Alibaba Cloud

Key Management Service Best Practices

Document Version: 20211227

C-J Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

- You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloudauthorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
- 2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.
- 3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
- 4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).
- 5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud and/or its affiliates Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.
- 6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

| Style | Description | Example |
|-----------------|--|--|
| <u>↑</u> 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. |
| O 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. |
| C) 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 listinstanceid 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.Use the exponential backoff method to retry requests | 05 |
|---|------------------|
| 2.Use a KMS CMK to encrypt and decrypt data online | <mark>0</mark> 8 |
| 3.Use envelope encryption to encrypt and decrypt local data | 12 |
| 4.Use KMS to protect ECS workloads with a few clicks | 17 |
| 5.Use KMS to encrypt Kubernetes Secrets at rest | 22 |

1.Use the exponential backoff method to retry requests

When you call API operations of Key Management Service (KMS), errors are returned sometimes. This topic describes how to use the exponential backoff method to retry the failed requests.

Background information

If an error occurs when you call an API operation, you can retry the request in the application.

Some Alibaba Cloud SDKs support automatic retries. For example, you can configure automatic retry policies for Alibaba Cloud SDKs for .NET. If your SDK does not support automatic retries, you can use the method described in this topic.

Retry methods

If a server error (5xx) is returned or your request is denied due to throttling, you can use one of the following methods:

• Simple retry

Retry a request at fixed intervals within a specified period of time.

• Exponential backoff

For consecutive errors, increase the waiting time between retries exponentially based on a maximum backoff time and a maximum number of retries. This method helps prevent constant errors returned during the retry process. For example, you can use this method to retry a request that was denied due to throttling. This reduces the number of throttling errors returned within a short period of time.

Pseudocode for exponential backoff

The following pseudocode demonstrates how to achieve exponential increase of waiting time between retries:

```
initialDelay = 200
retries = 0
D0
wait for (2^retries * initialDelay) milliseconds
status = CallSomeAPI()
IF status == SUCCESS
    retry = false // Succeeded, stop calling the API again.
ELSE IF status = THROTTLED || status == SERVER_NOT_READY
    retry = true // Failed because of throttling or server busy, try again.
ELSE
    retry = false // Some other error occurred, stop calling the API again.
END IF
    retries = retries + 1
WHILE (retry AND (retries < MAX RETRIES))</pre>
```

Use the exponential backoff method to process throttling errors in KMS

> Document Version: 20211227

The following Java code demonstrates how to use the exponential backoff method to process the throttling errors returned when the Decrypt operation is called.

- You can make simple modifications to the code to process certain server errors, such as HTTP error 503.
- You can estimate the number of requests that the client will initiate within a specific period of time. Then, adjust the initial delay (initialDelay) and maximum number of retries (maxRetries) in the code based on the estimation result.

```
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.exceptions.ClientException;
import com.aliyuncs.http.FormatType;
import com.aliyuncs.http.HttpClientConfig;
import com.aliyuncs.http.MethodType;
import com.aliyuncs.http.ProtocolType;
import com.aliyuncs.kms.model.v20160120.DecryptRequest;
import com.aliyuncs.kms.model.v20160120.DecryptResponse;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;
import java.io.*;
public class CmkDecrypt {
   private static DefaultAcsClient kmsClient(String regionId, String accessKeyId, String a
ccessKeySecret) {
        IClientProfile profile = DefaultProfile.getProfile(regionId, accessKeyId, accessKey
Secret);
       HttpClientConfig clientConfig = HttpClientConfig.getDefault();
        profile.setHttpClientConfig(clientConfig);
        return new DefaultAcsClient(profile);
    }
    private static String kmsDecrypt (String cipherTextBlob) throws ClientException {
        final DecryptRequest request = new DecryptRequest();
        request.setSysProtocol(ProtocolType.HTTPS);
       request.setAcceptFormat(FormatType.JSON);
       request.setSysMethod(MethodType.POST);
        request.setCiphertextBlob(cipherTextBlob);
        DecryptResponse response = kmsClient.getAcsResponse(request);
       return response.getPlaintext();
    }
    public static long getWaitTimeExponential(int retryCount) {
        final long initialDelay = 200L;
        long waitTime = ((long) Math.pow(2, retryCount) * initialDelay);
        return waitTime;
    public static void main(String[] args) {
        String regionId = "xxxxx"; //"cn-shanghai"
        String accessKeyId = "xxxxx";
        String accessKeySecret = "xxxxxx";
        String cipherTextBlob = "xxxxxx";
        int maxRetries = 5;
        kmsClient = kmsClient(regionId, accessKeyId, accessKeySecret);
        for (int i=0; i<maxRetries; i++) {</pre>
            try {
                String plainText = kmsDecrypt(cipherTextBlob);
                return;
```

```
} catch (ClientException e) {
    if (e.getErrCode().contains("Rejected.Throttling")) {//need retry
        try {
            Thread.sleep(getWaitTimeExponential(i+1));
        } catch (InterruptedException ignore) {
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

2.Use a KMS CMK to encrypt and decrypt data online

You must encrypt sensitive information in your IT assets that are deployed on Alibaba Cloud. You can call cryptographic API operations of Key Management Service (KMS) to encrypt or decrypt data less than 6 KB online.

Context

You can use a customer master key (CMK) to encrypt and decrypt data in but not limited to the following scenarios:

- Encrypt configuration files.
- Encrypt the private keys of SSL certificates.

This topic describes how to call KMS API operations to encrypt and decrypt the private keys of SSL certificates online.

Encryption and decryption procedure

The data of the user is transmitted to the KMS server over a secure channel. The KMS server encrypts or decrypts the data and returns the result to the user over a secure channel. The following figure shows the entire procedure.

Procedure:

- 1. Create a CMK in the KMS console or by calling the CreateKey operation.
- 2. Call the Encrypt operation of KMS to encrypt the private key of an SSL certificate. The ciphertext private key is returned.
- 3. Deploy the SSL certificate by using the ciphertext private key to an Elastic Compute Service (ECS) instance.
- 4. Call the Decrypt operation of KMS to decrypt the ciphertext when the ECS instance starts and needs to use the SSL certificate.

Related operations

You can call the following API operations to encrypt and decrypt data.

| Operation | Description |
|-------------|---|
| CreateKey | Creates a CMK. |
| CreateAlias | Creates an alias for a CMK. |
| Encrypt | Encrypts data by using a specific CMK. |
| Decrypt | Decrypts data that is encrypted by KMS. You do not need to specify a CMK. |

Encrypt and decrypt the private key of an SSL certificate

1. Call the CreateKey operation to create a CMK.

aliyun kms CreateKey

Expected output:

```
{
 "KeyMetadata": {
   "CreationDate": "2019-04-08T07:45:54Z",
   "Description": "",
   "KeyId": "1234abcd-12ab-34cd-56ef-12345678****",
   "KeyState": "Enabled",
   "KeyUsage": "ENCRYPT/DECRYPT",
   "DeleteDate": "",
   "Creator": "151266687691****",
   "Arn": "acs:kms:cn-hangzhou:151266687691****:key/1234abcd-12ab-34cd-56ef-12345678**
**",
   "Origin": "Aliyun KMS",
   "MaterialExpireTime": ""
 },
 "RequestId": "2a37b168-9fa0-4d71-aba4-2077dd9e80df"
}
```

2. (Optional)Create an alias for the CMK. We recommend that you perform this step.

Aliases are optional to CMKs. If a CMK does not have an alias, you can use the ID of the CMK.

```
aliyun kms CreateAlias --AliasName alias/Apollo/WorkKey --KeyId 1234abcd-12ab-34cd-56ef -12345678****
```

Note In this example, Apollo/WorkKey is created for the CMK used in the Apollo project as the alias. This alias is used in the subsequent sample code. You can use alias/Apoll o/WorkKey to reference the CMK when you call the Encrypt operation.

3. Call the Encrypt operation to encrypt the private key of an SSL certificate. Then, KMS encrypts the private key.

In the following sample code:

- alias/Apollo/WorkKey is the alias of the CMK.
- ./certs/key.pem is the plaintext private key.
- ./certs/key.pem.cipher is the ciphertext private key.

```
#!/usr/bin/env python
#coding=utf-8
import json
from aliyunsdkcore import client
from aliyunsdkkms.request.v20160120 import EncryptRequest
from aliyunsdkkms.request.v20160120 import DecryptRequest
def KmsEncrypt(client, plaintext, key alias):
 request = EncryptRequest.EncryptRequest()
 request.set accept format('JSON')
 request.set KeyId(key alias)
 request.set Plaintext(plaintext)
 response = json.loads(client.do action(request))
 return response.get("CiphertextBlob")
def ReadTextFile(in file):
 file = open(in file, 'r')
 content = file.read()
 file.close()
 return content
def WriteTextFile(out file, content):
 file = open(out file, 'w')
 file.write(content)
 file.close()
clt = client.AcsClient('<Access-Key-Id>','Access-Key-Secret','<Region-Id>')
key alias = 'alias/Apollo/WorkKey'
in file = './certs/key.pem'
out_file = './certs/key.pem.cipher'
# Read private key file in text mode
in_content = ReadTextFile(in_file)
# Encrypt
ciphertext = KmsEncrypt(clt, in content, key alias)
# Write encrypted key file in text mode
WriteTextFile(out_file, ciphertext)
```

4. Call the Decrypt operation to decrypt the ciphertext private key. Then, KMS decrypts the private key that you have deployed to your ECS instance.

In the following sample code:

- ./certs/key.pem.cipher is the ciphertext private key.
- ./certs/decrypted_key.pem is the plaintext private key.

```
#!/usr/bin/env python
#coding=utf-8
import json
from aliyunsdkcore import client
from aliyunsdkkms.request.v20160120 import EncryptRequest
from aliyunsdkkms.request.v20160120 import DecryptRequest
def KmsDecrypt(client, ciphertext):
 request = DecryptRequest.DecryptRequest()
 request.set accept format('JSON')
 request.set CiphertextBlob(ciphertext)
response = json.loads(client.do action(request))
return response.get("Plaintext")
def ReadTextFile(in file):
file = open(in_file, 'r')
 content = file.read()
 file.close()
 return content
def WriteTextFile(out file, content):
file = open(out file, 'w')
 file.write(content)
 file.close()
clt = client.AcsClient('<Access-Key-Id>', 'Access-Key-Secret', '<Region-Id>')
in file = './certs/key.pem.cipher'
out file = './certs/decrypted key.pem'
# Read encrypted key file in text mode
in_content = ReadTextFile(in_file)
# Decrypt
ciphertext = KmsDecrypt(clt, in_content)
# Write Decrypted key file in text mode
WriteTextFile(out file, ciphertext)
```

3.Use envelope encryption to encrypt and decrypt local data

You must encrypt sensitive information in your IT assets that are deployed on Alibaba Cloud. If you need to encrypt a large amount of local data, you can call cryptographic API operations of Key Management Service (KMS) to generate a data key online and then use the data key to encrypt the local data offline. This encryption mechanism is known as envelope encryption.

Context

You can use envelope encryption in, but not limited to, the following scenarios:

- Encrypt business data files.
- Encrypt all data stored on local disks.

This topic describes how to use envelope encryption to encrypt and decrypt local files.

How data encryption and decryption work

Use KMS to create a customer master key (CMK), use the CMK to generate a data key, and then use the data key to encrypt and decrypt local files. Envelope encryption is suitable for encrypting and decrypting a large amount of data. The following figure shows the procedure of envelope encryption.

• Envelope encryption



Procedure:

- i. Create a CMK in the KMS console or by calling the CreateKey operation.
- ii. Call the GenerateDataKey operation to generate a data key. KMS returns the plaintext and ciphertext of the data key.
- iii. Use the plaintext data key to encrypt the local files and then delete the plaintext data key from the memory.

iv. Store the ciphertext data key and encrypted data files on a persistent storage device or service.

• Envelope decryption



Procedure:

- i. Retrieve the ciphertext data key from the local files.
- ii. Call the Decrypt operation of KMS to decrypt the ciphertext data key. The plaintext of the data key is returned.
- iii. Use the plaintext data key to decrypt the local files and then delete the plaintext data key from the memory.

Encryption and decryption API operations

You can call the KMS API operations described in the following table to encrypt and decrypt local files.

| Operation | Description |
|-----------------|--|
| CreateKey | Creates a CMK. |
| CreateAlias | Creates an alias for a CMK. |
| GenerateDataKey | Generates a data key, uses a specified CMK to encrypt the data key, and then returns the plaintext and ciphertext of the data key. |
| Decrypt | Decrypts data that is encrypted in KMS, including the ciphertext data key generated by calling the GenerateDataKey operation. You do not need to specify a CMK. |

Encrypt and decrypt local files

You can create a CMK and encrypt and decrypt local files by using Alibaba Cloud CLI.

1. Call the CreateKey operation to create a CMK.

aliyun kms CreateKey

Expected output:

```
{
 "KeyMetadata": {
   "CreationDate": "2019-04-08T07:45:54Z",
   "Description": "",
   "KeyId": "1234abcd-12ab-34cd-56ef-12345678****",
   "KeyState": "Enabled",
   "KeyUsage": "ENCRYPT/DECRYPT",
   "DeleteDate": "",
   "Creator": "151266687691****",
   "Arn": "acs:kms:cn-hangzhou:151266687691****:key/1234abcd-12ab-34cd-56ef-12345678**
**",
   "Origin": "Aliyun_KMS",
   "MaterialExpireTime": ""
 },
 "RequestId": "2a37b168-9fa0-4d71-aba4-2077dd9e80df"
}
```

2. Optional. Create an alias for the CMK.

Aliases, which are optional, are used to identify CMKs. If a CMK does not have an alias, you can use the ID of the CMK to identify the CMK.

```
aliyun kms CreateAlias --AliasName alias/Apollo/WorkKey --KeyId 1234abcd-12ab-34cd-56ef -12345678****
```

Note In this example, Apollo/WorkKey is used as the alias of the CMK for key encryption in the Apollo project. You can use the alias alias/Apollo/WorkKey in subsequent sample code to call the Encrypt operation.

3. Encrypt a local file.

In the following sample code:

- alias/Apollo/WorkKey is the alias of the CMK.
- ./data/sales.csv is the plaintext data file.
- ./data/sales.csv.cipher is the returned ciphertext data file.

```
#!/usr/bin/env python
#coding=utf-8
import json
import base64
from Crypto.Cipher import AES
from aliyunsdkcore import client
from aliyunsdkkms.request.v20160120 import GenerateDataKeyRequest
def KmsGenerateDataKey(client, key alias):
    request = GenerateDataKeyRequest.GenerateDataKeyRequest()
   request.set accept format('JSON')
   request.set KeyId(key alias)
    request.set NumberOfBytes(32)
    response = json.loads(client.do action(request))
   datakey encrypted = response["CiphertextBlob"]
   datakey plaintext = response["Plaintext"]
   return (datakey_plaintext, datakey_encrypted)
def ReadTextFile(in file):
 file = open(in file, 'r')
 content = file.read()
 file.close()
  return content
def WriteTextFile(out_file, lines):
 file = open(out file, 'w')
 for ln in lines:
    file.write(ln)
   file.write('\n')
 file.close()
# Out file format (text)
# Line 1: b64 encoded data key
# Line 2: b64 encoded IV
# Line 3: b64 encoded ciphertext
# Line 4: b64 encoded authentication tag
def LocalEncrypt(datakey plaintext, datakey encrypted, in file, out file):
 data_key_binary = base64.b64decode(datakey_plaintext)
 cipher = AES.new(data key binary, AES.MODE EAX)
 in content = ReadTextFile(in file)
 ciphertext, tag = cipher.encrypt and digest(in content)
 lines = [datakey_encrypted, base64.b64encode(cipher.nonce), base64.b64encode(cipherte
xt), base64.b64encode(tag)];
 WriteTextFile(out file, lines)
clt = client.AcsClient('Access-Key-Id', 'Access-Key-Secret', 'Region-Id')
key_alias = 'alias/Apollo/WorkKey'
in file = './data/sales.csv'
out file = './data/sales.csv.cipher'
# Generate Data Key
datakey = KmsGenerateDataKey(clt, key alias)
# Locally Encrypt the sales record
LocalEncrypt(datakey[0], datakey[1], in_file, out_file)
```

4. Decrypt a local file.

In the following sample code:

• ./data/sales.csv.cipher is the ciphertext data file.

• ./data/decrypted_sales.csv is the returned plaintext data file.

```
#!/usr/bin/env python
#coding=utf-8
import json
import base64
from Crypto.Cipher import AES
from aliyunsdkcore import client
from aliyunsdkkms.request.v20160120 import DecryptRequest
def KmsDecrypt(client, ciphertext):
request = DecryptRequest.DecryptRequest()
 request.set accept format('JSON')
 request.set_CiphertextBlob(ciphertext)
 response = json.loads(client.do action(request))
 return response.get("Plaintext")
def ReadTextFile(in file):
 file = open(in_file, 'r')
 lines = []
 for ln in file:
   lines.append(ln)
 file.close()
  return lines
def WriteTextFile(out_file, content):
 file = open(out file, 'w')
 file.write(content)
 file.close()
def LocalDecrypt(datakey, iv, ciphertext, tag, out file):
 cipher = AES.new(datakey, AES.MODE EAX, iv)
 data = cipher.decrypt and verify(ciphertext, tag).decode('utf-8')
 WriteTextFile(out file, data)
clt = client.AcsClient('Access-Key-Id', 'Access-Key-Secret', 'Region-Id')
in file = './data/sales.csv.cipher'
out_file = './data/decrypted_sales.csv'
# Read encrypted file
in lines = ReadTextFile(in file)
# Decrypt data key
datakey = KmsDecrypt(clt, in lines[0])
# Locally decrypt the sales record
LocalDecrypt(
 base64.b64decode(datakey),
 base64.b64decode(in lines[1]), # IV
 base64.b64decode(in lines[2]), # Ciphertext
 base64.b64decode(in lines[3]), # Authentication tag
 out file
  )
```

4.Use KMS to protect ECS workloads with a few clicks

In most scenarios, the production data that you use Elastic Compute Service (ECS) workloads to process contains your business secrets, privacy information, or important credentials. Therefore, you must protect these workloads against information leaks. Key Management Service (KMS) supports quick encryption for ECS workloads to protect transient and persistent data generated in computing environments. This ensures data confidentiality and privacy and meets your requirements for compliance. KMS helps build a secure cloud computing environment at low costs.

Background information

Customers have data security requirements on business secrets and personal privacy. These types of data is essential to enterprises and is subject to regulatory compliance. For example, General Data Protection Regulation (GDPR) requires enterprises to protect personal privacy data. The data is stored in databases. Before you store the data of an application system in databases, you must encrypt the data to reduce the risk of data leaks caused by attacks such as dictionary attacks.

To ensure the security and compliance of encryption, you can use KMS or Data Encryption Service to encrypt business data of an application system. For more information about how to encrypt your business data at the application layer, see Use envelope encryption to encrypt and decrypt local data.

If you have encrypted your business data, the workloads that are used to encrypt and decrypt data become a weak link in your system. Take note of the following points, which may cause security risks:

- Your applications deployed on an ECS instance contain the important credentials that are used to access KMS, hardware security modules (HSMs), microservices, or subsystems.
- The system disks of your ECS instances may generate some temporary files, including sensitive data files involved in network transmission and local data processing.
- Disk backup based on automatic snapshot is enabled for your ECS instances to store multiple copies of sensitive data. This helps achieve data redundancy.

? Note The preceding risks are some examples. When you deploy business systems, other issues may occur. For example, if the application deployment and lifecycle change mechanisms are used in DevOps mode, O&M and security engineers are unaware of whether new types of sensitive data are generated for workloads. They cannot determine whether to introduce business logic to process the new sensitive data.

Benefits

Alibaba Cloud ECS instances use KMS to protect the resources that ECS workloads require. The resources include system disks and data disks of ECS instances and relevant images and snapshots.

You can authorize ECS instances to use customer master keys (CMKs) in KMS and encrypt the resources with a few clicks. This protects known, potential, transient, and persistent sensitive data against unauthorized access. In emergencies, you can disable KMS-based decryption on an ECS instance by revoking permissions or disabling CMKs.

O&M and security engineers can encrypt resources that ECS workloads require in DevOps mode. This is simple and efficient.

Encrypt a system disk

A system disk contains the operating system and application software required for business operations. It is always packaged as an image.

After you create a custom image that can run in a production environment and use the custom image as a baseline, you can copy and encrypt the image. This way, an encrypted system disk is created.

- 1. Log on to the ECS console.
- 2. In the left-side navigation pane, choose Instances & Images > Images.
- 3. In the top navigation bar, select a region.
- 4. On the Images page, click the Custom Image tab.
- 5. Find your image and click Copy Image in the Actions column.

? Note If the size of your image is greater than 500 GiB, you are prompted to submit a ticket to complete the operation when you click **Copy Image**.

6. In the Copy Image dialog box, select Encrypt. Then, select a key from the drop-down list.

By default, **Default Service CMK** is used, which indicates a service-managed CMK. You can also select a CMK that you create in KMS for encryption. The CMK that you create allows for more permissions on the data disk. In emergencies, you can disable KMS-based decryption on an ECS instance by revoking permissions or disabling CMKs.

⑦ Note If this is your first time to select a different custom encryption key, click Confirm Authorization Policy and select AliyunECSDiskEncryptDefaultRole to allow ECS to access your KMS resources. This step describes only how to configure the encryption settings when you copy a custom image. For more information about other settings, see Copy custom images.

| Copy Image | | |
|--|---|--------|
| Note that a snapshot w operation will incur add | vill be generated in the specified region when you copy an image and this ditional costs. | |
| lmages : | | |
| m-bp1aa3aak | 22:43Z | |
| Destination Region : | | |
| China (Hangzhou) | | \vee |
| Custom Image Name: | | |
| demo0704 | | G |
| Description : | | |
| The specified image is c | opied from image m-b f3v in the China (Hangzhou) region. | |
| Resource Group : | | |
| Select | ~ | |
| 🗸 Encrypt | | |
| alias/ecsdoc | | ~ |
| Default Service CMK | | |
| | | |
| alias/acs/ecs | | |

7. Click OK.

Encrypt a data disk

You can encrypt a data disk when you create an ECS instance or create the disk.

Encrypt a data disk when you create an ECS instance

- 1. Log on to the ECS console.
- 2. In the left-side navigation pane, choose **Instances & Images > Instances**.
- 3. On the Instances page, click Create Instance.
- 4. In the **Storage** section of the page that appears, create a data disk and configure the encryption settings.

? Note This step describes only how to configure the encryption settings when you create an ECS instance. For more information about other settings, see Create an instance by using the wizard.

i. Click Add Disk.

- ii. Specify the disk category and capacity.
- iii. Select Disk Encryption. Then, select a key from the drop-down list.

By default, **Default Service CMK** is used, which indicates a service-managed CMK. You can also select a CMK that you create in KMS for encryption. The CMK that you create allows for more permissions on the data disk. In emergencies, you can disable KMS-based decryption on an ECS instance by revoking permissions or disabling CMKs.

Note If this is your first time to select a different custom encryption key, click Confirm Authorization Policy and select AliyunECSDiskEncryptDefaultRole to allow ECS to access your KMS resources.

| Storage | System Disk | | | | |
|-------------------------|---|--|--|--|--|
| Disk specifications and | Ultra Disk 🔹 40 GiB 2120 IOPS 🗹 Release with Instance | | | | |
| performance | Click here for guidelines on how to select an appropriate disk for your scenario. | | | | |
| | Disk Backup (Recommended) | | | | |
| | You can periodically backup disks with an automatic snapshot policy to prevent risks such as viru | | | | |
| | Data Disk You have selected 1 disks and can select 15 more. | | | | |
| | Enhanced SSD (ESSD) 🔹 40 GiB 3800 IOPS Performance Level | | | | |
| | ✓ Disk Encryption Default Service CMK | | | | |
| | Disk Backup (Recommended) | | | | |
| | + Add Disk | | | | |

Encrypt a data disk when you create the disk

- 1. Log on to the ECS console.
- 2. In the left-side navigation pane, choose **Storage & Snapshots > Disks**.
- 3. In the upper-right corner of the **Disks** page, click **Create Disk**.
- 4. Specify the disk category and capacity.

Note This step describes only how to configure the encryption settings when you create a disk. For more information about other settings, see **Create a disk**.

5. In the **Storage** section of the page that appears, select **Disk Encryption**. Then, select a key from the drop-down list.

By default, **Default Service CMK** is used, which indicates a service-managed CMK. You can also select a CMK that you create in KMS for encryption. The CMK that you create allows for more permissions on the data disk. In emergencies, you can disable KMS-based decryption on an ECS instance by revoking permissions or disabling CMKs.

Note If this is your first time to select a different custom encryption key, click Confirm Authorization Policy and select AliyunECSDiskEncryptDefaultRole to allow ECS to access your KMS resources.

| Disk | Enhanced SSD (ESSD) 🔻 40 Gi8 3800 IOPS Performance Level 💮 : PL1 (up to 50,000 IOPS per disk) 💌 Create from Snapshot 💟 Disk Encryption Default Service CMK 💌 | |
|-------------|---|--|
| | You have purchased Enhanced SSD (ESSD) 0 GB in the region. Remaining quota: 65496 GB | |
| | Click here for guidelines on how to select an appropriate disk for your scenario. Learn how to create a subscription disk. | |
| Quantity | - 1 + Unit | |
| Disk Name | Enter a disk name. | |
| | The name must be 2 to 128 characters in length, and can contain letters, digits, periods (), underscores (), colons (), and hyphens (-). It must start with a letter. | |
| Description | Enter a disk description | |
| | The description must be 2 to 256 characters in length and cannot start with http:// or https://. | |

5.Use KMS to encrypt Kubernetes Secrets at rest

Container Service for Kubernetes (ACK) allows you to use a customer master key (CMK) in Key Management Service (KMS) to encrypt the Secrets of Kubernetes clusters at rest.

Scenarios

ACK provides powerful capabilities in operation orchestration management. It obtains Secrets such as passwords, certificates, credentials, and access keys across products, services, and modules. ACK uses Secret modules to store and manage the sensitive information of Kubernetes clusters and that of business applications in the clusters. It also stores sensitive information in etcd. The replication feature of etcd supports distributed storage.

A Kubernetes cluster in the initialized state (without business load) has about 50 Secrets. The leak of a Secret may cause immeasurable loss to the cluster, the business system, or even the entire enterprise. Therefore, you must protect the Secrets stored in Kubernetes clusters.

Encryption mechanism

Professional managed Kubernetes clusters allow you to use a CMK in KMS to encrypt Secrets. The KMS provider mechanism of Kubernetes is used during encryption. A KMS provider uses envelope encryption to encrypt or decrypt the keys of Secrets that are stored in etcd. For more information about envelope encryption, see What is envelope encryption? Procedures of key encryption and decryption:

- When you store a Kubernetes Secret by using Kubernetes Secret API, the API server generates a random data key to encrypt your business key. Then, the system uses a CMK in KMS to encrypt the data key and store the cyphertext of the data key in etcd.
- When you decrypt the Kubernetes Secret, the system calls the Decrypt operation of KMS to decrypt the data key first. Then, the system uses the plaintext of the data key to decrypt the Kubernetes Secret and returns the decrypted Secret.

Prerequisites

- The Alibaba Cloud account within which you use ACK is assigned the AliyunCSManagedSecurityRole role. If you use an Alibaba Cloud account that is not assigned the role to enable Secret encryption at rest for a new or existing professional managed Kubernetes cluster, you are prompted to assign the role to the Alibaba Cloud account first.
- The RAM user that you use to log on to the ACK console is granted the AliyunKMSCryptoAdminAccess permission. For more information, see Grant permissions to a RAM user.
- A CMK is created in the KMS console. For more information, see Create a CMK.

? Note Only CMKs of the Aliyun_AES_256 type are supported.

Create a professional managed Kubernetes cluster and enable Secret encryption at rest

- 1. Log on to the ACK console.
- 2. In the left-side navigation pane, click **Clusters**.
- 3. In the upper-right corner of the **Clusters** page, click **Cluster Template**.

- 4. In the Select Cluster Template dialog box, select Professional Managed Kubernetes Cluster and click Create.
- 5. On the **Managed Kubernetes** tab, find **Secret Encryption**, select **Select Key**, and then select a CMK ID from the drop-down list.
- 6. Set other parameters by following the on-screen instructions.

For more information, see Create a professional managed Kubernetes cluster.

Enable Secret encryption at rest for an existing professional managed Kubernetes cluster

- 1. On the **Clusters** page, click the name of the professional managed Kubernetes cluster for which you want to enable Secret encryption at rest.
- 2. Click the Basic Information tab. In the Basic Information section, turn on Secret Encryption.
- 3. In the Secret Encryption dialog box, select a CMK ID from the Existing Key drop-down list and click OK.

If the status of the cluster changes from **Updating** to **Running**, Secret encryption at rest is enabled for the cluster.

Results

If you can find encryption or decryption events performed by the AliyunCSManagedSecurityRole role on the **Event Detail Query** page of the ActionTrail console, Secret encryption at rest is enabled for the cluster. You can view all the KMS CMK calling records in the ActionTrail console.