ALIBABA CLOUD

Alibaba Cloud

Key Management Service

User Guide

Document Version: 20201106

Alibaba Cloud

Legal disclaimer

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings> Network> Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Manage CMKs

This topic describes how to use Key Management Service (KMS) to create and manage customer master keys (CMKs). CMKs are used to encrypt data.

Create a CMK

1. Log on to the KMS console.
2. In the upper-left corner of the **Keys** page, click **Create Key**.
3. In the **Create Key** dialog box, set **Alias Name** and **Description**.
4. Click **Advanced** and specify **Key Material Source**.
   - **Alibaba Cloud KMS**: Use KMS to generate key material.
   - **External**: Import key material from an external source. For more information about how to import key material, see Import key material.

     Note If you select External, you must also select **I understand the implications of using the external key materials key**.
5. Click **OK**.

Note After the CMK is created, you can view the detailed information, such as the CMK ID, status, and protection level on the Keys page.

Disable a CMK

After you create a CMK, it is in the Enabled state by default. You cannot use a disabled CMK to encrypt or decrypt data. Follow these steps to disable a CMK:

1. On the **Keys** page, find the CMK that you want to disable.
2. In the **Actions** column, click **Disable**.
3. In the **Disable Key** message, click **OK**.

Schedule a key deletion task

After a CMK is deleted, it cannot be recovered, and the data and ciphertext data keys encrypted by using this CMK cannot be decrypted. Therefore, to prevent you from deleting CMKs by mistake, KMS only allows you to schedule key deletion tasks. You cannot immediately delete CMKs. Rather than deleting a CMK, we recommend that you disable the CMK.

1. On the **Keys** page, find the CMK that you want to delete.
2. In the **Actions** column, choose **More > Schedule Key Deletion**.
3. In the **Schedule Key Deletion** dialog box, specify **Delete In (7-30 days)**. The value ranges from 7 to 30. Unit: days. Default value: 30.
4. Click **OK**.

Note

- A CMK in the Pending Deletion state cannot be used to encrypt data, decrypt data, or generate data keys.
- You can choose **More > Cancel Key Deletion** to cancel a scheduled key deletion task.

2.Use aliases

Aliases are optional to CMKs.

Aliases must be unique in a region for each Alibaba Cloud account. An Alibaba Cloud account can have identical aliases in different regions. An alias can be bound to only one CMK in a region, but a CMK can have multiple aliases.

Although aliases are bound to CMKs, aliases are resources independent of CMKs. Aliases have the following characteristics:

- You can call the UpdateAlias operation to bind an alias to a different CMK. This operation does not affect the CMK.
- If you delete an alias, the CMK to which the alias is bound is not deleted.
- RAM users must be authorized before they can perform operations on an alias. For more information, see Use RAM to control access to resources.
- Aliases cannot be modified. To change the alias of a CMK, you must delete the old alias and create a new one for the CMK.

You can replace the CMK ID in the request parameters for the following API operations with an alias that is bound to the CMK:

- DescribeKey
- Encrypt
- GenerateDataKey
- GenerateDataKeyWithoutPlaintext

To specify an alias instead of a CMK ID in the request parameters for the preceding operations, a RAM user must have the relevant permissions on the CMK. The RAM user does not need to have permissions on the alias.

You can perform the following alias-related operations:

- Create an alias
- Bind an alias to a different CMK
- Delete an alias
- Obtain all aliases
- Obtain the aliases that are bound to a specific CMK

When you use an alias, you must make sure that the alias is complete. Example:

```
//A complete alias must have the alias/ prefix.
alias/example
```

Create an alias

- An alias must contain the `alias/` prefix. An alias (excluding the prefix) can contain letters, digits, underscores (_), hyphens (-), and forward slashes (/). An alias (excluding the prefix) must be 1 to 255 characters in length.
- To create an alias, a RAM user must have permissions on both the alias and the CMK to which the alias is bound.

- Creating a new alias for a CMK does not affect the existing aliases of the CMK.
- You can call the CreateAlias operation to create an alias.

```
//Grant RAM user 123456 the permissions to create the alias/example alias for the 08ec3bb9-034f-485b-b1cd-3459baa8**** CMK.
{
 "Version": "1",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "kms:CreateAlias"
   ],
   "Resource": [
    "acs:kms:cn-hangzhou:123456:key/08ec3bb9-034f-485b-b1cd-3459baa8****",
    "acs:kms:cn-hangzhou:123456:alias/example"
   ]
  }
 ]
}

//Create an alias.
aliyun kms CreateAlias --KeyId 08ec3bb9-034f-485b-b1cd-3459baa8**** --AliasName alias/example
```

Bind an alias to a different CMK

- You can call the UpdateAlias operation to bind an existing alias to a different CMK.
- To bind an alias to a different CMK, a RAM user must have permissions on the alias, the CMK to which the alias is currently bound, and the CMK to which you want to bind the alias.

```
//Grant RAM user 123456 the permissions to bind the alias/example alias to the 127d2f84-ee5f-4f4d-9d41-dbc
1aca28788 CMK. The alias is originally bound to the 08ec3bb9-034f-485b-b1cd-3459baa8**** CMK.
{
 "Version": "1",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "kms:UpdateAlias"
   ],
   "Resource": [
    "acs:kms:cn-hangzhou:123456:key/08ec3bb9-034f-485b-b1cd-3459baa8****",
    "acs:kms:cn-hangzhou:123456:key/127d2f84-ee5f-4f4d-9d41-dbc1aca2****",
    "acs:kms:cn-hangzhou:123456:alias/example"
   ]
  }
 ]
}

//Bind an alias to a different CMK.
aliyun kms UpdateAlias --AliasName alias/example --KeyId 127d2f84-ee5f-4f4d-9d41-dbc1aca2****
```

Delete an alias

- You can call the DeleteAlias operation to delete an alias. Deleting an alias does not affect the CMK to which the alias is bound.
- To delete an alias, a RAM user must have permissions on both the alias and the CMK to which the alias is bound.

```
//Grant RAM user 123456 the permissions to delete the alias/example alias of the 127d2f84-ee5f-4f4d-9d41-d
bc1aca2**** CMK.
{
 "Version": "1",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "kms:DeleteAlias"
   ],
   "Resource": [
    "acs:kms:cn-hangzhou:123456:key/127d2f84-ee5f-4f4d-9d41-dbc1aca2****",
    "acs:kms:cn-hangzhou:123456:alias/example"
   ]
  }
 ]
}

//Delete an alias.
aliyun kms DeleteAlias --AliasName alias/example
```

Obtain all aliases

- You can call the ListAliases operation to obtain all aliases under your Alibaba Cloud account in the current region.
- To obtain all aliases, a RAM user must have permissions on alias resources.

```
//Grant RAM user 123456 the permissions to obtain all aliases.
{
 "Version": "1",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "kms:ListeAliases"
   ],
   "Resource": [
    "acs:kms:cn-hangzhou:123456:alias"
   ]
  }
 ]
}

//Obtain all aliases.
aliyun kms ListAliases
```

Obtain the aliases that are bound to a specific CMK

- You can call the ListAliasesByKeyId operation to obtain all aliases bound to a specific CMK.
- To obtain the aliases bound to a specific CMK, a RAM user must have permissions on the CMK.

```
//Grant RAM user 123456 the permissions to obtain all aliases bound to the 127d2f84-ee5f-4f4d-9d41-dbc1ac
a2**** CMK.
{
 "Version": "1",
 "Statement": [
  {
   "Effect": "Allow",
   "Action": [
    "kms:DeleteAlias"
   ],
   "Resource": [
    "acs:kms:cn-hangzhou:123456:key/127d2f84-ee5f-4f4d-9d41-dbc1aca2****"
   ]
  }
 ]
}

//Obtain the aliases that are bound to a specific CMK.
aliyun kms ListAliasesByKeyId --KeyId 127d2f84-ee5f-4f4d-9d41-dbc1aca2****
```

3.CMK overview

You can host different types of customer master keys (CMKs) in Key Management Service (KMS) based on your business requirements. For example, you can use a CMK to encrypt and decrypt data. You can also use a CMK to generate and verify a signature.

Key-based cryptographic algorithms

KMS allows you to use various algorithms to support cryptographic operations. The algorithms are classified into two types: symmetric key algorithms and asymmetric key algorithms, as listed in the following table.

Algorithm class	Algorithm subclass	Support encryption and decryption	Support signature generation and verification
Symmetric key algorithm	AES	Yes	No
Symmetric key algorithm	SM4Note	Yes	No
Asymmetric key algorithm	RSA	Yes	Yes
Asymmetric key algorithm	ECC	No	Yes
Asymmetric key algorithm	SM2Note	Yes	Yes

Note Managed HSM supports the SM4 and SM2 algorithms only in mainland China. For more information, see Supported regions.

Symmetric keys are used to encrypt or decrypt data. If you do not specify the KeySpec parameter during key creation, KMS creates a symmetric key. You can call the Encrypt or Decrypt operation to encrypt or decrypt data without the need to obtain the plaintext of a symmetric key. For more information, see Overview.

Asymmetric keys can be used to encrypt data, decrypt data, generate a signature, or verify a signature. An asymmetric CMK in KMS consists of a public key and a private key, which are cryptographically related to each other. The public key can be made available for anyone to use, but the private key must be kept secure. To keep private keys secure, KMS does not provide an API operation for you to export the private key of an asymmetric key pair. You can use a private key to decrypt data or generate a signature by calling the related operations. Anyone with a public key can use it to encrypt data or verify the signature generated by the corresponding private key. For more information, see Overview.

Protection levels

KMS provides the Managed HSM feature. You can set the protection level of your CMK to HSM to host the CMK in an HSM. Managed HSM uses HSMs as dedicated hardware to safeguard keys. For a CMK whose protection level is HSM, the plaintext of its key material is stored only inside an HSM. KMS calls an HSM-related API operation to perform cryptographic operations. During the operations, KMS and Alibaba Cloud O&M personnel cannot access the plaintext of the key material. The plaintext of the key material cannot be exported from the HSM. For more information, see Overview and Use Managed HSM.

If you set the protection level of your CMK to SOFTWARE, KMS uses a software module to protect the CMK and uses a trusted platform module (TPM) to provide root-of-trust protection for the software module.

Key managers

In most cases, you are the manager of your CMK in KMS, and its Creator attribute is set to the ID of your Alibaba Cloud account.

Alibaba Cloud services that are integrated with KMS can implement server-side encryption. In this scenario, an Alibaba Cloud service can automatically host an encryption key in KMS to encrypt and protect your data. This makes it easier for you to use the entry-level data encryption features and reduces your overhead for key lifecycle and permission management. These service-managed keys are called service keys. To facilitate identification, KMS sets the Creator attribute of the service key hosted by an Alibaba Cloud service to the code of this service and assigns an alias in the format of `acs/<Code of the Alibaba Cloud service>` to the service key. For example, the Creator attribute of the service key hosted by Alibaba Cloud Object Storage Service (OSS) is set to OSS and alias acs/oss is assigned to the service key. For more information, see Integration with KMS.

4.Use symmetric keys

4.1. Overview

This topic describes symmetric encryption, which is the most commonly used data encryption method. KMS provides easy-to-use API operations that allow you to encrypt and decrypt data on the cloud.

If you do not specify the KeySpec parameter during key creation, KMS creates a symmetric key. KMS supports popular symmetric key algorithms and provides high-level data security by using strong cryptography.

Types of symmetric keys

The following table lists the types of symmetric keys that KMS supports.

Algorithm	Key length	Key type	Data encryption mode	Protection level
AES	256 bits	Aliyun_AES_256	GCM	• Software • HSM
SM4Note	128 bits	Aliyun_SM4	GCM	HSM

Note KMS provides the SM4 algorithm by using Managed HSM. For more information, see Overview.

Encryption and decryption features

When you call the Encrypt, ReEncrypt, GenerateDataKey, or GenerateDataKeyWithoutPlaintext operation to encrypt data or data keys, you need to specify only a CMK ID or alias. KMS uses the specified CMK for encryption and returns ciphertext. When you call the Decrypt operation, you need to specify only the ciphertext that you want to decrypt. You do not need to specify a CMK.

AAD

Symmetric keys of KMS use GCM for block ciphers. You can use additional authenticated data (AAD) to provide supplemental protection for the integrity of encrypted data. KMS encapsulates AAD to enable you to customize authentication data. For more information, see EncryptionContext.

Envelope encryption

You can call the GenerateDataKey and GenerateDataKeyWithoutPlaintext operations in KMS to generate a two-level key hierarchy to accelerate envelope encryption. For more information, see the following topics:

- What is envelope encryption?
- Use envelope encryption to encrypt and decrypt local data

Rotation of symmetric keys

Each symmetric CMK that is generated in KMS supports multiple key versions. KMS automatically rotates CMKs by generating new key versions. You can customize the key rotation policy.

If a CMK has multiple versions, the latest version of the CMK is used to encrypt data or data keys in the Encrypt, GenerateDataKey, and GenerateDataKeyWithoutPlaintext operations. When you call the Decrypt operation, you do not need to specify a CMK ID or key version ID. KMS automatically identifies the CMK and its key version with which the corresponding data or data key is encrypted. Then, KMS uses the key material of the identified key version to decrypt the ciphertext.

KMS rotates a CMK by generating a new version of the CMK. After a rotation is complete, KMS automatically uses the new key version to encrypt data or data keys. However, the earlier key version is still available to decrypt the ciphertext generated before the rotation. For more information, see Configure automatic key rotation.

BYOK

KMS allows you to encrypt your data on the cloud by using the Bring Your Own Key (BYOK) feature. This feature helps you meet stringent security and compliance requirements. We recommend that you use Managed HSM to protect your keys. You can import your key material into a CMK whose protection level is HSM. Keys in a managed HSM can only be destroyed, and their plaintext cannot be exported. For more information, see Import key material.

4.2. EncryptionContext

EncryptionContext is a JSON string that can be used in KMS API operations, such as Encrypt, GenerateDataKey, and Decrypt.

Function of EncryptionContext

EncryptionContext is a JSON string, and it must be in the string-string format. EncryptionContext is used to ensure data integrity.

If this parameter is specified during encryption, you must specify an equivalent EncryptionContext value for decryption. You can call the Encrypt or GenerateDataKey operation for encryption and call the Decrypt operation for decryption. EncryptionContext is related to decryption, but it is not included in ciphertext, which corresponds to the CipherBlob parameter.

Valid values of EncryptionContext

A valid value of EncryptionContext is a JSON string of up to 8,192 characters in the string-string format. When you specify EncryptionContext for an API operation, consider the escape characters.

Example of valid EncryptionContext

```
{"ValidKey":"ValidValue"}
{"Key1":"Value1","Key2":"Value2"}
```

Example of invalid EncryptionContext

```
[{"Key":"Value"}] // JSON array
{"Key":12345} //String-int
{"Key":["value1","value2"]} // String-array
```

Equivalent EncryptionContext

EncryptionContext is a map or hash table in the string-string format. Two EncryptionContext values are considered to be equivalent if their key-value pairs are consistent. The sequences of the key-value pairs can be different. If EncryptionContext is specified during encryption, you can specify an equivalent EncryptionContext value to decrypt ciphertext. The EncryptionContext values do not have to be identical.

Example of equivalent EncryptionContext values

```
{"Key1":"Value1","Key2":"Value2"} is equivalent to {"Key2":"Value2","Key1":"Value1"}.
```

4.3. Import key material

If you set the Origin parameter to EXTERNAL, KMS does not create key material. In this case, you must import external key material to the CMK. This topic describes how to import external key material.

Prerequisites

An Alibaba Cloud account is created. To create an Alibaba Cloud account, visit the account registration page.

Context

Customer master keys (CMKs) are basic resources of KMS. A CMK is composed of a key ID, basic metadata (such as key state), and key material that is used to encrypt and decrypt data. When you call the CreateKey operation to create a CMK, if you use the default value Aliyun_KMS of the Origin parameter, KMS generates key material. If you set the Origin parameter to EXTERNAL, KMS does not generate key material. In this case, you must import external key material to the CMK.

You can call the DescribeKey operation to view the key material source of an existing CMK.

- If the value of Origin in KeyMetadata is *Aliyun_KMS*, the key material is generated by KMS. In this case, the CMK is considered a **common key**.
- If the value of Origin is *EXTERNAL*, the key material is imported from an external source. In this case, the CMK is considered an **external key**.

Before you import external key material, take note of the following points:

- Make sure that the source of randomness from which the key material is generated meets security requirements.
- Make sure that the key material is reliable.
  - KMS ensures the high availability of imported key material. However, it cannot ensure that the imported key material has the same reliability as the key material generated by KMS.

- You can call the DeleteKeyMaterial operation to delete the imported key material. You can also set an expiration time to automatically delete the key material after it expires. The CMK is not deleted. To delete key material generated by KMS, you can only call the ScheduleKeyDeletion operation to specify a waiting period of 7 to 30 days for deleting the CMK. The key material is deleted along with the relevant CMK after the waiting period ends.
- After you delete the imported key material, you can re-import the same key material to make the relevant CMK available again. Therefore, we recommend that you save a copy of the key material.

- Key material is unique for each CMK. When you import key material into a CMK, the CMK is associated with that key material. Even after the key material expires or is deleted, you cannot import different key material into that CMK. If you need to rotate a CMK that uses external key material, you must create a CMK and then import new key material.
- CMKs are independent. You cannot use a CMK to decrypt data that is encrypted by using another CMK, even if the two CMKs use the same key material.
- The key material to be imported must be a 256-bit symmetric key.

Import key material in the KMS console

1. Create an external key.
   i. Log on to the KMS console.
   ii. In the top navigation bar, select the region where you want to create an external key.
   iii. In the left-side navigation pane, click **Keys**.
   iv. In the upper-left corner of the page that appears, click **Create Key**.
   v. In the **Create Key** dialog box, specify **Key Spec**. Set Key Spec to Aliyun_AES_256. For more information about key types, see Types of symmetric keys.
   vi. Specify **Alias Name**, **Description**, and **Rotation Period**.
   vii. Click **Advanced** and set **Key Material Source** to **External**.
   viii. Select **I understand the implications of using the external key materials key.** and click **OK**.
2. Obtain the parameters that are used to import key material. These parameters include a public key and an import token. The public key is used to encrypt the key material.
   i. In the left-side navigation pane, click **Keys**.
   ii. Find the CMK to which you want to import key material and click its **alias** to go to the key management page.
   iii. In the **Key Material** section, click **Key Encryption Material**.
   iv. In the **Key Encryption Material** dialog box, set **Wrapping Algorithm** to **RSAES_OAEP_SHA_1** and click **Next**. KMS supports the following encryption algorithms: RSAES_PKCS1_V1_5 (default value), RSAES_OAEP_SHA_1, and RSAES_OAEP_SHA_256. In this example, RSAES_OAEP_SHA_1 is used.
   v. Click **Download** next to **Public Key** and that next to **Import Token** to download the required public key and import token. Then, click **Close**.
3. Use OpenSSL to encrypt key material. The public key is a 2048-bit Rivest-Shamir-Adleman (RSA) public key. The encryption algorithm must be consistent with the algorithm specified when you obtain the import parameters. The public key is Base64 encoded. You must first decode the public key. You can use the public key encrypted by using the OpenSSL to obtain your key material.
   i. Create key material and use OpenSSL to generate a 32-byte random number.

ii. Base64 decode the public key that is used to encrypt the key material.

iii. Use an encryption algorithm such as RSAES_OAEP_SHA_1 to encrypt the key material.

iv. Base64 encode the encrypted key material and save it as a text file.

```
openssl rand -out KeyMaterial.bin 32
openssl enc -d -base64 -A -in PublicKey_base64.txt -out PublicKey.bin
openssl rsautl -encrypt -in KeyMaterial.bin -oaep -inkey PublicKey.bin  -keyform DER  -pubin -out E
ncryptedKeyMaterial.bin
openssl enc -e -base64 -A -in EncryptedKeyMaterial.bin -out EncryptedKeyMaterial_base64.txt
```

4. Import key material.

   You can import key material into an external key that has never had key material. You can also reset the expiration time of key material or re-import key material that has expired or been deleted.

   Each import token is bound to a public key that is used to encrypt key material. A CMK is specified when an import token is generated. The import token can only be used to import key material into the specified CMK. The lifecycle of an import token is 24 hours. It can be used repeatedly within this period. After it expires, you must obtain a new import token and a new public key.

   i. In the left-side navigation pane, click **Keys**.

   ii. Find the CMK to which you want to import key material and click its **alias** to go to the key management page.

   iii. In the **Key Material** section, click **Import Key Material**.

   iv. In the **Import Key Material** dialog box, specify the following parameters:

   - **Encrypted Key Material**: Upload the key material text file generated in Step 3.
   - **Import Token**: Upload the import token text file obtained in Step 2.

   v. Specify **Valid Until** and click **OK**. After the key material is imported, the state of the key changes from **Pending Import** to **Enabled**.

Import key material by using Alibaba Cloud CLI

1. Create an external key. Run the **aliyun kms CreateKey** command to call the CreateKey operation to set Origin to *EXTERNAL*.

```
aliyun kms CreateKey --Origin EXTERNAL --Description "External key"
```

2. Obtain the parameters that are used to import key material. Run the **aliyun kms GetParametersForImport** command to call the GetParametersForImport operation to obtain the parameters that are used to import key material.

```
aliyun kms GetParametersForImport --KeyId 1339cb7d-54d3-47e0-b595-c7d3dba8**** --WrappingAlgori
thm RSAES_OAEP_SHA_1 --WrappingKeySpec RSA_2048
```

3. Import key material.

i. Use the public key to encrypt the key material.

   The public key is a 2048-bit Rivest-Shamir-Adleman (RSA) public key. The encryption algorithm must be consistent with the algorithm specified when you obtain the import parameters. The public key returned when you call the GetParametersForImport operation is Base64 encoded. You must first decode the public key. KMS supports the following encryption algorithms: RSAES_OAEP_SHA_1, RSAES_OAEP_SHA_256, and RSAES_PKCS1_V1_5.

ii. Base64 encode the encrypted key material.

iii. Run the **aliyun kms ImportKeyMaterial** command to call the ImportKeyMaterial operation to import the encoded key material and the import token to KMS.

```
aliyun kms ImportKeyMaterial --KeyId 1339cb7d-54d3-47e0-b595-c7d3dba8**** --EncryptedKeyMat
erial xxx --ImportToken xxxx
```

Sample code

- SDK for Java

```
//Use the latest KMS SDK for Java.
//KmsClient.java

import com.aliyuncs.kms.model.v20160120.*;
import com.aliyuncs.profile.DefaultProfile;

//KMS API encapsulation
public class KmsClient {
    DefaultAcsClient client;

    public KmsClient( String region_id, String ak, String secret) {
        DefaultProfile profile = DefaultProfile.getProfile(region_id, ak, secret);
        this.client = new DefaultAcsClient(profile);
    }

    public CreateKeyResponse createKey() throws Exception {
        CreateKeyRequest request = new CreateKeyRequest();
        request.setOrigin("EXTERNAL"); //Create an external key.
        return this.client.getAcsResponse(request);
    }
    //... Omitted. The remaining operations are the same as those in the API method.
}
//example.java
import com.aliyuncs.kms.model.v20160120.*;
import KmsClient
import java.security.KeyFactory;
import java.security.PublicKey;
import java.security.spec.MGF1ParameterSpec;
import javax.crypto.Cipher;
import javax.crypto.spec.OAEPParameterSpec;
import javax.crypto.spec.PSource.PSpecified;
import java.security.spec.X509EncodedKeySpec;
import java.util.Random;
import javax.xml.bind.DatatypeConverter;

public class CreateAndImportExample {
    public static void main(String[] args) {
        String regionId = "cn-hangzhou";
    String accessKeyId = "*** Provide your AccessKeyId ***";
    String accessKeySecret = "*** Provide your AccessKeySecret ***";
    KmsClient kmsclient = new KmsClient(regionId,accessKeyId,accessKeySecret);
    //Create an external key.
    try {
        CreateKeyResponse keyResponse = kmsclient.createKey();
        String keyId = keyResponse.KeyMetadata.getKeyId();
        //Generate a 32-byte random number.
        byte[] keyMaterial = new byte[32];
        new Random().nextBytes(keyMaterial);
        //Obtain the parameters that are used to import key material.
        GetParametersForImportResponse paramResponse = kmsclient.getParametersForImport(keyId,"
RSAES_OAEP_SHA_256");
        String importToekn = paramResponse.getImportToken();
        String encryptPublicKey = paramResponse.getPublicKey();
        //Base64 decode the public key.
        byte[] publicKeyDer = DatatypeConverter.parseBase64Binary(encryptPublicKey);
        //Use RSA to parse the public key.
        KeyFactory keyFact = KeyFactory.getInstance("RSA");
        X509EncodedKeySpec spec = new X509EncodedKeySpec(publicKeyDer);
        PublicKey publicKey = keyFact.generatePublic(spec);
        //Encrypt the key material.
        Cipher oaepFromAlgo = Cipher.getInstance("RSA/ECB/OAEPWithSHA-1AndMGF1Padding");
        String hashFunc = "SHA-256";
        OAEPParameterSpec oaepParams = new OAEPParameterSpec(hashFunc, "MGF1", new MGF1Para
meterSpec(hashFunc), PSpecified.DEFAULT);
        oaepFromAlgo.init(Cipher.ENCRYPT_MODE, publicKey, oaepParams);
        byte[] cipherDer = oaepFromAlgo.doFinal(keyMaterial);
        //Base64 encode the encrypted key material.
```

```
            String encryptedKeyMaterial = DatatypeConverter.printBase64Binary(cipherDer);
            //Import the key material.
            Long expireTimestamp = 1546272000L; //UNIX timestamp, precise to the second. 0 indicates that th
e key material does not expire.
                kmsClient.importKeyMaterial(keyId,encryptedKeyMaterial, expireTimestamp);
        } catch(Exception e) {
            //... Omitted.
        }
        }
}
```

- SDK for Go

```
package main

import (
  "crypto/rand"
  "crypto/rsa"
  "crypto/sha256"
  "crypto/x509"
  "encoding/base64"
  "fmt"
  "log"
  random "math/rand"
  "time"

  "github.com/aliyun/alibaba-cloud-sdk-go/services/kms"
)

//Encapsulation of the CreateKey operation
func kmsCreateKey(client *kms.Client) (string, error) {
  request := kms.CreateCreateKeyRequest()
  request.Scheme = "https"
  request.Origin = "EXTERNAL" //Create an external key.
  response, err := client.CreateKey(request)
  if err ! = nil {
    return "", fmt.Errorf("CreateKey error:%v", err)
  }
  return response.KeyMetadata.KeyId, nil
}

//Encapsulation of the GetParametersForImport operation
```

```
func kmsGetParametersForImport(client *kms.Client, keyId, wrappingKeySpec, wrappingAlgorithm string
) (string, string, error) {
    request := kms.CreateGetParametersForImportRequest()
    request.Scheme = "https"
    request.KeyId = keyId
    request.WrappingKeySpec = wrappingKeySpec
    request.WrappingAlgorithm = wrappingAlgorithm
    response, err := client.GetParametersForImport(request)
    if err ! = nil {
        return "", "", fmt.Errorf("GetParametersForImport error:%v", err)
    }
    return response.PublicKey, response.ImportToken, nil
}

//Encapsulation of the ImportKeyMaterial operation
func kmsImportKeyMaterial(client *kms.Client, keyId, importToken, encryptedKeyMaterial string) error {
    request := kms.CreateImportKeyMaterialRequest()
    request.Scheme = "https"
    request.KeyId = keyId
    request.ImportToken = importToken
    request.EncryptedKeyMaterial = encryptedKeyMaterial
    _, err := client.ImportKeyMaterial(request)
    if err ! = nil {
        return fmt.Errorf("ImportKeyMaterial error:%v", err)
    }
    return nil
}

func randBytes(n int) []byte {
    var r = random.New(random.NewSource(time.Now().UnixNano()))
    bytes := make([]byte, n)
    for i := range bytes {
        bytes[i] = byte(r.Intn(256))
    }
    return bytes
}

func main() {
    accessKeyId := "*** Provide your AccessKeyId ***"
    accessKeySecret := "*** Provide your AccessKeySecret ***"
```

```
  regionId := "cn-hangzhou"
  client, err := kms.NewClientWithAccessKey(regionId, accessKeyId, accessKeySecret)
  if err ! = nil {
    log.Fatalf("NewClientWithAccessKey error:%+v\n", err)
  }
  //Create an external key.
  keyId, err := kmsCreateKey(client)
  if err ! = nil {
    log.Fatalf("kmsCreateKey error:%+v\n", err)
  }
  //The following sample code is executed to generate a 32-byte random number. In actual scenarios, you
need to generate a key in your own key management system and use the public key in the parameters tha
t are used to import key material to encrypt the key material.
  keyMaterial := randBytes(32)
  //Obtain the parameters that are used to import key material.
  encryptPublicKey, importToken, err := kmsGetParametersForImport(client, keyId, "RSA_2048", "RSAES_
OAEP_SHA_256")
  if err ! = nil {
    log.Fatalf("kmsGetParametersForImport error:%v\n", err)
  }
  //Base64 decode the public key.
  publicKeyDer, err := base64.StdEncoding.DecodeString(encryptPublicKey)
  if err ! = nil {
    log.Fatalf("base64.StdEncoding.DecodeString error:%v\n", err)
  }
  //Use RSA to parse the public key.
  publicKey, err := x509.ParsePKIXPublicKey(publicKeyDer)
  if err ! = nil {
    log.Fatalf("x509.ParsePKIXPublicKey error:%v\n", err)
  }
  //Encrypt the key material.
  cipherDer, err := rsa.EncryptOAEP(sha256.New(), rand.Reader, publicKey.(*rsa.PublicKey), keyMaterial,
nil)
  if err ! = nil {
    log.Fatalf("rsa.EncryptOAEP error:%v\n", err)
  }
  //Base64 encode the encrypted key material.
  encryptedKeyMaterial := base64.StdEncoding.EncodeToString(cipherDer)
  //Import the key material.
  err = kmsImportKeyMaterial(client, keyId, importToken, encryptedKeyMaterial)
  if err ! = nil {
        log.Fatalf("ImportKeyMaterial error:%v", err)
    }
}
```

4.4. Delete key material

After you import key material, you can directly delete the key material. In this case, the key will no longer function and the ciphertext encrypted by using the key cannot be decrypted. This topic describes how to delete key material.

Prerequisites

Key material is imported. For information about how to import key material, see Import key material.

Context

After you import key material into an external key, you can use the external key just like a normal key. The only difference is that the key material of an external key may expire and can be independently deleted. After the key material of an external key expires or is deleted, the external key can no longer be used, and the ciphertext encrypted by using this external key cannot be decrypted. After you delete the imported key material, you can re-import the same key material to make the relevant CMK available again. Therefore, we recommend that you save a copy of the key material.

Delete key material in the KMS console

1. Log on to the KMS console.
2. In the top navigation bar, select the region where your key resides.
3. In the left-side navigation pane, click **Keys**.
4. Find your key and click its alias to go to the key management page.
5. In the **Key Material** section of the page that appears, click **Delete Key Material**. In the message that appears, click OK.After the key material is deleted, the key state changes from **Enabled** to **Pending Import**.

Delete key material by using Alibaba Cloud CLI

Run the **aliyun kms DeleteKeyMaterial** command to call the DeleteKeyMaterial operation to delete key material.

```
aliyun kms DeleteKeyMaterial --KeyId 1339cb7d-54d3-47e0-b595-c7d3dba8****
```

5.Use asymmetric keys

5.1. Overview

Unlike symmetric keys, asymmetric keys are mainly used to verify digital signatures or encrypt sensitive information between systems with different trust levels.

An asymmetric key pair consists of a public key and a private key, which are cryptographically related to each other. The public key is available for anyone to use, but the private key must be kept secure and used only by trusted users. Alibaba Cloud supports popular asymmetric key algorithms and provides high-level data security by using strong cryptography and digital signatures.

KMS generates a certificate signing request (CSR) file for an asymmetric CMK. A certificate applicant submits the CSR file to a certificate authority (CA). Then, the CA sends back a digital certificate that is signed by using the private key of the CA. The digital certificate can be used to ensure the security of emails, terminals, code signing, trusted website services, and identity authorization management systems.

Types of asymmetric keys

The following table lists the types of asymmetric keys that KMS supports.

Algorithm	Key type	Description	Purpose
RSA	RSA_2048	RSA asymmetric cryptosystem	• Encrypt or decrypt data. • Generate a digital signature.
ECC	• EC_P256: NIST-recommended elliptic curve P-256 • EC_P256K: SECG elliptic curve secp256k1	Elliptic-curve cryptography (ECC)	Generate a digital signature.
SM2	EC_SM2	ECC defined by GB/T 32918	• Encrypt or decrypt data. • Generate a digital signature.

Data encryption

In data encryption, asymmetric keys are used to transmit sensitive information. The following operations describe a typical scenario:

1. An information receiver distributes a public key to a transmitter.
2. The transmitter uses the public key to encrypt sensitive information.
3. The transmitter sends the ciphertext generated from the sensitive information to the information receiver.

4. The information receiver uses the private key to decrypt the ciphertext.

The private key can be used only by the information receiver. This ensures that the plaintext of sensitive information cannot be intercepted and decrypted by unauthorized parties during transmission. This encryption method is widely used to exchange keys. For example, session keys are exchanged in Transport Layer Security (TLS) handshakes, and encryption keys are exported and imported between different hardware security modules (HSMs).

For more information, see Encrypt and decrypt data by using an asymmetric CMK.

Digital signature

Asymmetric keys are also used to generate digital signatures. Private keys can be used to sign messages or information. Private keys are strictly protected and can be used only by trusted users to generate signatures. After a signature is generated, you can use the corresponding public key to verify the signature to achieve the following purposes:

- Verify data integrity. If the data does not match its signature, the data may be tampered with.
- Verify message authenticity. If a message does not match its signature, the message transmitter does not hold the private key.
- Provide non-repudiation for signatures. If the data matches its signature, the signer cannot deny this signature.

The following operations describe a typical signature verification scenario:

1. A signer sends a public key to a message receiver.
2. The signer uses the private key to sign data.
3. The signer sends the data and signature to the message receiver.
4. After receiving the data and signature, the message receiver uses the public key to verify the signature.

Digital signatures are widely used to defend against data tampering and authenticate identities.

- Case 1: You can use digital signatures to protect the integrity of your binary code and verify that the code has not been manipulated. This helps provide a trusted execution environment.
- Case 2: Digital signatures can also be used in digital certificate systems. In such a system, a certificate authority (CA) provides a signature for digital certificates to certify the entity information, public and private key information, key purpose, expiration date, and issuer. The private key holder of a certificate uses the private key to sign a message. The message receiver uses the public key contained in the certificate to verify the message signature and uses the public key of the certificate issuer to verify the certificate.

For more information, see Using asymmetric CMKs for digital signatures.

Key version

KMS does not support automatic rotation of asymmetric CMKs. You can call the CreateKeyVersion operation to create a key version in a specific CMK and generate a new pair of public and private keys. If you use a new key version to generate a digital signature or encrypt data, you must also distribute the new version of the public key.

In addition, unlike symmetric CMKs, asymmetric CMKs do not have a primary key version. Therefore, to call the operations related to asymmetric keys in KMS, you must specify the CMK ID or CMK alias and a key version.

Public key operation

In most cases, you can call the GetPublicKey operation to obtain a public key and distribute it to users for encryption or verification. Then, the users can use cryptographic libraries such as OpenSSL and Java Cryptography Extension (JCE) on the business end to perform local calculation.

You can also call the AsymmetricEncrypt or AsymmetricVerify operation to perform public key operations. If you call these operations, KMS records the logs of the calls and allows you to use Resource Access Management (RAM) to put limits on the use of public keys. Compared with local calculation on the business end, KMS offers flexible functions that better suit your needs.

Private key operation

You can call only the AsymmetricDecrypt or AsymmetricSign operation to use a private key to decrypt data or generate a digital signature.

5.2. Encrypt and decrypt data by using an asymmetric CMK

This topic describes how to use an asymmetric customer master key (CMK) to encrypt and decrypt data in Alibaba Cloud CLI.

Asymmetric encryption generally includes the following steps:

1. An information receiver distributes a public key to a transmitter.
2. The transmitter uses the public key to encrypt sensitive information.
3. The transmitter sends the ciphertext generated from the sensitive information to the receiver.
4. The receiver uses the private key to decrypt the ciphertext.

Prerequisites

You must call the CreateKey operation to create an asymmetric key in KMS. While you create an asymmetric key, set the KeySpec parameter to a desired key type and the Usage parameter to `ENCRYPT/DECRYPT` . For more information about CreateKey, see CreateKey.

Create an RSA encryption key.

```
$ aliyun kms CreateKey --KeySpec=RSA_2048 --KeyUsage=ENCRYPT/DECRYPT --ProtectionLevel=HSM
```

Obtain the public key

1. Call the GetPublicKey operation to obtain the public key of the asymmetric key pair.

```
$ aliyun kms GetPublicKey --KeyId=**** --KeyVersionId=****
```

Example output:

```
{
    "RequestId": "82c383eb-c377-4mf6-bxx8-81hkc1g5g7ab",
    "KeyId": "****",
    "KeyVersionId": "****",
    "PublicKey": "PublicKey-Data****"
}
```

2. Save the public key to the file rsa_publickey.pub. (PublicKey-Data**** is a placeholder. You must replace it with the obtained public key).

```
$ echo PublicKey-Data**** > rsa_publickey.pub
```

Use the public key to encrypt data

1. Create a sample plaintext file named plaintext-file.txt that contains "this is plaintext".

```
echo "this is plaintext" > plaintext-file.txt
```

2. Use OpenSSL to encrypt the file and write the obtained binary ciphertext into the plaintext-file.enc file.

```
openssl pkeyutl -encrypt -in plaintext-file.txt \
  -inkey rsa_publickey.pub -pubin \
  -pkeyopt rsa_padding_mode:oaep \
  -pkeyopt rsa_oaep_md:sha256 \
  -pkeyopt rsa_mgf1_md:sha256 \
  -out plaintext-file.enc
```

Call the KMS API to decrypt data

You must call the KMS API and use the private key to decrypt data.

1. Before you transmit the encrypted data over the network, encode it in Base64.

```
$ openssl base64 -in plaintext-file.enc
```

The following Base64-encoded ciphertext is returned:

```
5kdCB06HHeAwgfH9ARY4/9Nv5vlpQ94GXZcmaC9FE59Aw8v8RYdozT6ggSbyZbi+
8STKVq9402MEfmUDmwJLuu0qgAZsCe5wU4JWHh1y84Qn6HT068j0qOy5X2HIlrjs
fCdetgtMtVorSgb3bbERk2RV67nHWrDkecNbUaz+6ik4AlZxv2uWrV62eQ9yUBYm
Jb956LbqnfWdCFxUSHH/qB5QCnLpijzvPmfNlZr653H4nF08gpZjnmlF4FjTu3i2
mGLzK4J3Rh/l7PQHiVMdc4hSnXosg68QmMVdZBGLK9/cD9SYngPDiirU7z0q7Git
dIeloyCAUDFyuQC6a+SqzA==
```

2. Pass the Base64-encoded ciphertext to KMS to decrypt data.

```
aliyun kms AsymmetricDecrypt \
 --KeyId **** \
 --KeyVersionId **** \
 --Algorithm RSAES_OAEP_SHA_256 \
 --CiphertextBlob 5kdCB06HHeAwgfH9ARY4/9Nv5vlpQ94GXZcmaC9FE59Aw8v8RYdozT6ggSbyZbi+8STKVq94
02MEfmUDmwJLuu0qgAZsCe5wU4JWHh1y84Qn6HT068j0qOy5X2HIlrjsfCdetgtMtVorSgb3bbERk2RV67nHWr
DkecNbUaz+6ik4AlZxv2uWrV62eQ9yUBYmJb956LbqnfWdCFxUSHH/qB5QCnLpijzvPmfNlZr653H4nF08gpZjn
mlF4FjTu3i2mGLzK4J3Rh/l7PQHiVMdc4hSnXosg68QmMVdZBGLK9/cD9SYngPDiirU7z0q7GitdIeloyCAUDFyuQ
C6a+SqzA==
```

Example output:

```
{
    "KeyId": "*****",
    "KeyVersionId": "*****",
    "Plaintext": "dGhpcyBpcyBwbGFpbnRleHQgDQo=",
    "RequestId": "6be7a8e4-35b9-4549-ad05-c5b1b535a22c"
}
```

3. Decode the returned Base64-encoded plaintext in Base64.

```
echo dGhpcyBpcyBwbGFpbnRleHQgDQo= | openssl base64 -d
```

The following decrypted plaintext is returned:

```
this is plaintext
```

5.3. Using asymmetric CMKs for digital signatures

This topic uses Alibaba Cloud CLI as an example to describe how to use an asymmetric customer master key (CMK) to generate and verify a digital signature. You can also perform this operation by using the KMS SDK.

Asymmetric encryption generally includes the following steps:

1. A signer sends a public key to a receiver.
2. The signer uses the private key to sign data.
3. The signer sends the data and signature to the receiver.
4. After receiving the data and signature, the receiver uses the public key to verify the signature.

Before you start

You must call the CreateKey operation to create an asymmetric key in KMS. While you create an asymmetric key, set the KeySpec parameter to a desired key type and set the Usage parameter to `SIGN/VERIFY` .

- Create an RSA signature key:

```
$ aliyun kms CreateKey --KeySpec=RSA_2048 --KeyUsage=SIGN/VERIFY --ProtectionLevel=HSM
```

- Create a NIST P-256 signature key:

```
$ aliyun kms CreateKey --KeySpec=EC_P256 --KeyUsage=SIGN/VERIFY --ProtectionLevel=HSM
```

- Create a secp256k1 signature key:

```
$ aliyun kms CreateKey --KeySpec=EC_P256K --KeyUsage=SIGN/VERIFY --ProtectionLevel=HSM
```

Preprocess signature: compute a message digest

Both RSA and ECC signature operations involve first computing the digest of an unsigned message and then signing the digest.

Note The algorithm used to obtain a message digest must match the algorithm used to call KMS to compute a signature. For example, the ECDSA_SHA_256 signature algorithm must be used in conjunction with the SHA-256 digest algorithm. It does not support the SHA-384 digest algorithm.

The following example uses the SHA-256 digest algorithm.

1. Save the message "this is message" that needs to be signed into the file message-file.txt:

```
$ echo "this is message" > message-file.txt
```

2. Compute the SHA-256 digest of the message and save the binary digest to the file message-sha256.bin:

```
$ openssl dgst -sha256 -binary -out message-sha256.bin  message-file.txt
```

Call KMS to compute the signature

You must call the KMS API to compute the signature of a message with the private key.

1. Before you transmit the message digest over the network, encode it in Base64.

```
$ openssl base64 -in message-sha256.bin
```

The following Base64 encoded digest is returned:

```
hRP2cuRFSlfEoUXCGuPyi7kZr18VCTZeVOTw0jbUB6w=
```

2. Pass the Base64 encoded digest to KMS to generate a signature.

Note The parameters passed and the results generated vary depending on key types and signature algorithms. Each signature result generated in the example is stored in a different file.

- RSASSA-PSS

  For RSA keys, you can use the RSASSA-PSS signature algorithm and the SHA-256 digest algorithm to create a signature. Run the following command:

  ```
  $ aliyun kms AsymmetricSign --KeyId=**** --KeyVersionId=**** \
    --Algorithm=RSA_PSS_SHA_256 --Digest=hRP2cu...
  {
      "KeyId": "*****",
      "KeyVersionId": "*****",
      "Value": "J7xmdnZ...",
      "RequestId": "70f78da9-c1b6-4119-9635-0ce4427cd424"
  }
  ```

  Decode the signature value in Base64 and generate a binary signature. This signature is saved in the file rsa_pss_signature.bin:

  ```
  $ echo J7xmdnZ... | openssl base64 -d -out rsa_pss_signature.bin
  ```

- RSASSA_PKCS1_V1_5

  For RSA keys, you can use the RSASSA_PKCS1_V1_5 signature algorithm and the SHA-256 digest algorithm to create a signature. Run the following command:

  ```
  $ aliyun kms AsymmetricSign --KeyId=**** --KeyVersionId=**** \
    --Algorithm=RSA_PKCS1_SHA_256 --Digest=hRP2cu...
  {
      "KeyId": "*****",
      "KeyVersionId": "*****",
      "Value": "qreBkH/u...",
      "RequestId": "4be57288-f477-4ecd-b7be-ad8688390fbc"
  }
  ```

  Decode the signature value in Base64 and generate a binary signature. This signature is saved in the file rsa_pkcs1_signature.bin:

  ```
  echo qreBkH/u... | openssl base64 -d -out rsa_pkcs1_signature.bin
  ```

- NIST P-256

  For NIST curve P-256, you can use the ECDSA signature algorithm and the SHA-256 digest signature to create a signature. Run the following command:

```
$ aliyun kms AsymmetricSign --KeyId=**** --KeyVersionId=**** \
  --Algorithm=ECDSA_SHA_256 --Digest=hRP2cu...
{
    "KeyId": "*****",
    "KeyVersionId": "*****",
    "Value": "MEYCIQD33Y98...",
    "RequestId": "472d789c-d4be-4271-96bb-367f7f0f8ec3"
}
```

Decode the signature value in Base64 and generate a binary signature. This signature is saved in the file ec_p256_signature.bin:

```
echo MEYCIQD33Y98... | openssl base64 -d -out ec_p256_signature.bin
```

- secp256k1

  For SECG curve secp256k1, you can use the ECDSA signature algorithm and the SHA-256 digest algorithm to create a signature. Run the following command:

```
$ aliyun kms AsymmetricSign --KeyId=**** --KeyVersionId=**** \
  --Algorithm=ECDSA_SHA_256 --Digest=hRP2cu...
{
    "KeyId": "*****",
    "KeyVersionId": "*****",
    "Value": "MEYCIQDWuuI...",
    "RequestId": "fe41abed-91e7-4069-9f6b-0048f5bf4de5"
}
```

  Decode the signature Value in Base64 and generate a binary signature. This signature is saved in the file ec_p256k_signature.bin:

```
echo MEYCIQDWuuI... | openssl base64 -d -out ec_p256k_signature.bin
```

Obtain the public key

Refer to Obtain the public key to obtain the public key of the created asymmetric key pair from the KMS. The preceding example assumes that:

- The public key of the RSA key pair is saved to the file rsa_publickey.pub.
- The public key of the NIST P-256 key pair is saved to the file ec_p256_publickey.pub.
- The public key of the secp256k1 key pair is saved to the file ec_p256k_publickey.pub.

Use the public key to verify the signature

Run the following command lines to verify the signature (the command varies depending on the algorithm used to generate the public key):

- RSASSA-PSS

```
$ openssl dgst \
  -verify rsa_publickey.pub \
  -sha256 \
  -sigopt rsa_padding_mode:pss \
  -sigopt rsa_pss_saltlen:-1 \
  -signature rsa_pss_signature.bin \
  message-file.txt
```

- RSASSA_PKCS1_V1_5

```
$ openssl dgst \
  -verify rsa_publickey.pub \
  -sha256 \
  -signature rsa_pkcs1_signature.bin \
  message-file.txt
```

- NIST P-256

```
$ openssl dgst \
  -verify ec_p256_publickey.pub \
  -sha256 \
  -signature ec_p256_signature.bin \
  message-file.txt
```

- secp256k1

```
$ openssl dgst \
  -verify ec_p256k_publickey.pub \
  -sha256 \
  -signature ec_p256k_signature.bin \
  message-file.txt
```

If the verification succeeds, the system displays the following message:

```
Verified OK
```

6.Manage aliases

7.Managed HSM

7.1. Overview

Managed HSM is an important feature of Key Management Service (KMS) to enable easy access to certified hardware security modules (HSMs) provided by Alibaba Cloud.

An HSM is a highly secure hardware device that performs cryptographic operations and generates and stores keys. You can host the keys for your most sensitive Alibaba Cloud workloads and assets in HSMs that are managed on Alibaba Cloud (referred to as managed HSMs).

Supported regions

You can use Managed HSM in the following regions. This feature is scheduled to be available in more regions in the future.

Region	City	Certification type	Region ID
China (Beijing)	Beijing	State Cryptography Administration (SCA) certification	cn-beijing
China (Zhangjiakou-Beijing Winter Olympics)	Zhangjiakou	SCA certification	cn-zhangjiakou
China (Hangzhou)	Hangzhou	SCA certification	cn-hangzhou
China (Shanghai)	Shanghai	SCA certification	cn-shanghai
China (Shenzhen)	Shenzhen	SCA certification	cn-shenzhen
China (Hong Kong)	Hong Kong	Federal Information Processing Standards (FIPS) 140-2 Level 3	cn-hongkong
Singapore (Singapore)	Singapore	FIPS 140-2 Level 3	ap-southeast-1
Australia (Sydney)	Sydney	FIPS 140-2 Level 3	ap-southeast-2
Malaysia (Kuala Lumpur)	Kuala Lumpur	FIPS 140-2 Level 3	ap-southeast-3
Indonesia (Jakarta)	Jakarta	FIPS 140-2 Level 3	ap-southeast-5
US (Virginia)	Virginia	FIPS 140-2 Level 3	us-east-1

Compliance

Managed HSM helps you meet regulatory requirements. Based on different regulatory requirements in each local market, Alibaba Cloud offers HSMs certified by different third-party organizations to meet both your localization and internationalization requirements.

For regions in mainland China:

- SCA certification: Alibaba Cloud managed HSMs have passed the certification of the agencies designated by SCA.

- SCA compliance: Alibaba Cloud managed HSMs comply with the relevant technical requirements and specifications of SCA and provide Alibaba Cloud users with commercial cryptographic algorithms that comply with both national and industrial standards.

For regions outside mainland China:

- FIPS validation for hardware: Alibaba Cloud managed HSMs, including their hardware and firmware, have passed FIPS 140-2 Level 3 validation. For more information about the certificate issued by National Institute of Standards and Technology (NIST), visit Certificate #3254.
- FIPS 140-2 Level 3 compliance: Alibaba Cloud managed HSMs run in FIPS Approved Level 3 mode of operation.
- PCI DSS: Alibaba Cloud managed HSMs comply with Payment Card Industry Data Security Standard (PCI DSS) requirements.

High security assurance

- Hardware protection

  Managed HSMs use secure hardware mechanisms to help you protect keys in KMS. The plaintext key material of CMKs is processed only inside HSMs for key operations. It is kept within the hardware security boundary of HSMs.

- Secure key generation

  Randomness is crucial to the encryption strength of keys. Managed HSMs use a random number generation algorithm to generate key material. The algorithm is secure and licensed and has high system entropy seeds. This protects keys from being recovered or predicted by attackers.

Ease of operation

HSM hardware is fully managed by Alibaba Cloud. This eliminates the costs otherwise incurred by the following hardware management operations:

- Hardware lifecycle management
- HSM cluster management
- High availability and scalability management
- System patching
- Most disaster recovery operations

Ease of integration

Native key management capabilities allow you to use the following features:

- Key version management
- Automatic key rotation
- Resource tag management
- Controlled authorization

These features enable rapid integration of your applications with HSMs, as well as the integration of ECS, ApsaraDB for RDS, and other cloud services with Managed HSM. You can implement cloud data encryption at rest without the need to pay any R&D costs.

Key control

Managed HSM allows you to better control encryption keys on the cloud and move the most sensitive computing tasks and assets to the cloud.

If you use both Managed HSM and BYOK (Bring Your Own Key), you can have full control over the following items:

- Generation modes of key material.
- Processing of key material: The key material that you import to a managed HSM can be destroyed but cannot be exported.
- Lifecycle of keys.
- Persistence of keys.

Cost-effectiveness

You can benefit from the pay-as-you-go billing method of cloud computing. Compared with user-created key infrastructure by using your on-premises HSMs, Managed HSM eliminates hardware procurement costs, as well as subsequent R&D and O&M costs.

7.2. Use Managed HSM

This topic describes how to use the Managed HSM feature to create and use keys.

Prerequisites

An Alibaba Cloud account is created. To create an Alibaba Cloud account, visit the account registration page.

Context

Managed HSM is supported only in some regions. For more information about the supported regions, see Supported regions.

Create a key in the KMS console

1. Log on to the KMS console.
2. On the **Keys** page, click **Create Key**.
3. In the **Create Key** dialog box, enter an alias in the **Alias Name** field.
4. Select **HSM** from the **Protection Level** drop-down list.
5. Enter a description. Click **OK**.
   After the key is created, you can view its **Protection Level** attribute on the **Keys** page and **Key Details** page.

Use Alibaba Cloud CLI to create a key

1. Use Alibaba Cloud CLI to run the following command:

   ```
   aliyun kms CreateKey --ProtectionLevel HSM --Description "Key1 in Managed HSM"
   ```

2. Call the DescribeKey operation to check the protection level of the created key.

```
{
 "KeyMetadata": {
  "CreationDate": "2019-07-04T13:14:15Z",
  "Description": "Key1 in Managed HSM",
  "KeyId": "1234abcd-12ab-34cd-56ef-12345678****",
  "KeyState": "Enabled",
  "KeyUsage": "ENCRYPT/DECRYPT",
  "DeleteDate": "",
  "Creator": "111122223333",
  "Arn": "acs:kms:cn-hongkong:111122223333:key/1234abcd-12ab-34cd-56ef-12345678****",
  "Origin": "Aliyun_KMS",
  "MaterialExpireTime": "",
  "ProtectionLevel": "HSM"
 },
 "RequestId": "8eaeaa8b-4491-4f1e-a51e-f95a4e54620c"
}
```

Import external keys to a managed HSM

You can import a key from user-created key infrastructure to an HSM managed by Alibaba Cloud (referred to as managed HSM). The prerequisite is that you have set **Protection Level** to **HSM** when you create the external key. For more information about how to create an external key, see Import key material in the KMS console.

After you trigger the import, KMS performs the following operations:

- Calls the GetParametersForImport operation. During this process, KMS generates a key pair in a managed HSM to import the external key based on the **HSM** protection level and returns the public key of the key pair.
- Calls the ImportKeyMaterial operation. During this process, KMS imports the encrypted external key material to the managed HSM and then obtains the plaintext of the key material by using the key unwrapping mechanism of the managed HSM. The plaintext of the key material can no longer be exported.

Manage and use keys

You can apply all management and cryptographic features supported by KMS to keys created in managed HSMs. The features are as follows:

- Enable and disable keys.
- Manage the lifecycle of keys.
- Manage the aliases of keys.
- Manage the tags of keys.
- Call cryptographic API operations.

Integration with other Alibaba Cloud services

Keys in managed HSMs can be used to protect native data in other Alibaba Cloud services such as Elastic Compute Service (ECS), ApsaraDB for RDS, and Object Storage Service (OSS) over the standard API of KMS. The prerequisite is that the Alibaba Cloud service supports server-side encryption by using user-managed keys. To use this feature, you only need to configure a CMK for the Alibaba Cloud service to support server-side encryption and host the CMK in a managed HSM.

8.Key rotation

8.1. Overview

Keys are often used to protect data. The security of data is dependent on the security of its corresponding keys. You can use key versions and the periodic rotation mechanism to improve key security and implement security policies and best practices for data protection.

Security goals

You can use the periodic key rotation mechanism to:

- Reduce the amount of data encrypted by each key

  The security of a key is inversely proportional to the amount of data encrypted by it. This amount is usually defined by the total bytes of data or the total number of messages that are encrypted by the same key. For example, National Institute of Standards and Technology (NIST) defines the secure lifecycle of a key in GCM mode as the total number of messages encrypted based on the key. The periodic key rotation mechanism enables each key to remain secure and minimize vulnerability to cryptanalytic attacks.

- Respond in advance to security events

  In the early days of system design, key rotation was introduced as a routine O&M method. This provides the system with a method to handle security events when they occur, and complies with the fail early, fail often principle of software engineering. If key rotation is not executed until an emergency event has already occurred, the probability of system failure increases exponentially.

- Provide logical isolation of data

  Encrypted data is isolated with each key rotation from other data encrypted using different keys. The impact of key-related security events can be identified quickly and preventive measures can be taken.

- Reduce the window of time to crack keys

  Periodic rotation of encryption keys ensures that you can control and reduce the window of time for which the key and its encrypted data are vulnerable to being cracked. Attackers only have a limited period of time between rotation tasks during which they are able to crack the key. This practice greatly increases the security of your data against cryptanalytic attacks.

Regulatory compliance

The periodic key rotation mechanism facilitates compliance with various regulations, which include but are not limited to:

- Payment Card Industry Data Security Standard (PCI DSS)
- Cryptography-related industrial standards issued by State Cryptography Administration, such as GM/T 0051-2016
- Cryptography-related standards issued by NIST, such as NIST Publication 800-38D

8.2. Configure automatic key rotation

This topic describes how to configure automatic rotation of customer master keys (CMKs) in Key Management Service (KMS).

Key versions

A CMK may have multiple key versions. Each key version represents an independently generated key. Key versions of the same CMK do not have any cryptographic relation to each other. KMS automatically rotates CMKs by generating new key versions.

Types of key versions

There are two types of key versions:

- Primary key versions
  - The primary key version of a CMK is an active encryption key. Each CMK has only one primary key version at any point in time.
  - When you call an encryption API operation such as GenerateDataKey or Encrypt, KMS uses the primary key version of a specified CMK to encrypt the target plaintext.
  - You can call the DescribeKey operation to query the PrimaryKeyVersion attribute.
- Non-primary key versions
  - A non-primary key version of a CMK is an inactive encryption key. Each CMK can have any number of non-primary key versions.
  - Each non-primary key version was a primary key version and acted as the active encryption key in the past.
  - When a new primary key version is created, KMS does not delete or disable non-primary key versions because they will be used to decrypt data.

Note When you call an encryption API operation, the primary key version of a specified CMK is used. When you call a decryption API operation, the key version that was used to encrypt the ciphertext is used.

Generation of key versions

You can generate a key version in either of the following ways:

- Create a CMK.

  You can call the CreateKey operation to create a CMK. If you set the Origin parameter to *Aliyun_KMS*, KMS generates an initial key version and sets it as the primary key version.
- Execute an automatic rotation policy.

  After you configure an automatic rotation policy, KMS executes the policy on a regular basis to generate new key versions.

Automatic key rotation

Configure and query an automatic rotation policy

When you call the CreateKey operation to create a CMK, you can specify an automatic rotation policy for the CMK. You can call the UpdateRotationPolicy operation to update the current automatic rotation policy. When you call the API operations, you must configure the following parameters:

- EnableAutomaticRotation: specifies whether to enable automatic rotation.
- RotationInterval: indicates the time period for automatic rotation.

You can call the DescribeKey operation to view the configured automatic rotation policy. The following parameters are returned:

- AutomaticRotation: indicates whether automatic rotation is enabled. For more information, see Impact of CMK status on automatic rotation.
  - *Disabled*: indicates that automatic rotation is disabled.
  - *Enabled*: indicates that automatic rotation is enabled.
  - *Suspended*: indicates that KMS suspends the execution of automatic rotation although automatic rotation is enabled.
- RotationInterval: indicates the time period for automatic rotation.

Execute an automatic rotation policy

When automatic rotation is enabled, KMS calculates the time of the next rotation by using the following formula:

```
${NextRotationTime} = ${LastRotationTime} + ${RotationInterval}
```

where:

- `LastRotationTime` : specifies the time the last key version is created. You can call the DescribeKey operation and check the `LastRotationDate` parameter to obtain the time.
- `NextRotationTime` : specifies the time KMS performs the next rotation task to create a new key version. You can call the DescribeKey operation and check the `NextRotationDate` parameter to obtain the time.

Notice When you update the RotationInterval parameter of an automatic rotation policy, the value of NextRotationTime may be a point in time in the past. This does not affect the execution of the automatic rotation policy. If this situation occurs, KMS executes the automatic rotation policy immediately.

Impacts of automatic rotation on encryption and decryption

After a CMK is rotated, KMS uses the current version of the CMK to encrypt data. For data that is encrypted by using a historical version of a CMK, you can use only the historical version to decrypt the data.

You can call the ReEncrypt operation to decrypt the ciphertext that is encrypted by using a historical version of a CMK and encrypt the obtained plaintext data by using the current version of the CMK.

Impact of CMK status on automatic rotation

A new key version can be created for a CMK only if this CMK is in the *Enabled* state (the value of KeyState). You must take note of the following points:

- If a CMK is in the *Disabled* or *Pending Deletion* state, do not call the UpdateRotationPolicy operation to update its automatic rotation policy.
- If a CMK enters the *Disabled* or *Pending Deletion* state after you enable automatic rotation for it, KMS suspends the execution of automatic rotation. In this case, if you call the DescribeKey operation, the returned value of the AutomaticRotation parameter is *Suspended*. When the CMK enters the Enabled state again, its automatic rotation policy becomes active.

Limits

The following keys do not support multiple key versions:

- Service-managed keys: the default keys managed by KMS for specific cloud services. These keys belong to the users of cloud services and are used to provide basic encryption protection for user data.
- Keys based on BYOK: the keys that you imported to KMS. The Origin attribute of these keys is *EXTERNAL*. KMS does not generate key material or initiate rotation tasks for these keys. For more information about how to import key material, see Import key material.

These two types of keys do not support version-based manual or automatic key rotation. A BYOK-based key does not have multiple versions in KMS due to the following reasons:

- Users have strong control over the persistence and lifecycle of BYOK-based keys. This makes management of the keys difficult and error-prone. For example, you must have on-premises key management facilities, data must be synchronized between on-premises facilities and the cloud, and no grace period is provided for key material deletion on the cloud. The complexity of maintaining multiple versions of BYOK-based keys makes key management much more risky.
- Both primary and non-primary key versions may become unavailable at different points in time. For example, if key versions are deleted by KMS or imported again when they expire, CMKs may not be synchronized and encrypted data may not be decrypted by using the correct CMK. This affects system integrity.

8.3. Manual key rotation

If your Customer Master Keys (CMKs) do not support version-based automatic rotation, you can manually rotate the CMKs. This is an alternative solution that does not depend on whether automatic key rotation is supported.

Custom data encryption scenario

On-premise or cloud applications can call the API operation to implement custom data encryption. Examples:

- Encrypt sensitive data such as ID card numbers, credit card information, and home addresses before writing it to databases
- Encrypt data at the client side before uploading it to OSS
- Encrypt service profiles that contain sensitive data and SSL key certificates such as application profiles

You can use the key alias feature to rotate encryption keys within applications. The ID and alias of the key are not required when you call the Decrypt API operation.

In this scenario, you must perform the following steps:

1. Initial configuration
   i. The administrator creates a CMK, whose ID is CMK-A.
   ii. The administrator binds the alias/MyAppKey alias to CMK-A.
   iii. When the application encryption module calls the Encrypt API operation, the value of the KeyId parameter is *alias/MyAppKey*. KMS finds that alias/MyAppKey is bound to CMK-A and then uses CMK-A to encrypt data.

iv. When the application decryption module calls the Decrypt API operation, the KeyId parameter is not used. KMS uses the CMK used to encrypt the data to decrypt the data.

2. Manual rotation

   i. The administrator creates a CMK, whose ID is CMK-B.

   ii. The administrator calls the UpdateAlias API operation to bind the alias/MyAppKey alias to CMK-B.

   iii. When the application decryption module calls the Encrypt API operation, the value of the KeyId parameter is *alias/MyAppKey*. KMS finds that alias/MyAppKey is bound to CMK-B and uses CMK-B to encrypt data.

   iv. When the application decryption module calls the Decrypt API operation, the KeyId parameter is not used. KMS uses the CMK used to encrypt the data to decrypt the data.

Server encryption scenario

Other cloud services can encrypt their data by integrating KMS API operations. The following situations may occur in key rotation scenarios:

- Automatic rotation policies configured on KMS affect the server encryption of other cloud services

  Cause: After CMK encryption is configured for cloud services, they will call the GenerateDataKey API operation of KMS to generate data keys. When KMS generates a new primary key version, cloud services use the new version to generate new data key. A typical example of such cloud services is OSS. In this situation, if you want to enable automatic key rotation, you cannot use service managed keys or BYOKs imported to KMS because they do not support automatic rotation.

- Automatic rotation policies configured on KMS do not affect server encryption of other cloud services

  Cause: After CMK encryption is configured for cloud services, the services only call the GenerateDataKey API operation of KMS once to generate keys to encrypt specific resources. When KMS generates a new primary key version, cloud services will not use it. For example, when encrypting a cloud disk, ECS calls the GenerateDataKey API operation of KMS once to generate a volume encryption key. This key will not be updated again after it is created.

If automatic rotation policies configured on KMS do not affect server encryption of other cloud services or you want to rotate data keys when BYOKs are used, you can change configurations and copy data to achieve the same effect as key rotation. These methods depend on the features of different cloud services. For more information, see documentation of respective cloud services.