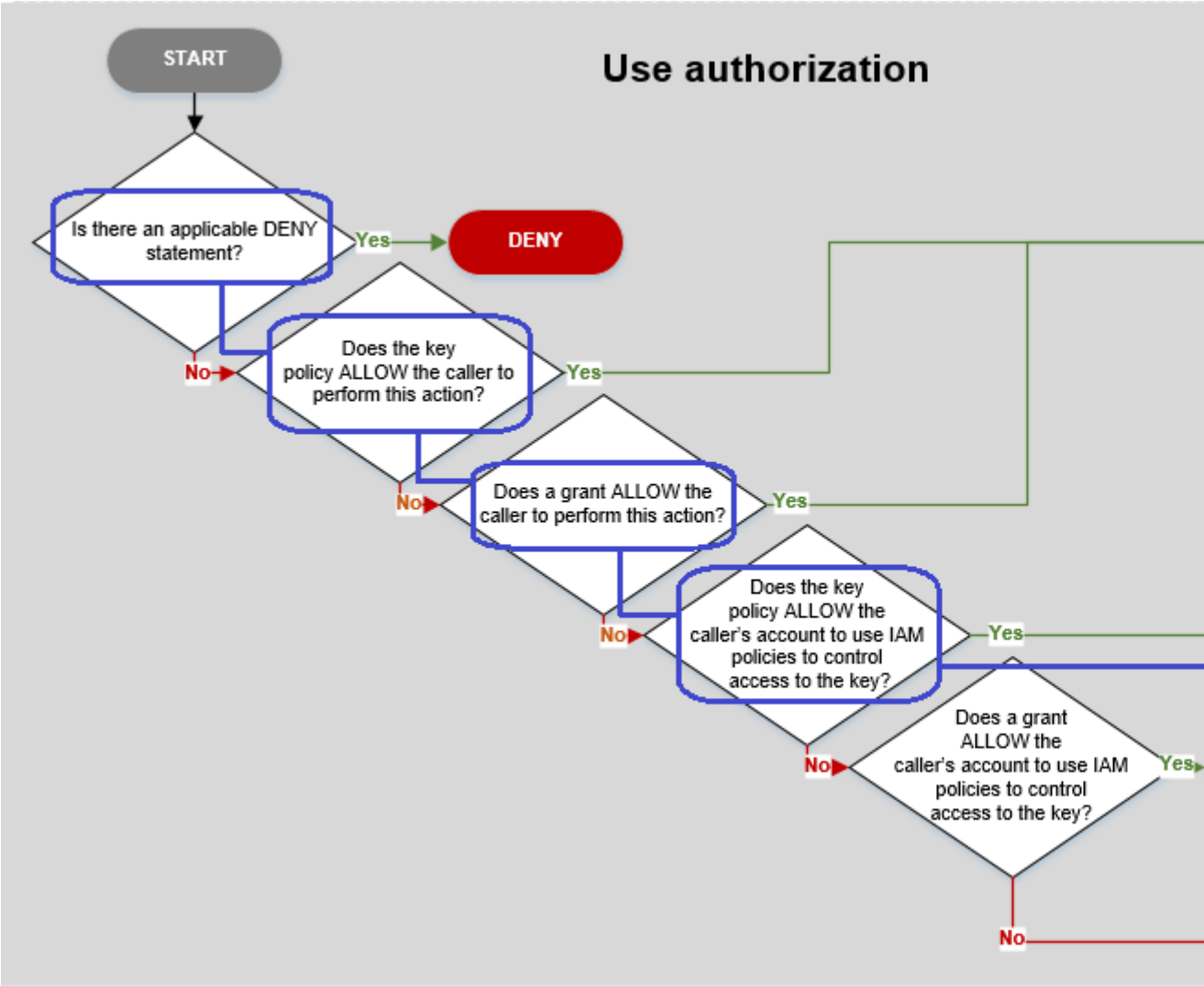
- However, no key policy, IAM policy, or grant gives Alice permission to use the CMK. Therefore, Alice is denied permission to use the CMK.

Example 2: User assumes role with permission to use a CMK in a different AWS account

Bob is a user in account 1 (111122223333). He is allowed to use a CMK in account 2 (444455556666) in cryptographic operations (p. 10). How is this possible?

> **Tip**
> When evaluating cross-account permissions, remember that the key policy is specified in the CMK's account. The IAM policy is specified in the caller's account, even when the caller is in a different account.

- The key policy for the CMK in account 2 allows account 2 to use IAM policies to control access to the CMK.
- The key policy for the CMK in account 2 allows account 1 to use the CMK in cryptographic operations. However, account 1 must use IAM policies to give its principals access to the CMK.
- An IAM policy in account 1 allows the `ExampleRole` role to use the CMK in account 2 for cryptographic operations.
- Bob, a user in account 1, has permission to assume the `ExampleRole` role.
- Bob can trust this CMK, because even though it is not in his account, an IAM policy in his account gives him explicit permission to use this CMK.



Consider the policies that let Bob, a user in account 1, use the CMK in account 2.

- The key policy for the CMK allows account 2 (444455556666, the account that owns the CMK) to use IAM policies to control access to the CMK. This key policy also allows account 1 (111122223333) to use the CMK in cryptographic operations (specified in the `Action` element of the policy statement). However, no one in account 1 can use the CMK in account 2 until account 1 defines IAM policies that give the principals access to the CMK.

  In the flowchart, this key policy in account 2 satisfies the *Does the key policy ALLOW the caller's account to use IAM policies to control access to the key?* condition.

```
{
    "Id": "key-policy-acct-2",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Permission to use IAM policies",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::444455556666:root"
            },
```

```
            "Action": "kms:*",
            "Resource": "*"
        },
        {
            "Sid": "Allow account 1 to use this CMK",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncryptFrom",
                "kms:ReEncryptTo",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext",
                "kms:DescribeKey"
            ],
            "Resource": "*"
        }
    ]
}
```

- An IAM policy in the caller's AWS account (account 1, 111122223333) gives the `ExampleRole` role in account 1 permission to perform cryptographic operations using the CMK in account 2 (444455556666). The `Action` element gives the role the same permissions that the key policy in account 2 gave to account 1.

  Cross-account IAM policies like this one are effective only when the key policy for the CMK in account 2 gives account 1 permission to use the CMK. Also, account 1 can only give its principals permission to perform the actions that the key policy gave to the account.

  In the flowchart, this satisfies the *Does an IAM policy allow the caller to perform this action?* condition.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Principal": { "arn:aws:iam::111122223333:role/ExampleRole" }
            "Effect": "Allow",
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncryptFrom",
                "kms:ReEncryptTo",
                "kms:GenerateDataKey",
                "kms:GenerateDataKeyWithoutPlaintext",
                "kms:DescribeKey"
            ],
            "Resource": [
                "arn:aws:kms:us-
west-2:444455556666:key/1234abcd-12ab-34cd-56ef-1234567890ab"
            ]
        }
    ]
}
```

- The last required element is the definition of the `ExampleRole` role in account 1. The `AssumeRolePolicyDocument` in the role allows Bob to assume the `ExampleRole` role.

```
{
    "Role": {
        "Arn": "arn:aws:iam::111122223333:role/ExampleRole",
```

```
        "CreateDate": "2019-05-16T00:09:25Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": {
                "Principal": {
                    "AWS": "arn:aws:iam::111122223333:user/bob"
                },
                "Effect": "Allow",
                "Action": "sts:AssumeRole"
            }
        },
        "Path": "/",
        "RoleName": "ExampleRole",
        "RoleId": "AROA4KJY2TU23Y7NK62MV"
    }
}
```

Using symmetric and asymmetric keys

AWS KMS protects the customer master keys (p. 3) (CMKs) that you use to protect your data and data keys. Your secret keys are generated and used only in hardware security modules designed so that no one, including AWS employees, can access the plaintext key material.

You can create and manage the CMKs in your AWS account, including setting the key policies (p. 59), IAM policies (p. 77), and grants (p. 126) that control access to your CMKs, enabling and disabling the CMKs, creating tags and aliases, and deleting the CMKs. You can use your CMKs to protect your resources in AWS services that are integrated with AWS KMS (p. 238). And, you can audit all operations that use or manage your CMKs in AWS CloudTrail logs (p. 303).

AWS KMS supports symmetric and asymmetric CMKs.

- Symmetric CMK (p. 142): Represents a single 256-bit secret encryption key that never leaves AWS KMS unencrypted. To use your symmetric CMK, you must call AWS KMS.
- Asymmetric CMK (p. 142): Represents a mathematically related public key and private key pair that you can use for encryption and decryption or signing and verification, but not both. The private key never leaves AWS KMS unencrypted. You can use the public key within AWS KMS by calling the AWS KMS API operations, or download the public key and use it outside of AWS KMS.

> **Note**
> Asymmetric CMKs and asymmetric data key pairs are supported in all AWS Regions that AWS KMS supports except for China (Beijing) and China (Ningxia).

AWS KMS also provides symmetric data keys (p. 5) and asymmetric data key pairs (p. 6) that are designed to be used for client-side cryptography outside of AWS KMS. The symmetric data key and the private key in an asymmetric data key pair are protected by a symmetric CMK in AWS KMS.

- Symmetric data key — A symmetric encryption key that you can use to encrypt data outside of AWS KMS. This key is protected by a symmetric CMK in AWS KMS.
- Asymmetric data key pair — An RSA or elliptic curve (ECC) key pair that consists of a public key and a private key. You can use your data key pair outside of AWS KMS to encrypt and decrypt data, or sign messages and verify signatures. The private key is protected by a symmetric CMK in AWS KMS.

For information about how to create and use data keys and data key pairs, see Data keys (p. 5) and Data key pairs (p. 6). To learn how to limit the types of data key pairs that principals in your account are permitted to generate, use the kms:DataKeyPairSpec (p. 105) condition key.

This topic explains how symmetric and asymmetric CMKs work, how they differ, and how to decide which type of CMK you need to protect your data. It also explains how symmetric data keys and asymmetric data key pairs work and how to use them outside of AWS KMS.

Learn more

- For a table that compares the AWS KMS API operations that apply to each type of CMK, see the section called “Comparing symmetric and asymmetric CMKs” (p. 150).
- To find out whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

- To examine the difference in the default key policy that the AWS KMS console sets for symmetric and asymmetric CMKs, see the section called "Allows key users to use the CMK with AWS services" (p. 67).
- To specify the key specs, key usage, encryption algorithms, and signing algorithms that principals in your account can use for CMKs, see the section called "AWS KMS condition keys" (p. 99).
- To learn about the request quotas that apply to different types of CMKs, see the section called "Request quotas" (p. 372).
- To learn how to sign messages and verify signatures with asymmetric CMKs, see Digital signing with the new asymmetric keys feature of AWS KMS in the *AWS Security Blog*.

Topics

- About symmetric and asymmetric CMKs (p. 142)
- How to choose your CMK configuration (p. 143)
- Viewing the cryptographic configuration of CMKs (p. 149)
- Comparing symmetric and asymmetric CMKs (p. 150)

About symmetric and asymmetric CMKs

In AWS KMS, you can create symmetric and asymmetric CMKs.

Symmetric customer master keys

When you create a customer master key (CMK) in KMS, by default, you get a symmetric CMK.

In AWS KMS, a *symmetric CMK* represents a 256-bit encryption key that never leaves AWS KMS unencrypted. To use a symmetric CMK, you must call AWS KMS. Symmetric keys are used in symmetric encryption, where the same key is used for encryption and decryption.

Unless your task explicitly requires asymmetric encryption, symmetric CMKs, which never leave AWS KMS unencrypted, are a good choice. For information about the cryptographic configuration or *key spec* for symmetric CMKs, see SYMMETRIC_DEFAULT Key Spec (p. 146). For help creating a symmetric CMK, see Creating symmetric CMKs (p. 25).

AWS services that are integrated with AWS KMS use symmetric CMKs to encrypt your data. These services do not support encryption with asymmetric CMKs. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

You can use a symmetric CMK in AWS KMS to encrypt, decrypt, and re-encrypt data, generate data keys and data key pairs, and generate random byte strings. You can import your own key material (p. 159) into a symmetric CMK and create symmetric CMKs in custom key stores (p. 184). For a table comparing the operations that you can perform on symmetric and asymmetric CMKs, see Comparing Symmetric and Asymmetric CMKs (p. 150).

Asymmetric customer master keys

> **Note**
> Asymmetric CMKs and asymmetric data key pairs are supported in all AWS Regions that AWS KMS supports except for China (Beijing) and China (Ningxia).

You can create an asymmetric CMK in AWS KMS. An *asymmetric CMK* represents a mathematically related public key and private key pair. You can give the public key to anyone, even if they're not trusted, but the private key must be kept secret.

In an asymmetric CMK, the private key is created in AWS KMS and never leaves AWS KMS unencrypted. To use the private key, you must call AWS KMS. You can use the public key within AWS KMS by calling the AWS KMS API operations. Or, you can download the public key (p. 53) and use it outside of AWS KMS.

If your use case requires encryption outside of AWS by users who cannot call AWS KMS, asymmetric CMKs are a good choice. However, if you are creating a CMK to encrypt the data that you store or manage in an AWS service, use a symmetric CMK. AWS services that are integrated with AWS KMS use symmetric CMKs to encrypt your data. These services do not support encryption with asymmetric CMKs.

AWS KMS supports two types of asymmetric CMKs.

- **RSA CMKs**: A CMK with an RSA key pair for encryption and decryption or signing and verification (but not both). KMS supports several key lengths for different security requirements.
- **Elliptic Curve (ECC) CMKs**: A CMK with an elliptic curve key pair for signing and verification. KMS supports several commonly-used curves.

For technical details about the encryption and signing algorithms that AWS KMS supports for RSA CMKs, see RSA Key Specs (p. 146). For technical details about the signing algorithms that AWS KMS supports for ECC CMKs, see Elliptic Curve Key Specs (p. 148).

For a table comparing the operations that you can perform on symmetric and asymmetric CMKs, see Comparing Symmetric and Asymmetric CMKs (p. 150). For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

How to choose your CMK configuration

The type of CMK that you create depends largely on how your plan to use the CMK, your security requirements, and your authorization requirements. When creating your CMK, remember that the cryptographic configuration of the CMK, including its key spec and key usage are established when you create the CMK and cannot be changed. For help with creating symmetric and asymmetric CMK, see the section called “Creating keys” (p. 24).

AWS KMS supports two CMK key types: **Symmetric** and **Asymmetric**. Each key type is associated particular key usage (p. 144) and key spec (p. 145) options.

Use the following guidance to determine which type of CMK you need based on your use case.

Encrypt and decrypt data

Use a symmetric CMK (p. 142) for most use cases that require encrypting and decrypting data. The symmetric encryption algorithm that AWS KMS uses is fast, efficient, and assures the confidentiality and authenticity of data. It supports authenticated encryption with additional authenticated data (AAD), defined as an encryption context (p. 19). This type of CMK requires both the sender and recipient of encrypted data to have valid AWS credentials to call AWS KMS.

If your use case requires encryption outside of AWS by users who cannot call AWS KMS, asymmetric CMKs (p. 142) are a good choice. You can distribute the public portion of the asymmetric CMK to allow these users to encrypt data. And your applications that need to decrypt that data can use the private portion of the asymmetric CMK within AWS KMS.

Sign messages and verify signatures

To sign messages and verify signatures, you must use an asymmetric CMK (p. 142). You can use a CMK with a key spec (p. 145) that represents an RSA key pair or an elliptic curve (ECC) key pair. The key spec you choose is determined by the signing algorithm that you want to use. In some cases, the users who will verify signatures are outside of AWS and can’t call the Verify operation. In that case,

choose a key spec (p. 145) associated with a signing algorithm that these users can support in their local applications.

Perform public key encryption

To perform public key encryption, you must use an asymmetric CMK (p. 142) with an RSA key spec (p. 147). Elliptic curve (ECC) key specs (p. 148) cannot be used for public key encryption. To encrypt data in AWS KMS with the public key of an RSA CMK, use the Encrypt operation. You can also download the public key (p. 53) and share it with the parties that need to encrypt data outside of AWS KMS.

When you download the public key of an asymmetric CMK, you can use it outside of AWS KMS. But it is no longer subject to the security controls that protect the CMK in AWS KMS. For example, you cannot use KMS key policies or grants to control use of the public key. Nor can you control whether the key is used only for encryption and decryption using the RSA encryption algorithms that AWS KMS supports. For more details, see Special Considerations for Downloading Public Keys (p. 53).

To decrypt data that was encrypted with the public key outside of AWS KMS, call the Decrypt operation. The `Decrypt` operation fails if the data was encrypted under a public key from a CMK with a key usage (p. 144) of `SIGN_VERIFY`. It will also fail if it was encrypted by using an algorithm that KMS does not support for RSA CMKs.

To avoid these errors, anyone using a public key outside of AWS KMS must store the key configuration. The AWS KMS console and the GetPublicKey response provide the information that you must include when you share the public key.

Use with integrated AWS services

To create a CMK for use with an AWS service that is integrated with AWS KMS (p. 238), consult the documentation for the service. All AWS services that encrypt data on your behalf require a symmetric CMK (p. 142).

In addition to these considerations, CMKs with different key specs have different prices and different request quotas. For information about AWS KMS pricing, see AWS Key Management Service Pricing. For information about request quotas, see Request quotas (p. 372).

Selecting the key usage

The key usage (p. 18) of a CMK determines whether the CMK is used for encryption and decryption -or- signing and verification. You cannot choose both. Using a CMK for more than one type of operations makes the product of both operations more vulnerable to attack.

As shown in the following table, symmetric CMKs can be used only for encryption and decryption. Elliptic curve (ECC) CMKs can be used only for signing and verification. Key usage decisions are really made only for RSA CMKs.

Valid key usage for CMK types

CMK type	Encrypt and decrypt	Sign and verify
Symmetric CMKs	✓	⊗
Asymmetric CMKs with RSA key pairs	✓	✓
Asymmetric CMKs with ECC key pairs	⊗	✓

In the AWS KMS console, you first choose the key type (symmetric or asymmetric), and then, for asymmetric CMKs, the key usage. If you select a symmetric key type, the key usage options do not appear, because symmetric CMKs only support encryption and decryption. The key usage that you choose determines which key specs (p. 145) are displayed.

To choose a key usage in the AWS KMS console:

- For CMKs with elliptic curve (ECC) key material, choose **Sign and verify**.
- For CMKs with RSA key material, choose **Encrypt and decrypt** or **Sign and verify**.

To determine the key usage that principals in your account are permitted to use for CMKs, use the kms:CustomerMasterKeyUsage (p. 104) condition key.

Selecting the key spec

When you create an asymmetric CMK, you select its key spec (p. 17). The *key spec*, which is a property of every customer master key (CMK), represents the cryptographic configuration of your CMK. You choose the key spec when you create the CMK, and you cannot change it. If you've selected the wrong key spec, delete the CMK (p. 172), and create a new one.

> **Note**
> In AWS KMS API operations, the key spec for CMKs is known as the `CustomerMasterKeySpec`. This distinguishes it from the key spec for data keys (`KeySpec`) and data key pairs (`KeyPairSpec`), and the key spec used when wrapping key material for import (`WrappingKeySpec`). Each key spec type has different values.

The key spec determines whether the CMK is symmetric or asymmetric, the type of key material in the CMK, and the encryption algorithms or signing algorithms that AWS KMS supports for the CMK. The key spec that you choose is typically determined by your use case and regulatory requirements.

To determine the key specs that principals in your account are permitted to use for CMKs, use the kms:CustomerMasterKeySpec (p. 102) condition key.

AWS KMS supports the following key specs for CMKs:

- Symmetric CMKs (p. 146) (default; encryption and decryption)
  - SYMMETRIC_DEFAULT
- RSA key specs (p. 146) (encryption and decryption -or- signing and verification)
  - RSA_2048
  - RSA_3072
  - RSA_4096
- Elliptic curve key specs (p. 148)
  - Asymmetric NIST-recommended elliptic curve key pairs (signing and verification)
    - ECC_NIST_P256 (secp256r1)
    - ECC_NIST_P384 (secp384r1)
    - ECC_NIST_P521 (secp521r1)
  - Other asymmetric elliptic curve key pairs (signing and verification)
    - ECC_SECG_P256K1 (secp256k1), commonly used for cryptocurrency.

Topics

The following topics provide technical information about the key specs.

- SYMMETRIC_DEFAULT Key Spec (p. 146)

- RSA Key Specs (p. 146)
- Elliptic Curve Key Specs (p. 148)

SYMMETRIC_DEFAULT key spec

The default key spec, SYMMETRIC_DEFAULT, is the key spec for symmetric CMKs. When you select the **Symmetric** key type in the AWS KMS console, it selects the `SYMMETRIC_DEFAULT` key spec. In the CreateKey operation, if you don't specify a `CustomerMasterKeySpec` value, SYMMETRIC_DEFAULT is selected. If you don't have a reason to use a different key spec, SYMMETRIC_DEFAULT is a good choice.

The encryption algorithm for symmetric CMKs is also known as SYMMETRIC_DEFAULT. Currently, this represents a symmetric algorithm based on Advanced Encryption Standard (AES) in Galois Counter Mode (GCM) with 256-bit keys, an industry standard for secure encryption. The ciphertext that this algorithm generates supports additional authenticated data (AAD), such as an encryption context (p. 19), and GCM provides an additional integrity check on the ciphertext. For technical details, see the AWS Key Management Service Cryptographic Details whitepaper.

Data encrypted under AES-256-GCM is protected now and in the future. Cryptographers consider this algorithm to be *quantum resistant*. Theoretical future, large-scale quantum computing attacks on ciphertexts created under 256-bit AES-GCM keys reduce the effective security of the key to 128 bits. But, this security level is sufficient to make brute force attacks on AWS KMS ciphertexts infeasible.

You can use a symmetric CMK in AWS KMS to encrypt, decrypt, and re-encrypt data, and generate data keys and data key pairs. AWS services that are integrated with AWS KMS use symmetric CMKs to encrypt your data at rest. You can import your own key material (p. 159) into a symmetric CMK and create symmetric CMKs in custom key stores (p. 184). For a table comparing the operations that you can perform on symmetric and asymmetric CMKs, see Comparing Symmetric and Asymmetric CMKs (p. 150).

RSA key specs

When you use an RSA key spec, AWS KMS creates an asymmetric CMK with an RSA key pair. The private key never leaves AWS KMS unencrypted. You can use the public key within AWS KMS, or download the public key for use outside of AWS KMS.

> **Warning**
> When you encrypt data outside of AWS KMS, be sure that you can decrypt your ciphertext. If you use the public key from a CMK that has been deleted from AWS KMS, the public key from a CMK configured for signing and verification, or an encryption algorithm that is not supported by the CMK, the data is unrecoverable.

In AWS KMS, you can use asymmetric CMKs with RSA key pairs for encryption and decryption, or signing and verification, but not both. This property, known as *key usage* (p. 144), is determined separately from the key spec, but you should make that decision before you select a key spec.

AWS KMS supports the following RSA key specs for encryption and decryption or signing and verification:

- RSA_2048
- RSA_3072
- RSA_4096

RSA key specs differ by the length of the RSA key in bits. The RSA key spec that you choose might be determined by your security standards or the requirements of your task. In general, use the largest key that is practical and affordable for your task. CMKs with different RSA key specs are priced differently and are subject to different request quotas. For information about AWS KMS pricing, see AWS Key Management Service Pricing. For information about request quotas, see Request quotas (p. 372).

RSA key specs for encryption and decryption

When an RSA asymmetric CMK is used for encryption and decryption, you encrypt with the public key and decrypt with the private key. When you call the `Encrypt` operation in AWS KMS for an RSA CMK, AWS KMS uses the public key in the RSA key pair and the encryption algorithm you specify to encrypt your data. To decrypt the ciphertext, call the `Decrypt` operation and specify the same CMK and encryption algorithm. AWS KMS then uses the private key in the RSA key pair to decrypt your data.

You can also download the public key and use it to encrypt data outside of AWS KMS. Be sure to use an encryption algorithm that AWS KMS supports for RSA CMKs. To decrypt the ciphertext, call the `Decrypt` function with the same CMK and encryption algorithm.

AWS KMS supports two encryption algorithms for CMKs with RSA key specs. These algorithms, which are defined in PKCS #1 v2.2, differ in the hash function they use internally. In AWS KMS, the RSAES_OAEP algorithms always use the same hash function for both hashing purposes and for the mask generation function (MGF1). You are required to specify an encryption algorithm when you call the Encrypt and Decrypt operations. You can choose a different algorithm for each request.

Supported encryption algorithms for RSA key specs

Encryption algorithm	Algorithm description
RSAES_OAEP_SHA_1	PKCS #1 v2.2, Section 7.1. RSA encryption with OAEP Padding using SHA-1 for both the hash and in the MGF1 mask generation function along with an empty label.
RSAES_OAEP_SHA_256	PKCS #1, Section 7.1. RSA encryption with OAEP Padding using SHA-256 for both the hash and in the MGF1 mask generation function along with an empty label.

You cannot configure a CMK to use a particular encryption algorithm. However, you can use the kms:EncryptionAlgorithm (p. 106) policy condition to specify the encryption algorithms that principals are allowed to use with the CMK.

To get the encryption algorithms for a CMK, view the cryptographic configuration (p. 33) of the CMK in the AWS KMS console or use the DescribeKey operation. AWS KMS also provides the key spec and encryption algorithms when you download your public key, either in the AWS KMS console or by using the GetPublicKey operation.

You might choose an RSA key spec based on the length of the plaintext data that you can encrypt in each request. The following table shows the maximum size, in bytes, of the plaintext that you can encrypt in a single call to the Encrypt operation. The values differ with the key spec and encryption algorithm. To compare, you can use a symmetric CMK to encrypt up to 4096 bytes at one time.

To compute the maximum plaintext length in bytes for these algorithms, use the following formula: (*`key_size_in_bits`* / 8) - (2 * *`hash_length_in_bits`*/8) - 2. For example, for RSA_2048 with SHA-256, the maximum plaintext size in bytes is (2048/8) - (2 * 256/8) -2 = 190.

Maximum plaintext size (in bytes) in an Encrypt operation

	Encryption algorithm	
RSA_2048	214	190
RSA_3072	342	318
RSA_4096	470	446

RSA key specs for signing and verification

When an RSA asymmetric CMK is used for signing and verification, you generate the signature for a message with the private key and verify the signature with the public key.

When you call the `Sign` operation in AWS KMS for an asymmetric CMK, AWS KMS uses the private key in the RSA key pair, the message, and the signing algorithm you specify, to generate a signature. To verify the signature, call the Verify operation. Specify the signature, plus the same CMK, message, and signing algorithm. AWS KMS then uses the public key in the RSA key pair to verify the signature. You can also download the public key and use it to verify the signature outside of AWS KMS.

AWS KMS supports the following signing algorithms for CMKs with RSA key spec. You are required to specify an signing algorithm when you call the Sign and Verify operations. You can choose a different algorithm for each request.

Supported signing algorithms for RSA key specs

Signing algorithm	Algorithm description
RSASSA_PKCS1_V1_5_SHA_256	PKCS #1 v2.2, Section 8.2, RSA signature with PKCS #1v1.5 Padding and SHA-256
RSASSA_PKCS1_V1_5_SHA_384	PKCS #1 v2.2, Section 8.2, RSA signature with PKCS #1v1.5 Padding and SHA-384
RSASSA_PKCS1_V1_5_SHA_512	PKCS #1 v2.2, Section 8.2, RSA signature with PKCS #1v1.5 Padding and SHA-512
RSASSA_PSS_SHA_256	PKCS #1 v2.2, Section 8.1, RSA signature with PSS padding using SHA-256 for both the message digest and the MGF1 mask generation function along with a 256-bit salt
RSASSA_PSS_SHA_384	PKCS #1 v2.2, Section 8.1, RSA signature with PSS padding using SHA-384 for both the message digest and the MGF1 mask generation function along with a 384-bit salt
RSASSA_PSS_SHA_512	PKCS #1 v2.2, Section 8.1, RSA signature with PSS padding using SHA-512 for both the message digest and the MGF1 mask generation function along with a 512-bit salt

You cannot configure a CMK to use particular signing algorithms. However, you can use the kms:SigningAlgorithm (p. 121) policy condition to specify the signing algorithms that principals are allowed to use with the CMK.

To get the signing algorithms for a CMK, view the cryptographic configuration (p. 33) of the CMK in the AWS KMS console or by using the DescribeKey operation. AWS KMS also provides the key spec and signing algorithms when you download your public key, either in the AWS KMS console or by using the GetPublicKey operation.

Elliptic curve key specs

When you use an elliptic curve (ECC) key spec, AWS KMS creates an asymmetric CMK with an ECC key pair for signing and verification. The private key that generates signature never leaves AWS KMS unencrypted. You can use the public key to verify signatures within AWS KMS, or download the public key (p. 164) for use outside of AWS KMS.

AWS KMS supports the following ECC key specs for asymmetric CMKs.

- Asymmetric NIST-recommended elliptic curve key pairs (signing and verification)
  - ECC_NIST_P256 (secp256r1)
  - ECC_NIST_P384 (secp384r1)
  - ECC_NIST_P521 (secp521r1)
- Other asymmetric elliptic curve key pairs (signing and verification)
  - ECC_SECG_P256K1 (secp256k1), commonly used for cryptocurrencies.

The ECC key spec that you choose might be determined by your security standards or the requirements of your task. In general, use the curve with the most points that is practical and affordable for your task.

If you're creating an asymmetric CMK to use with cryptocurrencies, use the ECC_SECG_P256K1 key spec. You can also use this key spec for other purposes, but it is required for Bitcoin, and other cryptocurrencies.

CMKs with different ECC key specs are priced differently and are subject to different request quotas. For information about AWS KMS pricing, see AWS Key Management Service Pricing. For information about request quotas, see Request quotas (p. 372).

The following table shows the signing algorithms that AWS KMS supports for each of the ECC key specs. You cannot configure a CMK to use particular signing algorithms. However, you can use the kms:SigningAlgorithm (p. 121) policy condition to specify the signing algorithms that principals are allowed to use with the CMK.

Supported signing algorithms for ECC key specs

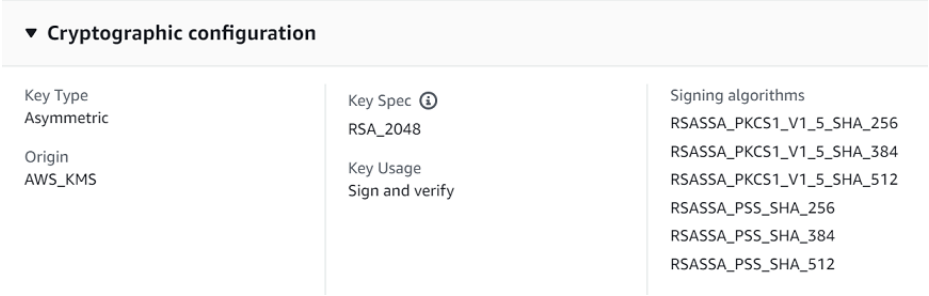
Key spec	Signing algorithm	Algorithm description
ECC_NIST_P256	ECDSA_SHA_256	NIST FIPS 186-4, Section 6.4, ECDSA signature using the curve specified by the key and SHA-256 for the message digest.
ECC_NIST_P384	ECDSA_SHA_384	NIST FIPS 186-4, Section 6.4, ECDSA signature using the curve specified by the key and SHA-384 for the message digest.
ECC_NIST_P521	ECDSA_SHA_512	NIST FIPS 186-4, Section 6.4, ECDSA signature using the curve specified by the key and SHA-512 for the message digest.
ECC_SECG_P256K1	ECDSA_SHA_256	NIST FIPS 186-4, Section 6.4, ECDSA signature using the curve specified by the key and SHA-256 for the message digest.

Viewing the cryptographic configuration of CMKs

After you create your CMK, you can view its cryptographic configuration. You cannot change the configuration of a CMK after it is created. If you prefer a different configuration, delete the CMK and create it again.

You can find the cryptographic configuration of your CMKs, include the key spec, key usage, and supported encryption or signing algorithms, in the AWS KMS console or by using the AWS KMS API. For details, see Identifying symmetric and asymmetric CMKs (p. 43).

In the AWS KMS console, the details page for each CMK includes a **Cryptographic configuration** section that displays cryptographic details about your CMKs. For example, the following image shows the **Cryptographic configuration** section for an RSA CMK used for signing and verification.



In the AWS KMS API, use the DescribeKey operation. The `KeyMetadata` structure in the response includes the cryptographic configuration of the CMK. For example, `DescribeKey` returns the following response for an RSA CMK used for signing and verification.

```
{
  "KeyMetadata": {
    "AWSAccountId": "111122223333",
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "Arn": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "CreationDate": 1571767572.317,
    "Enabled": false,
    "Description": "",
    "KeyUsage": "SIGN_VERIFY",
    "KeyState": "Disabled",
    "Origin": "AWS_KMS",
    "KeyManager": "CUSTOMER",
    "CustomerMasterKeySpec": "RSA_2048",
    "SigningAlgorithms": [
        "RSASSA_PKCS1_V1_5_SHA_256",
        "RSASSA_PKCS1_V1_5_SHA_384",
        "RSASSA_PKCS1_V1_5_SHA_512",
        "RSASSA_PSS_SHA_256",
        "RSASSA_PSS_SHA_384",
        "RSASSA_PSS_SHA_512"
    ]
  }
}
```

Comparing symmetric and asymmetric CMKs

You can create and manage symmetric and asymmetric CMKs by using the AWS KMS console and the AWS KMS API. However, AWS KMS supports different features for CMKs of different types.

For example, you can only use symmetric CMKs to generate symmetric data keys and asymmetric data key pairs. Also, importing key material (p. 159) and automatic key rotation (p. 154) are supported only for symmetric CMKs, and you can create only symmetric CMKs in a custom key store (p. 184).

The following table lists the AWS KMS operations that you can use to create and manage CMKs of each type. If you use the operation on a CMK that doesn't not support it, the operation fails.

AWS KMS operations with symmetric and asymmetric CMKs

AWS KMS API operation	Symmetric CMKs	Asymmetric CMKs (ENCRYPT_DECR	Asymmetric CMKs (SIGN_VERIFY)
CancelKeyDeletion	✓	✓	✓
CreateAlias	✓	✓	✓
CreateGrant	✓	✓	✓
CreateKey	✓	✓	✓
- With no key material (Origin = EXTERNAL)	✓	⊗	⊗
- In a custom key store (Origin = AWS_CLOUSDHSM)	✓	⊗	⊗
Decrypt	✓	✓	⊗
DeleteAlias	✓	✓	✓
DeleteImportedKeyMaterial	✓	⊗	⊗
DescribeKey	✓	✓	✓
DisableKey	✓	✓	✓
DisableKeyRotation	✓	⊗	⊗
EnableKey	✓	✓	✓
EnableKeyRotation	✓	⊗	⊗
Encrypt	✓	✓	⊗
GenerateDataKey	✓	⊗	⊗
GenerateDataKeyPair	✓ [1]	⊗	⊗

AWS KMS API operation	Symmetric CMKs	Asymmetric CMKs (ENCRYPT_DECR	Asymmetric CMKs (SIGN_VERIFY)
GenerateDataKeyPairWithoutPlaintext	✓ [1]	⊗	⊗
GenerateDataKeyWithoutPlaintext	✓	⊗	⊗
GetKeyPolicy	✓	✓	✓
GetKeyRotationStatus	✓	✓ (`KeyRotationEnabled` will always be `false`.)	✓ (`KeyRotationEnabled` will always be `false`.)
GetParametersForImport	✓	⊗	⊗
GetPublicKey	⊗	✓	✓
ImportKeyMaterial	✓	⊗	⊗
ListAliases	✓	✓	✓
ListGrants	✓	✓	✓
ListKeyPolicies	✓	✓	✓
ListResourceTags	✓	✓	✓
ListRetirableGrants	✓	✓	✓
PutKeyPolicy	✓	✓	✓
ReEncrypt	✓	✓	⊗
RetireGrant	✓	✓	✓
RevokeGrant	✓	✓	✓

AWS KMS API operation	Symmetric CMKs	Asymmetric CMKs (ENCRYPT_DECR	Asymmetric CMKs (SIGN_VERIFY)
ScheduleKeyDeletion	✓	✓	✓
Sign	⊗	⊗	✓
TagResource	✓	✓	✓
UntagResource	✓	✓	✓
UpdateAlias The current CMK and the new CMK must be the same type (both symmetric or both asymmetric) and they must have the same key usage (`ENCRYPT_DECRYPT` or `SIGN_VERIFY`).	✓	✓	✓
UpdateKeyDescription	✓	✓	✓
Verify	⊗	⊗	✓

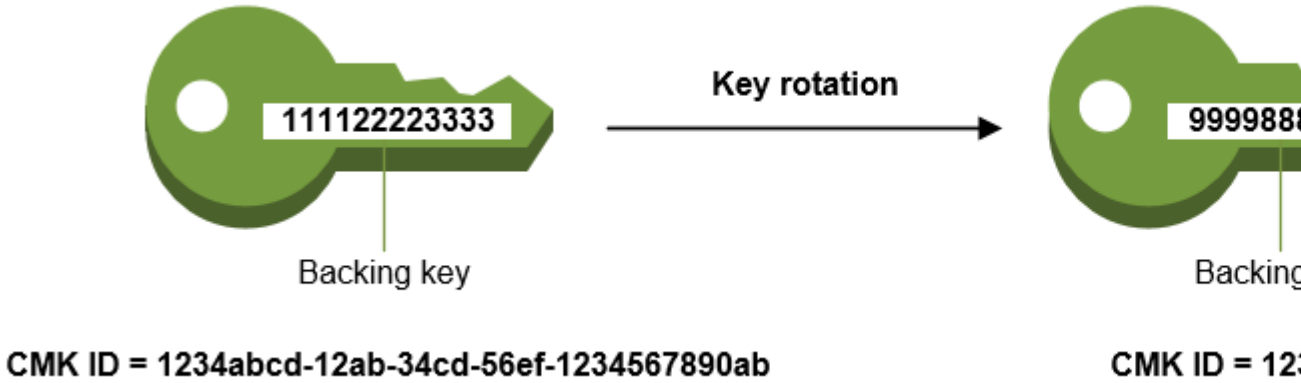
[1] `GenerateDataKeyPair` and `GenerateDataKeyPairWithoutPlaintext` generate an asymmetric data key pair that is protected by a symmetric CMK.

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 customer managed CMK (p. 4).

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 in perpetuity so it can be used to decrypt data that it encrypted. AWS KMS does not delete any rotated key material until you delete the CMK (p. 172).

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. 157). Manual rotation is a good choice when you want to control the key rotation schedule. It also provides a way to rotate CMKs that are not eligible for automatic key rotation, including asymmetric CMKs (p. 141), CMKs in custom key stores (p. 184), and CMKs with imported key material (p. 154).

Key rotation and pricing

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. 155)
- How to enable and disable automatic key rotation (p. 155)
- Rotating keys manually (p. 157)

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. AWS KMS supports optional automatic key rotation only for customer managed CMKs (p. 4).

- **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 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.
- **AWS managed CMKs.** You cannot manage key rotation for AWS managed CMKs (p. 4). AWS KMS automatically rotates AWS managed CMKs every three years (1095 days).
- **AWS owned CMKs.** You cannot manage key rotation for AWS owned CMKs. The key rotation (p. 154) strategy for an AWS owned CMK is determined by the AWS service that creates and manages the CMK. For details, see the *Encryption at Rest* topic in the user guide or developer guide for the service.
- **AWS services**. You can enable automatic key rotation on the customer managed CMKs (p. 4) that you use for server-side encryption in AWS services. The annual rotation is transparent and compatible with AWS services.
- **Monitoring key rotation.** When AWS KMS automatically rotates the key material for an AWS managed CMK (p. 4) or customer managed CMK (p. 4), it writes the **KMS CMK Rotation** event to Amazon CloudWatch Events. You can use this event to verify that the CMK was rotated.
- **Unsupported CMK types.** Automatic key rotation is *not* supported on the following types of CMKs, but you can rotate these CMKs manually (p. 157).
  - Asymmetric CMKs (p. 142)
  - CMKs in custom key stores (p. 184)
  - CMKs that have imported key material (p. 159)

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. 156)
- Enabling and disabling key rotation (AWS KMS API) (p. 156)

Enabling and disabling key rotation (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. Choose the **Key rotation** tab.

   The **Key rotation** tab only appears on the detail page of symmetric CMKs with key material that AWS KMS generated (the **Origin** is **AWS_KMS**). You cannot automatically rotate asymmetric CMKs, CMKs with imported key material (p. 159), or CMKs in custom key stores (p. 184). However, you can rotate them manually (p. 157).
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. 155) and Key state: Effect on your CMK (p. 233).
7. Choose **Save**.

Enabling and disabling key rotation (AWS 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.

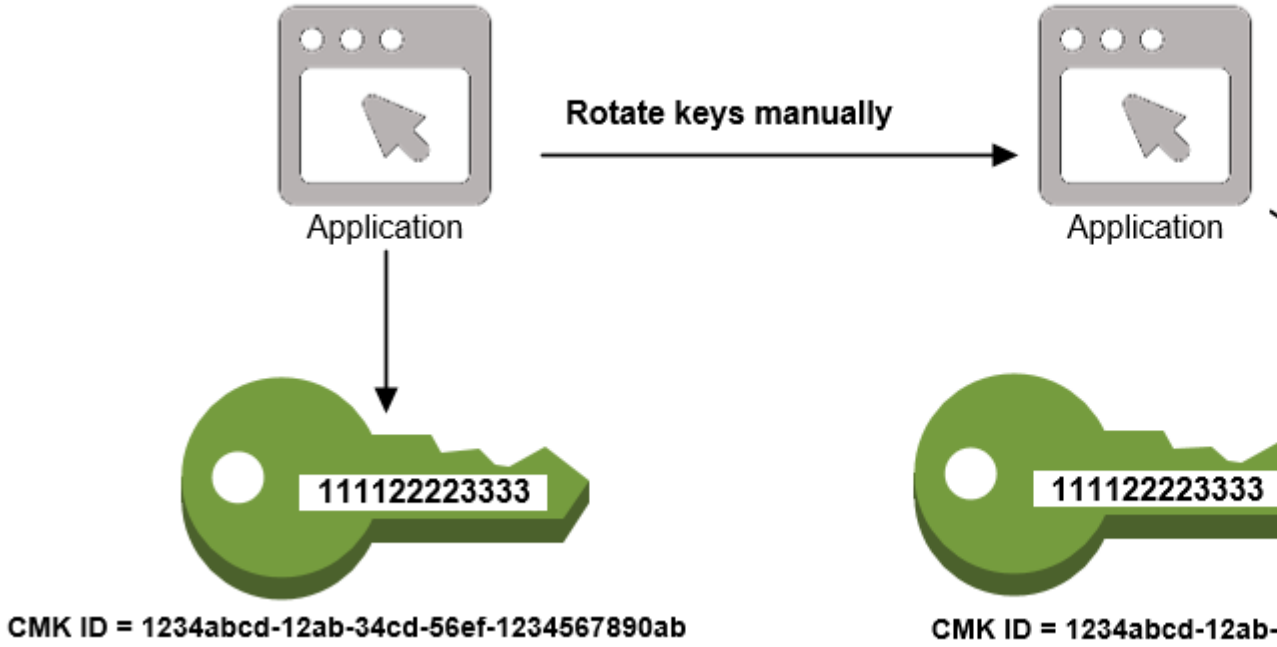
The EnableKeyRotation operation enables automatic key rotation for the specified CMK. The DisableKeyRotation operation disables it. To identify the CMK in these operations, use its key ID (p. 12) or key ARN (p. 12). By default, key rotation is disabled for customer managed CMKs.

The following example enables key rotation on the specified symmetric 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 asymmetric CMKs, CMKs in custom key stores (p. 184) and CMKs with imported key material (p. 159).

> **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 0987dcba-09fe-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)

An AWS KMS customer master key (p. 3)(CMK) is a logical representation of a master key. In addition to the CMK identifiers (p. 11) and other metadata, a CMK contains the *key material* used to encrypt and decrypt data. When you create a CMK (p. 24), 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, a feature often known as "bring your own key" (BYOK).

Imported key material is supported only for symmetric CMKs in AWS KMS key stores. It is not supported on asymmetric CMKs (p. 142) or CMKs in custom key stores (p. 184).

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. 169), but to also make it available again in the future. In contrast, scheduling key deletion (p. 172) 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. 159).

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

Topics

- About imported key material (p. 159)
- How to import key material (p. 160)
- How to reimport key material (p. 160)
- How to identify CMKs with imported key material (p. 161)

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. 160), but you cannot import different key material into that CMK. Also, you cannot enable automatic key rotation (p. 154) for a CMK with imported key material. However, you can manually rotate a CMK (p. 157) 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. 169), AWS KMS deletes the key material but does not delete the CMK or its metadata. In contrast, scheduling key deletion (p. 172) 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 symmetric CMK with no key material (p. 162) – To get started with importing key material, first create a symmetric 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. 164) – 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. 167) – 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. 168) – 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. 160) the first time, with the following exceptions.

- Use an existing CMK, instead of creating a new CMK. You can skip Step 1 (p. 162) of the import procedure.
- If the CMK contains key material, you must delete the existing key material (p. 169) 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. 164) 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)

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 (AWS 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
{
    "KeyMetadata": {
        "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
        "Origin": "EXTERNAL",
        "ExpirationModel": "KEY_MATERIAL_EXPIRES"
```

```
        "ValidTo": 1568894400.0,
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "AWSAccountId": "111122223333",
        "CreationDate": 1568289600.0,
        "Enabled": false,
        "Description": "",
        "KeyUsage": "ENCRYPT_DECRYPT",
        "KeyState": "PendingImport",
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ]
    }
}
```

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 Key state: Effect on your CMK (p. 233).

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. 162)
- Creating a CMK with no key material (AWS KMS API) (p. 163)

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. 164).

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. Choose **Symmetric**. You cannot import key material into an asymmetric CMK.

6. Expand **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, see About imported key material (p. 159).

   Choose **Next**.
8. Type an alias and (optionally) a description for the CMK.

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

   Choose **Next**.
10. 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. 62).

    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.

    Choose **Next**.
12. In the **This account** section, select the IAM users and roles in this AWS account who can use the CMK in cryptographic operations (p. 10). For more information, see Allows key users to use the CMK (p. 64).

    Note
    IAM policies can give other IAM users and roles permission to use the CMK.
13. (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 users in other accounts to use a CMK (p. 82).

    Choose **Next**.
14. 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. 165). To continue the process later, choose **Cancel**.

Next: Step 2: Download the public key and import token (p. 164).

Creating a CMK with no key material (AWS KMS API)

To use the AWS KMS API to create a symmetric 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": 1568289600.0,
        "Arn": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "AWSAccountId": "111122223333"
        "KeyManager": "CUSTOMER",
        "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
        "EncryptionAlgorithms": [
            "SYMMETRIC_DEFAULT"
        ]
    }
}
```

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. 164).

Importing key material step 2: Download the public key and import token

After you create a symmetric customer master key (CMK) with no key material (p. 162), 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 can also download these items when you want to reimport the same key material into a CMK. You might do this to change the expiration time for the key material, or to restore expired or deleted key material.

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. 167) proof-of-concept example in Step 3 (p. 167), 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. 165)
- Downloading the public key and import token (AWS KMS API) (p. 166)

Downloading the public key and import token (console)

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

1. If you just completed the steps to create a CMK with no key material (p. 162) and you are on the **Download wrapping key and import token** page, skip to Step 8.
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 only into a symmetric CMK with an **Origin** of **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. Expand the **Cryptographic configuration** section and view its values.

   You can only import key material into CMKs with a **Key type** of **Symmetric** and an **Origin** of **EXTERNAL**. For information about creating CMKs with imported key material, see, Importing key material in AWS Key Management Service (AWS KMS) (p. 159).
7. Expand the **Key material** section, and then choose **Download wrapping key and import token**.

The **Key material** section appears only for symmetric CMKs that have an **Origin** value of **EXTERNAL**.

8. 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 (p. 164).

   If you plan to try the Encrypt Key Material with OpenSSL (p. 167) proof-of-concept example in Step 3 (p. 167), choose RSAES_OAEP_SHA_1.

9. 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**.

10. 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.

11. To continue the process, see encrypt your key material (p. 167).

Downloading the public key and import token (AWS KMS API)

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

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

> **Note**
> If you plan to try the Encrypt Key Material with OpenSSL (p. 167) proof-of-concept example in Step 3 (p. 167), 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": 1568290320.0,
    "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. 167).

Importing key material step 3: Encrypt the key material

After you download the public key and import token (p. 164), 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. 164) (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. 164). 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 (AWS KMS API) (p. 166)) 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. 168).

Importing key material step 4: Import the key material

After you encrypt your key material (p. 167), 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. 167) and the import token that you downloaded at Step 2: Download the public key and import token (p. 164). 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. 168)
- Import key material (AWS KMS API) (p. 169)

Import key material (console)

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

1. If you are on the **Download wrapping key and import token** page, skip to Step 8.
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. Expand the **Cryptographic configuration** section and view its values.

You can only import key material into CMKs with a **Key type** of **Symmetric** and an **Origin** of **EXTERNAL**. For information about creating CMKs with imported key material, see Importing key material in AWS Key Management Service (AWS KMS) (p. 159).

7. Expand the **Key material** section and then choose **Upload key material**.

   The **Key material** section appears only for CMKs with a **Key type** of **Symmetric** and an **Origin** value of **EXTERNAL**.
8. In the **Encrypted key material and import token** section, under **Wrapped key material**, choose **Choose file**. Then upload the file that contains your wrapped (encrypted) key material.
9. In the **Encrypted key material and import token** section, under **Import token**, choose **Choose file**. Upload the file that contains the import token that you downloaded (p. 165).
10. In the **Expiration option** section, 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.
11. Choose **Finish** or **Upload key material**.

Import key material (AWS KMS API)

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

This example specifies an expiration time for the key material. To import key material with no expiration, replace `KEY_MATERIAL_EXPIRES` with `KEY_MATERIAL_DOES_NOT_EXPIRE` and omit the `--valid-to` parameter.

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 2019-09-17T12:00:00-08:00
```

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. 233) 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. 172) 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. 170)
- Delete key material (console) (p. 170)
- Delete key material (AWS KMS API) (p. 171)

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. 5) 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. 3) (CMK) in AWS KMS to generate a data key (p. 5), 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.

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. 233), changes to **Pending import**.

Delete key material (AWS KMS API)

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

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

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

Deleting customer master keys

Deleting a customer master key (CMK) in AWS Key Management Service (AWS KMS) is destructive and potentially dangerous. It deletes the key material and all metadata associated with the CMK and is irreversible. After a CMK is deleted, you can no longer decrypt the data that was encrypted under that 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. 51) 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. 181).

AWS KMS never deletes your CMKs unless you explicitly schedule them for deletion and the mandatory waiting period expires.

However, 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 CMK resource quota (p. 371)

> **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. 172)
- Scheduling and canceling key deletion (p. 174)
- Adding permission to schedule and cancel key deletion (p. 176)
- Creating an Amazon CloudWatch alarm to detect usage of a customer master key that is pending deletion (p. 177)
- Determining past usage of a customer master key (p. 181)

How deleting customer master keys works

Users who are authorized delete symmetric and asymmetric customer master keys (CMKs). The procedure is the same for both types of CMKs.

Because it is destructive and potentially dangerous to delete a 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 (p. 10).
- AWS KMS does not rotate the backing keys (p. 155) 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. 177) 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.

Deleting asymmetric CMKs

Users who are authorized (p. 176) can delete symmetric or asymmetric CMKs. The procedure to schedule the deletion of these CMKs is the same for both types of keys. However, because the public key of an asymmetric CMK can be downloaded (p. 53) and used outside of AWS KMS, the operation poses significant additional risks, especially for asymmetric CMKs used for encryption (the key usage is `ENCRYPT_DECRYPT`).

- When you schedule the deletion of a CMK, the key state of CMK changes to **Pending deletion**, and the CMK cannot be used in cryptographic operations (p. 10). However, scheduling deletion has no effect on public keys outside of AWS KMS. Users who have the public key can continue to use them to encrypt messages. They do not receive any notification that the key state is changed. Unless the deletion is canceled, ciphertext created with the public key cannot be decrypted.
- Alarms, logs, and other strategies that detect attempted use of CMK that is pending deletion cannot detect use of the public key outside of AWS KMS.
- When the CMK is deleted, all AWS KMS actions involving that CMK fail. However, users who have the public key can continue to use them to encrypt messages. These ciphertexts cannot be decrypted.

If you must delete an asymmetric CMK with a key usage of `ENCRYPT_DECRYPT`, use your CloudTrail Log entries to determine whether the public key has been downloaded and shared. If it has, verify that the public key is not being used outside of AWS KMS. Then, consider disabling the CMK (p. 51) instead of deleting it.

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. 3) (CMK) in AWS KMS to generate a data key (p. 5) 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. 51) 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. 176).

Ways to schedule and cancel key deletion

- Scheduling and canceling key deletion (console) (p. 174)
- Scheduling and canceling key deletion (AWS CLI) (p. 175)
- Scheduling and canceling key deletion (AWS SDK for Java) (p. 175)

Scheduling and canceling key deletion (console)

You can schedule and cancel key deletion in the AWS Management 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. Read and consider the warning, and the information about canceling the deletion during the waiting period. If you decide to cancel the deletion, choose **Cancel**.
7. For **Waiting period (in days)**, enter a number of days between 7 and 30.
8. Select the check box next to **Confirm you want to schedule this key for deletion in** ***<number of days>*** **days.**.

9. 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. 51).

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. 51).

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. 51).

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. 176)
- Adding permission to schedule and cancel key deletion (AWS CLI) (p. 177)

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.

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**.

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 Key state: Effect on your CMK (p. 233).

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. 303). 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*.

> **Warning**
> This Amazon CloudWatch alarm cannot detect use of the public key of an asymmetric CMK outside of AWS KMS. For details about the special risks of deleting asymmetric CMKs used for public key cryptography, including creating ciphertexts that cannot be decrypted, see Deleting asymmetric CMKs (p. 173).

Topics

- Requirements for a CloudWatch alarm (p. 178)
- Create the CloudWatch alarm (p. 178)

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.

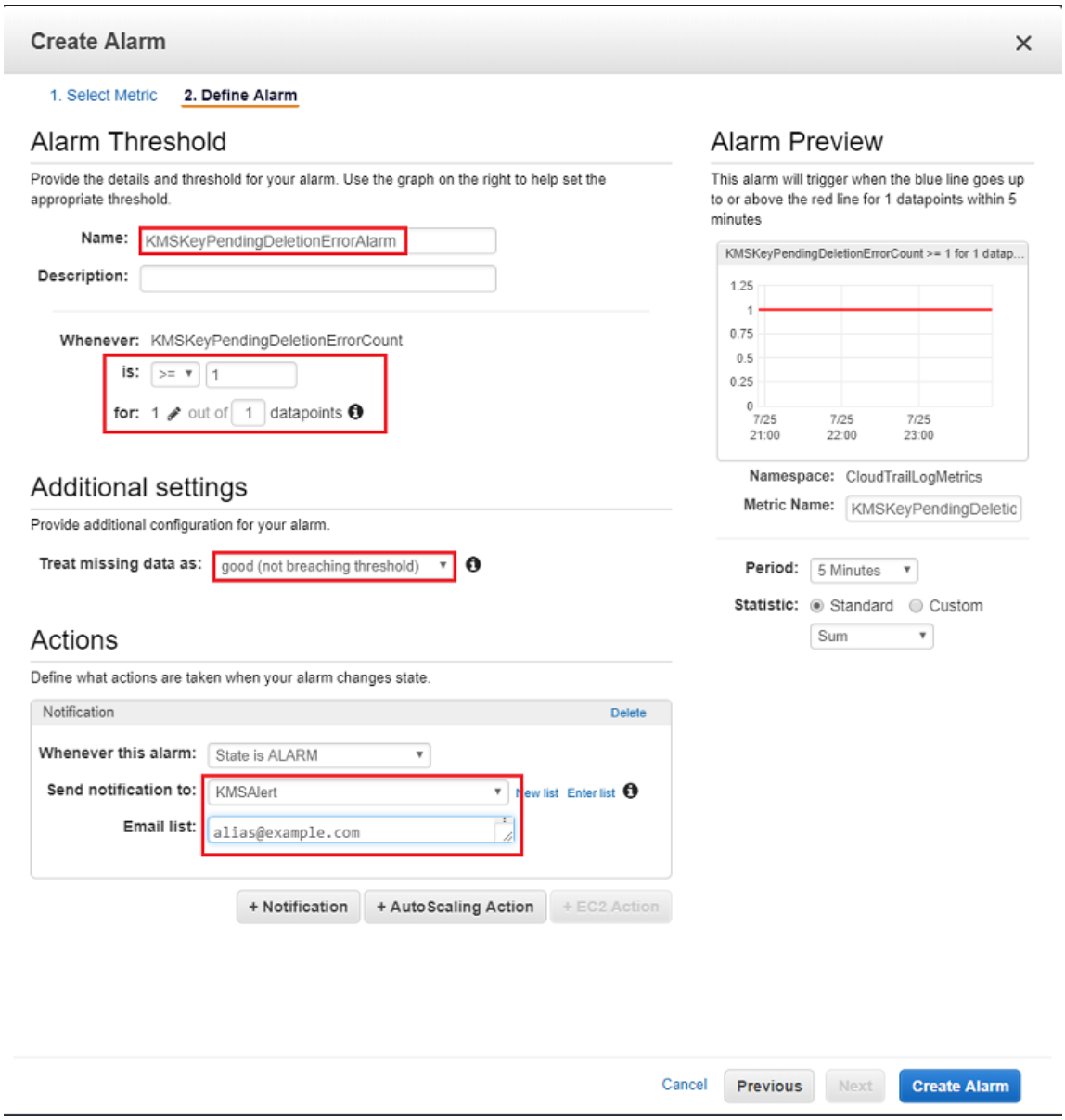
To create a CloudWatch alarm that monitors attempted usage of a KMS CMK that is pending deletion

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**.



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

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

After you complete this procedure, you will receive a notification each time this CloudWatch alarm enters the `ALARM` state. If you receive a notification for this alarm, it might mean that someone or something still needs to use this CMK. In that case, you should cancel deletion of the CMK (p. 174) 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.

> **Warning**
> These strategies for determining past and actual usage are effective only for AWS users and AWS KMS operations. They cannot detect use of the public key of an asymmetric CMK outside of AWS KMS. For details about the special risks of deleting asymmetric CMKs used for public key cryptography, including creating ciphertexts that cannot be decrypted, see Deleting asymmetric CMKs (p. 173).

Topics

- Examining CMK permissions to determine the scope of potential usage (p. 181)
- Examining AWS CloudTrail logs to determine actual usage (p. 181)

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. 130).

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. 275). 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` operation to request a unique data key for the object, and this request 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` 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. 303). For more information about CloudTrail go to the AWS CloudTrail User Guide.

Using a custom key store

AWS KMS supports custom key stores (p. 186) backed by AWS CloudHSM clusters. When you create an AWS KMS customer master key (p. 3) (CMK) in a custom key store, AWS KMS generates and stores non-extractable key material for the CMK in an AWS CloudHSM cluster that you own and manage. When you use a CMK in a custom key store, the cryptographic operations (p. 210) are performed in the HSMs in the cluster. This feature combines the convenience and widespread integration of AWS KMS with the added control of an AWS CloudHSM cluster in your AWS account.

AWS KMS provides full console and API support for creating, using, and managing your custom key stores. When you create CMKs in a custom key store, you can use them just as you would any CMK. For example, you can use the CMKs to generate data keys and encrypt data. You can also use the CMKs in your custom key store with AWS services that support customer managed CMKs.

Do I need a custom key store?

For most users, the default AWS KMS key store, which is protected by FIPS 140-2 validated cryptographic modules, fulfills their security requirements. There is no need to add an extra layer of maintenance responsibility or a dependency on an additional service.

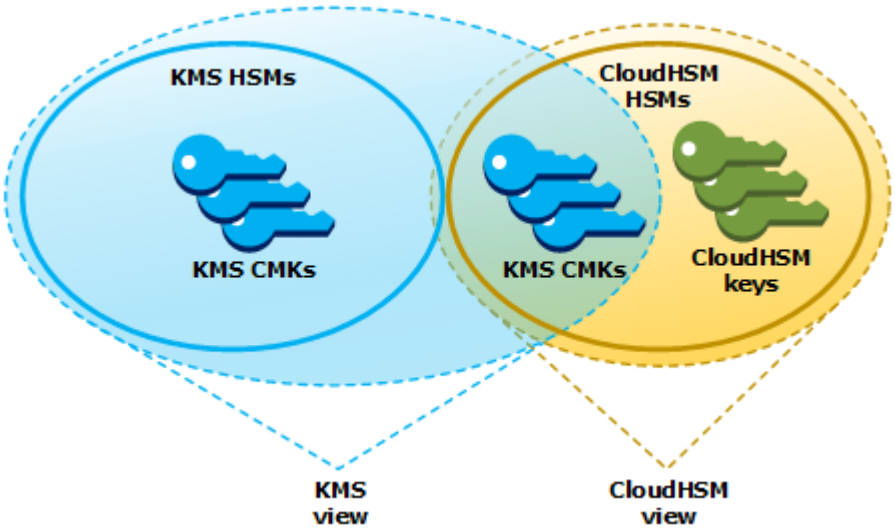
However, you might consider creating a custom key store if your organization has any of the following requirements:

- Key material cannot be stored in a shared environment.
- Key material must be backed up in multiple AWS Regions.
- Key material must be subject to a secondary, independent audit path.
- The HSMs that generate and store key material must be certified at FIPS 140-2 Level 3.

How do custom key stores work?

Each custom key store is associated with an AWS CloudHSM cluster in your AWS account. When you connect the custom key store to its cluster, AWS KMS creates the network infrastructure to support the connection. Then it logs into the key AWS CloudHSM client in the cluster using the credentials of a dedicated crypto user (p. 187) in the cluster.

You create and manage your custom key stores in AWS KMS and create and manage your HSM clusters in AWS CloudHSM. When you create customer master keys (CMKs) in an AWS KMS custom key store, you view and manage the CMKs in AWS KMS. But you can also view and manage their key material in AWS CloudHSM, just as you would do for other keys in the cluster.



You can create symmetric CMKs (p. 204) with key material generated by AWS KMS in your custom key store. Then use the same techniques to view and manage the CMKs in your custom key store that you use for CMKs in the AWS KMS key store. You can control access with IAM and key policies, create tags and aliases, enable and disable the CMKs, and schedule key deletion. You can use the CMKs for cryptographic operations (p. 210) and use them with AWS services that integrate with AWS KMS.

In addition, you have full control over the AWS CloudHSM cluster, including creating and deleting HSMs and managing backups. You can use the AWS CloudHSM client and supported software libraries to view, audit, and manage the key material for your CMKs. While the custom key store is disconnected, AWS KMS cannot access it, and users cannot use the CMKs in the custom key store for cryptographic operations. This added layer of control makes custom key stores a powerful solution for organizations that require it.

Where do I start?

To create and manage a custom key store, you use features of AWS KMS and AWS CloudHSM.

1. Start in AWS CloudHSM. Create an active AWS CloudHSM cluster or select an existing cluster. The cluster must have at least two active HSMs in different Availability Zones. Then create a dedicated crypto user (CU) account (p. 187) in that cluster for AWS KMS.
2. In AWS KMS, create a custom key store (p. 190) that is associated with your selected AWS CloudHSM cluster. AWS KMS provides a complete management interface (p. 194) that lets you create, view, edit, and delete your custom key stores.
3. When you're ready to use your custom key store, connect it to its associated AWS CloudHSM cluster (p. 198). AWS KMS creates the network infrastructure that it needs to support the connection. It then logs in to the cluster using the dedicated crypto user account credentials so it can generate and manage key material in the cluster.
4. Now, you can create symmetric CMKs in your custom key store (p. 204). Just specify the custom key store when you create the CMK.

If you get stuck at any point, you can find help in the Troubleshooting a custom key store (p. 215) topic. If your question is not answered, use the feedback link at the bottom of each page of this guide or post a question on the AWS Key Management Service Discussion Forum.

Quotas

There are no resource quotas for the number of custom key stores in an AWS account or Region. However, there are quotas on the number of AWS CloudHSM clusters in each AWS region, and request quotas (p. 372) on the rate of cryptographic operations using the CMKs in each custom key store.

Regions

AWS KMS supports custom key stores in all AWS Regions where both AWS KMS and AWS CloudHSM are available. For a list of AWS Regions that each service supports, see AWS Key Management Service Endpoints and Quotas and AWS CloudHSM Endpoints and Quotas in the *Amazon Web Services General Reference*.

Unsupported features

Custom key stores do not support asymmetric CMKs (p. 141), asymmetric data key pairs (p. 6), or CMKs with imported key material (p. 159), and you cannot enable automatic key rotation (p. 154) on a CMK in a custom key store.

Topics

- What is a custom key store? (p. 186)
- Controlling access to your custom key store (p. 188)
- Creating a custom key store (p. 190)
- Managing a custom key store (p. 194)
- Managing CMKs in a custom key store (p. 204)
- Troubleshooting a custom key store (p. 215)

What is a custom key store?

This topic explains some of the concepts used in AWS KMS custom key stores.

Topics

- AWS KMS custom key store (p. 186)
- AWS CloudHSM cluster (p. 187)
- kmsuser Crypto user (p. 187)
- CMKs in a custom key store (p. 188)

AWS KMS custom key store

A *key store* is a secure location for storing cryptographic keys. The default key store in AWS KMS also supports methods for generating and managing the keys that its stores. By default, the customer master keys (CMKs) that you create in AWS KMS are generated in and protected by hardware security modules (HSMs) that are FIPS 140-2 validated cryptographic modules. The CMKs never leave the modules unencrypted.

However, if you require even more control of the HSMs, you can create a custom key store that is backed by FIPS 140-2 Level 3 HSMs in an AWS CloudHSM cluster that you own and manage.

A *custom key store* is an AWS KMS resource that is associated with an AWS CloudHSM cluster. When you create an AWS KMS CMK in your custom key store, AWS KMS generates a 256-bit, persistent, non-exportable Advanced Encryption Standard (AES) symmetric key in the associated AWS CloudHSM cluster. This key material never leaves your HSMs unencrypted. When you use a CMK in a custom key store, the cryptographic operations are performed in the HSMs in the cluster.

Custom key stores combine the convenient and comprehensive key management interface of AWS KMS with the additional controls provided by an AWS CloudHSM cluster in your AWS account. This integrated feature lets you create, manage, and use CMKs in AWS KMS while maintaining full control of the HSMs

that store their key material, including managing clusters, HSMs, and backups. You can use the AWS KMS console and APIs to manage the custom key store and its CMKs. You can also use the AWS CloudHSM console, APIs, client software, and associated software libraries to manage the associated cluster.

You can view and manage (p. 194) your custom key store, edit its properties (p. 196), and connect and disconnect it (p. 198) from its associated AWS CloudHSM cluster. If you need to delete a custom key store (p. 202), you must first delete the CMKs in the custom key store by scheduling their deletion and waiting until the grace period expires. Deleting the custom key store removes the resource from AWS KMS, but it does not affect your AWS CloudHSM cluster.

AWS CloudHSM cluster

Every AWS KMS custom key store is associated with one AWS CloudHSM cluster. When you create a customer master key (CMK) in your custom key store, AWS KMS creates its key material in the associated cluster. When you use a CMK in your custom key store, the cryptographic operation is performed in the associated cluster.

Each AWS CloudHSM cluster can be associated with only one custom key store. The cluster that you choose cannot be associated with another key store or share a backup history with an associated cluster. The cluster must be initialized and active, and it must be in the same AWS account and Region as the AWS KMS custom key store. You can create a new cluster or use an existing one. AWS KMS does not need exclusive use of the cluster. To create CMKs in the custom key store, its associated cluster it must contain at least two active HSMs. All other operations require only one HSM.

You specify the cluster when you create the custom key store, and you cannot change it. However, you can substitute any cluster that shares a backup history with the original cluster. This lets you delete the cluster, if necessary, and replace it with a cluster created from one of its backups. You retain full control of the associated AWS CloudHSM cluster so you can manage users and keys, create and delete HSMs, and use and manage backups.

When you are ready to use your custom key store, you connect it to its associated AWS CloudHSM cluster. You can connect and disconnect your custom key store (p. 198) at any time. When a custom key store is connected, you can create and use its CMKs. When it is disconnected, you can view and manage the custom key store and its CMKs. But you cannot create new CMKs or use the CMKs in the custom key store for cryptographic operations.

`kmsuser` Crypto user

To create and manage key material in the associated AWS CloudHSM cluster on your behalf, AWS KMS uses a dedicated AWS CloudHSM crypto user (CU) in the cluster named `kmsuser`. The `kmsuser` CU is a standard CU account that is automatically synchronized to all HSMs in the cluster and is saved in cluster backups.

Before you create your custom key store, you create a `kmsuser` CU account (p. 191) in your AWS CloudHSM cluster using the createUser command in cloudhsm_mgmt_util. Then when you create the custom key store (p. 190), you provide the `kmsuser` account password to AWS KMS. When you connect the custom key store (p. 198), AWS KMS logs into the cluster as the `kmsuser` CU and rotates its password.

AWS KMS remains logged in as `kmsuser` as long as the custom key store is connected. You should not use this CU account for other purposes. However, you retain ultimate control of the `kmsuser` CU account. At any time, you can find the key handles (p. 213) of keys that `kmsuser` owns. If necessary, you can disconnect the custom key store (p. 198), change the `kmsuser` password, log into the cluster as `kmsuser` (p. 220), and view and manage the keys that `kmsuser` owns.

For instructions on creating your `kmsuser` CU account, see Create the `kmsuser` Crypto User (p. 191).

CMKs in a custom key store

You can use the AWS Management Console or AWS KMS API to create a customer master key (p. 3) (CMK) in a custom key store. You use the same technique that you would use on any AWS KMS CMK. The only difference is that you must identify the custom key store and specify that origin of the key material is the AWS CloudHSM cluster.

When you create a CMK in a custom key store (p. 204), AWS KMS creates the CMK in AWS KMS and it generates a 256-bit, persistent, non-exportable Advanced Encryption Standard (AES) symmetric backing key in its associated cluster. Although AWS CloudHSM supports symmetric and asymmetric keys of different types, AWS KMS and custom key stores only support AES symmetric keys.

You can view the CMKs in a custom key store in the AWS KMS console, and use the console options to display the custom key store ID. You can also use the DescribeKey operation to find the custom key store ID and AWS CloudHSM cluster ID.

The CMKs in a custom key store work just like any CMKs in AWS KMS. Authorized users need the same permissions to use and manage the CMKs. You use the same console procedures and API operations to view and manage the CMKs in a custom key store. These include enabling and disabling CMKs, creating and using tags and aliases, and setting and changing IAM and key policies. You can use the CMKs in a custom key store for cryptographic operations, and use them with integrated AWS services (p. 238) that support the use of customer managed CMKs. However, you cannot enable automatic key rotation (p. 154) or import key material (p. 159) into a CMK in a custom key store.

You also use the same process to schedule deletion (p. 214) of a CMK in a custom key store. After the waiting period expires, AWS KMS deletes the CMK from KMS. Then it makes a best effort to delete the key material for the CMK from the associated AWS CloudHSM cluster. However, you might need to manually delete the orphaned key material (p. 218) from the cluster and its backups.

Controlling access to your custom key store

You use IAM policies to control access to your AWS KMS custom key store and your AWS CloudHSM cluster. You can use IAM policies and key policies to control access to the customer master keys (CMKs) in your custom key store. We recommend that you provide users, groups, and roles only the permissions that they require for the tasks that they are likely to perform.

Topics

- Authorizing custom key store managers and users (p. 188)
- Authorizing AWS KMS to manage AWS CloudHSM and Amazon EC2 resources (p. 189)

Authorizing custom key store managers and users

When designing your custom key store, be sure that the principals who use and manage it have only the permissions that they require. The following list describes the minimum permissions required for custom key store managers and users.

- Principals who create and manage your custom key store require the following permission to use the custom key store API operations.
  - `cloudhsm:DescribeClusters`
  - `kms:CreateCustomKeyStore`
  - `kms:ConnectCustomKeyStore`
  - `kms:DisconnectCustomKeyStore`
  - `kms:UpdateCustomKeyStore`

  - `kms:DeleteCustomKeyStore`
  - `kms:DescribeCustomKeyStores`
  - `iam:CreateServiceLinkedRole`
- Principals who create and manage the AWS CloudHSM cluster that is associated with your custom key store need permission to create and initialize an AWS CloudHSM cluster. This includes permission to create or use a virtual private cloud, create subnets, and create an Amazon EC2 instance. They might also need to create and delete HSMs, and manage backups. For lists of the required permissions, see Restrict User Permissions to What's Necessary for AWS CloudHSM in the *AWS CloudHSM User Guide*.
- Principals who create and manage customer master keys (CMKs) in your custom key store require the same permissions as those who create and manage any CMK in AWS KMS. For example, those principals need an IAM policy with kms:CreateKey permission. No additional permissions are required. The default key policy (p. 61) for CMKs in a custom key store is identical to the default key policy for CMKs in AWS KMS.
- Principals who use the CMKs in your custom key store for cryptographic operations (p. 210) need permission to perform the cryptographic operation with the CMK, such as kms:Decrypt. You can provide these permissions in an IAM or key policy. But, they do not need any additional permissions to use a CMK in a custom key store.

Authorizing AWS KMS to manage AWS CloudHSM and Amazon EC2 resources

To support your custom key stores, AWS KMS needs permission to get information about your AWS CloudHSM clusters. It also needs permission to create the network infrastructure that connects your custom key store to its AWS CloudHSM cluster. To get these permissions, AWS KMS creates the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role in your AWS account. Users who create custom key stores must have the `iam:CreateServiceLinkedRole` permission that allows them to create service-linked roles.

Topics

- About the AWS KMS service-linked role (p. 189)
- Create the service-linked role (p. 190)
- Edit the service-linked role description (p. 190)
- Delete the service-linked role (p. 190)

About the AWS KMS service-linked role

A service-linked role is an IAM role that gives one AWS service permission to call other AWS services on your behalf. It's designed to make it easier for you to use the features of multiple integrated AWS services without having to create and maintain complex IAM policies.

For custom key stores, AWS KMS creates the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role with the **AWSKeyManagementServiceCustomKeyStoresServiceRolePolicy** policy. This policy grants the role the following permissions:

- cloudhsm:DescribeClusters
- ec2:AuthorizeSecurityGroupIngress
- ec2:CreateNetworkInterface
- ec2:CreateSecurityGroup
- ec2:DeleteSecurityGroup

- ec2:DescribeSecurityGroups
- ec2:RevokeSecurityGroupEgress

Because the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role trusts only `cks.kms.amazonaws.com`, only AWS KMS can assume this service-linked role. This role is limited to the operations that AWS KMS needs to view your AWS CloudHSM clusters and to connect a custom key store to its associated AWS CloudHSM cluster. It does not give AWS KMS any additional permissions. For example, AWS KMS does not have permission to create, manage, or delete your AWS CloudHSM clusters, HSMs, or backups.

Regions

Like the custom key stores feature, the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** role is supported in all AWS Regions where AWS KMS and AWS CloudHSM are available. For a list of AWS Regions that each service supports, see AWS Key Management Service Endpoints and Quotas and AWS CloudHSM Endpoints and Quotas in the *Amazon Web Services General Reference*.

For more information about how AWS services use service-linked roles, see Using Service-Linked Roles in the IAM User Guide.

Create the service-linked role

AWS KMS automatically creates the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role in your AWS account when you create a custom key store, if the role does not already exist. You cannot create or re-create this service-linked role directly.

Edit the service-linked role description

You cannot edit the role name or the policy statements in the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role, but you can edit role description. For instructions, see Editing a Service-Linked Role in the *IAM User Guide*.

Delete the service-linked role

AWS KMS does not delete the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role from your AWS account. If you have deleted all of your custom key stores (p. 202) and do not plan to create any new ones, you no longer need this service-linked role. AWS KMS does not assume this role or use its permissions unless you have active custom key stores. However, there is currently no procedure for deleting the **AWSServiceRoleForKeyManagementServiceCustomKeyStores** service-linked role.

Creating a custom key store

You can create one or several custom key stores (p. 186) in your account. Each custom key store is associated with one AWS CloudHSM cluster in the same AWS Region. Before you create your custom key store, you need to assemble the prerequisites (p. 191). Then, before you can use your custom key store, you must connect it (p. 198) to its AWS CloudHSM cluster.

> **Tip**
> You do not have to connect your custom key store immediately. You can leave it in a disconnected state until you are ready to use it. However, to verify that it is configured properly, you might want to connect it (p. 198), view its connection status (p. 194), and then disconnect it (p. 198).

Topics

- Assemble the prerequisites (p. 191)
- Create a custom key store (console) (p. 192)
- Create a custom key store (API) (p. 193)

Assemble the prerequisites

Each AWS KMS custom key store is backed by an AWS CloudHSM cluster. To create a custom key store, you must specify an active AWS CloudHSM cluster that is not already associated with another key store. You also need to create a dedicated crypto user (CU) in the cluster's HSMs that AWS KMS can use to create and manage keys on your behalf.

Before you create a custom key store, do the following:

Select an AWS CloudHSM cluster

Every custom key store is associated with exactly one AWS CloudHSM cluster (p. 187). When you create a customer master key (p. 3) (CMK) in your custom key store, AWS KMS creates the CMK metadata, such as an ID and Amazon Resource Name (ARN) in AWS KMS. It then creates the key material in the HSMs of the associated cluster. You can create a new AWS CloudHSM cluster or use an existing one. AWS KMS does not require exclusive access to the cluster.

The AWS CloudHSM cluster that you select is permanently associated with the custom key store. After you create the custom key store, you can change the cluster ID (p. 196) of the associated cluster, but the cluster that you specify must share a backup history with the original cluster. To use an unrelated cluster, you need to create a new custom key store.

The AWS CloudHSM cluster that you select must have the following characteristics:

- **The cluster must be active**.

  You must create the cluster, initialize it, install the AWS CloudHSM client software for your platform, and then activate the cluster. For detailed instructions, see the Getting Started section of the *AWS CloudHSM User Guide*.
- **The cluster must be in the same account and Region** as the AWS KMS custom key store. You cannot associate a custom key store in one region with a cluster in a different region. To create a multi-region key infrastructure, you must create key stores and clusters in each region.
- **The cluster cannot be associated with** another custom key store in the account. Each custom key store must be associated with a different AWS CloudHSM cluster. You cannot specify a cluster that is already associated with a custom key store or a cluster that shares a backup history with an associated cluster. Clusters that share a backup history have the same cluster certificate. To view the cluster certificate of a cluster, use the AWS CloudHSM console or the DescribeClusters operation.
- The cluster must be configured with private subnets in **at least two Availability Zones** in the Region. Because AWS CloudHSM is not supported in all Availability Zones, we recommend that you create private subnets in all Availability Zones in the region. You cannot reconfigure the subnets for an existing cluster, but you can create a cluster from a backup with different subnets in the cluster configuration.

  > **Important**
  > After you create your custom key store, do not delete any of the private subnets configured for its AWS CloudHSM cluster. If AWS KMS cannot find all of the subnets in the cluster configuration, attempts to connect to the custom key store (p. 198) fail with a `SUBNET_NOT_FOUND` connection error state. For details, see How to fix a connection failure (p. 216).
- The security group for the cluster (cloudhsm-cluster-*<cluster-id>*-sg) must include inbound rules and outbound rules that allow TCP traffic on ports 2223-2225. The **Source** in the inbound

rules and the **Destination** in the outbound rules must match the security group ID. These rules are set by default when you create the cluster. Do not delete or change them.

- **The cluster must contain at least two active HSMs** in different Availability Zones. To verify the number of HSMs, use the AWS CloudHSM console or the DescribeClusters operation. If necessary, you can add an HSM.

Find the trust anchor certificate

When you create a custom key store, you must upload the trust anchor certificate for the AWS CloudHSM cluster to AWS KMS. AWS KMS needs the trust anchor certificate to connect the custom key store to the cluster.

Every active AWS CloudHSM cluster has a *trust anchor certificate*. When you initialize the cluster, you generate this certificate, save it in the `customerCA.crt` file, and copy it to hosts that connect to the cluster.

Create the `kmsuser` crypto user for AWS KMS

To administer your custom key store, AWS KMS logs into the `kmsuser` crypto user (p. 187) (CU) account in the selected cluster. Before you create your custom key store, you must create the `kmsuser` CU. Then when you create your custom key store, you provide the password for `kmsuser` to AWS KMS. AWS KMS rotates the `kmsuser` password whenever you connect the custom key store to its associated AWS CloudHSM cluster.

> **Important**
> Do not specify the `2FA` option when you create the `kmsuser` CU. If you do, AWS KMS cannot log in and your custom key store cannot be connected to this AWS CloudHSM cluster. Once you specify 2FA, you cannot undo it. Instead, you must delete the CU and recreate it.

To create the `kmsuser` CU, use the following procedure.

1. Start cloudhsm_mgmt_util as described in the Prepare to run cloudhsm_mgmt_util section of the *AWS CloudHSM User Guide*.
2. Use the createUser command in cloudhsm_mgmt_util to create a CU named `kmsuser`. The password must consist of 7-32 alphanumeric characters. It is case-sensitive and cannot contain any special characters.

   For example, the following example command creates a `kmsuser` CU with a password of `kmsPswd`.

```
aws-cloudhsm> createUser CU kmsuser kmsPswd
```

Create a custom key store (console)

When you create a custom key store (p. 186) in the AWS Management Console, you can add and create the prerequisites (p. 191) as part of your workflow. However, the process is quicker when you have assembled them in advance.

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 **Custom key stores**.
4. Choose **Create key store**.
5. Enter a friendly name for the custom key store. The name must be unique in the account.
6. Select an AWS CloudHSM cluster (p. 187) for the custom key store. Or, to create a new AWS CloudHSM cluster, choose the **Create an AWS CloudHSM cluster** link.

The cluster must fulfill the requirements (p. 191) for association with a custom key store. The menu displays custom key stores in your account and region that are not already associated with a custom key store.

7. Choose **Upload file**, and then upload the trust anchor certificate for the AWS CloudHSM cluster that you chose. This is the `customerCA.crt` file that you created when you initialized the cluster.
8. Enter the password of the `kmsuser` crypto user (p. 187) (CU) that you created in the selected cluster.
9. Choose **Create**.

When the procedure is successful, the new custom key store appears in the list of custom key stores in the account and Region. If it is unsuccessful, an error message appears that describes the problem and provides help on how to fix it. If you need more help, see Troubleshooting a custom key store (p. 215).

Next: New custom key stores are not automatically connected. Before you can create customer master keys (CMKs) in the custom key store, you must connect the custom key store (p. 198) to its associated AWS CloudHSM cluster.

Create a custom key store (API)

The CreateCustomKeyStore operation creates a new custom key store (p. 186) that is associated with an AWS CloudHSM cluster in the account and Region. These examples use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

The `CreateCustomKeyStore` operation requires the following parameter values.

- CustomKeyStoreName – A friendly name for the custom key store that is unique in the account.
- CloudHsmClusterId – The cluster ID of a cluster that fulfills the requirements (p. 191) for association with a custom key store.
- KeyStorePassword – The password of `kmsuser` CU account in the specified cluster.
- TrustAnchorCertificate – The content of the `customerCA.crt` file that you created when you initialized the cluster.

The following example uses a fictitious cluster ID. Before running the command, replace it with a valid cluster ID.

```
$ aws kms create-custom-key-store
        --custom-key-store-name ExampleKeyStore \
        --cloud-hsm-cluster-id cluster-1a23b4cdefg \
        --key-store-password kmsPswd \
        --trust-anchor-certificate <certificate-goes-here>
```

If you are using the AWS CLI, you can specify the trust anchor certificate file, instead of its contents. In the following example, the `customerCA.crt` file is in the root directory.

```
$ aws kms create-custom-key-store
        --custom-key-store-name ExampleKeyStore \
        --cloud-hsm-cluster-id cluster-1a23b4cdefg \
        --key-store-password kmsPswd \
        --trust-anchor-certificate file://customerCA.crt
```

When the operation is successful, `CreateCustomKeyStore` returns the custom key store ID, as shown in the following example response.

```
{
    "CustomKeyStoreId": cks-1234567890abcdef0
}
```

If the operation fails, correct the error indicated by the exception, and try again. For additional help, see Troubleshooting a custom key store (p. 215).

Next, to use the custom key store, connect it to its AWS CloudHSM cluster (p. 198).

Managing a custom key store

Using the AWS Management Console and the AWS KMS API, you can manage a custom key store. For example, you can view a custom key store, edit its properties, connect and disconnect it from its associated AWS CloudHSM cluster, and delete the custom key store.

Topics

- Viewing a custom key store (p. 194)
- Editing custom key store settings (p. 196)
- Connecting and disconnecting a custom key store (p. 198)
- Deleting a custom key store (p. 202)

Viewing a custom key store

You can view the custom key stores in each account and region by using the AWS Management Console or the AWS KMS API.

For help with viewing the CMKs in your custom key store, see Viewing CMKs in a custom key store (p. 208).

Topics

- View a custom key store (console) (p. 194)
- View a custom key store (API) (p. 195)

View a custom key store (console)

When you view the custom key stores in the AWS Management Console, you can see the following:

- The custom key store name
- The ID of associated AWS CloudHSM cluster
- The number of HSMs in the cluster
- The current connection status

A connection status of **Disconnected** indicates that the custom key store is new and has never been connected, or it was intentionally disconnected from its AWS CloudHSM cluster (p. 198). However, if your attempts to use a CMK in a connected custom key store fail, that might indicate a problem with the custom key store or its AWS CloudHSM cluster. For help, see How to fix a failing CMK (p. 216).

To view the custom key stores in a given account and Region, use the following procedure.

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 **Custom key stores**.

To customize the display, click the gear icon that appears below the **Create key store** button.

View a custom key store (API)

To view your custom key stores, use the DescribeCustomKeyStores operation. By default, this operation returns all custom key stores in the account and Region. But you can use either the `CustomKeyStoreId` or `CustomKeyStoreName` parameter (but not both) to limit the output to a particular custom key store. The output consists of the custom key store ID and name, the ID of the associated AWS CloudHSM cluster, and the connection state. If the connection state indicates an error, the output also includes an error code that describes the reason for the error.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

For example, the following command returns all custom key stores in the account and Region. You can use the `Limit` and `Marker` parameters to page through the custom key stores in the output.

```
$ aws kms describe-custom-key-stores
```

The following example command uses the `CustomKeyStoreName` parameter to get only the custom key store with the `ExampleKeyStore` friendly name. You can use either the `CustomKeyStoreName` or `CustomKeyStoreId` parameter (but not both) in each command.

The following example output represents a custom key store that is connected to its AWS CloudHSM cluster. The `ConnectionState` element corresponds to the `Status` field in the console.

```
$ aws kms describe-custom-key-stores --custom-key-store-name ExampleKeyStore
{
   "CustomKeyStores": [
      {
         "CloudHsmClusterId": "cluster-1a23b4cdefg",
         "ConnectionState": "CONNECTED",
         "CreationDate": "1.499288695918E9",
         "CustomKeyStoreId": "cks-1234567890abcdef0",
         "CustomKeyStoreName": "ExampleKeyStore",
         "TrustAnchorCertificate": "<certificate appears here>"
      }
   ]
}
```

A `ConnectionState` of `Disconnected` indicates that a custom key store has never been connected or it was intentionally disconnected from its AWS CloudHSM cluster (p. 198). However, if attempts to use a CMK in a connected custom key store fail, that might indicate a problem with the custom key store or its AWS CloudHSM cluster. For help, see How to fix a failing CMK (p. 216).

If the `ConnectionState` of the custom key store is `FAILED`, the `DescribeCustomKeyStores` response includes a `ConnectionErrorCode` element that explains the reason for the error.

For example, in the following output, the `INVALID_CREDENTIALS` value indicates that the custom key store connection failed because the `kmsuser` password is invalid (p. 217). For help with this and other connection error failures, see Troubleshooting a custom key store (p. 215).

```
$ aws kms describe-custom-key-stores --custom-key-store-id cks-1234567890abcdef0
{
   "CustomKeyStores": [
      {
         "CloudHsmClusterId": "cluster-1a23b4cdefg",
         "ConnectionErrorCode": "INVALID_CREDENTIALS"
         "ConnectionState": "FAILED",
         "CustomKeyStoreId": "cks-1234567890abcdef0",
         "CustomKeyStoreName": "ExampleKeyStore",
         "CreationDate": "1.499288695918E9",
         "TrustAnchorCertificate": "<certificate appears here>"
      }
   ]
}
```

Editing custom key store settings

You can change the settings of an existing custom key store (p. 186). The custom key store must be disconnected from its AWS CloudHSM cluster.

To edit custom key store settings:

1. Disconnect the custom key store (p. 198) from its AWS CloudHSM cluster. While the custom key store is disconnected, you cannot create customer master keys (p. 3) (CMKs) in the custom key store and you cannot use the CMKs it contains for cryptographic operations (p. 210).
2. Edit one or more of the custom key store settings.
3. Reconnect the custom key store (p. 198) to its AWS CloudHSM cluster.

You can edit the following settings in a custom key store:

The friendly name of the custom key store.

Enter a new friendly name. The new name must be unique in your AWS account.

The cluster ID of the associated AWS CloudHSM cluster.

Edit this value to substitute a related AWS CloudHSM cluster for the original one. You can use this feature to repair a custom key store if its AWS CloudHSM cluster becomes corrupted or is deleted.

Specify an AWS CloudHSM cluster that shares a backup history with the original cluster and fulfills the requirements (p. 191) for association with a custom key store, including two active HSMs in different Availability Zones. Clusters that share a backup history have the same cluster certificate. To view the cluster certificate of a cluster, use the DescribeClusters operation. You cannot use the edit feature to associate the custom key store with an unrelated AWS CloudHSM cluster.

The current password of the `kmsuser` crypto user (p. 187) (CU).

Tells AWS KMS the current password of the `kmsuser` CU in the AWS CloudHSM cluster. This action does not change the password of the `kmsuser` CU in the AWS CloudHSM cluster.

If you change the password of the `kmsuser` CU in the AWS CloudHSM cluster, use this feature to tell AWS KMS the new `kmsuser` password. Otherwise, AWS KMS cannot log into the cluster and all attempts to connect the custom key store to the cluster fail.

Topics

- Edit a custom key store (console) (p. 197)
- Edit a custom key store (API) (p. 197)

Edit a custom key store (console)

When you edit the custom key store, you can change any or of the configurable values.

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 **Custom key stores**.
4. Choose the custom key store you want to edit.
5. If the value in the **Status** column is not **DISCONNECTED**, you must disconnect the custom key store before you can edit it. From the **Key store actions** menu, select **Disconnect custom key store**.
6. From the **Key store actions** menu, select **Edit custom key store settings**.
7. Do one or more of the following actions.
   - Type a new friendly name for the custom key store.
   - Type the cluster ID of a related AWS CloudHSM cluster.
   - Type the current password of the `kmsuser` crypto user in the associated AWS CloudHSM cluster.
8. Choose **Save**.

   When the procedure is successful, a message describes the settings that you edited. When it is unsuccessful, an error message appears that describes the problem and provides help on how to fix it. If you need more help, see Troubleshooting a custom key store (p. 215).
9. Reconnect the custom key store. (p. 198)

   To use the custom key store, you must reconnect it after editing. You can leave the custom key store disconnected. But while it is disconnected, you cannot create CMKs in the custom key store or use the CMKs in the custom key store in cryptographic operations (p. 210).

Edit a custom key store (API)

To change the properties of a custom key store, use the UpdateCustomKeyStore operation. You can change multiple properties of a custom key store in the same command. If the operation is successful, AWS KMS returns an HTTP 200 response and a JSON object with no properties.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

Begin by using DisconnectCustomKeyStore to disconnect the custom key store (p. 198) from AWS KMS. Replace the example custom key store ID, cks-1234567890abcdef0, with an actual ID.

```
$ aws kms disconnect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

The first example uses UpdateCustomKeyStore to change the friendly name of the custom key store to `DevelopmentKeys`. The command uses the `CustomKeyStoreId` parameter to identify the custom key store and the `CustomKeyStoreName` to specify the new name for the custom key store.

```
$ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 --new-custom-
key-store-name DevelopmentKeys
```

The following example changes the cluster that is associated with a custom key store to another backup of the same cluster. The command uses the `CustomKeyStoreId` parameter to identify the custom key store and the `CloudHsmClusterId` parameter to specify the new cluster ID.

```
$ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 --cloud-hsm-
cluster-id cluster-1a23b4cdefg
```

The following example tells AWS KMS that the current `kmsuser` password is `ExamplePassword`. The command uses the `CustomKeyStoreId` parameter to identify the custom key store and the `KeyStorePassword` parameter to specify the current password.

```
$ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 --key-store-
password ExamplePassword
```

The final command reconnects the custom key store to AWS KMS. You can leave the custom key store in the disconnected state, but you must connect it before you can create new CMKs or use existing CMKs for cryptographic operations (p. 210). Replace the example custom key store ID with an actual ID.

```
$ aws kms connect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

Connecting and disconnecting a custom key store

New custom key stores are not connected. Before you can create and use customer master keys (CMKs) in your custom key store, you need to connect it to its associated AWS CloudHSM cluster. You can connect and disconnect your custom key store at any time, and view its connection status (p. 194).

You are not required to connect your custom key store. You can leave a custom key store in a disconnected state indefinitely and connect it only when you need to use it. However, you might want to test the connection periodically to verify that the settings are correct and it can be connected.

> **Note**
> Custom key stores have a `DISCONNECTED` status only when the key store has never been connected or you explicitly disconnect it. If your custom key store status is `CONNECTED` but you are having trouble using it, make sure that its associated AWS CloudHSM cluster is active and contains at least one active HSMs. For help with connection failures, see the section called "Troubleshooting a custom key store" (p. 215).

Connecting a custom key store

When you connect a custom key store, AWS KMS finds the associated AWS CloudHSM cluster, connects to it, logs into the AWS CloudHSM client as the kmsuser crypto user (p. 187) (CU), and then rotates the `kmsuser` password. AWS KMS remains logged into the AWS CloudHSM client as long as the custom key store is connected.

To establish the connection, AWS KMS creates a security group named `kms-<custom key store ID>` in the virtual private cloud (VPC) of the cluster. The security group has a single rule that allows inbound traffic from the cluster security group. AWS KMS also creates an elastic network interface (ENI) in each Availability Zone of the private subnet for the cluster. AWS KMS adds the ENIs to the `kms-<cluster ID>` security group and the security group for the cluster. The description of each ENI is `KMS managed ENI for cluster <cluster-ID>`.

The connection process can take an extended amount of time to complete; up to 20 minutes.

Before you connect the custom key store, verify that it meets the requirements.

- Its associated AWS CloudHSM cluster must contain at least one active HSM. To find the number of HSMs in the cluster, view the cluster in the AWS CloudHSM console or use the DescribeClusters operation. If necessary, you can add an HSM.
- The cluster must have a kmsuser crypto user (p. 192) (CU) account, but that CU cannot be logged into the cluster when you connect the custom key store. For help with logging out, see How to log out and reconnect (p. 221).

- The connection status of the custom key store cannot be `DISCONNECTING` or `FAILED`. You can view the connection status (p. 194) in the console or by using the DescribeCustomKeyStores operation. If the connection status is `FAILED`, disconnect the custom key store, and then connect it.

When your custom key store is connected, you can create CMKs in it (p. 204) and use existing CMKs in cryptographic operations (p. 210).

Disconnecting a custom key store

When you disconnect a custom key store, AWS KMS logs out of the AWS CloudHSM client, disconnects from the associated AWS CloudHSM cluster, and removes the network infrastructure that it created to support the connection.

While a custom key store is disconnected, you can manage the custom key store and its customer master keys (CMKs), but you cannot create or use CMKs in the custom key store. The status of the key store is `DISCONNECTED` and the key state (p. 233) of CMKs in the custom key store is `Unavailable`, unless they are `PendingDeletion`. You can reconnect the custom key store at any time.

> **Note**
> While a custom key store is disconnected, all attempts to create customer master keys (CMKs) in the custom key store or to use existing CMKs in cryptographic operations will fail. This action can prevent users from storing and accessing sensitive data.

To better estimate the effect of disconnecting your key store, identify the CMKs (p. 211) in the custom key store and determine their past use (p. 181).

You might disconnect the custom key store for reasons such as the following:

- **To rotate of the `kmsuser` password.** AWS KMS changes the `kmsuser` password each time that it connects to the AWS CloudHSM cluster. To force a password rotation, just disconnect and reconnect.
- **To audit the key material** for the CMKs in the AWS CloudHSM cluster. When you disconnect the custom key store, AWS KMS logs out of the `kmsuser` crypto user (p. 187) account in the AWS CloudHSM client. This allows you to log into the cluster as the `kmsuser` CU and audit and manage the key material for the CMK.
- **To immediately disable all CMKs** in the custom key store. You can disable and re-enable CMKs (p. 51) in a custom key store by using the AWS Management Console or the DisableKey operation. These operations complete quickly, but they act on one CMK at a time. Disconnecting immediately changes the key state of all CMKs in the custom key to `Unavailable`, which prevents them from being used in any cryptographic operation.
- **To repair a failed connection attempt**. If an attempt to connect a custom key store fails (the connection status of the custom key store is `FAILED`), you must disconnect the custom key store before you try to connect it again.

Topics

- Connect a custom key store (console) (p. 199)
- Connect a custom key store (API) (p. 200)
- Disconnect a custom key store (console) (p. 201)
- Disconnect a custom key store (API) (p. 201)

Connect a custom key store (console)

To connect a custom key store in the AWS Management Console, begin by selecting the custom key store from the **Custom key stores** page. The process can take up to 20 minutes to complete.

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 **Custom key stores**.
4. Choose the custom key store you want to connect.
5. If the status of the custom key store is **FAILED**, you must disconnect the custom key store (p. 201) before you connect it.
6. From the **Key store actions** menu, select **Connect custom key store**.

AWS KMS begins the process of connecting your custom key store. It finds the associated AWS CloudHSM cluster, builds the required network infrastructure, connects to it, logs into the AWS CloudHSM cluster as the `kmsuser` CU, and rotates the `kmsuser` password. When the operation completes, the connection state changes to **CONNECTED**.

If the operation fails, an error message appears that describes the reason for the failure. Before you try to connect again, view the connection status (p. 194) of your custom key store. If it is **FAILED**, you must disconnect the custom key store (p. 201) before you connect it again. If you need help, see Troubleshooting a custom key store (p. 215).

Next: Create CMKs in a custom key store (p. 204).

Connect a custom key store (API)

To connect a disconnected custom key store, use the ConnectCustomKeyStore operation. The associated AWS CloudHSM cluster must contain at least one active HSM and the connection status cannot be `FAILED`.

The connection process takes an extended amount of time to complete; up to 20 minutes. Unless it fails quickly, the operation returns an HTTP 200 response and a JSON object with no properties. However, this initial response does not indicate that the connection was successful. To determine the connection status of the custom key store, use the DescribeCustomKeyStores operation.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

To identify the custom key store, use the custom key store ID. You can find the ID on the **Custom key stores** page in the console or by using the DescribeCustomKeyStores operation. Before running this example, replace the example ID with a valid one.

```
$ aws kms connect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

To verify that the custom key store is connected, use the DescribeCustomKeyStores operation. By default, this operation returns all custom keys stores in your account and Region. But you can use either the `CustomKeyStoreId` or `CustomKeyStoreName` parameter (but not both) to limit the response to particular custom key stores. The `ConnectionState` value of `CONNECTED` indicates that the custom key store is connected to its AWS CloudHSM cluster.

```
$ aws kms describe-custom-key-stores --custom-key-store-id cks-1234567890abcdef0
{
   "CustomKeyStores": [
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleKeyStore",
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "TrustAnchorCertificate": "<certificate string appears here>",
      "CreationDate": "1.499288695918E9",
```

```
        "ConnectionState": "CONNECTED"
    ],
}
```

If the `ConnectionState` value is failed, the `ConnectionErrorCode` element indicates the reason for the failure. In this case, AWS KMS could not find an AWS CloudHSM cluster in your account with the cluster ID `cluster-1a23b4cdefg`. If you deleted the cluster, you can restore it from a backup of the original cluster and then edit the cluster ID (p. 196) for the custom key store.

```
$ aws kms describe-custom-key-stores --custom-key-store-id cks-1234567890abcdef0
{
    "CustomKeyStores": [
        "CustomKeyStoreId": "cks-1234567890abcdef0",
        "CustomKeyStoreName": "ExampleKeyStore",
        "CloudHsmClusterId": "cluster-1a23b4cdefg",
        "TrustAnchorCertificate": "<certificate string appears here>",
        "CreationDate": "1.499288695918E9",
        "ConnectionState": "FAILED"
        "ConnectionErrorCode": "CLUSTER_NOT_FOUND"
    ],
}
```

Next: Create CMKs in a custom key store (p. 204).

Disconnect a custom key store (console)

To disconnect a connected custom key store in the AWS Management Console, begin by selecting the custom key store from the **Custom Key Stores** 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 **Custom key stores**.
4. Choose the custom key store you want to disconnect.
5. From the **Key store actions** menu, select **Disconnect custom key store**.

When the operation completes, the connection state changes from **DISCONNECTING** to **DISCONNECTED**. If the operation fails, an error message appears that describes the problem and provides help on how to fix it. If you need more help, see Troubleshooting a custom key store (p. 215).

Disconnect a custom key store (API)

To disconnect a connected custom key store, use the DisconnectCustomKeyStore operation. If the operation is successful, AWS KMS returns an HTTP 200 response and a JSON object with no properties.

The examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

This example disconnects a custom key store. Before running this example, replace the example ID with a valid one.

```
$ aws kms disconnect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

To verify that the custom key store is disconnected, use the DescribeCustomKeyStores operation. By default, this operation returns all custom keys stores in your account and Region. But you can use either

the `CustomKeyStoreId` and `CustomKeyStoreName` parameter (but not both) to limit the response to particular custom key stores. The `ConnectionState` value of `DISCONNECTED` indicates that the custom key store is not connected to its AWS CloudHSM cluster.

```
$ aws kms describe-custom-key-stores --custom-key-store-id cks-1234567890abcdef0
{
   "CustomKeyStores": [
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "ConnectionState": "DISCONNECTED",
      "CreationDate": "1.499288695918E9",
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleKeyStore",
      "TrustAnchorCertificate": "<certificate string appears here>"
   ],
}
```

Deleting a custom key store

When you delete a custom key store, AWS KMS deletes all metadata about the custom key store from KMS, including information about its association with an AWS CloudHSM cluster. This operation does not affect the AWS CloudHSM cluster, its HSMs, or its users. You can create a new custom key store that is associated with the specified cluster, but you cannot undo the delete operation.

You can only delete a custom key store that is disconnected from AWS KMS and does not contain any customer master keys (CMKs). Before you delete a custom key store, do the following.

- Verify that you will never need to use any of the CMKs in the key store for any cryptographic operations (p. 210). Then schedule deletion (p. 214) of all of the CMKs from the key store. For help finding the CMKs in a custom key store, see Find the CMKs in a custom key store (p. 211).
- Confirm that all CMKs have been deleted. To view the CMKs in a custom key store, see Viewing CMKs in a custom key store (p. 208).
- Disconnect the custom key store (p. 198) from AWS KMS.

Instead of deleting the custom key store, consider disconnecting it (p. 198) from its associated AWS CloudHSM cluster. While a custom key store is disconnected, you can manage the custom key store and its customer master keys (CMKs). But you cannot create or use CMKs in the custom key store. You can reconnect the custom key store at any time.

If you have deleted all custom key stores from all Regions of your AWS account and you do not plan to create any more, you should delete the service-linked role (p. 189) that AWS KMS uses for custom key stores.

Topics

- Delete a custom key store (console) (p. 202)
- Delete a custom key store (API) (p. 203)

Delete a custom key store (console)

To delete a custom key store in the AWS Management Console, begin by selecting the custom key store from the **Custom key stores** 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 **Custom key stores**.
4. Find the row that represents the custom key store that you want to remove. If the status of the custom key store is not **DISCONNECTED**, you must disconnect the custom key store (p. 198) before you delete the custom key store.
5. From the **Key store actions** menu, select **Delete custom key store**.

When the operation completes, a success message appears and the custom key store no longer appears in the custom key store list. If the operation is unsuccessful, an error message appears that describes the problem and provides help on how to fix it. If you need more help, see Troubleshooting a custom key store (p. 215).

Delete a custom key store (API)

To delete a custom key store, use the DeleteCustomKeyStore operation. If the operation is successful, AWS KMS returns an HTTP 200 response and a JSON object with no properties.

To begin, verify that the custom key store does not contain any AWS KMS customer master keys (CMKs). You cannot delete a custom key store that contains CMKs. The first example command uses ListKeys and DescribeKey to search for AWS KMS customer master keys in the custom key store with the *cks-1234567890abcdef0* fictitious key store ID. In this case, the command does not return any CMKs. If it does, use the ScheduleKeyDeletion operation to schedule deletion of each of the CMKs.

Bash

```
for key in $(aws kms list-keys --query 'Keys[*].KeyId' --output text) ;
do aws kms describe-key --key-id $key |
grep '"CustomKeyStoreId": "cks-1234567890abcdef0"' --context 100; done
```

PowerShell

```
PS C:\> (Get-KMSKeyList).KeyArn | foreach {Get-KMSKey -KeyId $_} | where
 CustomKeyStoreId -eq 'cks-1234567890abcdef0'
```

Next, disconnect the custom key store. This example command uses the DisconnectCustomKeyStore operation to disconnect the custom key store from its AWS CloudHSM cluster. Before running this command, replace the example custom key store ID with a valid one.

Bash

```
$ aws kms disconnect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

PowerShell

```
PS C:\> Disconnect-KMSCustomKeyStore -CustomKeyStoreId cks-1234567890abcdef0
```

After the custom key store is disconnected, you can use the DeleteCustomKeyStore operation to delete it.

Bash

```
$ aws kms delete-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

PowerShell

```
PS C:\> Remove-KMSCustomKeyStore -CustomKeyStoreId cks-1234567890abcdef0
```

Managing CMKs in a custom key store

You can create, view, manage, use, and schedule deletion of the customer master keys (CMKs) in a custom key store. The procedures that you use are very similar to those that you use for CMKs in AWS KMS. The only difference is that you specify a custom key store when you create the CMK. Then, AWS KMS creates non-extractable key material for the CMK in the AWS CloudHSM cluster that is associated with the custom key store. When you use a CMK in a custom key store, the cryptographic operations (p. 210) are performed in the HSMs in the cluster.

> **Note**
> AWS KMS custom key stores support only symmetric keys. Although AWS CloudHSM supports asymmetric keys, you cannot create asymmetric CMKs or asymmetric data key pairs in a custom key store.
> You cannot import key material (p. 159) into a CMK in a custom key store. AWS KMS generates the key material for the CMK in the AWS CloudHSM cluster.

In addition to the procedures discussed in this section, you can do the following with CMKs in a custom key store:

- Use key policies, IAM policies, and grants to authorize access (p. 56) to the CMK.
- Assign tags (p. 49) to the CMKs and create aliases (p. 360) that refer to the CMKs.
- Use the CMKs for cryptographic operations (p. 10), including encrypting, decrypting, re-encrypting, and generating data keys.
- Use the CMKs with AWS services that integrate with AWS KMS (p. 238) and support customer managed CMKs.
- Track your CMK use in AWS CloudTrail logs (p. 303) and Amazon CloudWatch monitoring tools (p. 296).

However, you cannot import key material into a CMK in a custom key store.

Topics

- Creating CMKs in a custom key store (p. 204)
- Viewing CMKs in a custom key store (p. 208)
- Using CMKs in a custom key store (p. 210)
- Finding CMKs and key material (p. 210)
- Scheduling deletion of CMKs from a custom key store (p. 214)

Creating CMKs in a custom key store

After you have created a custom key store, you can create customer master keys (p. 3) (CMKs) in your key store. They must be symmetric CMKs (p. 142) with key material that AWS KMS generates. You cannot create asymmetric CMKs (p. 142) or CMKs with imported key material (p. 159), and you cannot use symmetric CMKs in a custom key store to generate asymmetric data key pairs.

Use and manage the CMKs in your custom key store the same way that you use and manage any CMK in AWS KMS. For example, you can do any of the following:

- Use the CMKs for cryptographic operations (p. 10).
- Set IAM and key policies on the CMKs.
- Create aliases are associated with the CMKs.
- Attach tags to the CMKs.
- Enable and disable the CMKs.
- Schedule deletions of the CMKs.

To create a CMK in a custom key store, the custom key store must be connected to its associated AWS CloudHSM cluster (p. 198) and the cluster must contain at least two active HSMs in different Availability Zones. To find the connection status and number of HSMs, view the custom key stores page (p. 194) in the AWS Management Console. When using the API operations, use the DescribeCustomKeyStores operation to verify that the custom key store is connected. Use the AWS CloudHSM DescribeClusters operation to get the number of active HSMs in the cluster and their Availability Zones.

When you create a CMK in your custom key store, AWS KMS creates the CMK in AWS KMS. But, it creates the key material for the CMK in the associated AWS CloudHSM cluster. Specifically, AWS KMS signs into the cluster as the `kmsuser` CU that you created (p. 191). Then it creates a persistent, non-extractable, 256-bit Advanced Encryption Standard (AES) symmetric key in the cluster. AWS KMS sets the value of the key label attribute, which is visible only in the cluster, to Amazon Resource Name (ARN) of the CMK.

When the command succeeds, the key state (p. 233) of the new CMK is `Enabled` and its origin is `AWS_CLOUDHSM`. You cannot change the origin of any CMK after you create it. When you view a CMK in a custom key store in the console or by using the DescribeKey operation, you can see typical properties, like its key ID, key state, and creation date. But you can also see the custom key store ID and (optionally) the AWS CloudHSM cluster ID. For details, see Viewing CMKs in a custom key store (p. 208).

If your attempt to create a CMK in your custom key store fails, use the error message to help you determine the cause. It might indicate that the custom key store is not connected (`CustomKeyStoreInvalidStateException`) or the associated AWS CloudHSM cluster doesn't have the two active HSMs that are required for this operation (`CloudHsmClusterInvalidConfigurationException`). For help see Troubleshooting a custom key store (p. 215).

Topics

- Create a CMK in a custom key store (console) (p. 205)
- Create a CMK in a custom key store (API) (p. 207)

Create a CMK in a custom key store (console)

Use the following procedure to create a customer master key (CMK) in a custom key store.

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. Choose **Symmetric**.

   You cannot create an asymmetric CMK in a custom key store.
6. Choose **Advanced options**.

7. For **Key material origin**, choose **Custom key store (CloudHSM)**.
8. Choose **Next**.
9. Select a custom key store for your new CMK. To create a new custom key store, choose **Create custom key store**.

   The custom key store that you select must have a status of **CONNECTED**. Its associated AWS CloudHSM cluster must be active and contain at least two active HSMs in different Availability Zones.

   For help with connecting a custom key store, see Connecting and disconnecting a custom key store (p. 198). For help with adding HSMs, see Adding an HSM in the *AWS CloudHSM User Guide*.
10. Choose **Next**.
11. Type an alias and an optional description for the CMK.
12. (Optional). On the **Add Tags** page, add tags that identify or categorize your CMK.

    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. 49).
13. Choose **Next**.
14. 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. 62).

    Note
    IAM policies can give other IAM users and roles permission to use the CMK.
15. (Optional) To prevent these key administrators from deleting this CMK, clear the box at the bottom of the page for **Allow key administrators to delete this key.**
16. Choose **Next**.
17. In the **This account** section, select the IAM users and roles in this AWS account who can use the CMK in cryptographic operations (p. 10). For more information, see Allows Key Users to Use the CMK (p. 64).

    Note
    IAM policies can give other IAM users and roles permission to use the CMK.
18. (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
    Administrators of the other AWS accounts must also allow access to the CMK by creating IAM policies for their users. For more information, see Allowing users in other accounts to use a CMK (p. 82).
19. Choose **Next**.
20. On the **Review and edit key policy** page, review and edit the policy document for the new CMK. When you're done, choose **Finish**.

When the procedure succeeds, the display shows the new CMK in the custom key store that you chose. When you choose the name or alias of the new CMK, its detail page displays the origin of the CMK (**CloudHSM**), the name and ID of the custom key store, and the ID of the AWS CloudHSM cluster. If the procedure fails, an error message appears that describes the failure.

Tip
To make it easier to identify CMKs in a custom key store, on the **Customer managed keys** page, add the **Custom key store ID** column to the display. Click the gear icon in the upper-right and select **Custom key store ID**.

Create a CMK in a custom key store (API)

To create a new customer master key (p. 3) (CMK) in your custom key store, use the CreateKey operation. Use the `CustomKeyStoreId` parameter to identify your custom key store and specify an `Origin` value of `AWS_CLOUDHSM`.

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 examples in this section use the AWS Command Line Interface (AWS CLI), but you can use any supported programming language.

The following example begins with a call to the DescribeCustomKeyStores operation to verify that the custom key store is connected to its associated AWS CloudHSM cluster. By default, this operation returns all custom keys stores in your account and Region. To describe only a particular custom key store, use the `CustomKeyStoreId` or `CustomKeyStoreName` parameter (but not both).

Before running this command, replace the example custom key store ID with a valid ID.

```
$ aws kms describe-custom-key-stores --custom-key-store-id cks-1234567890abcdef0
{
   "CustomKeyStores": [
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleKeyStore",
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "TrustAnchorCertificate": "<certificate string appears here>",
      "CreationDate": "1.499288695918E9",
      "ConnectionState": "CONNECTED"
   ],
}
```

The next example command uses the DescribeClusters operation to verify that the AWS CloudHSM cluster that is associated with the `ExampleKeyStore` (cluster-1a23b4cdefg) has at least two active HSMs. If the cluster has fewer than two HSMs, the `CreateKey` operation fails.

```
$ aws cloudhsmv2 describe-clusters
{
    "Clusters": [
        {
            "SubnetMapping": {
               ...
            },
            "CreateTimestamp": 1507133412.351,
            "ClusterId": "cluster-1a23b4cdefg",
            "SecurityGroup": "sg-865af2fb",
            "HsmType": "hsm1.medium",
            "VpcId": "vpc-1a2b3c4d",
            "BackupPolicy": "DEFAULT",
            "Certificates": {
                "ClusterCertificate": "-----BEGIN CERTIFICATE-----\...\n-----END
 CERTIFICATE-----\n"
            },
            "Hsms": [
                {
                    "AvailabilityZone": "us-west-2a",
                    "EniIp": "10.0.1.11",
                    "ClusterId": "cluster-1a23b4cdefg",
                    "EniId": "eni-ea8647e1",
                    "StateMessage": "HSM created.",
                    "SubnetId": "subnet-a6b10bd1",
```

```
                    "HsmId": "hsm-abcdefghijk",
                    "State": "ACTIVE"
                },
                {
                    "AvailabilityZone": "us-west-2b",
                    "EniIp": "10.0.0.2",
                    "ClusterId": "cluster-1a23b4cdefg",
                    "EniId": "eni-ea8647e1",
                    "StateMessage": "HSM created.",
                    "SubnetId": "subnet-b6b10bd2",
                    "HsmId": "hsm-zyxwvutsrqp",
                    "State": "ACTIVE"
                },
            ],
            "State": "ACTIVE"
        }
    ]
}
```

This example command uses the CreateKey operation to create a CMK the custom key store. To create a CMK in a custom key store, you must provide the ID of the custom key store name and specify an `Origin` value of `AWS_CLOUDHSM`.

The response includes the IDs of the custom key store and the AWS CloudHSM cluster.

Before running this command, replace the example custom key store ID with a valid ID.

```
$ aws kms create-key --origin AWS_CLOUDHSM --custom-key-store-id cks-1234567890abcdef0
{
  "KeyMetadata": {
    "AWSAccountId": "111122223333",
    "Arn": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
    "CreationDate": 1.499288695918E9,
    "Description": "Example key",
    "Enabled": true,
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
    "KeyManager": "CUSTOMER",
    "KeyState": "Enabled",
    "KeyUsage": "ENCRYPT_DECRYPT",
    "Origin": "AWS_CLOUDHSM"
    "CloudHsmClusterId": "cluster-1a23b4cdefg",
    "CustomKeyStoreId": "cks-1234567890abcdef0"
    "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
    "EncryptionAlgorithms": [
        "SYMMETRIC_DEFAULT"
    ]
  }
}
```

Viewing CMKs in a custom key store

To view the customer master keys (CMKs) in a custom key store, use the same techniques that you would use to view any AWS KMS customer managed CMKs (p. 3). To learn the basics, see Viewing keys (p. 29). To identify the keys in your AWS CloudHSM cluster that serve as key material for your CMK, see Finding CMKs and key material (p. 210).

In the AWS Management Console, the CMKs in your custom key store are displayed along with all other customer managed CMKs your AWS account and Region.

However, the following values are specific to CMKs in a custom key store.

- The name and ID of the custom key store that stores the CMK.
- The cluster ID of the associated AWS CloudHSM cluster that contains their key material.
- An `Origin` value of `CloudHSM` in the AWS Management Console or `AWS_CLOUDHSM` in API responses.
- The key state (p. 233) value can be `Unavailable`. For help resolving the status, see How to fix unavailable CMKs (p. 215).

To view the CMKs in a custom key store (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. In the navigation pane, choose **Customer managed keys**.
4. In the upper-right corner, choose the gear icon, choose **Custom key store ID** and **Origin**, then choose **Confirm**.
5. To identify CMKs in any custom key store, look for CMKs with an **Origin** value of **AWS_CLOUDHSM**. To identify CMKs in a particular custom key store, view the values in the **Custom key store ID** column.
6. Choose the alias or key ID of a CMK in a custom key store.

   This page displays detailed information about the CMK, including its Amazon Resource Name (ARN), key policy, and tags.
7. Expand **Cryptographic configuration**.

   This section includes information about the CMK's custom key store and cluster.

To view the CMKs in a custom key store (API)

You use the same AWS KMS API operations to view the CMKs in a custom key store that you would use for any CMK, including ListKeys, DescribeKey, and GetKeyPolicy. For example, the following `describe-key` operation in the AWS CLI shows the special fields for a CMK in a custom key store. Before running a command like this one, replace the example CMK ID with a valid value.

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

{
   "KeyMetadata": {
      "AWSAccountId": "111122223333",
      "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab",
      "Arn": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
      "CreationDate": 1537582718.431,
      "Enabled": true,
      "KeyManager": "CUSTOMER",
      "KeyState": "Enabled",
      "KeyUsage": "ENCRYPT_DECRYPT",
      "Origin": "AWS_CLOUDHSM",
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "Description": "CMK in custom key store"
      "CustomerMasterKeySpec": "SYMMETRIC_DEFAULT",
      "EncryptionAlgorithms": [
         "SYMMETRIC_DEFAULT"
      ]
   }
}
```

For help finding the CMKs in a custom key store or identifying the keys in your AWS CloudHSM cluster that serve as key material for your CMK, see Finding CMKs and key material (p. 210).

Using CMKs in a custom key store

After you create a symmetric CMK in a custom key store (p. 204), you can use it for the following cryptographic operations:

- Encrypt
- Decrypt
- GenerateDataKey
- GenerateDataKeyWithoutPlaintext
- ReEncrypt

Asymmetric CMKs and asymmetric data key pairs are not supported in a custom key store. As a result, you cannot use the operations that are specific to asymmetric CMKs — Sign, Verify, and GetPublicKey. Also, the operations that generate asymmetric data key pairs, GenerateDataKeyPair and GenerateDataKeyPairWithoutPlaintext, are not supported in a custom key store.

When you use your CMK in a request, identify the CMK by its ID or alias; you do not need to specify the custom key store or AWS CloudHSM cluster. The response includes the same fields that are returned for any symmetric CMK.

However, when you use a CMK in a custom key store, the cryptographic operation is performed entirely within the AWS CloudHSM cluster that is associated with the custom key store. The operation uses the key material in the cluster that is associated with the CMK that you chose.

To make this possible, the following conditions are required.

- The key state (p. 233) of the CMK must be `Enabled`. To find the key state, use the **Status** field in the AWS Management Console (p. 208) or the `KeyState` field in the DescribeKey response.
- The custom key store must be connected to its AWS CloudHSM cluster. Its **Status** in the AWS Management Console (p. 194) or `ConnectionState` in the DescribeCustomKeyStores response must be `CONNECTED`.
- The AWS CloudHSM cluster that is associated with the custom key store must contain at least one active HSM. To find the number of active HSMs in the cluster, use the AWS KMS console (p. 194), the AWS CloudHSM console, or the DescribeClusters operation.
- The AWS CloudHSM cluster must contain the key material for the CMK. If the key material was deleted from the cluster, or an HSM was created from a backup that did not include the key material, the cryptographic operation will fail.

If these conditions are not met, the cryptographic operation fails, and AWS KMS returns a `KMSInvalidStateException` exception. Typically, you just need to reconnect the custom key store (p. 198). For additional help, see How to fix a failing CMK (p. 216).

When using the CMKs in a custom key store, be aware that the CMKs in each custom key store share a per-second quota (p. 374) on requests for cryptographic operations. If you exceed the quota, AWS KMS returns a `ThrottlingException`. If the AWS CloudHSM cluster that is associated with the custom key store is processing numerous commands, including those unrelated to the custom key store, you might get a `ThrottlingException` at an even lower rate. If you get a `ThrottlingException` for any request, lower your request rate and try the commands again. For details about the request quota for cryptographic operations in a custom key store, see Custom key store quota (p. 374).

Finding CMKs and key material

If you manage a custom key store, you might need to identify the CMKs in each custom key store. For example, you might need to do some of the following tasks.

- Track the CMKs in custom key store in AWS CloudTrail logs.
- Predict the effect on CMKs of disconnecting a custom key store.
- Schedule deletion of CMKs before you delete a custom key store.

In addition, you might want to identify the keys in your AWS CloudHSM cluster that serve as key material for your CMKs. Although AWS KMS manages the CMKs and their key material, you still retain control of and responsibility for the management of your AWS CloudHSM cluster, its HSMs and backups and the keys in the HSMs. You might need to identify the keys in order to audit the key material, protect it from accidental deletion, or delete it from HSMs and cluster backups after deleting the CMK.

All key material for the CMKs in your custom key store is owned by the `kmsuser` crypto user (p. 187) (CU). AWS KMS sets the key label attribute, which is viewable only in AWS CloudHSM, to the Amazon Resource Name (ARN) of the CMK.

To find CMKs and key material, use any of the following techniques.

- Find the CMKs in a custom key store (p. 211) — How to identify the CMKs in one or all of your custom key stores.
- Find all keys for a custom key store (p. 212) — How to find all keys in your cluster that serve as key material for the CMKs in your custom key store.
- Find the key for a CMK (p. 213) — How to find the key in your cluster that serves as key material for a particular CMK in your custom key store.
- Find the CMK for a key (p. 213) — How to find the CMK for a particular key in your cluster.

Find the CMKs in a custom key store

If you manage a custom key store, you might need to identify the CMKs in each custom key store. You can use this information track the CMK operations in AWS CloudTrail logs, predict the effect on CMKs of disconnecting a custom key store, or schedule deletion of CMKs before you delete a custom key store.

To find the CMKs in a custom key store (console)

To find the CMKs in a particular custom key store, on the **Customer Managed Keys** page, view the values in the **Custom Key Store Name** or **Custom Key Store ID** fields. To identify CMKs in any custom key store, look for CMKs with an **Origin** value of **CloudHSM**. To add optional columns to the display, choose the gear icon in the upper right corner of the page.

To find the CMKs in a custom key store (API)

To find the CMKs in a custom key store, use the ListKeys and DescribeKey operations and then filter the `CustomKeyStoreId` value. Before running the examples, replace the fictitious custom key store ID values with a valid value.

Bash

To find CMKs in a particular custom key store, get all of your CMKs in the account and Region. Then filter the ID of the custom key store.

```
for key in $(aws kms list-keys --query 'Keys[*].KeyId' --output text) ;
do aws kms describe-key --key-id $key |
grep '"CustomKeyStoreId": "cks-1234567890abcdef0"' --context 100; done
```

To get CMKs in any custom key store in the account and Region, search for `CustomKeyStoreId` values that begin with `cks-`.

```
for key in $(aws kms list-keys --query 'Keys[*].KeyId' --output text) ;
```

```
do aws kms describe-key --key-id $key |
grep '"CustomKeyStoreId": "cks-"' --context 100; done
```

PowerShell

To find CMKs in a particular custom key store, use the Get-KmsKeyList Get-KmsKey cmdlets to get all of your CMKs in the account and Region. Then filter for the ID of the custom key store.

```
PS C:\> (Get-KMSKeyList).KeyArn | foreach {Get-KMSKey -KeyId $_} | where
 CustomKeyStoreId -eq 'cks-1234567890abcdef0'
```

To get CMKs in any custom key store in the account and Region, use the `-like` comparison operator. All custom key store identifiers begin with `cks-`.

```
PS C:\> (Get-KMSKeyList).KeyArn | foreach {Get-KMSKey -KeyId $_} | where
 CustomKeyStoreId -like 'cks*'
```

Find all keys for a custom key store

You can identify the keys in your AWS CloudHSM cluster that serve as key material for your custom key store. To do that, use the findAllKeys command in cloudhsm_mgmt_util to find the key handles of all keys that `kmsuser` owns or shares. Unless you have logged in as `kmsuser` and created keys outside of AWS KMS, all of the keys that `kmsuser` owns represent key material for AWS KMS CMKs.

Any crypto officer in the cluster can run this command without disconnecting the custom key store.

1. Start cloudhsm_mgmt_util by using the procedure described in the Prepare to run cloudhsm_mgmt_util topic.
2. Log into cloudhsm_mgmt_util using a crypto officer (CO) account.
3. Use the listUsers command to find the user ID of the `kmsuser` crypto user.

   In this example, `kmsuser` has user ID 3.

```
aws-cloudhsm> listUsers
Users on server 0(10.0.0.1):
Number of users found:3

    User Id             User Type       User Name            MofnPubKey
 LoginFailureCnt         2FA
         1              PCO             admin                      NO               0
             NO
         2              AU              app_user                   NO               0
             NO
         3              CU              kmsuser                    NO               0
             NO
```

4. Use the findAllKeys command to find the key handles of all keys that `kmsuser` owns or shares. Replace the example user ID with the actual user ID of `kmsuser` in your cluster.

   The example output shows that `kmsuser` owns keys with key handles 8, 9, and 262162 on both HSMs in the cluster.

```
aws-cloudhsm> findAllKeys 3 0
Keys on server 0(10.0.0.1):
Number of keys found 3
number of keys matched from start index 0::6
8,9,262162
findAllKeys success on server 0(10.0.0.1)
```

```
Keys on server 1(10.0.0.2):
Number of keys found 6
number of keys matched from start index 0::6
8,9,262162
findAllKeys success on server 1(10.0.0.2)
```

Find the CMK for a key

If you know the key handle of a key that `kmsuser` owns in the cluster, you can use the key label to identify the associated CMK in your custom key store.

When AWS KMS creates the key material for a CMK in your AWS CloudHSM cluster, it writes the Amazon Resource Name (ARN) of the CMK in the key label. Unless you have changed the label value, you can use the getAttribute command in key_mgmt_util or cloudhsm_mgmt_util to associate the key with its CMK.

To run this procedure, you need to disconnect the custom key store temporarily so you can log in as the `kmsuser` CU.

> **Note**
> While a custom key store is disconnected, all attempts to create customer master keys (CMKs) in the custom key store or to use existing CMKs in cryptographic operations will fail. This action can prevent users from storing and accessing sensitive data.

1. Disconnect the custom key store, if it is not already disconnected., then log into the key_mgmt_util as `kmsuser`, as explained in How to disconnect and log in (p. 220).
2. Use the `getAttribute` command in key_mgmt_util or cloudhsm_mgmt_util to get the label attribute (`OBJ_ATTR_LABEL`, attribute `3`) for a particular key handle.

   For example, this command uses `getAttribute` in cloudhsm_mgmt_util to get the label attribute (attribute `3`) of the key with key handle `262162`. The output shows that key `262162` serves as key material for the CMK with ARN `arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`. Before running this command, replace the example key handle with a valid one.

   For a list of key attributes, use the listAttributes command or see the Key Attribute Reference in the *AWS CloudHSM User Guide*.

```
aws-cloudhsm> getAttribute 262162 3

Attribute Value on server 0(10.0.1.10):
OBJ_ATTR_LABEL
arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

Attribute Value on server 1(10.0.1.12):
OBJ_ATTR_EXTRACTABLE
arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

3. Log out of key_mgmt_util or cloudhsm_mgmt_util and reconnect the custom key store as explained in How to log out and reconnect (p. 221).

Find the key for a CMK

You can use the CMK ID of a CMK in a custom key store to identify the key in your cluster that serves as its key material. Then you can use its key handle to identify the key in AWS CloudHSM client commands.

When AWS KMS creates the key material for a CMK in your AWS CloudHSM cluster, it writes the Amazon Resource Name (ARN) of the CMK in the key label. Unless you have changed the label value, you can

use the findKey command in key_mgmt_util to get the key handle of the key material for the CMK. To run this procedure, you need to disconnect the custom key store temporarily so you can log in as the `kmsuser` CU.

> **Note**
> While a custom key store is disconnected, all attempts to create customer master keys (CMKs) in the custom key store or to use existing CMKs in cryptographic operations will fail. This action can prevent users from storing and accessing sensitive data.

1. Disconnect the custom key store, if it is not already disconnected, then log into the key_mgmt_util as `kmsuser`, as explained in How to disconnect and log in (p. 220).
2. Use the findKey command in key_mgmt_util to search for a key with a label that matches the ARN of a CMK in your custom key store. Replace the example CMK ARN in the value of the `-l` (lower-case L for 'label') parameter with a valid CMK ARN.

   For example, this command finds the key with a label that matches the example CMK ARN, `arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab`. The example output shows that the key with key handle `262162` has the specified CMK ARN in its label. You can now use this key handle in other key_mgmt_util commands.

```
Command: findKey -l arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
Total number of keys present 1

 number of keys matched from start index 0::1
262162

        Cluster Error Status
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS

        Cfm3FindKey returned: 0x00 : HSM Return: SUCCESS
```

3. Log out of key_mgmt_util and reconnect the custom key store as explained in How to log out and reconnect (p. 221).

Scheduling deletion of CMKs from a custom key store

When you are certain that you will not need to use a customer master key (CMK) for any cryptographic operation, you can schedule the deletion of the CMK (p. 172). Use the same procedure that you would use to schedule the deletion of any CMK from AWS KMS. In addition, keep your custom key store connected so AWS KMS can delete the corresponding key material from the associated AWS CloudHSM cluster when the waiting period expires.

> **Warning**
> Deleting a CMK is a destructive and potentially dangerous operation that prevents you from recovering all data encrypted under the CMK. Before scheduling deletion of the CMK, examine past usage (p. 181) of the CMK and create a Amazon CloudWatch alarm (p. 177) that alerts you when someone tries to use the CMK while it is pending deletion. Whenever possible, disable the CMK (p. 51), instead of deleting it.

If you schedule deletion of a CMK from a custom key store, its key state (p. 233) changes to **Pending deletion**. The CMK remains in the **Pending deletion** state throughout the waiting period, even if the CMK becomes unavailable because you have disconnected the custom key store (p. 198). This allows you to cancel the deletion of the CMK at any time during the waiting period.

When the waiting period expires, AWS KMS deletes the CMK from AWS KMS. Then AWS KMS makes a best effort to delete the key material from the associated AWS CloudHSM cluster. If AWS KMS cannot

delete the key material, such as when the key store is disconnected from AWS KMS, you might need to manually delete the orphaned key material (p. 218) from the cluster.

AWS KMS does not delete the key material from cluster backups. Even if you delete the CMK from AWS KMS and delete its key material from your AWS CloudHSM cluster, clusters created from backups might contain the deleted key material. To permanently delete the key material view the creation date (p. 208) of the CMK. Then delete all cluster backups that might contain the key material.

Troubleshooting a custom key store

Custom key stores are designed to be available and resilient. However, there are some error conditions that you might have to repair to keep your custom key store operational.

Topics

- How to fix unavailable CMKs (p. 215)
- How to fix a failing CMK (p. 216)
- How to fix a connection failure (p. 216)
- How to fix invalid kmsuser credentials (p. 217)
- How to delete orphaned key material (p. 218)
- How to recover deleted key material for a CMK (p. 219)
- How to log in as kmsuser (p. 220)

How to fix unavailable CMKs

The key state (p. 233) of customer master keys (CMKs) in a custom key store is typically `Enabled`. Like all CMKs, the key state changes when you disable the CMKs in a custom key store or schedule them for deletion. However, unlike other CMKs, the CMKs in a custom key store can also have a key state (p. 233) of `Unavailable`.

A key state of `Unavailable` indicates that the CMK is in a custom key store that was intentionally disconnected from its AWS CloudHSM cluster (p. 198) and attempts to reconnect it, if any, failed. While a CMK is unavailable, you can view and manage the CMK, but you cannot use it for cryptographic operations (p. 210).

To find the key state of a CMK, on the **Customer managed keys** page, view the **Status** field of the CMK. Or, use the DescribeKey operation and view the `KeyState` element in the response. For details, see Viewing keys (p. 29).

The CMKs in a disconnected custom key store will have a key state of `Unavailable` or `PendingDeletion`. CMKs that are scheduled for deletion from a custom key store have a `Pending Deletion` key state, even when the custom key store is disconnected from its AWS CloudHSM cluster. This allows you to cancel the scheduled key deletion without reconnecting the custom key store.

To fix an unavailable CMK, reconnect the custom key store (p. 198). After the custom key store is reconnected, the key state of the CMKs in the custom key store is automatically restored to its previous state, such as `Enabled` or `Disabled`. CMKs that are pending deletion remain in the `PendingDeletion` state. However, while the problem persists, enabling and disabling an unavailable CMK (p. 51) does not change its key state. The enable or disable action takes effect only when the key becomes available.

For help with failed connections, see How to fix a connection failure (p. 216).

How to fix a failing CMK

Problems with creating and using CMKs in custom key stores can be caused by a problem with your custom key store, its associated AWS CloudHSM cluster, the CMK, or its key material.

When a custom key store is disconnected from its AWS CloudHSM cluster, the key state of CMKs in the custom key store is `Unavailable`. All requests to create CMKs in a disconnected custom key store return a `CustomKeyStoreInvalidStateException` exception. All requests to encrypt, decrypt, re-encrypt, or generate data keys return a `KMSInvalidStateException` exception. To fix the problem, reconnect the custom key store (p. 198).

However, your attempts to use a custom key store CMK for cryptographic operations (p. 210) might fail even when its key state is `Enabled` and the connection status of the custom key store is `Connected`. This might be caused by any of the following conditions.

- The key material for the CMK might have been deleted from the associated AWS CloudHSM cluster. To investigate, find the key handle (p. 208) of the key material for a CMK and, if necessary, try to recover the key material (p. 219).
- All HSMs were deleted from the AWS CloudHSM cluster that is associated with the custom key store. To use a CMK in a custom key store in a cryptographic operation, its AWS CloudHSM cluster must contain at least one active HSM. To verify the number and state of HSMs in an AWS CloudHSM cluster, use the AWS CloudHSM console or the DescribeClusters operation. To add an HSM to the cluster, use the AWS CloudHSM console or the CreateHsm operation.
- The AWS CloudHSM cluster associated with the custom key store was deleted. To fix the problem, create a cluster from a backup that is related to the original cluster, such as a backup of the original cluster, or a backup that was used to create the original cluster. Then, edit the cluster ID (p. 196) in the custom key store settings. For instructions, see How to recover deleted key material for a CMK (p. 219).

How to fix a connection failure

If you try to connect a custom key store (p. 198) to its AWS CloudHSM cluster, but the operation fails, the connection status of the custom key store changes to `FAILED`. To find the status of a custom key store, view the **Status** column of the custom key store in the AWS Management Console or the `ConnectionState` element the DescribeCustomKeyStores response.

Alternatively, some connection attempts fail quickly due to easily detected cluster configuration errors. In this case, the **Status** or `ConnectionState` is still `DISCONNECTED`. These failures return an error message or exception that explains why the attempt failed. Review the exception description and cluster requirements (p. 191), fix the problem, update the custom key store (p. 196), if necessary, and try to connect again.

When the connection status is `FAILED`, run the DescribeCustomKeyStores operation and see the `ConnectionErrorCode` element in the response.

> **Note**
> When the connection status of a custom key store is `FAILED`, you must disconnect the custom key store (p. 198) before attempting to reconnect it. You cannot connect a custom key store with a `FAILED` connection status.

- `CLUSTER_NOT_FOUND` indicates that AWS KMS cannot find an AWS CloudHSM cluster with the specified cluster ID. This might occur because the wrong cluster ID was provided to an API operation or the cluster was deleted and not replaced. To fix this error, verify the cluster ID, such as by using the AWS CloudHSM console or the DescribeClusters operation. If the cluster was deleted, create a cluster from a recent backup of the original. Then, disconnect the custom key store (p. 198), edit the custom key store (p. 196) cluster ID setting, and reconnect the custom key store (p. 198) to the cluster.

- INSUFFICIENT_CLOUDHSM_HSMS indicates that the associated AWS CloudHSM cluster does not contain any HSMs. To connect, the cluster must have at least one HSM. To find the number of HSMs in the cluster, use the DescribeClusters operation. To resolve this error, add at least one HSM to the cluster. If you add multiple HSMs, it's best to create them in different Availability Zones.
- INTERNAL_ERROR indicates that AWS KMS could not complete the request due to an internal error. Retry the request. For ConnectCustomKeyStore requests, disconnect the custom key store before trying to connect again.
- INVALID_CREDENTIALS indicates that AWS KMS cannot log into the associated AWS CloudHSM cluster because it doesn't have the correct kmsuser account password. For help with this error, see How to fix invalid kmsuser credentials (p. 217).
- NETWORK_ERRORS usually indicates transient network issues. Disconnect the custom key store (p. 198), wait a few minutes, and try to connect again.
- SUBNET_NOT_FOUND indicates that at least one subnet in the AWS CloudHSM cluster configuration was deleted. If AWS KMS cannot find all of the subnets in the cluster configuration, attempts to connect the custom key store to the AWS CloudHSM cluster fail.

  To fix this error, create a cluster from a recent backup of the same AWS CloudHSM cluster. (This process creates a new cluster configuration with a VPC and private subnets.) Verify that the new cluster meets the requirements for a custom key store (p. 191), and note the new cluster ID. Then, to associate the new cluster with your custom key store, disconnect the custom key store (p. 198), change the cluster ID (p. 196) of the custom key store to the ID of the new cluster, and try to connect again.

  > **Tip**
  > To avoid resetting the kmsuser password (p. 217), use the most recent backup of the AWS CloudHSM cluster.

- USER_LOCKED_OUT indicates that the kmsuser crypto user (CU) account (p. 187) is locked out of the associated AWS CloudHSM cluster due to too many failed password attempts. For help with this error, see How to fix invalid kmsuser credentials (p. 217).

  To fix this error, disconnect the custom key store (p. 198) and use the changePswd command in cloudhsm_mgmt_util to change the kmsuser account password. Then, edit the kmsuser password setting (p. 196) for the custom key store, and try to connect again. For help, use the procedure described in the How to fix invalid kmsuser credentials (p. 217) topic.
- USER_LOGGED_IN indicates that the kmsuser CU account is logged into the associated AWS CloudHSM cluster. This prevents AWS KMS from rotating the kmsuser account password and logging into the cluster. To fix this error, log the kmsuser CU out of the cluster. If you changed the kmsuser password to log into the cluster, you must also and update the key store password value for the custom key store. For help, see How to log out and reconnect (p. 221).
- USER_NOT_FOUND indicates that AWS KMS cannot find a kmsuser CU account in the associated AWS CloudHSM cluster. To fix this error, create a kmsuser CU account (p. 192) in the cluster, and then update the key store password value (p. 196) for the custom key store. For help, see How to fix invalid kmsuser credentials (p. 217).

How to fix invalid kmsuser credentials

When you connect a custom key store (p. 198), AWS KMS logs into the associated AWS CloudHSM cluster as the kmsuser crypto user (p. 187) (CU). It remains logged in until the custom key store is disconnected. The DescribeCustomKeyStores response shows a ConnectionState of FAILED and ConnectionErrorCode value of INVALID_CREDENTIALS, as shown in the following example.

If you disconnect the custom key store and change the kmsuser password, AWS KMS cannot log into the AWS CloudHSM cluster with the credentials of the kmsuser CU account. As a result, all attempts to connect the custom key store fail. The DescribeCustomKeyStores response shows a

`ConnectionState` of `FAILED` and `ConnectionErrorCode` value of `INVALID_CREDENTIALS`, as shown in the following example.

```
$ aws kms describe-custom-key-stores --custom-key-store-name ExampleKeyStore
{
   "CustomKeyStores": [
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "ConnectionErrorCode": "INVALID_CREDENTIALS"
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleKeyStore",
      "TrustAnchorCertificate": "<certificate string appears here>",
      "CreationDate": "1.499288695918E9",
      "ConnectionState": "FAILED"
   ],
}
```

Also, after five failed attempts to log into the cluster with an incorrect password, AWS CloudHSM locks the user account. To log into the cluster, you must change the account password.

If AWS KMS gets a lockout response when it tries to log into the cluster as the `kmsuser` CU, the request to connect the custom key store fails. The DescribeCustomKeyStores response includes a `ConnectionState` of `FAILED` and `ConnectionErrorCode` value of `USER_LOCKED_OUT`, as shown in the following example.

```
$ aws kms describe-custom-key-stores --custom-key-store-name ExampleKeyStore
{
   "CustomKeyStores": [
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "ConnectionErrorCode": "USER_LOCKED_OUT"
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleKeyStore",
      "TrustAnchorCertificate": "<certificate string appears here>",
      "CreationDate": "1.499288695918E9",
      "ConnectionState": "FAILED"
   ],
}
```

To repair any of these conditions, use the following procedure.

1. Disconnect the custom key store (p. 198).
2. Run the DescribeCustomKeyStores operation and view the value of the `ConnectionErrorCode` element in the response.
   - If the `ConnectionErrorCode` value is `INVALID_CREDENTIALS`, determine the current password for the `kmsuser` account. If necessary, use the changePswd command in cloudhsm_mgmt_util to set the password to a known value.
   - If the `ConnectionErrorCode` value is `USER_LOCKED_OUT`, you must use the changePswd command in cloudhsm_mgmt_util to change the `kmsuser` password.
3. Edit the `kmsuser` password setting (p. 196) so it matches the current `kmsuser` password in the cluster. This action tells AWS KMS which password to use to log into the cluster. It does not change the `kmsuser` password in the cluster.
4. Connect the custom key store (p. 198).

How to delete orphaned key material

After scheduling deletion of a CMK from a custom key store, you might need to manually delete the corresponding key material from the associated cluster.

When you create a CMK in a custom key store, AWS KMS creates the CMK metadata in AWS KMS and generates the key material in the associated AWS CloudHSM cluster. When you schedule deletion of a CMK in a custom key store, after the waiting period, AWS KMS deletes the CMK metadata. Then AWS KMS makes a best effort to delete the corresponding key material from the cluster. AWS KMS does not attempt to delete key material from cluster backups.

If AWS KMS cannot delete the key material, such as when the custom key store is disconnected, AWS KMS writes an entry to your AWS CloudTrail logs. The entry includes the CMK ID, the AWS CloudHSM cluster ID, and the key handle of the key material.

To delete the key material from the associated AWS CloudHSM cluster, use a procedure like the following one. This example uses the AWS CLI and AWS CloudHSM command line tools, but you can use the AWS Management Console instead of the CLI.

1. Disconnect the custom key store, if it is not already disconnected, then log into the key_mgmt_util, as explained in How to disconnect and log in (p. 220).
2. Use the deleteKey command in key_mgmt_util to delete the key from the HSMs in the cluster.

   For example, this command deletes key `262162` from the HSMs in the cluster. The key handle is listed in the CloudTrail log entry.

```
Command: deleteKey -k 262162

        Cfm3DeleteKey returned: 0x00 : HSM Return: SUCCESS

        Cluster Error Status
        Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
        Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

3. Log out of key_mgmt_util and reconnect the custom key store as described in How to log out and reconnect (p. 221).

How to recover deleted key material for a CMK

If the key material for a customer master key is deleted, the CMK is unusable and all ciphertext that was encrypted under the CMK cannot be decrypted. This can happen if the key material for a CMK in a custom key store is deleted from the associated AWS CloudHSM cluster. However, it might be possible to recover the key material.

When you create a customer master key (CMK) in a custom key store, AWS KMS logs into the associated AWS CloudHSM cluster and creates the key material for the CMK. It also changes the password to a value that only it knows and remains logged in as long as the custom key store is connected. Because only the key owner, that is, the CU who created a key, can delete the key, it is unlikely that the key will be deleted from the HSMs accidentally.

However, if the key material for a CMK is deleted from the HSMs in a cluster, the CMK key state eventually changes to `UNAVAILABLE`. If you attempt to use the CMK for a cryptographic operation, the operation fails with a **KMSInvalidStateException** exception. Most importantly, any data that was encrypted under the CMK cannot be decrypted.

Under certain circumstances, you can recover deleted key material by creating a cluster from a backup that contains the key material. This strategy works only when at least one backup was created while the key existed and before it was deleted.

Use the following process to recover the key material.

1. Find a cluster backup that contains the key material. The backup must also contain all users and keys that you need to support the cluster and its encrypted data.

Use the DescribeBackups operation to list the backups for a cluster. Then use the backup timestamp to help you select a backup. To limit the output to the cluster that is associated with the custom key store, use the `Filters` parameter, as shown in the following example.

```
$ aws cloudhsmv2 describe-backups --filters clusterIds=<cluster ID>
{
    "Backups": [
        {
            "ClusterId": "cluster-1a23b4cdefg",
            "BackupId": "backup-9g87f6edcba",
            "CreateTimestamp": 1536667238.328,
            "BackupState": "READY"
        },
             ...
    ]
}
```

2. Create a cluster from the selected backup. Verify that the backup contains the deleted key and other users and keys that the cluster requires.
3. Disconnect the custom key store (p. 198) so you can edit its properties.
4. Edit the cluster ID (p. 196) of the custom key store. Enter the cluster ID of the cluster that you created from the backup. Because the cluster shares a backup history with the original cluster, the new cluster ID should be valid.
5. Reconnect the custom key store (p. 198).

How to log in as `kmsuser`

To create and manage the key material in the AWS CloudHSM cluster for your custom key store, AWS KMS uses the kmsuser crypto user (CU) account (p. 187). You create the `kmsuser` CU account (p. 191) in your cluster and provide its password to AWS KMS when you create your custom key store.

In general, AWS KMS manages the `kmsuser` account. However, for some tasks, you need to disconnect the custom key store, log into the cluster as the `kmsuser` CU, and use the cloudhsm_mgmt_util and key_mgmt_util command line tools.

> **Note**
> While a custom key store is disconnected, all attempts to create customer master keys (CMKs) in the custom key store or to use existing CMKs in cryptographic operations will fail. This action can prevent users from storing and accessing sensitive data.

This topic explains how to disconnect your custom key store and log in (p. 220) as `kmsuser`, run the AWS CloudHSM command line tool, and log out and reconnect your custom key store (p. 221).

Topics

- How to disconnect and log in (p. 220)
- How to log out and reconnect (p. 221)

How to disconnect and log in

Use the following procedure each time to need to log into an associated cluster as the `kmsuser` CU.

1. Disconnect the custom key store, if it is not already disconnected. You can use the AWS Management Console or AWS KMS API.

   While your custom key is connected, AWS KMS is logged in as the `kmsuser`. This prevents you from logging in as `kmsuser` or changing the `kmsuser` password.

For example, this command uses DisconnectCustomKeyStore to disconnect an example key store. Replace the example custom key store ID with a valid one.

```
$ aws kms disconnect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

2. Start cloudhsm_mgmt_util. Use the procedure described in Prepare to run cloudhsm_mgmt_util section of the *AWS CloudHSM User Guide*.
3. Log into cloudhsm_mgmt_util on the AWS CloudHSM cluster as a crypto officer (CO).

   For example, this command logs in as a CO named admin. Replace the example CO user name and password with valid values.

```
aws-cloudhsm>loginHSM CO admin <password>
loginHSM success on server 0(10.0.2.9)
loginHSM success on server 1(10.0.3.11)
loginHSM success on server 2(10.0.1.12)
```

4. Use the changePswd command to change the password of the `kmsuser` account to one that you know. (AWS KMS rotates the password when you connect your custom key store.) The password must consist of 7-32 alphanumeric characters. It is case-sensitive and cannot contain any special characters.

   For example, this command changes the `kmsuser` password to `tempPassword`.

```
aws-cloudhsm>changePswd CU kmsuser tempPassword

*************************CAUTION*********************************
This is a CRITICAL operation, should be done on all nodes in the
cluster. Cav server does NOT synchronize these changes with the
nodes on which this operation is not executed or failed, please
ensure this operation is executed on all nodes in the cluster.
*****************************************************************

Do you want to continue(y/n)?y
Changing password for kmsuser(CU) on 3 nodes
```

5. Log into key_mgmt_util or cloudhsm_mgmt_util as `kmsuser` using the password that you set. For detailed instructions, see Getting Started with cloudhsm_mgmt_util and Getting Started with key_mgmt_util. The tool that you use depends on your task.

   For example, this command logs into key_mgmt_util.

```
Command: loginHSM -u CU -s kmsuser -p tempPassword
Cfm3LoginHSM returned: 0x00 : HSM Return: SUCCESS

Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
Node id 2 and err state 0x00000000 : HSM Return: SUCCESS
```

How to log out and reconnect

1. Perform the task, then log out of the command line tool. If you do not log out, attempts to reconnect your custom key store will fail.

```
Command:  logoutHSM
Cfm3LogoutHSM returned: 0x00 : HSM Return: SUCCESS
```

```
Cluster Error Status
Node id 0 and err state 0x00000000 : HSM Return: SUCCESS
Node id 1 and err state 0x00000000 : HSM Return: SUCCESS
```

2. Edit the kmsuser password setting (p. 196) for the custom key store.

   This tells AWS KMS the current password for kmsuser in the cluster. If you omit this step, AWS KMS will not be able to log into the cluster as kmsuser, and all attempts to reconnect your custom key store will fail. You can use the AWS Management Console or the KeyStorePassword parameter of the UpdateCustomKeyStore operation.

   For example, this command tells AWS KMS that the current password is tempPassword. Replace the example password with the actual one.

```
$ aws kms update-custom-key-store --custom-key-store-id cks-1234567890abcdef0 --key-
store-password tempPassword
```

3. Reconnect the custom key store to AWS KMS. Replace the example custom key store ID with a valid one. During the connection process, AWS KMS changes the kmsuser password to a value that only it knows.

   The ConnectCustomKeyStore operation returns quickly, but the connection process can take an extended period of time. The initial response does not indicate the success of the connection process.

```
$ aws kms connect-custom-key-store --custom-key-store-id cks-1234567890abcdef0
```

4. Use the DescribeCustomKeyStores operation to verify that the custom key store is connected. Replace the example custom key store ID with a valid one.

   In this example, the connection state field shows that the custom key store is now connected.

```
$ aws kms describe-custom-key-stores --custom-key-store-
id cks-1234567890abcdef0
{
   "CustomKeyStores": [
      "CustomKeyStoreId": "cks-1234567890abcdef0",
      "CustomKeyStoreName": "ExampleKeyStore",
      "CloudHsmClusterId": "cluster-1a23b4cdefg",
      "TrustAnchorCertificate": "<certificate string appears here>",
      "CreationDate": "1.499288695918E9",
      "ConnectionState": "CONNECTED"
   ],
}
```

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.

Supported AWS Regions

AWS KMS supports VPC endpoints in all AWS Regions where both Amazon VPC and AWS KMS are available.

Topics

- Considerations for AWS KMS VPC endpoints (p. 223)
- Creating a VPC endpoint for AWS KMS (p. 223)
- Connecting to an AWS KMS VPC endpoint (p. 224)
- Using a VPC endpoint in a policy statement (p. 224)
- Auditing your VPC endpoint (p. 226)

Considerations for AWS KMS VPC endpoints

Before you set up an interface VPC endpoint for AWS KMS, review the Interface endpoint properties and limitations topic in the *Amazon VPC User Guide*.

AWS KMS supports the following features to support a VPC endpoint.

- You can use your VPC interface endpoint to call all AWS KMS API operations from your VPC.
- AWS KMS does not support creating a VPC interface endpoint to an AWS KMS FIPS endpoint.
- You can use AWS CloudTrail logs to audit your use of AWS KMS customer master keys (CMKs) through the VPC endpoint. For details, see Auditing your VPC endpoint (p. 226).

Creating a VPC endpoint for AWS KMS

You can create a VPC endpoint for AWS KMS by using the Amazon VPC console or the Amazon VPC API. For more information, see Creating an interface endpoint in the *Amazon VPC User Guide*.

To create a VPC endpoint for AWS KMS, use the following service name:

```
com.amazonaws.region.kms
```

For example, in the US West (Oregon) Region (`us-west-2`), the service name would be:

```
com.amazonaws.us-west-2.kms
```

To make it easier to use the VPC endpoint, you can enable a private DNS hostname for your VPC endpoint. If you do, the standard AWS KMS DNS hostname (`https://kms.<region>.amazonaws.com`) resolves to your VPC endpoint. Also, the AWS CLI and AWS 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.

You can enable private DNS hostnames when you create or modify your VPC endpoint. For more information about using private DNS hostnames, see Private DNS names for endpoint services in the *Amazon VPC User Guide*.

Connecting to an AWS KMS VPC endpoint

You can connect to AWS KMS through the VPC endpoint by using an AWS SDK, the AWS CLI or AWS Tools for PowerShell. 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 private DNS hostnames is enabled on the endpoint, you do not need to specify the VPC endpoint URL in your CLI commands or application configuration. For more information about using private DNS hostnames, see Private DNS names for endpoint services in the *Amazon VPC User Guide*.

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. 99).
> 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:sourceVpce": "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": "*",
        }
    ]
}
```

Auditing your VPC endpoint

When a request to AWS KMS uses a VPC endpoint, the VPC endpoint ID appears in the AWS CloudTrail log (p. 303) 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"
}
```

Using hybrid post-quantum TLS with AWS KMS

AWS Key Management Service (AWS KMS) now supports a hybrid post-quantum key exchange option for the Transport Layer Security (TLS) network encryption protocol. You can use this TLS option when you connect to KMS API endpoints. We're offering this feature before post-quantum algorithms are standardized so you can begin testing the effect of these key exchange protocols on AWS KMS calls. These optional hybrid post-quantum key exchange features are at least as secure as the TLS encryption we use today and are likely to provide additional security benefits. However, they affect latency and throughput compared to the classic key exchange protocols in use today.

The data that you send to AWS Key Management Service (AWS KMS) is protected in transit by the encryption provided by a Transport Layer Security (TLS) connection. The classic cipher suites that AWS KMS supports for TLS sessions make brute force attacks on the key exchange mechanisms infeasible with current technology. However, if large-scale quantum computing becomes practical in the future, the classic cipher suites used in TLS key exchange mechanisms will be susceptible to these attacks. If you're developing applications that rely on the long-term confidentiality of data passed over a TLS connection, you should consider a plan to migrate to post-quantum cryptography before large-scale quantum computers become available for use. AWS is working to prepare for this future, and we want you to be well-prepared, too.

To protect data encrypted today against potential future attacks, AWS is participating with the cryptographic community in the development of quantum-resistant or *post-quantum* algorithms. We've implemented hybrid post-quantum key exchange cipher suites in AWS KMS endpoints. These hybrid cipher suites, which combine classic and post-quantum elements, ensure that your TLS connection is at least as strong as it would be with classic cipher suites.

These hybrid cipher suites are available for use on your production workloads in most AWS Regions (p. 228). However, because the performance characteristics and bandwidth requirements of hybrid cipher suites are different from those of classic key exchange mechanisms, we recommend that you test them on your AWS KMS API calls (p. 231) under different conditions.

Feedback

As always, we welcome your feedback and participation in our open-source repositories. We'd especially like to hear how your infrastructure interacts with this new variant of TLS traffic.

- To provide feedback on this topic, use the **Feedback** link in the lower right corner of this page. You can also create an issue or a pull request in the aws-kms-developer-docs repository in GitHub.
- We're developing these hybrid cipher suites in open source in the s2n repository on GitHub. To provide feedback on the usability of the cipher suites, or share novel test conditions or results, create an issue in the s2n repository.
- We're writing code samples for using hybrid post-quantum TLS with AWS KMS in the aws-kms-pq-tls-example GitHub repository. To ask questions or share ideas about configuring your HTTP client or AWS KMS client to use the hybrid cipher suites, create an issue in the `aws-kms-pq-tls-example` repository.

Supported AWS Regions

Post-quantum TLS for AWS KMS is available in all AWS Regions except for AWS GovCloud (US-East), AWS GovCloud (US-West), China (Beijing), and China (Ningxia).

For a list of AWS KMS endpoints for each AWS Region, see AWS Key Management Service Endpoints and Quotas in the *Amazon Web Services General Reference*. For information about FIPS endpoints, see AWS Service Endpoints in the *Amazon Web Services General Reference*..

About hybrid post-quantum key exchange in TLS

AWS KMS supports hybrid post-quantum key exchange cipher suites. You can use the AWS SDK for Java 2.x and AWS common runtime to configure an HTTP client to use these cipher suites. Then, whenever you connect to a AWS KMS endpoint with your HTTP client, the hybrid cipher suites are used.

This HTTP client uses s2n, which is an open source implementation of the TLS protocol. s2n includes the pq-crypto module, which includes implementations of hybrid post-quantum algorithms for encryption in transit.

The hybrid cipher suites in s2n are implemented only for key exchange, not for direct data encryption. During *key exchange*, the client and server calculate the key they will use to encrypt and decrypt the data on the wire.

The algorithms that s2n uses are a *hybrid* that combines Elliptic Curve Diffie-Hellman (ECDH), a classic key exchange algorithm used today in TLS, with Bit Flipping Key Encapsulation (BIKE), a proposed post-quantum algorithm. This mechanism uses each of the algorithms independently to generate a key. Then it combines the two keys cryptographically. With s2n, you can configure an HTTP client with a *cipher preference* that places ECDH with BIKE first in the preference list. Classic key exchange algorithms are included in the preference list to ensure compatibility, but they are lower in the preference order.

If ongoing research reveals that the BIKE algorithm lacks the anticipated post-quantum strength, the hybrid key is still at least as strong as the single ECDH key currently in use. The National Institute for Standards and Technology (NIST) has not yet standardized post-quantum algorithms. They are still in the process of evaluating candidate approaches. Until that process is complete, we recommend using hybrid algorithms, rather than using post-quantum algorithms alone.

Using hybrid post-quantum TLS with AWS KMS

You can use hybrid post-quantum TLS for your calls to AWS KMS. When setting up your HTTP client test environment, be aware of the following information:

Encryption in Transit

The hybrid cipher suites in s2n are used only for encryption in transit. They protect your data while it is traveling from your client to the AWS KMS endpoint. AWS KMS does not use these cipher suites to encrypt data under customer master keys (CMKs).

Instead, when AWS KMS encrypts your data under CMKs, it uses symmetric cryptography with 256-bit keys and the Advanced Encryption Standard in Galois Counter Mode (AES-GCM) algorithm, which is already quantum resistant. Theoretical future, large-scale quantum computing attacks on ciphertexts created under 256-bit AES-GCM keys reduce the effective security of the key to 128 bits. This security level is sufficient to make brute force attacks on AWS KMS ciphertexts infeasible.

Supported Systems

Use of the hybrid cipher suites in s2n is currently supported only on Linux systems. In addition, these cipher suites are supported only in SDKs that support the AWS common runtime, such as the AWS SDK for Java 2.x. For an example, see How to configure hybrid post-quantum TLS (p. 230).

AWS KMS Endpoints

When using the hybrid cipher suites, use the standard AWS KMS endpoint. The hybrid cipher suites in s2n are not compatible with the FIPS 140-2 validated endpoints for AWS KMS. Post-quantum algorithms are not allowed in a validated cryptographic module.

When you configure a HTTP client with the hybrid post-quantum cipher preference in s2n, the post-quantum ciphers are first in the cipher preference list. However, the preference list includes the classic, non-hybrid ciphers lower in the preference order for compatibility. If you were to use this cipher preference with an AWS KMS FIPS 140-2 validated endpoint, s2n negotiates a classic, non-hybrid key exchange cipher.

For a list of AWS KMS endpoints for each AWS Region, see AWS Key Management Service Endpoints and Quotas in the *Amazon Web Services General Reference*. For information about FIPS endpoints, see AWS Service Endpoints in the *Amazon Web Services General Reference*.

Expected Performance

Our early benchmark testing shows that the hybrid cipher suites in s2n are slower than classic TLS cipher suites. The effect varies based on the network profile, CPU speed, the number of cores, and your call rate. For performance test results, see Post-quantum TLS now supported in AWS KMS.

How to configure hybrid post-quantum TLS

In this procedure, you get the `aws-crt-dev-preview` (developer preview) branch of the AWS SDK for Java 2.x from its GitHub repository. Next, you build an AWS common runtime client and add the AWS common runtime to your dependencies. Then you can configure an HTTP client that uses the hybrid post-quantum cipher preference and create an AWS KMS client that uses the HTTP client.

To see a complete working examples of configuring and using hybrid post-quantum TLS with AWS KMS, see the aws-kms-pq-tls-example repository.

1. Clone the developer preview branch (`aws-crt-dev-preview`) of the AWS SDK for Java 2.x.

   The AWS SDK for Java 2.x is being developed in the aws-sdk-java-v2 GitHub repository.

   > **Note**
   > The `aws-crt-dev-preview` branch is a beta release. Your use of this library is subject to Section 1.10 ("Beta Service Participation") of the AWS Service Terms.

   ```
   $ git clone git@github.com:aws/aws-sdk-java-v2.git --branch aws-crt-dev-preview
   ```

2. Install and build the AWS common runtime client (`aws-crt-client`) JAR.

   ```
   $ cd aws-sdk-java-v2
   $ mvn install -Pquick
   ```

3. Add the AWS common runtime client to your Maven dependencies. We recommend using the latest available version.

   For example, this statement adds version `2.10.7-SNAPSHOT` of the AWS common runtime client to your Maven dependencies.

   ```
   <dependency>
       <groupId>software.amazon.awssdk</groupId>
       <artifactId>aws-crt-client</artifactId>
       <version>2.10.7-SNAPSHOT</version>
   </dependency>
   ```

4. To enable the hybrid post-quantum cipher suites, add the AWS SDK for Java 2.x to your project and initialize it. Then enable the hybrid cipher suites as shown in the following example.

This code ensures that you are working on a system that supports the hybrid cipher suite. The code then creates an HTTP client with the `TLS_CIPHER_KMS_PQ_TLSv1_0_2019_06` cipher preference that prioritizes the ECDH with BIKE hybrid cipher suite. Finally, it creates an AWS KMS client that uses the HTTP client for data transmission.

This code uses the AWS KMS asynchronous client, `KmsAsyncClient`, which calls AWS KMS asynchronously. For information about this client, see the KmsAsyncClient Javadoc.

After this code completes, your AWS KMS API requests on the AWS KMS asynchronous client use the hybrid cipher suite for TLS.

```
// Check platform support
if(!TLS_CIPHER_KMS_PQ_TLSv1_0_2019_06.isSupported()){
    throw new RuntimeException("Hybrid post-quantum cipher suites are not supported on
 this platform");
}

// Configure HTTP client
SdkAsyncHttpClient awsCrtHttpClient = AwsCrtAsyncHttpClient.builder()
          .tlsCipherPreference(TLS_CIPHER_KMS_PQ_TLSv1_0_2019_06)
          .build();

// Create the KMS async client
KmsAsyncClient kmsAsync = KmsAsyncClient.builder()
         .httpClient(awsCrtHttpClient)
         .build();
```

5. Test your KMS calls with post-quantum TLS.

   When you call AWS KMS API operations on the configured AWS KMS client, your calls are transmitted to the AWS KMS endpoint using hybrid post-quantum TLS. To test your configuration, run a simple KMS API call, such as `ListKeys`.

```
ListKeysReponse keys = kmsAsync.listKeys().get();
```

Testing hybrid post-quantum TLS with AWS KMS

Consider running the following tests with hybrid cipher suites on your applications that call AWS KMS.

- Run load tests and benchmarks. The hybrid cipher suites perform differently than traditional key exchange algorithms. You might need to adjust your connection timeouts to allow for the longer handshake times. If you're running inside an AWS Lambda function, extend the execution timeout setting.
- Try connecting from different locations. Depending on the network path your request takes, you might discover that intermediate hosts, proxies, or firewalls with deep packet inspection (DPI) block the request. This might result from using the new cipher suites in the ClientHello part of the TLS handshake, or from the larger key exchange messages. If you have trouble resolving these issues, work with your security team or IT administrators to update the relevant configuration and unblock the new TLS cipher suites.

Learn more about post-quantum TLS in AWS KMS

For more information about using hybrid post-quantum TLS in AWS KMS, see the following resources.

- For more information about using hybrid post-quantum TLS cipher suites with AWS KMS, including performance data, see Post-quantum TLS now supported in AWS KMS.
- For information about the AWS SDK for Java 2.x, see the AWS SDK for Java 2.x Developer Guide and the AWS SDK for Java 2.x released blog post.
- For information about s2n, see Introducing s2n, a New Open Source TLS Implementation and Using s2n.
- For information about the post-quantum cryptography project at the National Institute for Standards and Technology (NIST), see Post-Quantum Cryptography.
- For technical information about using hybrid post-quantum key exchange in TLS, see Hybrid Post-Quantum Key Encapsulation Methods (PQ KEM) for Transport Layer Security 1.2 (TLS).

Key state: Effect on your CMK

Customer master keys (CMKs) are always in one of the following states: `Enabled`, `Disabled`, `PendingImport`, `PendingDeletion`, or `Unavailable`.

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

Symmetric CMKs can be in the `Enabled`, `Disabled`, `PendingImport`, `PendingDeletion`, or `Unavailable` states. Asymmetric CMKs can be in the `Enabled`, `Disabled`, or `PendingDeletion` key state.

The `Unavailable` state applies only to a CMK in a custom key store (p. 184). A CMK in a custom key store is `Unavailable` when the custom key store is intentionally disconnected from its AWS CloudHSM cluster. You can view and manage unavailable CMKs, but you cannot use them in cryptographic operations.

The following API operations do not appear in the table because they do not use an existing CMK.

- ConnectCustomKeyStore
- CreateCustomKeyStore
- CreateKey
- DeleteCustomKeyStore
- DescribeCustomKeyStores
- DisconnectCustomKeyStore
- GenerateRandom
- UpdateCustomKeyStore

API	Enabled	Disabled	Pending import	Pending deletion	Unavailable
CancelKeyDeletion	✗ [4]	✗ [4]	✗ [4]	✓	✗ [4], [13]
CreateAlias	✓	✓	✓	✗ [3]	✓
CreateGrant	✓	✗ [1]	✗ [5]	✗ [2] or [3]	✓
Decrypt	✓	✗ [1]	✗ [5]	✗ [2] or [3]	✗ [11]

API	Enabled	Disabled	Pending import	Pending deletion	Unavailable
DeleteAlias	✓	✓	✓	✓	✓
DeleteImportedKeyMaterial	? [9]	? [9]	✓ (No effect)	⊗ [9]	⊗ [9]
DescribeKey	✓	✓	✓	✓	✓
DisableKey	✓	✓	⊗ [5]	⊗ [3]	✓ [12]
DisableKeyRotation	? [7]	⊗ [1] or [7]	⊗ [6]	⊗ [3] or [7]	⊗ [7]
EnableKey	✓	✓	⊗ [5]	⊗ [3]	✓ [12]
EnableKeyRotation	? [7]	⊗ [1] or [7]	⊗ [6]	⊗ [3] or [7]	⊗ [7]
Encrypt	✓	⊗ [1]	⊗ [5]	⊗ [2] or [3]	⊗ [11]
GenerateDataKey	✓	⊗ [1]	⊗ [5]	⊗ [2] or [3]	⊗ [11]
GenerateDataKeyPair	✓	⊗ [1]	⊗ [5]	⊗ [2] or [3]	⊗ [11]
GenerateDataKeyPairWithoutPlaintext	✓	⊗ [1]	⊗ [5]	⊗ [2] or [3]	⊗ [11]

API	Enabled	Disabled	Pending import	Pending deletion	Unavailable
GenerateDataKeyWithoutPlaintext	✓	✗ [1]	✗ [5]	✗ [2] or [3]	✗ [11]
GetKeyPolicy	✓	✓	✓	✓	✓
GetKeyRotationStatus	? [7]	? [7]	✗ [6]	? [7]	✗ [7]
GetParametersForImport	? [9]	? [9]	✓	✗ [8] or [9]	✗ [9]
GetPublicKey	✓	✗ [1]	N/A	✗ [2] or [3]	N/A
ImportKeyMaterial	? [9]	? [9]	✓	✗ [8] or [9]	✗ [9]
ListAliases	✓	✓	✓	✓	✓
ListGrants	✓	✓	✓	✓	✓
ListKeyPolicies	✓	✓	✓	✓	✓
ListKeys	✓	✓	✓	✓	✓
ListResourceTags	✓	✓	✓	✓	✓
ListRetirableGrants	✓	✓	✓	✓	✓
PutKeyPolicy	✓	✓	✓	✓	✓
ReEncrypt	✓	✗ [1]	✗ [5]	✗ [2] or [3]	✗ [11]

API	Enabled	Disabled	Pending import	Pending deletion	Unavailable
RetireGrant	✓	✓	✓	✓	✓
RevokeGrant	✓	✓	✓	✓	✓
ScheduleKeyDeletion	✓	✓	✓	✗ [3]	✓
Sign	✓	✗ [1]	N/A	✗ [2] or [3]	N/A
TagResource	✓	✓	✓	✗ [3]	✓
UnTagResource	✓	✓	✓	✗ [3]	✓
UpdateAlias	✓	✓	✓	? [10]	✓
UpdateKeyDescription	✓	✓	✓	✗ [3]	✓
Verify	✓	✗ [1]	N/A	✗ [2] or [3]	N/A

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 or is in a custom key store: `UnsupportedOperationException`.
- [8] If the CMK has imported key material: `KMSInvalidStateException`
- [9] If the CMK cannot or 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.`
- [11] `KMSInvalidStateException: <CMK ARN> is unavailable.` You cannot perform this operation on an unavailable CMK.
- [12] The operation succeeds, but the key state of the CMK does not change until it becomes available.
- [13] While a CMK in a custom key store is pending deletion, its key state remains `PendingDeletion` even if the CMK becomes unavailable. This allows you to cancel deletion of the CMK at any time during the waiting period.

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.

> **Important**
> AWS services that are integrated with AWS KMS use symmetric CMKs to encrypt your data. These services do not support encryption with asymmetric CMKs. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

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*.

> **Important**
> AWS CloudTrail and Amazon S3 support only symmetric customer master keys (p. 142) (CMKs). You cannot use an asymmetric CMK (p. 142) to encrypt your CloudTrail Logs. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

Topics

- Understanding when your CMK is used (p. 238)
- Understanding how often your CMK is used (p. 242)

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. 274) 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. 239)

- CloudTrail puts a log file into your S3 bucket (p. 240)
- You get an encrypted log file from your S3 bucket (p. 241)

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. 19):

- 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` operation (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",
    "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. 19):

- 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` operation (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",
    "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. 19):

- 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` operation (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", (1)
    "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", (2)
  "eventName": "Decrypt", (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)
    }
  },
  "responseElements": null,
  "requestID": "16a0590a-88ba-11e5-b406-436f15c3ac01",
  "eventID": "9525bee7-5145-42b0-bed5-ab7196a16daa",
  "readOnly": true,
  "resources": [{
    "ARN": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab", (6)
    "accountId": "111122223333"
  }],
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}
```

Understanding how often your CMK is used

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

When you encrypt CloudTrail log files with AWS KMS–Managed Keys (SSE-KMS), each time CloudTrail puts a log file into your S3 bucket (p. 240) 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 calls that result from `PUT` requests. They do not count the decrypt calls to AWS KMS that result from `GET` requests when you get an encrypted log file from your S3 bucket.

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 the encryption at rest server-side encryption feature.

With *encryption at rest*, DynamoDB transparently encrypts all customer data in a DynamoDB 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 your table, DynamoDB decrypts the table data transparently. You do not need to change your applications to use or manage encrypted tables.

Encryption at rest also protects DynamoDB streams, global tables, and backups whenever these objects are saved to durable media. Statements about tables in this topic apply to these objects, too.

All DynamoDB tables are encrypted. There is no option to enable or disable encryption for new or existing tables. By default, all tables are encrypted under an AWS owned customer master key (p. 4) (CMK) in the DynamoDB service account. However, you can select an option to encrypt some or all of your tables under a customer managed CMK (p. 4) or the AWS managed CMK (p. 4) for DynamoDB in your account.

> **Note**
> Before November 2018, encryption at rest was an optional feature that supported only the AWS managed CMK for DynamoDB. If you enabled encryption at rest on any of your DynamoDB tables, they will continue to be encrypted under the AWS managed CMK unless you use the AWS Management Console or UpdateTable operation to switch to a customer managed CMK or an AWS owned CMK.

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 table data before submitting it to 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 the DynamoDB Encryption Client along with encryption at rest. To help you decide if this strategy is right 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. 243)
- Authorizing use of your CMK (p. 245)
- DynamoDB encryption context (p. 249)
- Monitoring DynamoDB interaction with AWS KMS (p. 249)

Using CMKs and data keys

The DynamoDB encryption at rest feature uses an AWS KMS customer master key (CMK) and a hierarchy of data keys to protect your table data. DynamoDB uses the same key hierarchy to protect DynamoDB streams, global tables, and backups when they are written to durable media.

Customer master key (CMK)

Encryption at rest protects your DynamoDB tables under an AWS KMS customer master key (CMK). By default, it uses an AWS owned CMK (p. 4), a multi-tenant key that is created and managed in a DynamoDB service account. But DynamoDB supports an option to encrypt some or all of your tables under a customer managed CMKs (p. 4) or the AWS managed CMK (p. 4) for DynamoDB (`aws/dynamodb`) in your AWS account. You can select the CMK for a table when you create or update the table, and you can make a different choice for each table.

> **Important**
> DynamoDB supports only symmetric CMKs (p. 142). You cannot use an asymmetric CMK (p. 142) to encrypt your DynamoDB tables. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

You can choose your CMK in the DynamoDB console or by using DynamoDB API. When you select a CMK for a table, the local and global secondary indexes, streams, and backups are encrypted with the same CMK. However, you cannot use a customer managed CMK to encrypt DynamoDB global table replicas. To encrypt replicas, use an AWS owned CMK or an AWS managed CMK.

You can change the CMK for a table at any time, either in the DynamoDB console or by using the UpdateTable operation. The process of switching keys is seamless and does not require downtime or degrade service.

Use a customer managed CMK to get the following features:

- You create and manage the CMK, including setting the key policies (p. 59), IAM policies (p. 77) and grants (p. 126) to control access to the CMK. You can enable and disable (p. 51) the CMK, enable and disable automatic key rotation (p. 154), and delete the CMK (p. 172) when it is no longer in use.
- You can use a customer managed CMK with imported key material (p. 159) or a customer managed CMK in a custom key store (p. 184) that you own and manage.
- You can audit the encryption and decryption of your DynamoDB table by examining the DynamoDB API calls to AWS KMS in AWS CloudTrail logs (p. 249).

Use the AWS managed CMK if you need any of the following features:

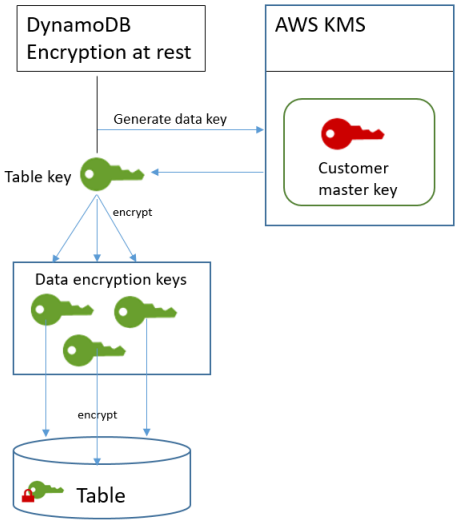
- You can view the CMK (p. 29) and view its key policy (p. 71). (You cannot change the key policy.)
- You can audit the encryption and decryption of your DynamoDB table by examining the DynamoDB API calls to AWS KMS in AWS CloudTrail logs (p. 249).

However, the AWS owned CMK is free of charge and its use does not count against AWS KMS resource or request quotas (p. 370). Customer managed CMKs and AWS managed CMKs incur a charge for each API call and AWS KMS quotas apply to these CMKs.

Table keys

DynamoDB uses the CMK for the table to generate and encrypt a unique data key (p. 5) for the table, known as the *table key*. The table key persists for the lifetime of the encrypted table.

The table key is used as a key encryption key. DynamoDB uses this table key to protect data encryption keys that are used to encrypt the table data. 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 first access an encrypted table, DynamoDB sends a request to AWS KMS to use the CMK 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. Then, it stores the encrypted keys with the encrypted data so they are available to decrypt the table data on demand.

If you change the CMK for your table, DynamoDB generates a new table key. Then, it uses the new table key to re-encrypt the data encryption keys.

Table key caching

To avoid calling AWS KMS for every DynamoDB operation, DynamoDB caches the plaintext table keys for each connection 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 made to the access policies of the CMK in AWS KMS or AWS Identity and Access Management (IAM) since the last request to decrypt the table key.

Authorizing use of your CMK

If you use a customer managed CMK (p. 4) or the AWS managed CMK (p. 4) in your account to protect your DynamoDB table, the policies on that CMK must give DynamoDB permission to use it on your behalf. The authorization context on the AWS managed CMK for DynamoDB includes its key policy and grants that delegate the permissions to use it.

You have full control over the policies and grants on a customer managed CMK. Because the AWS managed CMK is in your account, you can view its policies and grants. But, because it is managed by AWS, you cannot change the policies.

DynamoDB does not need additional authorization to use the default AWS owned CMK (p. 3) to protect the DynamoDB tables in your AWS account.

Topics

- AWS managed CMK key policy (p. 246)
- Customer managed CMK key policy (p. 247)

- Using grants to authorize DynamoDB (p. 248)

AWS managed CMK key policy

When DynamoDB uses the AWS managed CMK (p. 4) for DynamoDB (`aws/dynamodb`) in cryptographic operations, it does so on behalf of the user who is accessing the DynamoDB resource. The key policy on the AWS managed CMK gives all users in the account permission to use the AWS managed CMK for specified operations. But permission is granted only when DynamoDB makes the request on the user's behalf. The ViaService condition (p. 122) in the key policy does not allow any user to use the AWS managed CMK unless the request originates with the DynamoDB service.

This key policy, like the policies of all AWS managed keys, is established by AWS. You cannot change it, but you can view it at any time. For details, see Viewing a key policy (p. 71).

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

- Allow users in the account to use the AWS managed CMK for DynamoDB in cryptographic operations when the request comes from DynamoDB on their behalf. The policy also allows users to create grants (p. 248) for the CMK.
- Allows the AWS account root user to view the properties of the AWS managed CMK for DynamoDB and to revoke the grant that allows DynamoDB to use the CMK. DynamoDB uses grants (p. 248) for ongoing maintenance operations.
- Allows DynamoDB to perform read-only operations to find the AWS managed CMK for DynamoDB 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" : "*"
  } ]
}
```

Customer managed CMK key policy

When you select a customer managed CMK (p. 4) to protect a DynamoDB table, DynamoDB gets permission to use the CMK on behalf of the principal who makes the selection. That principal, a user or role, must have the permissions on the CMK that DynamoDB requires. You can provide these permissions in a key policy (p. 59), an IAM policy (p. 77), or a grant (p. 126).

At a minimum, DynamoDB requires the following permissions on a customer managed CMK:

- kms:Encrypt
- kms:Decrypt
- kms:ReEncrypt* (for kms:ReEncryptFrom and kms:ReEncryptTo)
- kms:GenerateDataKey* (for kms:GenerateDataKey and kms:GenerateDataKeyWithoutPlaintext)
- kms:DescribeKey
- kms:CreateGrant

For example, the following example key policy provides only the required permissions. The policy has the following effects:

- Allows DynamoDB to use the CMK in cryptographic operations and create grants, but only when it is acting on behalf of principals in the account who have permission to use DynamoDB. If the principals specified in the policy statement don't have permission to use DynamoDB, the call fails, even when it comes from the DynamoDB service.
- The kms:ViaService (p. 122) condition key allows the permissions only when the request comes from DynamoDB on behalf of the principals listed in the policy statement. These principals can't call these operations directly. Note that the `kms:ViaService` value, `dynamodb.*.amazonaws.com`, has an asterisk (*) in the Region position. DynamoDB requires the permission to be independent of any particular AWS Region so it can make cross-Region calls to support DynamoDB global tables.
- Gives the CMK administrators (users who can assume the `db-team` role) read-only access to the CMK and permission to revoke grants, including the grants that DynamoDB requires (p. 248) to protect the table.
- Gives DynamoDB read-only access to the CMK. In this case, DynamoDB can call these operations directly. It does not have to act on behalf of an account principal.

Before using an example key policy, replace the example principals with actual principals from your AWS account.

```
{
  "Id": "key-policy-dynamodb",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid" : "Allow access through Amazon DynamoDB for all principals in the account that
 are authorized to use Amazon DynamoDB",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:user/db-lead"},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
```

```
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
         "StringLike": {
           "kms:ViaService" : "dynamodb.*.amazonaws.com"
         }
      }
    },
    {
      "Sid":  "Allow administrators to view the CMK and revoke grants",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/db-team"
       },
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
      ],
      "Resource": "*"
    },
    {
      "Sid":  "Allow DynamoDB to get information about the CMK",
      "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 a customer managed CMK or the AWS managed CMK for DynamoDB (aws/dynamodb). To view the grants on a CMK in your account, use the ListGrants operation. DynamoDB does not need grants, or any additional permissions, to use the AWS owned CMK (p. 4) to protect your table.

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. 243).

Each grant is specific to a table. If the account includes multiple tables encrypted under the same CMK, there is a grant of each type for each table. The grant is constrained by the DynamoDB encryption context (p. 249), which includes the table name and the AWS account ID, and it includes permission to the retire the grant if it is no longer needed.

To create the grants, DynamoDB must have permission to call `CreateGrant` on behalf of the user who created the encrypted table. For AWS managed CMKs, DynamoDB gets `kms:CreateGrant` permission from the key policy (p. 246), which allows account users to call CreateGrant on the CMK only when DynamoDB makes the request on an authorized user's behalf.

The key policy can also allow the account to revoke the grant on the CMK. 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. 19) 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 AWS KMS cryptographic operations. If you use a customer managed CMK (p. 4) or an AWS managed CMK (p. 4) to protect your DynamoDB table, you can use the encryption context to identify use of the CMK 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 the encryption context to constrain the grants (p. 248) that allow access to the customer managed CMK or AWS managed CMK in your account and region.

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

If you use a customer managed CMK (p. 4) or an AWS managed CMK (p. 4) to protect your DynamoDB tables, you can use AWS CloudTrail 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 the CMK you selected 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 CMK for the table.

The event that records the `GenerateDataKey` operation is similar to the following example event. The user is the DynamoDB service account. The parameters include the Amazon Resource Name (ARN) of the CMK, a key specifier that requires a 256-bit key, and the encryption context (p. 249) 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 CMK for the table.

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. 249) that identifies the table and the AWS account. AWS KMS derives the ID of the CMK from the ciphertext.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE: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": "AROAIGDT3HGFQZX4RY6RU",
                "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

When you use a customer managed CMK (p. 4) or an AWS managed CMK (p. 4) to protect your DynamoDB table, DynamoDB uses grants (p. 248) to allow the service to perform continuous data protection and maintenance and durability tasks. These grants are not required on AWS owned CMKs (p. 4).

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 CMK for the table, 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. 249).

```
{
    "eventVersion": "1.05",
    "userIdentity":
    {
```

```
        "type": "AssumedRole",
        "principalId": "AROAIGDTESTANDEXAMPLE: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": "AROAIGDTESTANDEXAMPLE",
                "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-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
        "retiringPrincipal": "dynamodb.us-west-2.amazonaws.com",
        "constraints": {
            "encryptionContextSubset": {
                "aws:dynamodb:tableName": "Books",
                "aws:dynamodb:subscriberId": "111122223333"
            }
        },
        "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-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333"
}
```

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. 253)
- Using CMKs and data keys (p. 253)
- Amazon EBS encryption context (p. 254)
- Detecting Amazon EBS failures (p. 254)
- Using AWS CloudFormation to create encrypted Amazon EBS volumes (p. 255)

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. 4) for Amazon EBS in your account. However, you can specify a customer managed CMK (p. 4).

> **Important**
> Amazon EBS supports only symmetric CMKs (p. 142). You cannot use an asymmetric CMK (p. 142) to encrypt an Amazon EBS volume. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

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 (p. 19) 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 its key state (p. 233) 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. 29) 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. 51).
- If the CMK's key state is pending import, import key material (p. 160).
- If the CMK's key state is pending deletion, cancel key deletion (p. 174).

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. 255)
- Decrypting the input file (p. 256)
- Encrypting the output file (p. 257)
- HLS content protection (p. 258)
- Elastic Transcoder encryption context (p. 258)

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 a symmetric (p. 142) AWS KMS customer master key (p. 3) (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 (p. 4) for Amazon S3, a master key that is owned by your account, but managed by AWS
- Any symmetric (p. 142) customer managed CMK (p. 4) that you create by using AWS KMS

> **Important**
> For both client-side and server-side encryption, Elastic Transcoder supports only symmetric CMKs (p. 142). You cannot use an asymmetric CMK (p. 142) to encrypt your Elastic Transcoder files. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

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 protect your input file by using the AWS managed CMK for Amazon S3 in your account or a customer managed CMK, 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 decrypt the media file. The process is opaque to the user.
S3-AWS-KMS	Default	Amazon S3 uses a data key encrypted by the default AWS managed CMK for Amazon S3 in your account to encrypt the media file.
S3-AWS-KMS	Custom (with ARN)	Amazon S3 uses a data key encrypted by the specified customer managed CMK to encrypt the media file.

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

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

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

Encrypting the output file

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

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

When you specify that the AWS managed CMK for Amazon S3 in your account or a customer managed CMK is 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 under 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**
> AWS never stores your private encryption keys. Therefore, it is important that you manage your keys safely and securely. 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.

When you enable *HLS content protection*, each media segment is encrypted using a 128-bit AES encryption key. When the content is viewed, during the playback process, the player downloads the key and decrypts the media segments.

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. 19) 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.

> **Important**
> Amazon EMR supports only symmetric CMKs (p. 142). You cannot use an asymmetric CMK (p. 142) to encrypt data at rest in an Amazon EMR cluster. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

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 Management Guide*.

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

Topics

- Encrypting data on the EMR file system (EMRFS) (p. 259)
- Encrypting data on the storage volumes of cluster nodes (p. 261)
- Encryption context (p. 262)

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 Encryption Options in the *Amazon EMR Management 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. 260).

- 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. 260).

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 symmetric customer managed CMK that you create. With CSE-KMS, you must choose a symmetric customer managed CMK that you create. When you choose a customer managed CMK, you must ensure that your Amazon EMR cluster has permission to use the CMK. For more information, see Using AWS KMS Customer Master Keys (CMKs) for Encryption in the *Amazon EMR Management Guide*.

For both SSE-KMS and CSE-KMS, the CMK you choose is the master key in an envelope encryption (p. 18) 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. 260)
- Process for encrypting data on EMRFS with CSE-KMS (p. 260)

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. 262).
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 an encryption context (p. 19).
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. 262).
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 (p. 19).
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 customer managed CMK (p. 4) that you create; you cannot use an AWS managed CMK (p. 4). If you choose a customer managed CMK, you must ensure that your Amazon EMR cluster has permission to use the CMK. For more information, see Using AWS KMS Customer Master Keys (CMKs) for Encryption in the *Amazon EMR Management 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 an encryption context (p. 19) 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.

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

Encryption context for EMRFS encryption with SSE-KMS

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

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

Encryption context for EMRFS encryption with CSE-KMS

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

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

Encryption context for local disk encryption with LUKS

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

How Amazon Redshift uses AWS KMS

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

Topics

- Amazon Redshift encryption (p. 262)
- Encryption context (p. 263)

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.

The master key encrypts the cluster key. You can use a AWS KMS customer master key (p. 3) (CMK) as the master key for Amazon Redshift. You can request encryption by checking the appropriate box in the Amazon Redshift console. You can specify a customer managed CMK (p. 4) to use by choosing one from the list that appears below the encryption box. If you do not specify a customer managed CMK, Amazon Redshift uses the AWS managed CMK (p. 4) for Amazon Redshift under your account.

> **Important**
> Amazon Redshift supports only symmetric CMKs. You cannot use an asymmetric CMK as the master key in an Amazon Redshift encryption workflow. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

Encryption context

Each service that is integrated with AWS KMS specifies an encryption context (p. 19) when requesting data keys, encrypting, and decrypting. The encryption context is additional authenticated data (AAD) 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 (CMK). The operations include cluster encryption, cluster decryption, and generating data keys.

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 (p. 3) (CMK) in AWS KMS. To learn how to encrypt your Amazon RDS resources under an AWS KMS CMK, see Encrypting Amazon RDS Resources in the *Amazon RDS User Guide*.

> **Important**
> Amazon RDS supports only symmetric CMKs (p. 142). You cannot use an asymmetric CMK (p. 142) to encrypt data in an Amazon RDS database. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

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. 253).

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. 19). The encryption context is additional authenticated data (AAD) that AWS KMS uses to ensure data integrity. When an encryption context is specified for an encryption operation, the service must specify the same encryption context for the decryption operation. Otherwise, decryption fails. 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. 5) that is protected by an AWS KMS customer master key (p. 3) (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. 265)
- Encrypting and decrypting secrets (p. 265)

- Using your AWS KMS CMK (p. 267)
- Authorizing use of the CMK (p. 268)
- Secrets Manager encryption context (p. 269)
- Monitoring Secrets Manager interaction with AWS KMS (p. 270)

Protecting the secret value

To protect a secret, Secrets Manager encrypts the *secret value* in a secret.

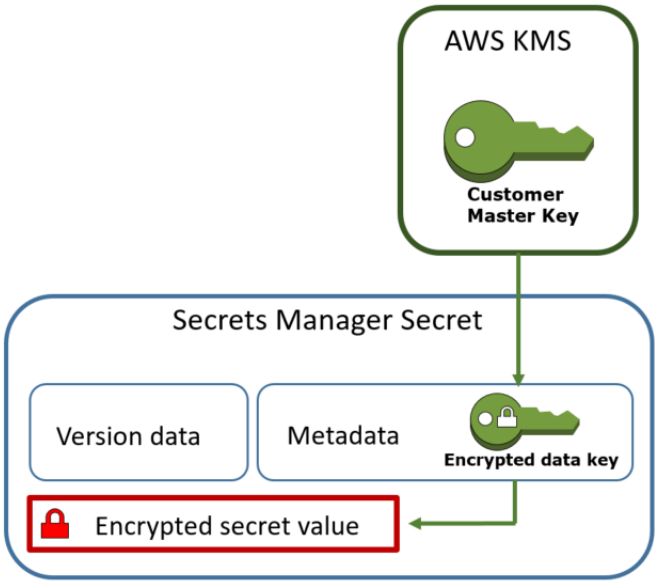
In Secrets Manager, a secret consists of a *secret value*, also known as *protected secret text* or *encrypted secret data*, and related metadata and version information. The secret value can be any string or binary data of up to 65,536 bytes, but it is typically a collection of name-value pairs that 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. 18) with AWS KMS customer master keys (p. 3) (CMKs) and data keys (p. 5).

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. 3) (CMK). Customer managed CMKs allow authorized users to control access (p. 56) to the CMK through policies and grants, manage automatic rotation (p. 154), and use imported key material (p. 159).

> **Important**
> Secrets Manager supports only symmetric CMKs (p. 142). You cannot use an asymmetric CMK (p. 142) to encrypt your secrets. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

When you create a new secret, you can specify any symmetric 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. 5). 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. 3) (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. 270), 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 the 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. 269) 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. 3) (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. 122) with the `secretsmanager.<region>.amazonaws.com` value.

You can also use the keys or values in the encryption context (p. 269) 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 managed CMK

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. 3), is established by the service. You cannot change the key policy, but you can view it at any time. For details, see Viewing a key policy (p. 71).

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. 126) 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. 19) 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. 267) 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. 269) that identifies the secret and version.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AROAIGDTESTANDEXAMPLE: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. 267) 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. 269) 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": "AROAIGDTESTANDEXAMPLE: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-a1b2c3",
            "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 AWS KMS customer master key (p. 3) (CMK) under which Amazon SES encrypts the messages. You can choose the AWS managed CMK (p. 4) for Amazon SES (its alias is **aws/ses**), or you can choose a symmetric customer managed CMK (p. 4) that you created in AWS KMS.

> **Important**
> Amazon SES supports only symmetric CMKs (p. 142). You cannot use an asymmetric CMK (p. 142) to encrypt your Amazon SES email messages. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

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. 272)
- Amazon SES encryption context (p. 273)
- Giving Amazon SES permission to use your AWS KMS customer master key (CMK) (p. 273)
- Getting and decrypting email messages (p. 274)

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. 272) through Step 6 (p. 273), 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 Getting and decrypting email messages (p. 274).

Amazon SES encryption context

When Amazon SES requests a data key to encrypt your received email messages (Step 3 (p. 272) in the Overview of Amazon SES encryption using AWS KMS (p. 272)), it includes an encryption context (p. 19) in the request. The encryption context provides additional authenticated data (AAD) 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 encryption context:

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

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

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

Giving Amazon SES permission to use your AWS KMS customer master key (CMK)

To encrypt your email messages, you can use the AWS managed customer master key (CMK) (p. 4) in your account for Amazon SES (**aws/ses**), or you can use a customer managed CMK (p. 4) that you create. Amazon SES already has permission to use the AWS managed CMK on your behalf. However, if you specify a customer managed CMK when you add the S3 action to your Amazon SES receipt rule, you must give Amazon SES permission to use the CMK to encrypt your email messages.

To give Amazon SES permission to use your customer managed CMK, add the following statement to that CMK's key policy (p. 59):

```
{
  "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, see Amazon SES encryption context (p. 273). For general information about how AWS KMS uses the encryption context, see encryption context (p. 19).

Getting 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 get 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. 272) in the Overview of Amazon SES encryption using AWS KMS (p. 272)). 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 master key or you can use an AWS KMS customer master key (CMK) with the Amazon S3 Encryption Client to protect your data on the client side.

Topics

- Server-Side Encryption: Using SSE-KMS (p. 275)
- Using the Amazon S3 encryption client (p. 276)
- Encryption context (p. 276)

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 Amazon S3-Managed Encryption Keys (SSE-S3).
- 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 customer master key (p. 3) (CMK) 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 select a CMK by using the Amazon S3 console or API. In the console, check the appropriate box to perform encryption and select your CMK from the list. For the Amazon S3 API, specify encryption and choose your CMK 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).

> **Important**
> Amazon S3 supports only symmetric CMKs (p. 142). You cannot use an asymmetric CMK (p. 142) to encrypt your data in Amazon S3. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

You can choose a customer managed CMK (p. 4) or the AWS managed CMK (p. 4) for Amazon S3 in 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 under the specified CMK.
- 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 in 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 Amazon S3 service receives your encrypted data; it 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 (p. 3) (CMK) that you and, if so, returns a new plaintext data key and the data key encrypted under the CMK. The Amazon S3 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 (p. 19) when requesting data keys, encrypting, and decrypting. The encryption context is additional authenticated data (AAD) that AWS KMS uses to check for data integrity.

When an encryption context is specified for an encryption operation, Amazon S3 specifies the same encryption context for the decryption operation. Otherwise, the decryption fails.

If you use SSE-KMS or the Amazon S3 encryption client for 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 the following one.

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

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 error-prone changes 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 AWS managed CMK (p. 4) that Parameter Store creates for your account or specify your own customer managed CMK (p. 4).

> **Important**
> Parameter Store supports only symmetric CMKs (p. 142). You cannot use an asymmetric CMK (p. 142) to encrypt your parameters. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

Parameter Store supports two tiers of secure string parameters: *standard* and *advanced*. Standard parameters, which cannot exceed 4096 bytes, are encrypted and decrypted directly under the CMK that you specify. To encrypt and decrypt advanced secure string parameters, Parameter Store uses envelope encryption with the AWS Encryption SDK. You can convert a standard secure string parameter to an advanced parameter, but you cannot convert an advanced parameter to a standard one. For more information about the difference between standard and advanced secure string parameters, see About Systems Manager Advanced Parameters in the AWS Systems Manager User Guide.

Topics

- Protecting standard secure string parameters (p. 277)
- Protecting advanced secure string parameters (p. 279)
- Setting permissions to encrypt and decrypt parameter values (p. 282)
- Parameter Store encryption context (p. 283)
- Troubleshooting CMK issues in Parameter Store (p. 285)

Protecting standard 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 standard secure string parameter value, Parameter Store calls the AWS KMS Encrypt operation. This operation uses a symmetric AWS KMS CMK directly to encrypt the parameter value instead of using the CMK to generate a data key (p. 5).

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 AWS managed CMK that Systems Manager automatically creates in your account. This CMK has the `aws/ssm` alias.

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 standard secure string parameter, use the PutParameter operation in the Systems Manager API. Omit the `Tier` parameter or specify a value of `Standard`, which is the default. Include a `Type` parameter with a value of `SecureString`. To specify an AWS KMS CMK, use the `KeyId` parameter. The default is the AWS managed CMK for your account, `aws/ssm`.

Parameter Store then calls the AWS KMS `Encrypt` operation 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 `--tier` and `--key-id` parameters, Parameter Store creates a standard secure string parameter and encrypts it under the AWS managed CMK.

```
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 (p. 4). The example uses a CMK ID to identify the CMK, but you can use any valid CMK identifier. Because the command omits the `Tier` parameter (`--tier`), Parameter Store creates a standard secure string parameter, not an advanced one.

```
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
    }
}
```

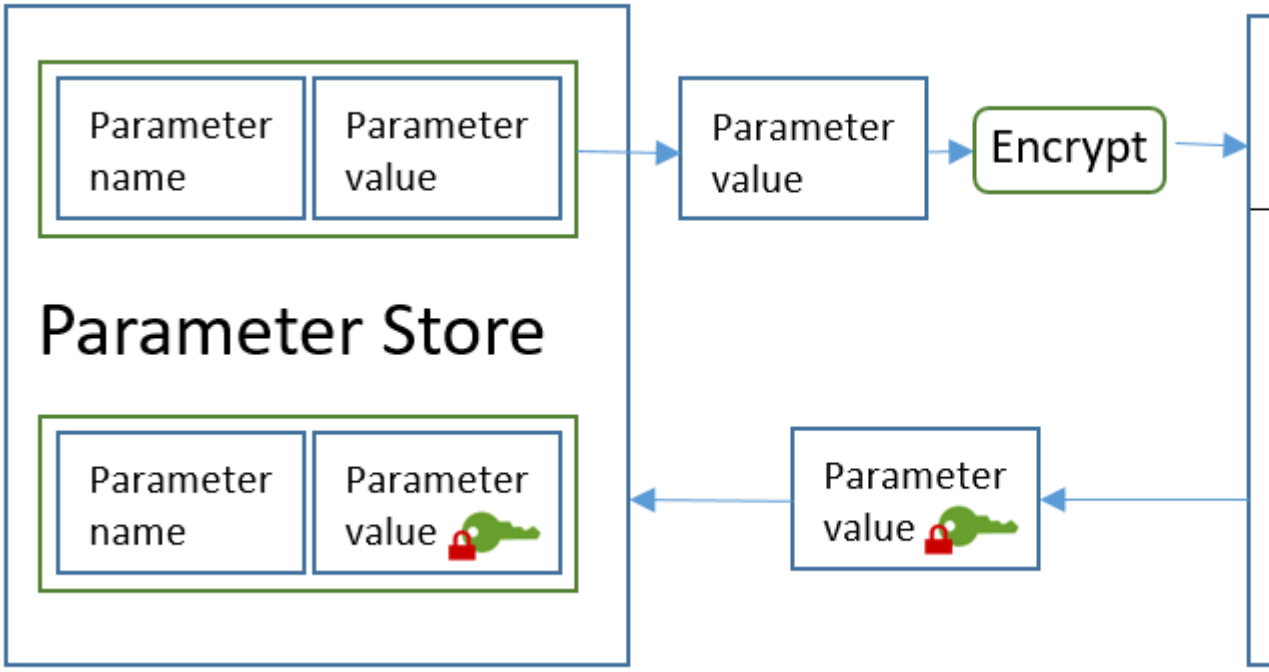
To decrypt the parameter value before returning it, set the `WithDecryption` parameter of `GetParameter` to `true`. When you use `WithDecryption`, Parameter Store calls the AWS KMS Decrypt operation on your behalf to decrypt the parameter value. As a result, the `GetParameter` request returns the parameter 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 to encrypt and decrypt a standard secure string parameter.

Encrypt a standard parameter

1. When you use `PutParameter` to create a secure string parameter, Parameter Store sends an `Encrypt` request to AWS KMS. That request includes the plaintext parameter value, the CMK that you chose, and the Parameter Store encryption context (p. 283). 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 encryption context. It returns the ciphertext to Parameter Store, which stores the parameter name and its encrypted value.



Decrypt a standard parameters

1. When you include the `WithDecryption` parameter in a `GetParameter` request, Parameter Store sends a `Decrypt` request to AWS KMS with the encrypted secure string parameter value and the Parameter Store encryption context (p. 283).
2. AWS KMS uses the same CMK and the supplied encryption context 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.

Protecting advanced secure string parameters

When you use `PutParameter` to create an advanced secure string parameter, Parameter Store uses envelope encryption with the AWS Encryption SDK and a symmetric AWS KMS customer master key (CMK) to protect the parameter value. Each advanced parameter value is encrypted under a unique data key, and the data key is encrypted under an AWS KMS CMK. You can use the AWS managed CMK (p. 4) for the account (`aws/ssm`) or any customer managed CMK.

The AWS Encryption SDK is an open-source, client-side library that helps you to encrypt and decrypt data using industry standards and best practices. It's supported on multiple platforms and in multiple programming languages, including a command-line interface. You can view the source code and contribute to its development in GitHub.

For each secure string parameter value, Parameter Store calls the AWS Encryption SDK to encrypt the parameter value using a unique data key that AWS KMS generates (GenerateDataKey). The AWS Encryption SDK returns to Parameter Store an encrypted message that includes the encrypted parameter value and an encrypted copy of the unique data key. Parameter Store stores the entire encrypted message in the secure string parameter value. Then, when you get an advanced secure string parameter

value, Parameter Store uses the AWS Encryption SDK to decrypt the parameter value. This requires a call to AWS KMS to decrypt the encrypted data key.

To create an advanced secure string parameter, use the PutParameter operation in the Systems Manager API. Set the value of `Tier` parameter to `Advanced`. Include a `Type` parameter with a value of `SecureString`. To specify an AWS KMS CMK, use the `KeyId` parameter. The default is the AWS managed CMK for your account, `aws/ssm`.

```
aws ssm put-parameter --name MyParameter --value "secret_value" --type SecureString --tier
 Advanced
```

The following similar example uses the `--key-id` parameter to specify a customer managed CMK (p. 4). The example uses the Amazon Resource Name (ARN) of the CMK, but you can use any valid CMK identifier.

```
aws ssm put-parameter --name MyParameter --value "secret_value"
 --type SecureString --tier Advanced --key-id arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab
```

When you get a secure string parameter from Parameter Store, its value is the encrypted message that the AWS Encryption SDK returned. To get a parameter, use the GetParameter operation in the Systems Manager API.

The following example uses the Systems Manager `GetParameter` operation 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 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"
    }
}
```

You cannot convert an advanced secure string parameter to a standard one, but you can convert a standard secure string to an advanced one. To convert a standard secure string parameter to an advanced secure string, use the `PutParameter` operation with the `Overwrite` parameter. The `Type` must be `SecureString` and the `Tier` value must be `Advanced`. The `KeyId` parameter, which identifies a customer managed CMK, is optional. If you omit it, Parameter Store uses the AWS managed CMK for the account. You can specify any CMK that the principal has permission to use, even if you used a different CMK to encrypt the standard parameter.

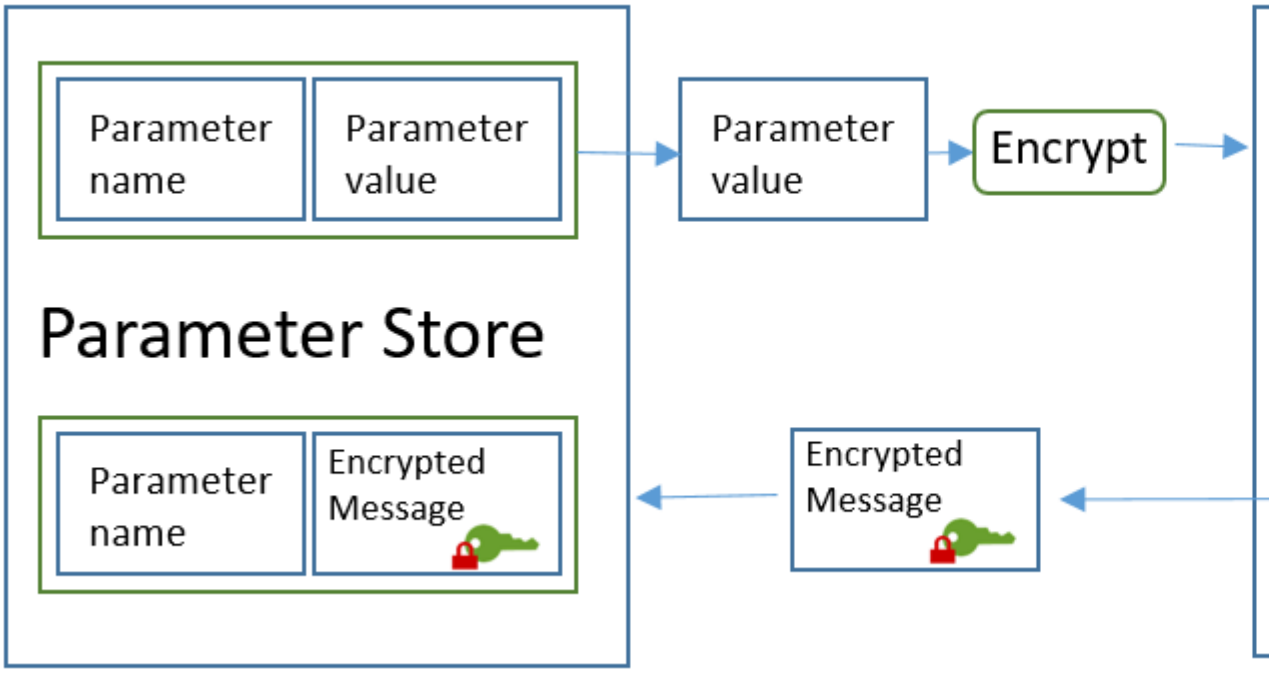
When you use the `Overwrite` parameter, Parameter Store uses the AWS Encryption SDK to encrypt the parameter value. Then it stores the newly encrypted message in Parameter Store.

```
$ aws ssm put-parameter --name myStdParameter --value "secret_value"  --type SecureString
 --tier Advanced --key-id 1234abcd-12ab-34cd-56ef-1234567890ab --overwrite
```

The following workflow shows how Parameter Store uses an AWS KMS CMK to encrypt and decrypt an advanced secure string parameter.

Encrypt an advanced parameter

1. When you use `PutParameter` to create an advanced secure string parameter, Parameter Store uses the AWS Encryption SDK and AWS KMS to encrypt the parameter value. Parameter Store calls the AWS Encryption SDK with the parameter value, the AWS KMS CMK that you specified, and the Parameter Store encryption context (p. 283).
2. The AWS Encryption SDK sends a GenerateDataKey request to AWS KMS with the identifier of the CMK that you specified and the Parameter Store encryption context. AWS KMS returns two copies of the unique data key: one in plaintext and one encrypted under the CMK. (The encryption context is used when encrypting the data key.)
3. The AWS Encryption SDK uses the plaintext data key to encrypt the parameter value. It returns an encrypted message that includes the encrypted parameter value, the encrypted data key, and other data, including the Parameter Store encryption context.
4. Parameter Store stores the encrypted message as the parameter value.



Decrypt an advanced parameter

1. You can include the `WithDecryption` parameter in a `GetParameter` request to get an advanced secure string parameter. When you do, Parameter Store passes the encrypted message from the parameter value to a decryption method of the AWS Encryption SDK.
2. The AWS Encryption SDK calls the KMS Decrypt operation. It passes in the encrypted data key and the Parameter Store encryption context from the encrypted message.

3. AWS KMS uses the CMK and the Parameter Store encryption context to decrypt the encrypted data key. Then it returns the plaintext (decrypted) data key to the AWS Encryption SDK.
4. The AWS Encryption SDK uses the plaintext data key to decrypt the parameter value. It returns the plaintext parameter value to Parameter Store.
5. Parameter Store verifies the encryption context and returns the plaintext parameter value to you in the `GetParameter` response.

Setting permissions to encrypt and decrypt parameter values

To encrypt a standard secure string parameter value, the user needs `kms:Encrypt` permission. To encrypt an advanced secure string parameter value, the user needs `kms:GenerateDataKey` permission. To decrypt either type of secure string parameter value, the user needs `kms:Decrypt` permission.

You can use IAM policies to allow or deny permission for a user to call the Systems Manager `PutParameter` and `GetParameter` operations.

If you are using customer managed CMKs to encrypt your secure string parameter values, 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. 56).

The following example shows an IAM policy designed for standard secure string parameters. It allows the user to call the Systems Manager `PutParameter` operation on all parameters in the `FinancialParameters` path. The policy also allows the user to call the AWS KMS `Encrypt` operation on an example customer managed CMK.

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

The next example shows an IAM policy that is designed for advanced secure string parameters. It allows the user to call the Systems Manager `PutParameter` operation on all parameters in the `ReservedParameters` path. The policy also allows the user to call the AWS KMS `GenerateDataKey` operation on an example customer managed CMK.

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

```
        {
            "Effect": "Allow",
            "Action": [
                "ssm:PutParameter"
            ],
            "Resource": "arn:aws:ssm:us-west-2:111122223333:parameter/ReservedParameters/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:us-
west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
        }
    ]
}
```

The final example also shows an IAM policy that can be used for standard or advanced secure string parameters. It allows the user to call the Systems Manager `GetParameter` operations (and related operations) on all parameters in the `ITParameters` path. The policy also allows the user to call the AWS KMS `Decrypt` operation on an example customer managed CMK.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ssm:GetParameter*"
            ],
            "Resource": "arn:aws:ssm:us-west-2:111122223333:parameter/ITParameters/*"
        },
        {
            "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.

The AWS Encryption SDK also takes an encryption context, although it handles it differently. Parameter Store supplies the encryption context to the encryption method. The AWS Encryption SDK cryptographically binds the encryption context to the encrypted data. It also includes the encryption context in plain text in the header of the encrypted message that it returns. However, unlike AWS KMS, the AWS Encryption SDK decryption methods do not take an encryption context as input. Instead, when it decrypts data, the AWS Encryption SDK gets the encryption context from the encrypted message. Parameter Store verifies that the encryption context includes the value that it expects before returning the plaintext parameter value to you.

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 decrypt Parameter Store parameter values by using the `GetParameter` operation with the `WithDecryption` parameter.

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 ARNs 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. 56).
- The CMK is not found.

  This typically happens when you use an incorrect identifier for the CMK. Find the correct identifiers (p. 40) 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. 51). If it is `Pending Import`, complete the import procedure (p. 159). If the key state is `Pending Deletion`, cancel the key deletion (p. 174) or use a different CMK.

  To find the key state (p. 233) of a CMK in the AWS KMS console, on the **Customer managed keys** or **AWS managed keys** page, see the Status column (p. 30). To use the AWS KMS API to find the status of a CMK, 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. 285)
- Amazon WorkMail encryption (p. 286)
- Authorizing use of the CMK (p. 288)
- Amazon WorkMail encryption context (p. 290)
- Monitoring Amazon WorkMail interaction with AWS KMS (p. 290)

Amazon WorkMail overview

Amazon WorkMail is a secure, managed business email and calendaring service with support for existing desktop and mobile email clients. You can create an Amazon WorkMail organization and assign to it one or more email domains that you own. Then you can create mailboxes for the email users and distribution groups in the organization.

Amazon WorkMail transparently encrypts all messages in the mailboxes of all Amazon WorkMail organizations before the messages are written to disk and transparently decrypts the messages when users access them. There is no option to disable encryption. To protect the encryption keys that protect the messages, Amazon WorkMail is integrated with AWS Key Management Service (AWS KMS).

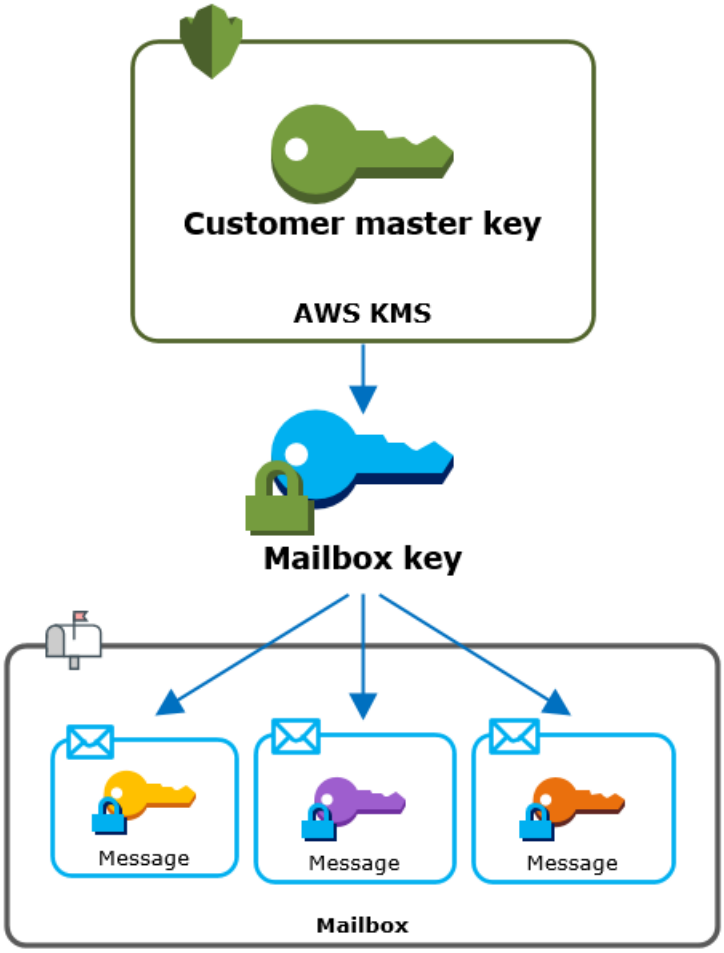
Amazon WorkMail also provides an option for enabling users to send signed or encrypted email. This encryption feature does not use AWS KMS.

Amazon WorkMail encryption

In Amazon WorkMail, each organization can contain multiple mailboxes, one for each user in the organization. All messages, including email and calendar items, are stored in the user's mailbox.

To protect the contents of the mailboxes in your Amazon WorkMail organizations, Amazon WorkMail encrypts all mailbox messages before they are written to disk. No customer-provided information is stored in plaintext.

Each message is encrypted under a unique data encryption key. The message key is protected by a mailbox key, which is a unique encryption key that is used only for that mailbox. The mailbox key is encrypted under an AWS KMS customer master key (CMK) for the organization that never leaves AWS KMS unencrypted. The following diagram shows the relationship of the encrypted messages, encrypted message keys, encrypted mailbox key, and the CMK for the organization in AWS KMS.



A CMK for the organization

When you create an Amazon WorkMail organization, you can select an AWS KMS customer master key (CMK) for the organization. This CMK protects all mailbox keys in that organization.

If you use the Quick Setup procedure to create your organization, Amazon WorkMail uses the AWS managed CMK (p. 3) for Amazon WorkMail (`aws/workmail`) in your AWS account. If you use the Standard Setup, you can select the AWS managed CMK for Amazon WorkMail or a customer managed CMK (p. 3) that you own and manage. You can select the same CMK or a different CMK for each of your organizations, but you cannot change the CMK once you have selected it.

> **Important**
> Amazon WorkMail supports only symmetric CMKs. You cannot use an asymmetric CMK to encrypt data in Amazon WorkMail. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

To find the CMK for your organization, use the AWS CloudTrail log entry that records calls to AWS KMS.

A unique encryption key for each mailbox

When you create a new mailbox, Amazon WorkMail generates a unique 256-bit Advanced Encryption Standard (AES) symmetric encryption key for the mailbox, known as its *mailbox key*, outside of AWS KMS. Amazon WorkMail uses the mailbox key to protect the encryption keys for each message in the mailbox.

To protect the mailbox key, Amazon WorkMail calls AWS KMS to encrypt the mailbox key under the CMK for the organization. Then it stores the encrypted mailbox key in the mailbox metadata.

> **Note**
> Amazon WorkMail uses a symmetric mailbox encryption key to protect message keys. Previously, Amazon WorkMail protected each mailbox with an asymmetric key pair. It used the public key to encrypt each message key and the private key to decrypt it. The private mailbox key was protected by the CMK for the organization. Existing mailboxes might still use an asymmetric mailbox key pair. This change does not affect the security of the mailbox or its messages.

A unique encryption key for each message

When a message is added to the mailbox, Amazon WorkMail generates a unique 256-bit AES symmetric encryption key for the message outside of AWS KMS. It uses this *message key* to encrypt the message. Amazon WorkMail encrypts the message key under the mailbox key and stores the encrypted message key with the message. Then, it encrypts the mailbox key under the CMK for the organization.

Creating a new mailbox

When Amazon WorkMail creates a new mailbox, it uses the following process to prepare the mailbox to hold encrypted messages.

- Amazon WorkMail generates a unique 256-bit AES symmetric encryption key for the mailbox outside of AWS KMS.
- Amazon WorkMail calls the AWS KMS Encrypt operation. It passes in the mailbox key and the identifier of the customer master key (CMK) for the organization. AWS KMS returns a ciphertext of the mailbox key encrypted under the CMK.
- Amazon WorkMail stores the encrypted mailbox key with the mailbox metadata.

Encrypting a mailbox message

To encrypt a message, Amazon WorkMail uses the following process.

1. Amazon WorkMail generates a unique 256-bit AES symmetric key for the message. It uses the plaintext message key and the Advanced Encryption Standard (AES) algorithm to encrypt the message outside of AWS KMS.
2. To protect the message key under the mailbox key, Amazon WorkMail needs to decrypt the mailbox key, which is always stored in its encrypted form.

Amazon WorkMail calls the AWS KMS Decrypt operation and passes in the encrypted mailbox key. AWS KMS uses the CMK for the organization to decrypt the mailbox key and it returns the plaintext mailbox key to Amazon WorkMail.

3. Amazon WorkMail uses the plaintext mailbox key and the Advanced Encryption Standard (AES) algorithm to encrypt the message key outside of AWS KMS.
4. Amazon WorkMail stores the encrypted message key in the metadata of the encrypted message so it is available to decrypt it.

Decrypting a mailbox message

To decrypt a message, Amazon WorkMail uses the following process.

1. Amazon WorkMail calls the AWS KMS Decrypt operation and passes in the encrypted mailbox key. AWS KMS uses the CMK for the organization to decrypt the mailbox key and it returns the plaintext mailbox key to Amazon WorkMail.
2. Amazon WorkMail uses the plaintext mailbox key and the Advanced Encryption Standard (AES) algorithm to decrypt the encrypted message key outside of AWS KMS.
3. Amazon WorkMail uses the plaintext message key to decrypt the encrypted message.

Caching mailbox keys

To improve performance and minimize calls to AWS KMS, Amazon WorkMail caches each plaintext mailbox key for each client locally for up to one minute. At the end of the caching period, the mailbox key is removed. If the mailbox key for that client is required during the caching period, Amazon WorkMail can get it from the cache instead of calling AWS KMS. The mailbox key is protected in the cache and is never written to disk in plaintext.

Authorizing use of the CMK

When Amazon WorkMail uses a customer master key (CMK) in cryptographic operations, it acts on behalf of the mailbox administrator.

To use the AWS KMS customer master key (CMK) for a secret on your behalf, the administrator must have the following permissions. You can specify these required permissions in an IAM policy or key policy.

- `kms:Encrypt`
- `kms:Decrypt`
- `kms:CreateGrant`

To allow the CMK to be used only for requests that originate in Amazon WorkMail, you can use the kms:ViaService (p. 122) condition key with the `workmail.<region>.amazonaws.com` value.

You can also use the keys or values in the encryption context (p. 290) 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 for the AWS managed CMK

The key policy for the AWS managed CMK for Amazon WorkMail gives users permission to use the CMK for specified operations only when Amazon WorkMail 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, is established by the service. You cannot change the key policy, but you can view it at any time. For details, see Viewing a key policy (p. 71).

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

- Allow users in the account and Region to use the CMK for cryptographic operations and to create grants, but only when the request comes from Amazon WorkMail 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.

The following is a key policy for an example AWS managed CMK for Amazon WorkMail.

```
{
  "Version" : "2012-10-17",
  "Id" : "auto-workmail-1",
  "Statement" : [ {
    "Sid" : "Allow access through WorkMail for all principals in the account that are
 authorized to use WorkMail",
    "Effect" : "Allow",
    "Principal" : {
      "AWS" : "*"
    },
    "Action" : [ "kms:Decrypt", "kms:CreateGrant", "kms:ReEncrypt*", "kms:DescribeKey",
 "kms:Encrypt" ],
    "Resource" : "*",
    "Condition" : {
      "StringEquals" : {
        "kms:ViaService" : "workmail.us-east-1.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:List*", "kms:Get*", "kms:RevokeGrant" ],
    "Resource" : "*"
  } ]
}
```

Using grants to authorize Amazon WorkMail

In addition to key policies, Amazon WorkMail uses grants to add permissions to the CMK for each organization. To view the grants on the CMK in your account, use the ListGrants operation.

Amazon WorkMail uses grants to add the following permissions to the CMK for the organization.

- Add the `kms:Encrypt` permission to allow Amazon WorkMail to encrypt the mailbox key.
- Add the `kms:Decrypt` permission to allow Amazon WorkMail to use the CMK to decrypt the mailbox key. Amazon WorkMail requires this permission in a grant because the request to read mailbox messages uses the security context of the user who is reading the message. The request does not use the credentials of the AWS account. Amazon WorkMail creates this grant when you select a CMK for the organization.

To create the grants, Amazon WorkMail calls CreateGrant on behalf of the user who created the organization. Permission to create the grant comes from the key policy. This policy allows account users

to call `CreateGrant` on the CMK for the organization when Amazon WorkMail makes the request on an authorized user's behalf.

The key policy also allows the account root to revoke the grant on the AWS managed key. However, if you revoke the grant, Amazon WorkMail cannot decrypt the encrypted data in your mailboxes.

Amazon WorkMail encryption context

An encryption context (p. 19) 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.

Amazon WorkMail uses the same encryption context format in all AWS KMS cryptographic operations. You can use the encryption context to identify a cryptographic operation in audit records and logs, such as AWS CloudTrail, and as a condition for authorization in policies and grants.

In its Encrypt and Decrypt requests to AWS KMS, Amazon WorkMail uses an encryption context where the key is `aws:workmail:arn` and the value is the Amazon Resource Name (ARN) of the organization.

```
"aws:workmail:arn":"arn:aws:workmail:region:account ID:organization/organization ID"
```

For example, the following encryption context includes an example organization ARN in the US East (Ohio) (`us-east-2`) Region.

```
"aws:workmail:arn":"arn:aws:workmail:us-east-2:111122223333:organization/
m-68755160c4cb4e29a2b2f8fb58f359d7"
```

Monitoring Amazon WorkMail interaction with AWS KMS

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

Encrypt

When you create a new mailbox, Amazon WorkMail generates a mailbox key and calls AWS KMS to encrypt the mailbox key. Amazon WorkMail sends an Encrypt request to AWS KMS with the plaintext mailbox key and an identifier for the CMK of the Amazon WorkMail organization.

The event that records the `Encrypt` operation is similar to the following example event. The user is the Amazon WorkMail service. The parameters include the CMK ID (`keyId`) and the encryption context for the Amazon WorkMail organization. Amazon WorkMail also passes in the mailbox key, but that is not recorded in the CloudTrail log.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "workmail.eu-west-1.amazonaws.com"
    },
    "eventTime": "2019-02-19T10:01:09Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Encrypt",
    "awsRegion": "eu-west-1",
    "sourceIPAddress": "workmail.eu-west-1.amazonaws.com",
```

```
    "userAgent": "workmail.eu-west-1.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:workmail:arn": "arn:aws:workmail:eu-west-1:111122223333:organization/m-
c6981ff7642446fa8772ba99c690e455"
        },
        "keyId": "arn:aws:kms:eu-
west-1:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d"
    },
    "responseElements": null,
    "requestID": "76e96b96-7e24-4faf-a2d6-08ded2eaf63c",
    "eventID": "d5a59c18-128a-4082-aa5b-729f7734626a",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:eu-
west-1:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d",
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "sharedEventID": "d08e60f1-097e-4a00-b7e9-10bc3872d50c"
}
```

Decrypt

When you add, view, or delete a mailbox message, Amazon WorkMail asks AWS KMS to decrypt the mailbox key. Amazon WorkMail sends an Decrypt request to AWS KMS with the encrypted mailbox key and an identifier for the CMK of the Amazon WorkMail organization.

The event that records the `Decrypt` operation is similar to the following example event. The user is the Amazon WorkMail service. The parameters include the encrypted mailbox key (as a ciphertext blob), which is not recorded in the log, and the encryption context for the Amazon WorkMail organization. AWS KMS derives the ID of the CMK from the ciphertext.

```
{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "workmail.eu-west-1.amazonaws.com"
    },
    "eventTime": "2019-02-20T11:51:10Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "eu-west-1",
    "sourceIPAddress": "workmail.eu-west-1.amazonaws.com",
    "userAgent": "workmail.eu-west-1.amazonaws.com",
    "requestParameters": {
        "encryptionContext": {
            "aws:workmail:arn": "arn:aws:workmail:eu-west-1:111122223333:organization/m-
c6981ff7642446fa8772ba99c690e455"
        }
    },
    "responseElements": null,
    "requestID": "4a32dda1-34d9-4100-9718-674b8e0782c9",
    "eventID": "ea9fd966-98e9-4b7b-b377-6e5a397a71de",
    "readOnly": true,
    "resources": [
        {
            "ARN": "arn:aws:kms:eu-
west-1:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d",
```

```
            "accountId": "111122223333",
            "type": "AWS::KMS::Key"
        }
    ],
    "eventType": "AwsApiCall",
    "recipientAccountId": "111122223333",
    "sharedEventID": "241e1e5b-ff64-427a-a5b3-7949164d0214"
}
```

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 (p. 3) (CMK) to use for the encryption. You can choose the AWS managed CMK (p. 4) for Amazon WorkSpaces (**aws/workspaces**) or a symmetric customer managed CMK (p. 4).

> **Important**
> Amazon WorkSpaces supports only symmetric CMKs. You cannot use an asymmetric CMK to encrypt the volumes in an Amazon WorkSpaces. For help determining whether a CMK is symmetric or asymmetric, see Identifying symmetric and asymmetric CMKs (p. 43).

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. 292)
- Amazon WorkSpaces encryption context (p. 293)
- Giving Amazon WorkSpaces permission to use a CMK on your behalf (p. 293)

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. 253) 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. 126) 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. 292)). This action creates a grant (p. 126) 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. Amazon EBS sends the encrypted data key to AWS KMS with a `Decrypt` request and specifies the WorkSpace user's `Sid`, its directory ID, and the the volume ID, which is used as the encryption context (p. 293).
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. 293)) 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 an encryption context (p. 19). However, when Amazon EBS requests an encrypted data key for the encrypted volumes of your WorkSpaces (Step 3 (p. 292) in the Overview of Amazon WorkSpaces encryption using AWS KMS (p. 292)) and when it requests a plaintext copy of that data key (Step 5 (p. 293)), it includes encryption context in the request. The encryption context provides additional authenticated data (AAD) 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"
}
```

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. 294)
2. Give the WorkSpaces administrators additional permissions with an IAM policy (p. 294)

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.

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 (AWS 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. 64). 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. 61), WorkSpaces administrators need permission to create grants (p. 126) 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, see Managed Policies and 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. 296) 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 (p. 10) 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. 159) 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. 297).

- **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. 300) 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 with the CloudTrail Processing Library, 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 dashboards provide an at-a-glance view of the state of your AWS environment.

You can customize (p. 30) the **AWS Managed Keys** and **Customer Managed Keys** pages of the AWS KMS console to display the following information about each CMK:

- Key ID
- Status
- Creation date
- Expiration date (for CMKs with imported key material (p. 159))
- Origin
- Custom key store ID (for CMKs in custom key stores (p. 184))

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. 298)
- Creating CloudWatch alarms to monitor AWS KMS metrics (p. 299)
- AWS KMS events (p. 300)

AWS KMS metrics and dimensions

When you import key material into a CMK (p. 159) 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. 299).

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. 299)
- Create a CloudWatch alarm to monitor usage of CMKs that are pending deletion (p. 300)

Create a CloudWatch alarm to monitor the expiration of imported key material

When you import key material into a CMK (p. 159), 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.

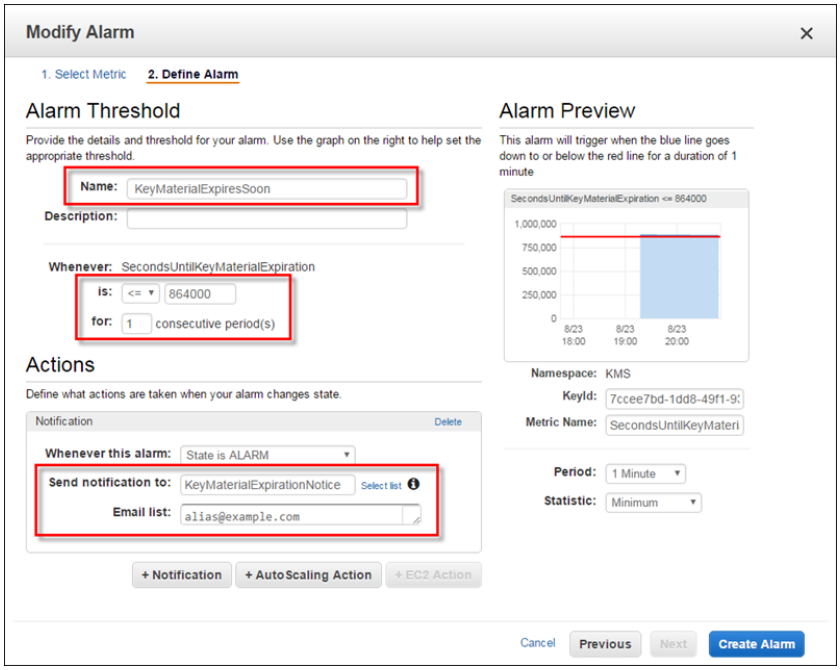
To create an alarm to monitor the expiration of imported key material (AWS Management Console)

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. 172) 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. 177).

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. 301)
- KMS imported key material expiration (p. 301)
- KMS CMK deletion (p. 302)

KMS CMK rotation

When you enable automatic key rotation (p. 154) for a customer managed CMK (p. 4), AWS KMS creates new key material for the CMK each year. The key material for AWS managed CMKs (p. 4) is automatically rotated every three years.

Whenever AWS KMS rotates key material, it sends a KMS CMK Rotation 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. 159), 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. 172) 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.

Although, by default, all AWS KMS actions are logged as CloudTrail events, you can exclude AWS KMS actions from a CloudTrail trail. For details, see Excluding AWS KMS events from a trail (p. 304).

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. 296).

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 (p. 10), 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.

Excluding AWS KMS events from a trail

Most AWS KMS users rely on the events in a CloudTrail trail to provide a record of the use and management of their AWS KMS resources. The trail can be an valuable source of data for auditing critical events, such as creating, disabling, and deleting customer master keys (CMKs), changing key policy, and the use of your CMKs by AWS services on your behalf. In some cases, the metadata in a CloudTrail log entry, such as the encryption context (p. 19) in an encryption operation, can help you to avoid or resolve errors.

However, because AWS KMS can generate a large number of events, AWS CloudTrail lets you exclude AWS KMS events from a trail. This per-trail setting excludes all AWS KMS events; you cannot exclude particular AWS KMS events.

> **Warning**
> Excluding AWS KMS events from a CloudTrail Log can obscure actions that use your CMKs. Be cautious when giving principals the `cloudtrail:PutEventSelectors` permission that is required to perform this operation.

To exclude AWS KMS events from a trail:

- In the CloudTrail console, use the **Log Key Management Service events** setting when you create a trail or update a trail. For instructions, see Logging Management Events with the AWS Management Console in the AWS CloudTrail User Guide.
- In the CloudTrail API, use the PutEventSelectors operation. Add the `ExcludeManagementEventSources` attribute to your event selectors with a value of `kms.amazonaws.com`. For an example, see Example: A trail that does not log AWS Key Management Service events in the AWS CloudTrail User Guide.

You can disable this exclusion at any time by changing the console setting or the event selectors for a trail. The trail will then start recording AWS KMS events. However, it cannot recover AWS KMS events that occurred while the exclusion was effective.

When you exclude KMS events by using the console or API, the resulting CloudTrail `PutEventSelectors` API operation is also logged in your CloudTrail Logs. If KMS events don't appear in your CloudTrail Logs, look for a `PutEventSelectors` event with the `ExcludeManagementEventSources` attribute set to `kms.amazonaws.com`.

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 CloudTrail log entries for each API request, see the following topics.

Topics

- CreateAlias (p. 305)
- CreateGrant (p. 306)
- CreateKey (p. 307)
- Decrypt (p. 308)
- DeleteAlias (p. 308)
- DescribeKey (p. 309)
- DisableKey (p. 310)
- EnableKey (p. 311)
- Encrypt (p. 312)
- GenerateDataKey (p. 312)
- GenerateDataKeyWithoutPlaintext (p. 313)
- GenerateRandom (p. 314)
- GetKeyPolicy (p. 314)
- ListAliases (p. 315)
- ListGrants (p. 316)
- ReEncrypt (p. 316)
- Amazon EC2 example one (p. 317)
- Amazon EC2 example two (p. 319)

CreateAlias

The following example shows a log file generated by calling `CreateAlias`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice",
            "sessionContext": {
                "attributes": {
                    "mfaAuthenticated": "false",
                    "creationDate": "2014-11-04T00:52:27Z"
                }
            }
        },
        "eventTime": "2014-11-04T00:52:27Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "CreateAlias",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "aliasName": "alias/my_alias",
            "targetKeyId": "arn:aws:kms:us-east-1:123456789012:key/64e07f97-2489-4d04-
bfdf-41723ad130bd"
        },
        "responseElements": null,
        "requestID": "d9472f40-63bc-11e4-bc2b-4198b6150d5c",
```

```
        "eventID": "f72d3993-864f-48d6-8f16-e26e1ae8dff0",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/64e07f97-2489-4d04-
bfdf-41723ad130bd",
            "accountId": "123456789012"
        },
        {
            "ARN": "arn:aws:kms:us-east-1:123456789012:alias/my_alias",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```


CreateGrant

The following example shows a log file generated by calling `CreateGrant`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:53:12Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "CreateGrant",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "keyId": "arn:aws:kms:us-east-1:123456789012:key/65f61d18-
c45c-41ca-90c9-179982e9b716",
            "constraints": {
                "encryptionContextSubset": {
                    "ContextKey1": "Value1"
                }
            },
            "operations": ["Encrypt",
            "RetireGrant"],
            "granteePrincipal": "EX_PRINCIPAL_ID"
        },
        "responseElements": {
            "grantId": "f020fe75197b93991dc8491d6f19dd3cebb24ee62277a05914386724f3d48758"
        },
        "requestID": "f3c08808-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "5d529779-2d27-42b5-92da-91aaea1fc4b5",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/65f61d18-
c45c-41ca-90c9-179982e9b716",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
```

```
        "recipientAccountId": "123456789012"
    }
  ]
}
```

CreateKey

The following example shows a log file generated by calling `CreateKey`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:59Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "CreateKey",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "policy": "{\n  \"Version\":\"2012-10-17\",\n  \"Statement\":[{\n    \"Effect
\":\"Allow\",\n    \"Principal\":{\"AWS\":\"arn:aws:iam::123456789012:user/Alice\"},\n
    \"Action\":\"kms:*\",\n    \"Resource\":\"*\"\n  }, {\n    \"Effect\":\"Allow\",\n
    \"Principal\":{\"AWS\":\"arn:aws:iam::012345678901:user/Bob\"},\n    \"Action\":
\"kms:CreateGrant\",\n    \"Resource\":\"*\"\n  }, {\n    \"Effect\":\"Allow\",\n
 \"Principal\":{\"AWS\":\"arn:aws:iam::012345678901:user/Charlie\"},\n    \"Action\":
\"kms:Encrypt\",\n    \"Resource\":\"*\"\n}]\n}",
            "description": "",
            "keyUsage": "ENCRYPT_DECRYPT"
        },
        "responseElements": {
            "keyMetadata": {
                "AWSAccountId": "123456789012",
                "enabled": true,
                "creationDate": "Nov 4, 2014 12:52:59 AM",
                "keyId": "06dc80ca-1bdc-4d0b-be5b-b7009cd14f13",
                "keyUsage": "ENCRYPT_DECRYPT",
                "description": "",
                "arn": "arn:aws:kms:us-east-1:123456789012:key/06dc80ca-1bdc-4d0b-be5b-
b7009cd14f13"
            }
        },
        "requestID": "ebe8ee68-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "ba116326-1792-4784-87dd-a688d1cb42ec",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/06dc80ca-1bdc-4d0b-be5b-
b7009cd14f13",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

Decrypt

The following example shows a log file generated by calling `Decrypt`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:20Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "Decrypt",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "errorCode": "InvalidCiphertextException",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "d5239dea-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "954983cf-7da9-4adf-aeaa-261a1292c0aa",
        "readOnly": true,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/e17cebae-e7a6-4864-
b92f-0365f2feff38",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```


DeleteAlias

The following example shows a log file generated by calling `DeleteAlias`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice",
            "sessionContext": {
                "attributes": {
                    "mfaAuthenticated": "false",
                    "creationDate": "2014-11-04T00:52:27Z"
                }
            }
        },
        "eventTime": "2014-11-04T00:52:27Z",
```

```
        "eventSource": "kms.amazonaws.com",
        "eventName": "DeleteAlias",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "aliasName": "alias/my_alias"
        },
        "responseElements": null,
        "requestID": "d9542792-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "12f48554-bb04-4991-9cfc-e7e85f68eda0",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:alias/my_alias",
            "accountId": "123456789012"
        },
        {
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/64e07f97-2489-4d04-
bfdf-41723ad130bd",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```


DescribeKey

The following example shows a log file that records multiple calls to DescribeKey. These calls were the result of viewing keys (p. 29) 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": "939fcefb-dc14-4a52-b918-73045fe97af3",
      "readOnly": true,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-east-1:123456789012:key/e7b6d35a-b551-4c8f-
b51a-0460ebc04565",
          "accountId": "123456789012"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    }
  ]
}
```


DisableKey

The following example shows a log file generated by calling `DisableKey`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
```

```
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:43Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "DisableKey",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "keyId": "262d9fcb-f1a0-4447-af16-3714cff61ec1"
        },
        "responseElements": null,
        "requestID": "e26552bc-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "995c4653-3c53-4a06-a0f0-f5531997b741",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/262d9fcb-f1a0-4447-
af16-3714cff61ec1",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```


EnableKey

The following example shows a log file generated by calling `EnableKey`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:20Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "EnableKey",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "keyId": "e17cebae-e7a6-4864-b92f-0365f2feff38"
        },
        "responseElements": null,
        "requestID": "d528a6fb-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "be393928-3629-4370-9634-567f9274d52e",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/e17cebae-e7a6-4864-
b92f-0365f2feff38",
```

```
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

Encrypt

The following example shows a log file generated by calling `Encrypt`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:53:11Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "Encrypt",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "encryptionContext": {
                "ContextKey1": "Value1"
            },
            "keyId": "arn:aws:kms:us-east-1:012345678901:key/8d3acf57-6bba-480a-9459-
ed1b8e79d3d0"
        },
        "responseElements": null,
        "requestID": "f3423043-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "91235988-eb87-476a-ac2c-0cdc244e6dca",
        "readOnly": true,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:012345678901:key/8d3acf57-6bba-480a-9459-
ed1b8e79d3d0",
            "accountId": "012345678901"
        }],
        "eventType": "AwsServiceEvent",
        "recipientAccountId": "012345678901"
    }
  ]
}
```

GenerateDataKey

The following example shows a log file created by calling `GenerateDataKey`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
```

```
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:40Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "GenerateDataKey",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "keyId": "637e8678-3d08-4922-a650-e77eb1591db5",
            "numberOfBytes": 32
        },
        "responseElements": null,
        "requestID": "e0eb83e3-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "a9dea4f9-8395-46c0-942c-f509c02c2b71",
        "readOnly": true,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/637e8678-3d08-4922-a650-
e77eb1591db5",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```


GenerateDataKeyWithoutPlaintext

The following example shows a log file created by calling `GenerateDataKeyWithoutPlaintext`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:23Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "GenerateDataKeyWithoutPlaintext",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "errorCode": "InvalidKeyUsageException",
        "requestParameters": {
            "keyId": "d4f2a88d-5f9c-4807-b71d-4d0ee5225156",
            "numberOfBytes": 16
        },
        "responseElements": null,
        "requestID": "d6b8e411-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "f7734272-9ec5-4c80-9f36-528ebbe35e4a",
        "readOnly": true,
```

```
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/d4f2a88d-5f9c-4807-
b71d-4d0ee5225156",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

GenerateRandom

The following example shows a log file created by calling `GenerateRandom`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:37Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "GenerateRandom",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": null,
        "responseElements": null,
        "requestID": "df1e3de6-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "239cb9f7-ae05-4c94-9221-6ea30eef0442",
        "readOnly": true,
        "resources": [],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

GetKeyPolicy

The following example shows a log file generated by calling `GetKeyPolicy`.

```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
```

```
        "eventTime": "2014-11-04T00:50:30Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "GetKeyPolicy",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "keyId": "arn:aws:kms:us-east-1:123456789012:key/e923fe55-
d3ef-4f9c-89a1-2752f98c3a70",
            "policyName": "default"
        },
        "responseElements": null,
        "requestID": "93746dd6-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "4aa7e4d5-d047-452a-a5a6-2cce282a7e82",
        "readOnly": true,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/e923fe55-
d3ef-4f9c-89a1-2752f98c3a70",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```


ListAliases

The following example shows a log file generated by calling `ListAliases`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:51:45Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "ListAliases",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "limit": 5,
            "marker":
 "eyJiIjoiYWxpYXMvZTU0Y2MxOTMtYTMwNC00YzEwLTliZWItYTJjZjA3NjA2OTJhIiwiYSI6ImFsaWFzL2U1NGNjMTkzLWEzMDQtN
        },
        "responseElements": null,
        "requestID": "bfe6c190-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "a27dda7b-76f1-4ac3-8b40-42dfba77bcd6",
        "readOnly": true,
        "resources": [],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```

ListGrants

The following example shows a log file generated by calling `ListGrants`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:49Z",
        "eventSource": "kms.amazonaws.com",
        "eventName": "ListGrants",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "keyId": "arn:aws:kms:us-east-1:123456789012:key/ea22a751-
e707-40d0-92ac-13a28fa9eb11",
            "marker":
 "eyJncmFudElkIjoiMWY4M2U2ZmM0YTY2NDgxYjQ2Yzc4MTdhM2Y4YmQwMDFkZDNiYmQ1MGVlYTMyY2RmOWFiNWY1Nzc1NDNjYmNmM
\u003d\u003d",
            "limit": 10
        },
        "responseElements": null,
        "requestID": "e5c23960-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "d24380f5-1b20-4253-8e92-dd0492b3bd3d",
        "readOnly": true,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/ea22a751-
e707-40d0-92ac-13a28fa9eb11",
            "accountId": "123456789012"
        }],
        "eventType": "AwsApiCall",
        "recipientAccountId": "123456789012"
    }
  ]
}
```


ReEncrypt

The following example shows a log file generated by calling `ReEncrypt`.


```
{
    "Records": [
    {
        "eventVersion": "1.02",
        "userIdentity": {
            "type": "IAMUser",
            "principalId": "EX_PRINCIPAL_ID",
            "arn": "arn:aws:iam::123456789012:user/Alice",
            "accountId": "123456789012",
            "accessKeyId": "EXAMPLE_KEY_ID",
            "userName": "Alice"
        },
        "eventTime": "2014-11-04T00:52:19Z",
```

```
        "eventSource": "kms.amazonaws.com",
        "eventName": "ReEncrypt",
        "awsRegion": "us-east-1",
        "sourceIPAddress": "192.0.2.0",
        "userAgent": "AWS Internal",
        "requestParameters": {
            "destinationKeyId": "arn:aws:kms:us-east-1:123456789012:key/116b8956-
a086-40f1-96d6-4858ef794ba5"
        },
        "responseElements": null,
        "requestID": "d3eeee63-63bc-11e4-bc2b-4198b6150d5c",
        "eventID": "627c13b4-8791-4983-a80b-4c28807b964c",
        "readOnly": false,
        "resources": [{
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/ff0c0fc1-cbaa-41ab-
a267-69481da8a4c8",
            "accountId": "123456789012"
        },
        {
            "ARN": "arn:aws:kms:us-east-1:123456789012:key/116b8956-
a086-40f1-96d6-4858ef794ba5",
            "accountId": "123456789012"
        }],
        "eventType": "AwsServiceEvent",
        "recipientAccountId": "123456789012"
    }
  ]
}
```


Amazon EC2 example one

The following example demonstrates an IAM user creating an encrypted volume using the default volume key in the Amazon EC2 management console.

The following example shows a CloudTrail log entry that demonstrates the user Alice creating an encrypted volume using a default volume key in AWS EC2 Management Console. The EC2 log file record includes a `volumeId` field with a value of `"vol-13439757"`. The AWS KMS record contains an `encryptionContext` field with a value of `"aws:ebs:id": "vol-13439757"`. Similarly, the `principalId` and `accountId` between the two records match. The records reflect the fact that creating an encrypted volume generates a data key that is used to encrypt the volume content.


```
{
  "Records": [
     {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T20:40:44Z"
          }
        },
        "invokedBy": "signin.amazonaws.com"
      },
      "eventTime": "2014-11-05T20:50:18Z",
      "eventSource": "ec2.amazonaws.com",
```

```
      "eventName": "CreateVolume",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "72.72.72.72",
      "userAgent": "signin.amazonaws.com",
      "requestParameters": {
        "size": "10",
        "zone": "us-east-1a",
        "volumeType": "gp2",
        "encrypted": true
      },
      "responseElements": {
        "volumeId": "vol-13439757",
        "size": "10",
        "zone": "us-east-1a",
        "status": "creating",
        "createTime": 1415220618876,
        "volumeType": "gp2",
        "iops": 30,
        "encrypted": true
      },
      "requestID": "1565210e-73d0-4912-854c-b15ed349e526",
      "eventID": "a3447186-135f-4b00-8424-bc41f1a93b4f",
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T20:40:44Z"
          }
        },
        "invokedBy": "AWS Internal"
      },
      "eventTime": "2014-11-05T20:50:19Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "GenerateDataKeyWithoutPlaintext",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "AWS Internal",
      "userAgent": "AWS Internal",
      "requestParameters": {
        "encryptionContext": {
          "aws:ebs:id": "vol-13439757"
        },
        "numberOfBytes": 64,
        "keyId": "alias/aws/ebs"
      },
      "responseElements": null,
      "requestID": "create-123456789012-758241111-1415220618",
      "eventID": "4bd2a696-d833-48cc-b72c-05e61b608399",
      "readOnly": true,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-east-1:123456789012:key/
e29ddfd4-1bf6-4e1b-8ecb-08216bd70d07",
          "accountId": "123456789012"
        }
      ],
```

```
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    }
  ]
}
```

Amazon EC2 example two

In the following example, an IAM user running an Amazon EC2 instance creates and mounts a data volume that is encrypted under an AWS KMS customer master key (CMK). This action generates multiple CloudTrail log records.

When the volume is created, Amazon EC2, acting on behalf of the customer, gets an encrypted data key from AWS KMS (`GenerateDataKeyWithoutPlaintext`). Then it creates a grant (`CreateGrant`) that allows it to decrypt the data key. When the volume is mounted, Amazon EC2 calls AWS KMS to decrypt the data key (`Decrypt`).

The `instanceId` of the Amazon EC2 instance, `"i-81e2f56c"`, appears in the `RunInstances` event. The same instance ID qualifies the `granteePrincipal` of the grant that is created (`"123456789012:aws:ec2-infrastructure:i-81e2f56c"`) and the assumed role that is the principal in the `Decrypt` call (`"arn:aws:sts::123456789012:assumed-role/aws:ec2-infrastructure/i-81e2f56c"`).

The key ARN (p. 12) of the CMK that protects the data volume, `arn:aws:kms:us-east-1:123456789012:key/e29ddfd4-1bf6-4e1b-8ecb-08216bd70d07`, appears in all three AWS KMS calls (`CreateGrant`, `GenerateDataKeyWithoutPlaintext`, and `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",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T21:34:36Z"
          }
        },
        "invokedBy": "signin.amazonaws.com"
      },
      "eventTime": "2014-11-05T21:35:27Z",
      "eventSource": "ec2.amazonaws.com",
      "eventName": "RunInstances",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "72.72.72.72",
      "userAgent": "signin.amazonaws.com",
      "requestParameters": {
        "instancesSet": {
          "items": [
            {
              "imageId": "ami-b66ed3de",
              "minCount": 1,
              "maxCount": 1
            }
          ]
        },
```

```
        "groupSet": {
          "items": [
            {
              "groupId": "sg-98b6e0f2"
            }
          ]
        },
        "instanceType": "m3.medium",
        "blockDeviceMapping": {
          "items": [
            {
              "deviceName": "/dev/xvda",
              "ebs": {
                "volumeSize": 8,
                "deleteOnTermination": true,
                "volumeType": "gp2"
              }
            },
            {
              "deviceName": "/dev/sdb",
              "ebs": {
                "volumeSize": 8,
                "deleteOnTermination": false,
                "volumeType": "gp2",
                "encrypted": true
              }
            }
          ]
        },
        "monitoring": {
          "enabled": false
        },
        "disableApiTermination": false,
        "instanceInitiatedShutdownBehavior": "stop",
        "clientToken": "XdKUT141516171819",
        "ebsOptimized": false
      },
      "responseElements": {
        "reservationId": "r-5ebc9f74",
        "ownerId": "123456789012",
        "groupSet": {
          "items": [
            {
              "groupId": "sg-98b6e0f2",
              "groupName": "launch-wizard-2"
            }
          ]
        },
        "instancesSet": {
          "items": [
            {
              "instanceId": "i-81e2f56c",
              "imageId": "ami-b66ed3de",
              "instanceState": {
                "code": 0,
                "name": "pending"
              },
              "amiLaunchIndex": 0,
              "productCodes": {

              },
              "instanceType": "m3.medium",
              "launchTime": 1415223328000,
              "placement": {
                "availabilityZone": "us-east-1a",
                "tenancy": "default"
```

```
            },
            "monitoring": {
              "state": "disabled"
            },
            "stateReason": {
              "code": "pending",
              "message": "pending"
            },
            "architecture": "x86_64",
            "rootDeviceType": "ebs",
            "rootDeviceName": "/dev/xvda",
            "blockDeviceMapping": {

            },
            "virtualizationType": "hvm",
            "hypervisor": "xen",
            "clientToken": "XdKUT1415223327917",
            "groupSet": {
              "items": [
                {
                  "groupId": "sg-98b6e0f2",
                  "groupName": "launch-wizard-2"
                }
              ]
            },
            "networkInterfaceSet": {

            },
            "ebsOptimized": false
          }
        ]
      }
    },
    "requestID": "41c4b4f7-8bce-4773-bf0e-5ae3bb5cbce2",
    "eventID": "cd75a605-2fee-4fda-b847-9c3d330ebaae",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
  },
  {
    "eventVersion": "1.02",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "EX_PRINCIPAL_ID",
      "arn": "arn:aws:iam::123456789012:user/Alice",
      "accountId": "123456789012",
      "accessKeyId": "EXAMPLE_KEY_ID",
      "userName": "Alice",
      "sessionContext": {
        "attributes": {
          "mfaAuthenticated": "false",
          "creationDate": "2014-11-05T21:34:36Z"
        }
      },
      "invokedBy": "AWS Internal"
    },
    "eventTime": "2014-11-05T21:35:35Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
      "constraints": {
        "encryptionContextSubset": {
          "aws:ebs:id": "vol-f67bafb2"
        }
```

```
        },
        "granteePrincipal": "123456789012:aws:ec2-infrastructure:i-81e2f56c",
        "keyId": "arn:aws:kms:us-east-1:123456789012:key/
e29ddfd4-1bf6-4e1b-8ecb-08216bd70d07"
      },
      "responseElements": {
        "grantId": "6caf442b4ff8a27511fb6de3e12cc5342f5382112adf75c1a91dbd221ec356fe"
      },
      "requestID": "41c4b4f7-8bce-4773-bf0e-5ae3bb5cbce2",
      "eventID": "c1ad79e3-0d3f-402a-b119-d5c31d7c6a6c",
      "readOnly": false,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-east-1:123456789012:key/
e29ddfd4-1bf6-4e1b-8ecb-08216bd70d07",
          "accountId": "123456789012"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    {
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "IAMUser",
        "principalId": "EX_PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "EXAMPLE_KEY_ID",
        "userName": "Alice",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T21:34:36Z"
          }
        },
        "invokedBy": "AWS Internal"
      },
      "eventTime": "2014-11-05T21:35:32Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "GenerateDataKeyWithoutPlaintext",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "AWS Internal",
      "userAgent": "AWS Internal",
      "requestParameters": {
        "encryptionContext": {
          "aws:ebs:id": "vol-f67bafb2"
        },
        "numberOfBytes": 64,
        "keyId": "alias/aws/ebs"
      },
      "responseElements": null,
      "requestID": "create-123456789012-758247346-1415223332",
      "eventID": "ac3cab10-ce93-4953-9d62-0b6e5cba651d",
      "readOnly": true,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-east-1:123456789012:key/
e29ddfd4-1bf6-4e1b-8ecb-08216bd70d07",
          "accountId": "123456789012"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    },
    {
```

```
      "eventVersion": "1.02",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "123456789012:aws:ec2-infrastructure:i-81e2f56c",
        "arn": "arn:aws:sts::123456789012:assumed-role/aws:ec2-infrastructure/i-81e2f56c",
        "accountId": "123456789012",
        "accessKeyId": "",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2014-11-05T21:35:38Z"
          },
          "sessionIssuer": {
            "type": "Role",
            "principalId": "123456789012:aws:ec2-infrastructure",
            "arn": "arn:aws:iam::123456789012:role/aws:ec2-infrastructure",
            "accountId": "123456789012",
            "userName": "aws:ec2-infrastructure"
          }
        }
      },
      "eventTime": "2014-11-05T21:35:47Z",
      "eventSource": "kms.amazonaws.com",
      "eventName": "Decrypt",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "172.172.172.172",
      "requestParameters": {
        "encryptionContext": {
          "aws:ebs:id": "vol-f67bafb2"
        }
      },
      "responseElements": null,
      "requestID": "b4b27883-6533-11e4-b4d9-751f1761e9e5",
      "eventID": "edb65380-0a3e-4123-bbc8-3d1b7cff49b0",
      "readOnly": true,
      "resources": [
        {
          "ARN": "arn:aws:kms:us-east-1:123456789012:key/
e29ddfd4-1bf6-4e1b-8ecb-08216bd70d07",
          "accountId": "123456789012"
        }
      ],
      "eventType": "AwsApiCall",
      "recipientAccountId": "123456789012"
    }
  ]
}
```

Programming the AWS KMS API

You can use the AWS KMS API to perform the following actions, and more.

- Create, describe, list, enable, and disable keys.
- Create, delete, list, and update aliases.
- Encrypt, decrypt, and re-encrypt content.
- Set, list, and retrieve key policies.
- Create, retire, revoke, and list grants.
- Retrieve key rotation status.
- Update key descriptions.
- Generate data keys with or without plaintext.
- Generate random data.

The sample code in the following topics show how to use the AWS SDKs to call the AWS KMS API.

For information about using the AWS KMS console to perform some of these tasks, see Getting started (p. 24).

Topics

- Creating a client (p. 324)
- Working with keys (p. 325)
- Encrypting and decrypting data keys (p. 336)
- Working with key policies (p. 343)
- Working with grants (p. 351)
- Working with aliases (p. 360)

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.standard().build();
```

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 KMSClient class in the AWS SDK for PHP API Reference.

```
// Create a KMSClient
$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 (p. 3) (CMKs), and to generate data keys (p. 5).

Topics

- Creating a customer master key (p. 325)
- Generating a data key (p. 327)
- Viewing a custom master key (p. 329)
- Getting key IDs and key ARNs of CMKs (p. 331)
- Enabling customer master keys (p. 333)
- Disabling customer master keys (p. 334)

Creating a customer master key

To create a customer master key (p. 3) (CMK), use the CreateKey operation. The examples in this section create a symmetric CMK. The `Description` parameter used in these examples is optional.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

For help with creating CMKs in the AWS KMS console, see Creating keys (p. 24).

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) => {
  ...
});
```

PowerShell

To create a CMK in PowerShell, use the New-KmsKey cmdlet.

```
# Create a CMK

$desc = 'Key for protecting critical data'
New-KmsKey -Description $desc
```

Generating a data key

To generate a symmetric data key, use the GenerateDataKey operation. This operation takes a symmetric CMK and returns a plaintext data key and a copy of that data key encrypted under the CMK that you specified. You must specify either a `KeySpec` or `NumberOfBytes` (but not both) in each command.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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

response = kmsClient.generate_data_key({
  key_id: keyId,
  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;
      ...
  }
});
```

PowerShell

To generate a symmetric data key, use the New-KMSDataKey cmdlet.

In the output, the plaintext key (in the `Plaintext` property) and the encrypted key (in the `CiphertextBlob` property) are MemoryStream objects. To convert them to strings, use the methods of the `MemoryStream` class, or a cmdlet or function that converts `MemoryStream` objects to strings, such as the ConvertFrom-MemoryStream and ConvertFrom-Base64 functions in the Convert module.

```
# 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'

$response = New-KmsDataKey -KeyId $keyId -KeySpec $keySpec
$plaintextKey = $response.Plaintext
$encryptedKey = $response.CiphertextBlob
```

Viewing a custom master key

To get detailed information about a customer master key (CMK), including the CMK ARN and key state (p. 233), use the DescribeKey operation.

`DescribeKey` does not get aliases. To get aliases, use the ListAliases operation. For examples, see Working with aliases (p. 360).

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

For help with viewing CMKs in the AWS KMS console, see Viewing keys (p. 29).

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) => {
  ...
});
```

PowerShell

To get detailed information about a CMK, use the Get-KmsKey cmdlet.

```
# 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'
Get-KmsKey -KeyId $keyId
```

Getting key IDs and key ARNs of CMKs

To get the key IDs (p. 12) and key ARNs (p. 12) of the customer master keys, use the ListKeys operation. These examples use the optional `Limit` parameter, which sets the maximum number of CMKs returned in each call. For help identifying a CMK in an AWS KMS operations, see Key identifiers (KeyId) (p. 11).

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

For help with finding key IDs and key ARNs in the AWS KMS console, see Finding the key ID and ARN (p. 40).

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) => {
  ...
});
```

PowerShell

To get the key ID and key ARN of all CMKs in the account and Region, use the Get-KmsKeyList cmdlet.

To limit the number of output objects, this example uses the Select-Object cmdlet, instead of the `Limit` parameter, which is being deprecated in list cmdlets. For help with paginating output in AWS Tools for PowerShell, see Output Pagination with AWS Tools for PowerShell.

```
# List CMKs in this account

$limit = 10
Get-KmsKeyList | Select-Object -First $limit
```

Enabling customer master keys

To enable a disabled customer master key (CMK), use the EnableKey operation.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

For help with enabling and disabling CMKs in the AWS KMS console, see Enabling and disabling keys (p. 51).

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) => {
  ...
});
```

PowerShell

To enable a CMK, use the Enable-KmsKey cmdlet.

```
# 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'
Enable-KmsKey -KeyId $keyId
```

Disabling customer master keys

To disable a CMK, use the DisableKey operation. Disabling a CMK prevents it from being used in cryptographic operations (p. 10).

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

For help with enabling and disabling CMKs in the AWS KMS console, see Enabling and disabling keys (p. 51).

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) => {
  ...
});
```

PowerShell

To disable a CMK, use the Disable-KmsKey cmdlet.

```
# 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'
Disable-KmsKey -KeyId $keyId
```

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. 5). They use an AWS KMS customer master key (p. 3) (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. 336)
- Decrypting a data key (p. 339)
- Re-encrypting a data key under a different customer master key (p. 341)

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 different Region and want to encrypt its data key with a CMK in the new Region.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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;
      ...
  }
});
```

PowerShell

To encrypt a data key under an AWS KMS CMK, use the Invoke-KMSEncrypt cmdlet. It returns the ciphertext as a `MemoryStream` (System.IO.MemoryStream) object. You can use the `MemoryStream` object as the input to the Invoke-KMSDecrypt cmdlet.

AWS KMS also returns data keys as `MemoryStream` objects. In this example, to simulate a plaintext data key, we create a byte array and write it to a `MemoryStream` object.

Note that the `Plaintext` parameter of `Invoke-KMSEncrypt` takes a byte array (`byte[]`); it does not require a `MemoryStream` object. Beginning in AWSPowerShell version 4.0, parameters in all AWSPowerShell modules that take byte arrays and `MemoryStream` objects accept byte arrays, `MemoryStream` objects, strings, string arrays, and `FileInfo` (System.IO.FileInfo) objects. You can pass any of these types to `Invoke-KMSEncrypt`.

```
# 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'

# Simulate a data key
```

```
 # Create a byte array
[byte[]] $bytes = 1, 2, 3, 4, 5, 6, 7, 8, 9, 0

 # Create a MemoryStream
$plaintext = [System.IO.MemoryStream]::new()

 # Add the byte array to the MemoryStream
$plaintext.Write($bytes, 0, $bytes.length)

# Encrypt the simulated data key
$response = Invoke-KMSEncrypt -KeyId $keyId -Plaintext $plaintext

# Get the ciphertext from the response
$ciphertext = $response.CiphertextBlob
```

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, or the `PrivateKeyCiphertextBlob` field from a GenerateDataKeyPair or GenerateDataKeyPairWithoutPlaintext response. You can also use the `Decrypt` operation to decrypt data encrypted outside of AWS KMS by the public key in an asymmetric CMK.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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;
    ...
  }
});
```

PowerShell

To decrypt a data key, use the Invoke-KMSEncrypt cmdlet.

This cmdlet returns the plaintext as a `MemoryStream` (System.IO.MemoryStream) object. To convert it to a byte array, use cmdlets or functions that convert `MemoryStream` objects to byte arrays, such as the functions in the Convert module.

Because this example uses the ciphertext that an AWS KMS encryption cmdlet returned, it uses a `MemoryStream` object for the value of the `CiphertextBlob` parameter. However, the `CiphertextBlob` parameter of `Invoke-KMSDecrypt` takes a byte array (`byte[]`); it does not require a `MemoryStream` object. Beginning in AWSPowerShell version 4.0, parameters in all AWSPowerShell modules that take byte arrays and `MemoryStream` objects accept byte arrays, `MemoryStream` objects, strings, string arrays, and `FileInfo` (System.IO.FileInfo) objects. You can pass any of these types to `Invoke-KMSDecrypt`.

```
# Decrypt a data key

[System.IO.MemoryStream]$ciphertext = Read-Host 'Place your cipher text blob here'

$response = Invoke-KMSDecrypt -CiphertextBlob $ciphertext
$plaintext = $response.Plaintext
```

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, or the `PrivateKeyCiphertextBlob` field from a GenerateDataKeyPair or GenerateDataKeyPairWithoutPlaintext response. You can also use the `ReEncrypt` operation to re-encrypt data encrypted outside of AWS KMS by the public key in an asymmetric CMK.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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) => {
  ...
});
```

PowerShell

To re-encrypt a ciphertext under the same or a different CMK, use the Invoke-KMSReEncrypt cmdlet.

Because this example uses the ciphertext that an AWS KMS encryption cmdlet returned, it uses a `MemoryStream` object for the value of the `CiphertextBlob` parameter. However, the `CiphertextBlob` parameter of `Invoke-KMSReEncrypt` takes a byte array (`byte[]`); it does not require a `MemoryStream` object. Beginning in AWSPowerShell version 4.0, parameters in all AWSPowerShell modules that take byte arrays and `MemoryStream` objects accept byte arrays, `MemoryStream` objects, strings, string arrays, and `FileInfo` (System.IO.FileInfo) objects. You can pass any of these types to `Invoke-KMSReEncrypt`.

```
# Re-encrypt a data key

[System.IO.MemoryStream]$ciphertextBlob = Read-Host 'Place your cipher text blob here'

# Replace the following fictitious CMK ARN with a valid CMK ID or ARN
$destinationKeyId = 'arn:aws:kms:us-west-2:111122223333:key/0987dcba-09fe-87dc-65ba-
ab0987654321'

$response = Invoke-KMSReEncrypt -Ciphertext $ciphertextBlob -DestinationKeyId
 $destinationKeyId
$reEncryptedCiphertext = $response.CiphertextBlob
```

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. 56).

Topics

- Listing key policy names (p. 343)
- Getting a key policy (p. 345)
- Setting a key policy (p. 347)

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**.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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) => {
  ...
});
```

PowerShell

To list the name of the default key policy, use the Get-KMSKeyPolicyList cmdlet.

```
# 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 = Get-KMSKeyPolicyList -KeyId $keyId
```

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**.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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) => {
  ...
});
```

PowerShell

To get the key policy for a CMK, use the Get-KMSKeyPolicy cmdlet. This cmdlet returns the key policy as a string (System.String) that you can use in a Write-KMSKeyPolicy (`PutKeyPolicy`) command. To convert the policies in the JSON string to `PSCustomObject` objects, use the ConvertFrom-JSON cmdlet.

```
# 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 = Get-KMSKeyPolicy -KeyId $keyId -PolicyName $policyName
```

Setting a key policy

To create or replace the key policy for a CMK, use the PutKeyPolicy operation.

`PutKeyPolicy` requires a policy name. The only valid policy name is **default**.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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
\n    "Version": "2012-10-17",
\n    "Statement": [{
\n        "Sid": "Allow access for ExampleUser",
\n        "Effect": "Allow",
\n        "Principal": {"AWS": "arn:aws:iam::111122223333:user/ExampleUser"},
\n        "Action": [
\n            "kms:Encrypt",
\n            "kms:GenerateDataKey*",
\n            "kms:Decrypt",
\n            "kms:DescribeKey",
\n            "kms:ReEncrypt*"
\n        ],
\n        "Resource": "*"
\n    }]
\n}\n"

response = kmsClient.put_key_policy({
  key_id: keyId,
  policy: policy,
  policy_name: policyName
})
```

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": "*" }
        ]
    } '
]);
```

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) => {
  ...
});
```

PowerShell

To set a key policy for a CMK, use the Write-KMSKeyPolicy cmdlet. This cmdlet doesn't return any output. To verify that the command was effective, use the Get-KMSKeyPolicy cmdlet.

The `Policy` parameter takes a string. Enclose the string in single quotes to make it a literal string. You don't have to use continuation characters or escape characters in the literal string.

```
# 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 = '{
    "Version": "2012-10-17",
    "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": "*"
        }]
    }'

Write-KMSKeyPolicy -KeyId $keyId -PolicyName $policyName -Policy $policy
```

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). For more details about using grants in AWS KMS, see Using grants (p. 126).

Topics

- Creating a grant (p. 352)
- Viewing a grant (p. 354)
- Retiring a grant (p. 356)
- Revoking a grant (p. 358)

Creating a grant

To create a grant for an AWS KMS customer master key, use the CreateGrant operation. The response includes only the grant ID and grant token. To get detailed information about the grant, use the ListGrants operation, as shown in Viewing a grant (p. 354).

These examples create a grant that allows Alice, an IAM user in the account, to call the GenerateDataKey operation on the CMK identified by the `KeyId` parameter.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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.GenerateDataKey.toString();

CreateGrantRequest request = new CreateGrantRequest()
    .withKeyId(keyId)
    .withGranteePrincipal(granteePrincipal)
    .withOperations(operation);

CreateGrantResult result = kmsClient.createGrant(request);
```

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.GenerateDataKey;

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 = ['GenerateDataKey']

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 = ['GenerateDataKey']

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 = ['GenerateDataKey']

$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: ["GenerateDataKey"];
kmsClient.createGrant({ KeyId, GranteePrincipal, Operations }, (err, data) => {
  ...
});
```

PowerShell

To create a grant, use the New-KMSGrant cmdlet.

```
# 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 = 'GenerateDataKey'

$response = New-KMSGrant -GranteePrincipal $granteePrincipal -KeyId $keyId -Operation
 $operation
```

Viewing a grant

To get detailed information about the grants on an AWS KMS customer master key, use the ListGrants operation. These examples use the optional `Limits` parameter, which determines how many grants the operation returns.

> **Note**
> The `GranteePrincipal` field in the `ListGrants` response usually contains the grantee principal of the grant. However, when the grantee principal in the grant is an AWS service, the `GranteePrincipal` field contains the service principal, which might represent several different grantee principals.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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) => {
  ...
});
```

PowerShell

To view the details of all AWS KMS grants for a CMK, use the Get-KMSGrantList cmdlet.

To limit the number of output objects, this example uses the Select-Object cmdlet, instead of the `Limit` parameter, which is being deprecated in list cmdlets. For help with paginating output in AWS Tools for PowerShell, see Output Pagination with AWS Tools for PowerShell.

```
# 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

$response = Get-KMSGrantList -KeyId $keyId | Select-Object -First $limit
```

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.

To retire a grant, provide the grant token, or both the grant ID and CMK ID. For this operation, the CMK ID must be Amazon Resource Name (ARN) of the CMK (p. 40). The grant token is returned by the CreateGrant operation. The grant ID is returned by the CreateGrant and ListGrants operations.

RetireGrant doesn't return a response. To verify that it was effective, use the ListGrants operation.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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) => {
  ...
});
```

PowerShell

To retire a grant, use the Disable-KMSGrant cmdlet. To get the grant token, use the New-KMSGrant cmdlet. The `GrantToken` parameter takes a string, so you don't need to convert output that the Read-Host cmdlet returns.

```
# Retire a grant

$grantToken = Read-Host -Message Place your grant token here
```

```
Disable-KMSGrant -GrantToken $grantToken
```

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.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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";

// &fake-grant-id;
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";

// &fake-grant-id;
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'

# &fake-grant-id;
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'

# &fake-grant-id;
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';

// Replace the following fictitious grant ID with a valid one
$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';

// Replace the following fictitious grant ID with a valid one
const GrantId = 'grant1';
kmsClient.revokeGrant({ GrantId, KeyId }, (err, data) => {
  ...
});
```

PowerShell

To revoke a grant, use the Revoke-KMSGrant cmdlet.

```
# 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'

# Replace the following fictitious grant ID with a valid one
$grantId = 'grant1'

Revoke-KMSGrant -KeyId $keyId -GrantId $grantId
```

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. 3). You can use an alias name (p. 14) or alias ARN (p. 12) as the value of the `KeyId` parameter in cryptographic operations (p. 10), such as `Encrypt` and `Decrypt`.

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, and each alias must be associated with a CMK in its 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 create, list, 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.

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.

```
// Alias ARN
arn:aws:kms:us-west-2:111122223333:alias/ExampleAlias

// Alias name
alias/ExampleAlias
```

Topics

- Creating an alias (p. 360)
- Listing aliases (p. 362)
- Updating an alias (p. 366)
- Deleting an alias (p. 368)

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.

The alias name must begin with `alias/` followed by a name, such as `alias/ExampleAlias`. It can contain only alphanumeric characters, forward slashes (/), underscores (_), and dashes (-). The alias name cannot begin with `alias/aws/`. The `alias/aws/` prefix is reserved for AWS managed CMKs (p. 3).

The `CreateAlias` operation does not return any output. To verify that the alias was created, use the ListAliases operation.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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) => {
  ...
});
```

PowerShell

To create an alias, use the New-KMSAlias cmdlet. The alias name is case-sensitive.

```
# 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'

New-KMSAlias -TargetKeyId $targetKeyId -AliasName $aliasName
```

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. 3), and aliases that AWS created and associated with your AWS managed CMKs (p. 3). 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.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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) => {
  ...
});
```

PowerShell

To list the aliases in the account and Region, use the Get-KMSAliasList cmdlet.

To limit the number of output objects, this example uses the Select-Object cmdlet, instead of the `Limit` parameter, which is being deprecated in list cmdlets. For help with paginating output in AWS Tools for PowerShell, see Output Pagination with AWS Tools for PowerShell.

```
# List the aliases in this AWS account
$limit = 10

$result = Get-KMSAliasList | Select-Object -First $limit
```

To list only the aliases that are associated with a particular CMK, use the `KeyId` parameter. Its value can be the key ID (p. 12) or key ARN (p. 12) 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) => {
  ...
});
```

PowerShell

To list the aliases for a CMK, use the `KeyId` parameter of the Get-KMSAliasList cmdlet.

```
# 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 = Get-KmsAliasList -KeyId $keyId
```

Updating an alias

To associate an existing alias with a different CMK, use the UpdateAlias operation.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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-2:111122223333: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-2:111122223333: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-2:111122223333: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) => {
  ...
});
```

PowerShell

To change the CMK that an alias points to, use the Update-KMSAlias cmdlet. The alias name is case-sensitive.

The `Update-KMSAlias` cmdlet does not return any output. To verify that the command worked, use the Get-KMSAliasList cmdlet.

```
# 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'

Update-KMSAlias -AliasName $aliasName -TargetKeyID $keyId
```

Deleting an alias

To delete an alias, use the DeleteAlias operation. Deleting an alias has no effect on the associated CMK.

In languages that require a client object, these examples use the AWS KMS client object that you created in Creating a client (p. 324).

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) => {
  ...
});
```

PowerShell

To delete an alias, use the Remove-KMSAlias cmdlet. The alias name is case-sensitive.

Because this cmdlet permanently deletes the alias, PowerShell prompts you to confirm the command. The `ConfirmImpact` is `High`, so you cannot use a `ConfirmPreference` to suppress this prompt. If you must suppress the confirmation prompt, add the `Confirm` common parameter with a value of `$false`, for example: `-Confirm:$false`.

The `Remove-KMSAlias` cmdlet doesn't return any output. To verify that the command was effective, use the Get-KMSAliasList cmdlet.

```
# Delete an alias for a CMK

$aliasName = 'alias/projectKey1'
Remove-KMSAlias -AliasName $aliasName
```

Quotas

To make AWS KMS responsive and performant for all users, AWS KMS applies two types of quotas. Each quota is calculated independently for each Region of each AWS account.

> **Important**
> If you need to exceed a quota, you can request a quota increase in Service Quotas. Use the Service Quotas console or the RequestServiceQuotaIncrease operation. For details, see Requesting a quota increase in the *Service Quotas User Guide*. If Service Quotas for AWS KMS are not available in the AWS Region, please visit the AWS Support Center and create a case.
> For help requesting an increase in an AWS KMS quota, see Request an AWS KMS Quota Increase (p. 378).

- Resource quotas (p. 370): Limit the number of each type of AWS KMS resource.
- Request quotas (p. 372): Limit the number of requests for AWS KMS API operations in a specified interval.

Resource quotas

AWS KMS establishes resource quotas to ensure that it can provide fast and resilient service to all of our customers. Some resource quotas apply only to resources that you create, but not to resources that AWS services create for you. Resources that you use, but that aren't in your AWS account, such as AWS owned CMKs (p. 4), do not count against these quotas.

If you have reached a resource limit, requests to create an additional resource of that type generate an `LimitExceededException` error message.

The following table lists and describes the AWS KMS resource quotas in each AWS account and Region. If you need to exceed a quota, you can request a quota increase in Service Quotas. Use the Service Quotas console or the RequestServiceQuotaIncrease operation. For details, see Requesting a quota increase in the *Service Quotas User Guide*. If Service Quotas for AWS KMS are not available in the AWS Region, please visit the AWS Support Center and create a case.

For help requesting an increase in an AWS KMS quota, see Request an AWS KMS Quota Increase (p. 378).

Quota name	Default value	Applies to
Customer Master Keys (CMKs) (p. 371)	10,000	Customer managed CMKs
Aliases (p. 371)	10,000	Customer created aliases
Grants per CMK (p. 371)	10,000	Customer managed CMKs
Grants for a given principal per CMK (p. 371)	500	Customer managed CMKs AWS managed CMKs
Key policy document size (p. 372)	32 KB (32,768 bytes)	Customer managed CMKs AWS managed CMKs

In addition to resource quotas, AWS KMS uses request quotas to ensure the responsiveness of the service. For details, see the section called "Request quotas" (p. 372).

Customer master keys (CMKs): 10,000

You can have up to 10,000 customer managed CMKs (p. 4) in each Region of your AWS account. This quota applies to all symmetric and asymmetric customer managed CMKs regardless of their key state (p. 233). Each CMK — whether symmetric or asymmetric — is considered to be one resource. AWS managed CMKs (p. 4) and AWS owned CMKs (p. 4) do not count against this quota.

If you need to exceed this quota, request a quota increase in Service Quotas. 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 an AWS Region, manage them programmatically with the AWS SDKs or AWS Command Line Tools.

Aliases: 10,000

You can create up to 10,000 aliases in each Region of your account. Aliases that AWS creates in your account, such as aws/*<service-name>*, do not count against this quota.

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 increase your CMK resource quota, you might also need to increase your aliases resource quota. For help with requesting a quota increase, see Requesting a Quota Increase in the *Service Quotas User Guide*.

Grants per CMK: 10,000

Each customer managed CMK (p. 4) can have up to 10,000 grants, including the grants created by AWS services that are integrated with AWS KMS. This quota does not apply to AWS managed CMKs (p. 4) or AWS owned CMKs (p. 4).

One effect of this quota is that you cannot perform more than 10,000 grant-authorized operations that use the same CMK at the same time. After you reach the quota, you can create new grants on the CMK only when an active grant is retired or revoked.

For example, when you attach an Amazon Elastic Block Store (Amazon EBS) volume to an Amazon Elastic Compute Cloud (Amazon EC2) instance, the volume is decrypted so you can read it. To get permission to decrypt the data, Amazon EBS creates a grant for each volume. However, you cannot have more than 10,000 grants on each CMK. Therefore, if all of your Amazon EBS volumes use the same CMK, you cannot attach more than 10,000 volumes at one time.

Grants (p. 126) are an alternative to key policy (p. 59). Like a key policy, a grant is attached to a CMK. You (or an AWS service integrated with AWS KMS) can use a grant to allow a principal to use or manage the CMK. 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

A CMK cannot have more than 500 grants for the same grantee principal. The *grantee principal* is the identity that gets the permissions in the grant.

This quota is calculated separately for each CMK in the account. It applies to customer managed CMKs (p. 4) and AWS managed CMKs (p. 4), but not to AWS owned CMKs (p. 4).

> **Note**
> Be careful when using the output from the ListGrants operation to calculate the number of grants with the same grantee principal.
> The `GranteePrincipal` field in the `ListGrants` response usually contains the grantee principal of the grant. However, when the grantee principal in the grant is an AWS service, the `GranteePrincipal` field contains the service principal, which might represent several different grantee principals.

This quota can have practical consequences for your use of AWS resource. For example, it can prevent you from attaching more than 500 Amazon EBS volumes to an Amazon EC2 instance at the same time. When you attach an Amazon EBS volume to an Amazon EC2 instance, EBS creates a grant that allows it to decrypt the volume so you can read it. Each grant is unique, but all of the grants have the same grantee principal. Because of this quota, if you use the same CMK for every volume, you cannot attach more than 500 volumes at once.

Key policy document size: 32 KB

The maximum length of each key policy document is 32 KB (32,768 bytes). If you use a larger policy document to create or update the key policy for a CMK, the operation fails.

If you must exceed this quota, request a quota increase in Service Quotas. For details, see Requesting a Quota Increase in the *Service Quotas User Guide*.

A key policy document (p. 60) 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 use a key policy document whenever you create or change a key policy by using the default view (p. 75) or policy view (p. 75) in the AWS Management Console, or the PutKeyPolicy operation. This quota applies to your key policy document, even if you use the default view (p. 75) in the AWS KMS console, where you don't edit the JSON statements directly.

Request quotas

AWS KMS establishes quotas for the number of API operations requested in each second. For a table that lists the per-second request quota for each API operation, see Request quotas for each AWS KMS API operation (p. 375).

When you exceed an API request quota, AWS KMS *throttles* the request, that is, it rejects an otherwise valid request and returns a `ThrottlingException` error like the following one. To respond, use a backoff and retry strategy.

```
You have exceeded the rate at which you may call KMS. Reduce the frequency of your calls.
(Service: AWSKMS; Status Code: 400; Error Code: ThrottlingException; Request ID: <ID>
```

The request quotas differ with the API operation, the AWS Region, and other factors, such as the CMK type.

> **Note**
> If you need to exceed a quota, you can request a quota increase in Service Quotas. Use the Service Quotas console or the RequestServiceQuotaIncrease operation. For details, see Requesting a quota increase in the *Service Quotas User Guide*. If Service Quotas for AWS KMS are not available in the AWS Region, please visit the AWS Support Center and create a case.
> For help requesting an increase in an AWS KMS quota, see Request an AWS KMS Quota Increase (p. 378).

If you are exceeding the request quota for the GenerateDataKey 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.

In addition to request quotas, AWS KMS uses resource quotas to ensure capacity for all users. For details, see Resource quotas (p. 370).

Topics

- Applying request quotas (p. 373)
- Shared quotas for cryptographic operations (p. 373)
- API requests made on your behalf (p. 374)
- Cross-account requests (p. 374)
- Custom key store quota (p. 374)
- Request quotas for each AWS KMS API operation (p. 375)

Applying request quotas

When reviewing request quotas, keep in mind the following information.

- Request quotas apply to both customer managed CMKs (p. 4) and AWS managed CMKs (p. 4). The use of AWS owned CMKs (p. 4) does not count against request quotas for your AWS account, even when they are used to protect resources in your account.
- Throttling is based on all requests on CMKs of all types in the Region. This total includes requests from all principals in the AWS account, including requests from AWS services on your behalf.
- Each request quota is calculated independently. For example, requests for the CreateKey operation have no effect on the request quota for the CreateAlias operation. If your `CreateAlias` requests are throttled, your `CreateKey` requests can still complete successfully.
- Although cryptographic operations share a quota, the shared quota is calculated independently of quotas for other operations. For example, calls to the Encrypt and Decrypt operations share a request quota, but that quota is independent of the quota for management operations, such as EnableKey. For example, in the Europe (London) Region, you can perform 10,000 cryptographic operations on symmetric CMKs *plus* 5 `EnableKey` operations per second without being throttled.

Shared quotas for cryptographic operations

AWS KMS cryptographic operations (p. 10) share request quotas. You can request any combination of the cryptographic operations that are supported by the CMK, just so the total number of cryptographic operations doesn't exceed the request quota for that type of CMK. The exceptions are GenerateDataKeyPair and GenerateDataKeyPairWithoutPlaintext, which share a separate quota.

The quotas for different types of CMKs are calculated independently. Each quota applies to all requests for these operations in the AWS account and Region with the given key type in each one-second interval.

- *Cryptographic operations (symmetric) request rate* is the shared request quota for cryptographic operations using symmetric CMKs in an account and region.

  For example, you might be using symmetric CMKs (p. 142) in an AWS Region with a shared quota of 10,000 requests per second. When you make 7,000 GenerateDataKey requests per second and 2,000 Decrypt requests per second, AWS KMS doesn't throttle your requests. However, when you make 9,500 `GenerateDataKey` requests and 1,000 Encrypt and requests per second, AWS KMS throttles your requests because they exceed the shared quota.
- *Cryptographic operations (RSA) request rate* is the shared request quota for cryptographic operations using RSA asymmetric CMKs (p. 146).

For example, with a request quota of 500 operations per second, you can make 200 Encrypt requests and 100 Decrypt requests with RSA CMKs that can encrypt and decrypt, plus 50 Sign requests and 150 Verify requests with RSA CMKs that can sign and verify.

- *Cryptographic operations (ECC) request rate* is the shared request quota for cryptographic operations using elliptic curve (ECC) asymmetric CMKs (p. 148).

  For example, with a request quota of 300 operations per second, you can make 100 Sign requests and 200 Verify requests with RSA CMKs that can sign and verify.

The quotas for different key types are also calculated independently. For example, in the Asia Pacific (Singapore) Region, if you use both symmetric and asymmetric CMKs, you can make up to 10,000 calls per second with symmetric CMKs *plus* up to 500 additional calls per second with your RSA asymmetric CMKs, *plus* up to 300 additional requests per second with your ECC-based CMKs.

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 quota 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 quota, so AWS KMS throttles the requests if you exceed a combined total of 5,500 (or 10,000 or 30,000 depending upon your AWS Region) 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, it'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, if an application in account A uses a CMK in account B, the CMK use applies only to the quotas in account A.

Custom key store quota

AWS KMS custom key stores support only symmetric CMKs. The cryptographic operations that use the CMKs in a custom key store (p. 184) share a request quota of 1,800 operations per second for each custom key store. However, not all operations use the quota equally. The `GenerateDataKey`, `GenerateDataKeyWithoutPlaintext`, and `GenerateRandom` operations use approximately three times as much of the per-second quota as the `Encrypt`, `Decrypt`, and `ReEncrypt` operations.

For example, if you are requesting only `Encrypt` and `Decrypt` operations, you can perform approximately 1,800 operations per second. If, instead, you request repeated `GenerateDataKey` operations, your performance might be closer to 600 operations per second. For applications patterns that consist of roughly equal numbers of `GenerateDataKey` and `Decrypt` operations, you can expect about 1,200 operations per second.

Unlike other AWS KMS quotas, the custom key store quota is not adjustable. You cannot increase it by using Service Quotas or by creating a case in AWS Support.

> **Note**
> If the AWS CloudHSM cluster that is associated with the custom key store is processing numerous commands, including those unrelated to the custom key store, you might get an AWS KMS `ThrottlingException` at a lower-than-expected rate. If this occurs, lower your request

rate to AWS KMS, reduce the unrelated load, or use a dedicated AWS CloudHSM cluster for your custom key store.

Request quotas for each AWS KMS API operation

This table lists the Service Quotas quota code and the default value for each AWS KMS request quota.

Quota name	Default value (per second)

`Cryptographic operations (symmetric) request rate`
 	
Applies to:
 	
• `Decrypt`
• `Encrypt`
• `GenerateDataKey`
• `GenerateDataKeyWithoutPlaintext`
• `GenerateRandom`
• `ReEncrypt`
	These shared quotas vary with the AWS Region and the type of CMK used in the request. Each quota is calculated separately.
	
	• 5,500 (shared)
	• 10,000 (shared) in the following Regions:
	  • US East (Ohio), us-east-2
	  • Asia Pacific (Singapore), ap-southeast-1
	  • Asia Pacific (Sydney), ap-southeast-2
	  • Asia Pacific (Tokyo), ap-northeast-1
	  • Europe (Frankfurt), eu-central-1
	  • Europe (London), eu-west-2
	• 30,000 (shared) in the following Regions:
	  • US East (N. Virginia), us-east-1
	  • US West (Oregon), us-west-2
	  • Europe (Ireland), eu-west-1
	
	Custom key stores quota (symmetric CMKs):
	
	• 1,800 (shared) for each custom key store. For details, see Custom key store quota (p. 374).

`Cryptographic operations (RSA) request rate` Applies to: • `Decrypt` • `Encrypt` • `ReEncrypt` • `Sign` • `Verify`	500 (shared) for RSA CMKs
`Cryptographic operations (ECC) request rate` Applies to: • `Sign` • `Verify`	300 (shared) for elliptic curve (ECC) CMKs
`CancelKeyDeletion request rate`	5
`ConnectCustomKeyStore request rate`	5

Quota name	Default value (per second)
`CreateAlias request rate`	5
`CreateCustomKeyStore request rate`	5
`CreateGrant request rate`	50
`CreateKey request rate`	5
`DeleteAlias request rate`	5
`DeleteCustomKeyStore request rate`	5
`DeleteImportedKeyMaterial request rate`	5
`DescribeCustomKeyStores request rate`	5
`DescribeKey request rate`	100
`DisableKey request rate`	5
`DisableKeyRotation request rate`	5
`DisconnectCustomKeyStore request rate`	5
`EnableKey request rate`	5
`EnableKeyRotation request rate`	5
`GenerateDataKeyPair (ECC_NIST_P256) request rate` Applies to: • `GenerateDataKeyPair` • `GenerateDataKeyPairWithoutPlaintext`	25
`GenerateDataKeyPair (ECC_NIST_P384) request rate` Applies to: • `GenerateDataKeyPair` • `GenerateDataKeyPairWithoutPlaintext`	10
`GenerateDataKeyPair (ECC_NIST_P521) request rate` Applies to: • `GenerateDataKeyPair` • `GenerateDataKeyPairWithoutPlaintext`	5
`GenerateDataKeyPair (ECC_SECG_P256K1) request rate` Applies to: • `GenerateDataKeyPair` • `GenerateDataKeyPairWithoutPlaintext`	25

Quota name	Default value (per second)
`GenerateDataKeyPair (RSA_2048) request rate` Applies to: • `GenerateDataKeyPair` • `GenerateDataKeyPairWithoutPlaintext`	1
`GenerateDataKeyPair (RSA_3072) request rate` Applies to: • `GenerateDataKeyPair` • `GenerateDataKeyPairWithoutPlaintext`	0.5 (1 in each 2-second interval)
`GenerateDataKeyPair (RSA_4096) request rate` Applies to: • `GenerateDataKeyPair` • `GenerateDataKeyPairWithoutPlaintext`	0.1 (1 in each 10-second interval)
`GetKeyPolicy request rate`	30
`GetKeyRotationStatus request rate`	30
`GetParametersForImport request rate`	0.25 (1 in each 4-second interval)
`GetPublicKey request rate`	5
`ImportKeyMaterial request rate`	5
`ListAliases request rate`	100
`ListGrants request rate`	100
`ListKeyPolicies request rate`	100
`ListKeys request rate`	100
`ListResourceTags request rate`	100
`ListRetirableGrants request rate`	5
`PutKeyPolicy request rate`	5
`RetireGrant request rate`	15
`RevokeGrant request rate`	15
`ScheduleKeyDeletion request rate`	5
`TagResource request rate`	5
`UntagResource request rate`	5
`UpdateAlias request rate`	5

Quota name	Default value (per second)
`UpdateCustomKeyStore request rate`	5
`UpdateKeyDescription request rate`	5

Request an AWS KMS Quota Increase

AWS KMS resource quotas and request quotas are adjustable, except for the custom key store quota (p. 374).

If you need to exceed a quota, you can request a quota increase in Service Quotas. Use the Service Quotas console or the RequestServiceQuotaIncrease operation. For details, see Requesting a quota increase in the *Service Quotas User Guide*. If Service Quotas for AWS KMS are not available in the AWS Region, please visit the AWS Support Center and create a case.

Using the Service Quotas console

To request an increase for an AWS KMS quota, you can use the Service Quotas console. For instructions, see Requesting a quota increase in the *Service Quotas User Guide*.

1. For service name, choose **AWS Key Management Service (AWS KMS)**.
2. Choose the quota name of the quota you want to increase. Use the detailed information about the quota to confirm that you have chosen the quota you want to increase.

   You can search for the quota name in the Service Quotas console. There are several pages of AWS KMS quotas. You can also find the quota names and descriptions of AWS KMS quotas in the resource quota (p. 370) and request quota (p. 372) tables.

   For example, to request an increase to the quota for cryptographic operations on a symmetric CMK, choose **Cryptographic operations (symmetric) request rate**.
3. Choose **Request quota increase**.

Using the Service Quotas API

To request an increase in an AWS KMS quota, you can use the Service Quotas API. The `RequestServiceQuotaIncrease` operation, which submits the request, requires the quota code for the quota. So begin by getting the quota code.

1. Find the quota name of the quota you want to increase. You can find the quota names and descriptions of AWS KMS quotas in the resource quota (p. 370) and request quota (p. 372) tables.
2. To get the quota code for an AWS KMS quota, use the ListServiceQuotas operation.

   Set `ServiceCode` to `kms`.

   The response includes the `QuotaName` and `QuotaCode` for each quota.

   For example, to get only the quota information for the `Cryptographic operations (RSA) request rate` quota, use a command like the following one. It uses the `query` parameter in the AWS Command Line Interface (AWS CLI) to get only the quota with the specified quota name.

```
$ aws service-quotas list-service-quotas \
        --service-code kms \
        --query 'Quotas[?QuotaName==`Cryptographic operations (RSA) request rate`]'
```

```
{
    "Quotas": [
        {
            "ServiceCode": "kms",
            "ServiceName": "AWS Key Management Service (AWS KMS)",
            "QuotaArn": "arn:aws:servicequotas:us-east-2:111122223333:kms/L-2AC98190",
            "QuotaCode": "L-2AC98190",
            "QuotaName": "Cryptographic operations (RSA) request rate",
            "Value": 500,
            "Unit": "None",
            "Adjustable": true,
            "GlobalQuota": false
        }
    ]
}
```

3. To request an increase for an AWS KMS quota, use the RequestServiceQuotaIncrease operation. To identify the quota, use the quota code.

   For example, the following command requests an increase in the `Cryptographic operations (RSA) request rate` quota to `700` requests per second. It uses the required quota code, `L-2AC98190`, to identify the quota.

   If the command completes successfully, the `Status` field displays the current status of the request. To get the updated status of the request, use the GetRequestedServiceQuotaChange, ListRequestedServiceQuotaChangeHistory or ListRequestedServiceQuotaChangeHistoryByQuota operations.

```
$ aws service-quotas request-service-quota-increase \
        --service-code kms \
        --quota-code L-2AC98190 \
        --desired-value 700

{
    "RequestedQuota": {
        "Id": "a12345",
        "ServiceCode": "kms",
        "ServiceName": "AWS Key Management Service (AWS KMS)",
        "QuotaCode": "L-2AC98190",
        "QuotaName": "Cryptographic operations (RSA) request rate",
        "DesiredValue": 700,
        "Status": "PENDING",
        "Created": 1580446904.067,
        "Requester": "{\"accountId\":\"111122223333\",\"callerArn\":
\"arn:aws:iam::111122223333:root\"}",
        "QuotaArn": "arn:aws:servicequotas:us-east-2:111122223333:kms/L-2AC98190",
        "GlobalQuota": false,
        "Unit": "None"
    }
}
```

Document history

This topic describes significant updates to the *AWS Key Management Service Developer Guide*.

Topics

- Recent updates (p. 380)
- Earlier updates (p. 381)

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 feature	Added support for asymmetric customer master keys and asymmetric data keys.	November 25, 2019
Updated feature	You can view the key policy of AWS managed CMKs in the AWS KMS console. This feature used to be limited to customer managed CMKs.	November 15, 2019
New feature	Explains how to use hybrid post-quantum key exchange algorithms in TLS for your calls to AWS KMS.	November 4, 2019
Quota change	Increased the resource quotas for some APIs that manage CMKs.	September 18, 2019
Quota change	Changed the resource quotas for customer master keys (CMKs), aliases, and grants per CMK.	March 27, 2019
Quota change	Changed the shared per-second request quota for cryptographic operations that use customer master keys (CMKs) in a custom key store.	March 7, 2019
New feature	Explains how to create and manage AWS KMS custom key stores. Each key store is backed by an AWS CloudHSM cluster that you own and control.	November 26, 2018

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
Quota change	Changed the shared request quota for use of 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. 49).	February 15, 2017
New content	Added documentation about Monitoring customer master keys (p. 296) and Monitoring with Amazon CloudWatch (p. 297).	August 31, 2016
New content	Added documentation about Importing key material (p. 159).	August 11, 2016
New content	Added the following documentation: Overview of managing access (p. 57), Using IAM policies (p. 77), AWS KMS API permissions reference (p. 87), and Using policy conditions (p. 98).	July 5, 2016

Change	Description	Date
Update	Updated portions of the documentation in the Authentication and access control (p. 56) chapter.	July 5, 2016
Update	Updated the Quotas (p. 370) page to reflect new default quotas.	May 31, 2016
Update	Updated the Quotas (p. 370) page to reflect new default quotas, and updated the Grant tokens (p. 23) documentation to improve clarity and accuracy.	April 11, 2016
New content	Added documentation about Allowing multiple IAM users to access a CMK (p. 76) and Using the IP address condition (p. 98).	February 17, 2016
Update	Updated the Using key policies in AWS KMS (p. 59) and Changing a key policy (p. 74) pages to improve clarity and accuracy.	February 17, 2016
Update	Updated the Getting started (p. 24) topic pages to improve clarity.	January 5, 2016
New content	Added documentation about How AWS CloudTrail uses AWS KMS (p. 238).	November 18, 2015
New content	Added instructions for Changing a key policy (p. 74).	November 18, 2015
Update	Updated the documentation about How Amazon Relational Database Service (Amazon RDS) uses AWS KMS (p. 263).	November 18, 2015
New content	Added documentation about How Amazon WorkSpaces uses AWS KMS (p. 292).	November 6, 2015
Update	Updated the Using key policies in AWS KMS (p. 59) page to improve clarity.	October 22, 2015

Change	Description	Date
New content	Added documentation about Deleting customer master keys (p. 172), including supporting documentation about Creating an Amazon CloudWatch alarm (p. 177) and Determining past usage of a customer master key (p. 181).	October 15, 2015
New content	Added documentation about Determining access to an AWS KMS customer master key (p. 130).	October 15, 2015
New content	Added documentation about Key state: Effect on your CMK (p. 233).	October 15, 2015
New content	Added documentation about How Amazon Simple Email Service (Amazon SES) uses AWS KMS (p. 272).	October 1, 2015
Update	Updated the Quotas (p. 370) page to explain the new request quotas.	August 31, 2015
New content	Added information about the charges for using AWS KMS. See AWS KMS Pricing (p. 2).	August 14, 2015
New content	Added request quotas to the AWS KMS Quotas (p. 370).	June 11, 2015
New content	Added a new Java code sample demonstrating use of the `UpdateAlias` operation. See Updating an alias (p. 366).	June 1, 2015
Update	Moved the AWS Key Management Service regions table to the *AWS General Reference*.	May 29, 2015
New content	Added documentation about How Amazon EMR uses AWS KMS (p. 259).	January 28, 2015
New content	Added documentation about How Amazon WorkMail uses AWS KMS (p. 285).	January 28, 2015
New content	Added documentation about How Amazon Relational Database Service (Amazon RDS) uses AWS KMS (p. 263).	January 6, 2015

Change	Description	Date
New content	Added documentation about How Amazon Elastic Transcoder uses AWS KMS (p. 255).	November 24, 2014
New guide	Introduced the *AWS Key Management Service Developer Guide*.	November 12, 2014