ALIBABA CLOUD

阿里云

密钥管理服务

最佳实践

文档版本：20201112

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	**危险** 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	**警告** 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	**注意** 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	**说明** 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.使用指数退避方法对请求错误进行重试 05
2.使用KMS主密钥在线加密、解密数据 08
3.使用KMS信封加密在本地加密或解密数据 12
4.使用KMS一键保护ECS工作负载 18
5.使用KMS保护支付宝应用和小程序 21
6.使用KMS一键加密Kubernetes集群Secret 26

1.使用指数退避方法对请求错误进行重试

当您调用KMS的API时，有时会返回错误信息。本文介绍了如何使用指数退避方法对请求错误进行重试。

背景信息

当您调用服务接口时，有时会在某一环节出现错误，此时您可以在应用程序中进行重试。

一些阿里云SDK支持通过配置，自动实现对请求的错误重试。例如：使用阿里云的.NET SDK可以配置重试的策略。当自动重试方式不适用时，您可以使用本文介绍的重试方法对请求错误进行重试。

重试策略

请求出现错误时，如果是服务器错误（5xx）或请求限流错误，则可以通过如下重试策略对请求错误进行重试：

- 简单重试。

  例如：总共重试10秒钟，每秒钟重试一次。

- 指数退避。

  对于连续错误响应，重试等待间隔越来越长，您需要按照最长延迟间隔和最大重试次数进行重试。指数退避可以防止在重试过程中持续不断的发生冲突。例如：在短时间发出超过限流配额数的请求时，通过指数退避的方式，可以有效规避持续的限流错误。

指数退避的伪代码

以下代码介绍了如何使用增量延迟方法重试某个操作。

```
initialDelay = 200
retries = 0

DO
  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))
```

使用指数退避方法处理KMS限流

以下Java示例介绍了如何使用指数退避的方式，处理KMS调用Decrypt接口时遇到的限流错误。

- 您可以通过简单修改，对特定类型的服务器错误（例如：HTTP 503）进行重试。
- 您可以通过精细的预估客户端在特定时间段内发出的请求数，调整初始延迟值（ `initialDelay` ）和重试次数（ `maxRetries` ）。

```
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 accessKeySecret) {
    IClientProfile profile = DefaultProfile.getProfile(regionId, accessKeyId, accessKeySecret);
    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 int initialDelay = 200;
```

```
    final int 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++) {
      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.使用KMS主密钥在线加密、解密数据

阿里云用户在云上部署IT资产，需要对敏感数据进行加密保护。如果被加密的数据对象较小（小于6KB），则可以通过密钥管理服务（Key Management Service，简称KMS）的密码运算API，在线对数据直接加解密。

前提条件

进行操作前，请确保您已经注册了阿里云账号。如还未注册，请先完成账号注册。

背景信息

典型的使用场景包括（但不限于）：

- 对配置文件的加密
- 对SSL私钥的加密

本文以加密SSL证书私钥为例，介绍如何调用KMS API实现对数据的在线加密、解密。

产品架构

用户的数据会通过安全信道传递到KMS服务端，服务端完成加密、解密后，操作结果通过安全信道返回给用户。具体架构如下图所示。

操作流程如下：

1. 通过KMS控制台或者调用CreateKey接口，创建一个用户主密钥。
2. 调用KMS服务的Encrypt接口，将明文证书加密为密文证书。
3. 将密文证书部署在云服务器上。
4. 当服务器启动需要使用证书时，调用KMS服务的Decrypt接口将密文证书解密为明文证书。

相关API

您可以调用以下KMS API，完成对数据的加密或解密操作。

API名称	说明
CreateKey	创建用户主密钥（CMK）。
CreateAlias	为指定用户主密钥创建一个别名。
Encrypt	指定CMK，由KMS加密数据。
Decrypt	解密KMS直接加密的数据，不需要指定CMK。

加密/解密证书密钥

1. 通过调用CreateKey，创建用户主密钥。

```
$ aliyun kms CreateKey
{
 "KeyMetadata": {
  "CreationDate": "2019-04-08T07:45:54Z",
  "Description": "",
  "KeyId": "1234abcd-12ab-34cd-56ef-12345678****",
  "KeyState": "Enabled",
  "KeyUsage": "ENCRYPT/DECRYPT",
  "DeleteDate": "",
  "Creator": "111122223333",
  "Arn": "acs:kms:cn-hangzhou:111122223333:key/1234abcd-12ab-34cd-56ef-12345678****",
  "Origin": "Aliyun_KMS",
  "MaterialExpireTime": ""
 },
 "RequestId": "2a37b168-9fa0-4d71-aba4-2077dd9e80df"
}
```

2. （可选）给主密钥添加别名（推荐步骤）。 别名是用户主密钥的可选标识。如果用户不创建别名，也可以直接使用密钥的ID。

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

> **说明** 其中，`Apollo/WorkKey` 表示Apollo项目中的工作密钥（当前被用于加密的密钥），并在后续示例代码中使用此别名。即表示应用可以使用 `alias/Apollo/WorkKey` 调用加密API。

3. 加密证书私钥（业务系统对SSL私钥证书进行加密保护）。

   示例代码中：

   - 用户主密钥：别名为 `alias/Apollo/WorkKey`
   - 明文证书文件：*./certs/key.pem*
   - 输出的密文证书文件：*./certs/key.pem.cipher*

```
#!/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):
```

```
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(clt.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. 解密证书私钥（业务系统对部署在云上的密文证书私钥进行解密）。

   示例代码中：

   - 部署的密文证书文件：*./certs/key.pem.cipher*
   - 输出的明文证书文件：*./certs/decrypted_key.pem*

```
#!/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(clt.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.使用KMS信封加密在本地加密或解密数据

阿里云用户在云上部署IT资产，需要对敏感数据进行加密保护。如果被加密的数据对象较大，则可以通过KMS的密码运算API在线生成数据密钥，用离线数据密钥在本地加密大量数据。这类加密模式叫作信封加密。

前提条件

进行操作前，请确保您已经注册了阿里云账号。如还未注册，请先完成账号注册。

背景信息

典型的场景包括（但不限于）：

- 对业务数据文件的加密
- 对全磁盘数据加密

本文以加密本地文件为例，介绍如何使用KMS实现对数据的信封加密，以及如何解密被信封加密的数据。

产品架构

使用KMS创建一个主密钥，使用主密钥生成一个数据密钥，再使用数据密钥在本地加解密数据。这种场景适用于大量数据的加解密。具体架构如下所示。

- 信封加密

  操作流程如下：

  i. 通过KMS控制台，或者调用CreateKey接口，创建一个用户主密钥。
  ii. 调用GenerateDataKey接口创建一个数据密钥。KMS会返回一个明文的数据密钥和一个密文的数据密钥。
  iii. 使用明文的数据密钥加密文件，产生密文文件，然后销毁内存中的明文密钥。
  iv. 用户将密文数据密钥和密文文件一同存储到持久化存储设备或服务中。

- 信封解密

  操作流程如下：

  i. 从本地文件中读取密文数据密钥。
  ii. 调用KMS服务的Decrypt接口，将加密过的密钥解密为明文密钥。
  iii. 用明文密钥为本地数据解密，再销毁内存中的明文密钥。

相关API

您可以调用以下KMS API，在本地对数据进行加解密。

API名称	说明
CreateKey	创建用户主密钥（CMK）。
CreateAlias	为指定用户主密钥创建一个别名。

API名称	说明
GenerateDataKey	在线生成数据密钥，用指定CMK加密数据密钥后，返回数据密钥的密文和明文。
Decrypt	解密KMS直接加密的数据（包括GenerateDataKey产生的数据密钥的密文），不需要指定CMK。

加密或解密本地文件

1. 创建用户主密钥。

```
$ aliyun kms CreateKey
{
 "KeyMetadata": {
  "CreationDate": "2019-04-08T07:45:54Z",
  "Description": "",
  "KeyId": "1234abcd-12ab-34cd-56ef-12345678****",
  "KeyState": "Enabled",
  "KeyUsage": "ENCRYPT/DECRYPT",
  "DeleteDate": "",
  "Creator": "111122223333",
  "Arn": "acs:kms:cn-hangzhou:111122223333:key/1234abcd-12ab-34cd-56ef-12345678****",
  "Origin": "Aliyun_KMS",
  "MaterialExpireTime": ""
 },
 "RequestId": "2a37b168-9fa0-4d71-aba4-2077dd9e80df"
}
```

2. （可选）给主密钥添加别名。

   别名是用户主密钥的可选标识。如果用户不创建别名，也可以直接使用密钥的ID。

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

   > **说明** 其中，Apollo/WorkKey表示Apollo项目中的工作密钥（当前被用于加密的密钥），并在后续示例代码中使用此别名。即表示应用可以使用alias/Apollo/WorkKey调用加密API。

3. 加密本地文件。

   示例代码中：

   - 用户主密钥：别名为 `alias/Apollo/WorkKey`
   - 明文数据文件：*./data/sales.csv*
   - 输出的密文数据文件：*./data/sales.csv.cipher*

```
#!/usr/bin/env python
```

```
#!/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(ciphertext), 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. 解密本地文件。

   示例代码中：

   - 密文数据文件：*./data/sales.csv.cipher*
   - 输出的明文数据文件：*./data/decrypted_sales.csv*

```
#!/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_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.使用KMS一键保护ECS工作负载

当ECS工作负载用于处理生产数据时，通常会接触到您的业务机密、隐私信息或者关键凭证，因此需要对工作负载进行保护以防范信息泄露。KMS支持对ECS工作负载进行一键加密，保护计算环境中产生的临时和持久数据，满足您对数据安全、隐私以及合规的要求，让您可以高效、低成本的构建安全的云上计算环境。

背景信息

客户的数据安全需求为保护业务机密和个人隐私。这类数据是企业的核心价值所在，通常受到监管合规的约束。例如：GDPR会要求企业保护个人的隐私数据。这类数据通常存储在数据库，应用系统应当在存储之前将其加密，降低数据库面临撞库拖库等攻击之后泄露的风险。

为了保证加密的安全性与合规性，应用系统可以使用KMS或者加密服务完成业务数据的加密。应用层加密业务数据详情，请参见使用KMS信封加密在本地加密或解密数据。

如果您已经采取了上述保护手段，那么处理加解密的工作负载就替代了数据库，成为了您的系统中新的薄弱环节。工作负载携带的风险如下：

- 您的ECS应用中，有访问KMS或者密码机，以及访问其他微服务、子系统的关键凭证。
- 您的ECS系统盘，可能产生一些临时文件，包含网络传输和本地处理过程中接触到的敏感数据。
- 您的ECS云盘，开启了基于自动快照的云盘备份，对敏感数据进行大量冗余存储。

> **说明** 实际的业务系统部署会面临更多的问题，例如：在研发（DevOps）自治的应用部署和生命周期变更机制下，运维与安全负责人并不知道工作负载是否产生了新的敏感数据类型，是否引入新的业务逻辑处理敏感数据。

产品价值

阿里云ECS基于KMS加密，提供保护工作负载所属资源的能力，例如：ECS系统盘、数据盘，以及和它们相关的镜像、快照。

您可以授权ECS使用KMS中的用户主密钥（CMK），一键加密这些资源，保护已知、未知、临时和持久性的敏感数据，防范它们被恶意者获取。您也可以根据需求，通过撤销授权、禁用密钥等手段，撤销ECS使用KMS解密的能力，获得应急响应的能力。

> **说明** 对运维与安全负责人而言，加密ECS工作负载的资源是研发（DevOps）模式下，简单而有效的安全兜底方案。

加密系统盘

由于系统盘包含操作系统以及业务所需要的应用软件，因此它通常被打包为一个镜像。

当您制作完成这个具备在生产环境运行的自定义镜像并作为基线之后，即可通过复制镜像的方式，产生一个加密镜像，为系统盘进行加密。

1. 登录ECS管理控制台。
2. 在左侧导航栏，单击**实例与镜像 > 镜像**。
3. 在顶部菜单栏左上角处，选择地域。
4. 在**镜像列表**处，单击**自定义镜像**页签。
5. 选择需要复制的镜像，在**操作**列中，单击**复制镜像**。

> **说明** 如果自定义镜像大于500GiB，单击**复制镜像**时，请根据系统引导提交工单。

6. 在**复制镜像**对话框，勾选**加密**，在下拉列表中选择一个密钥。阿里云默认使用托管的服务密钥（**Default Service CMK**）进行加密，您也可以将事先在KMS服务中创建的自带密钥（BYOK）指定为该云盘的加密密钥。建议您使用自带密钥（BYOK）进行加密。

   > **说明** 首次选择更多类型密钥时，单击**同意授权**，根据页面引导为ECS授权AliyunECSDiskEncryptDefaultRole角色，允许ECS访问您的KMS资源。本步骤仅描述复制镜像时如何配置加密选项，其余配置详情，请参见复制镜像。

7. 单击**确定**。

加密数据盘

您可以在创建实例或者创建云盘时加密数据盘。

创建实例时加密数据盘

1. 登录ECS管理控制台。
2. 在左侧导航栏，单击**实例与镜像 > 实例**。
3. 在**实例列表**页面的右上角，单击**创建实例**。
4. 在**基础配置**页面的**存储**区域，按以下步骤操作。

   > **说明** 本步骤仅描述创建实例时如何配置加密选项，其余配置详情，请参见使用向导创建实例。

   i. 单击**增加一块数据盘**。
   ii. 选择数据盘的云盘类型以及容量等配置。
   iii. 勾选**加密**，在下拉列表中选择一个密钥。阿里云默认使用托管的服务密钥（**Default Service CMK**）进行加密，您也可以将事先在KMS服务中创建的自带密钥（BYOK）指定为该云盘的加密密钥。建议您使用自带密钥（BYOK）进行加密。

   > **说明** 首次选择更多类型密钥时，单击**同意授权**，根据页面引导为ECS授权AliyunECSDiskEncryptDefaultRole角色，允许ECS访问您的KMS资源。

创建云盘时加密数据盘

1. 登录ECS管理控制台。
2. 在左侧导航栏，单击**存储与快照 > 云盘**。
3. 在**云盘**页面右上角，单击**创建云盘**。
4. 配置云盘类型和容量等具体信息。

   > **说明** 本步骤仅描述创建云盘时如何配置加密选项，具体配置详情，请参见创建云盘。

5. 在**存储**区域，勾选**加密**，在下拉列表中选择一个密钥。阿里云默认使用托管的服务密钥（**Default Service CMK**）进行加密，您也可以将事先在KMS服务中创建的自带密钥（BYOK）指定为该云盘的加密密钥。建议您使用自带密钥（BYOK）进行加密。

> **说明** 首次选择更多类型密钥时，单击**同意授权**，根据页面引导为ECS授权AliyunECSDiskEncryptDefaultRole角色，允许ECS访问您的KMS资源。

5.使用KMS保护支付宝应用和小程序

在支付宝开放平台的应用体系中，应用私钥是最核心的安全要素，使用密钥管理服务KMS（Key Management Service）保护私钥，可以极大的提高支付宝应用和小程序的安全性，帮助应用开发者保障业务和资金安全。

背景信息

支付宝开放平台的应用管理体系采用公私钥的机制，以保障商户应用和支付宝交互的安全性。这一机制包括以下两部分：

- 商户应用使用自己的私钥对消息加签后，将消息和签名传递给支付宝，支付宝则使用应用的公钥验证消息的真实性（来自于合法应用的真实消息）。

  支付宝机制

- 对于支付宝返回消息给商户应用的情形，应用则使用支付宝的平台公钥来验证返回消息的真实性。

该机制的前提是：商户应用必须保障应用私钥的安全性，从而保障应用和和支付宝交互的安全性。反之，一旦私钥发生泄露，商户会面临较大的安全风险。如果应用和支付宝的交互涉及到资金类接口则风险更大。

产品价值

相比于在应用中使用应用私钥的明文来对消息进行加签，KMS存在以下优势：

- 保障私钥安全性：用户可以将签名私钥安全存放在KMS托管密码机内。用户通过KMS的OpenAPI使用私钥加签时，私钥会在密码机的硬件安全边界之内，完成运算后返回签名值，从而防止私钥的泄露。托管密码机详情，请参见托管密码机概述。
- 控制私钥使用者：用户可以通过阿里云访问控制RAM（Resource Access Management），集中管控KMS密钥使用成员，用于应用加签。
- 审计对私钥的调用日志：通过阿里云操作审计（ActionTrail）可以查看每一次调用KMS的记录；而商家应用自行保管私钥则很难产生客观的审计事件。
- 灵活响应安全事件：在应用系统遭遇恶意者攻击等情形下，可以通过多种手段阻止恶意者对私钥的非法使用。例如：用户可以通过RAM撤销对私钥的使用权限，或者通过KMS禁用私钥等。

KMS价值

使用KMS控制台为支付宝应用或小程序加签

1. 在KMS中创建密钥。
   i. 登录密钥管理服务控制台。
   ii. 在页面左上角的地域下拉列表，选择密钥所在的地域。建议您选择和支付宝应用或小程序相同的地域。
   iii. 在左侧导航栏，单击**用户主密钥**。
   iv. 单击**创建密钥**。

v. 在弹出的**创建密钥**对话框，根据以下表格进行配置。

配置项	说明
密钥类型	选择**RSA_2048**。
密钥用途	选择**Sign/Verify**。
保护级别	选择**Hsm**：通过KMS系统的硬件加密机产生和保护密钥。
描述	输入密钥描述信息。
轮转周期	默认为**不开启**。

vi. 单击**确认**。

2. 在支付宝配置密钥。支付宝开放平台提供了普通公钥方式和公钥证书方式两种密钥配置方法。公钥证书方式是对普通公钥方式的增强机制，从数字签名的角度来看，二者机制大同小异，商户应用只需要选择其中一种即可。

> **说明** 对于涉及到资金往来的商户应用，应当使用公钥证书的方式。

- **方法一：普通公钥方式**

  从KMS获取应用公钥，注册到支付宝开放平台对应的应用中。

  a. 登录密钥管理服务控制台。
  b. 在页面左上角的地域下拉列表，选择密钥所在的地域。
  c. 在左侧导航栏，单击**用户主密钥**。
  d. 找到已创建的RSA_2048类型密钥，单击别名进入详情页。
  e. 在**密钥版本**区域，单击**查看公钥**。
  f. 在弹出的**查看公钥**对话框，复制或下载公钥。
  g. 登录支付宝开放平台。
  h. 打开需要加密的应用，在左侧导航栏单击**开发设置**，并单击**开发设置**页签。
  i. 在**开发设置**页签，单击接口加密方式右侧的**设置**，在弹出的对话框输入手机验证码进行验证。
  j. 在**加密管理**对话框择选择加签模式为**公钥**，在**填写公钥字符**区域输入公钥，单击**保存设置**完成公钥的配置。

- **方法二：公钥证书方式**

  从KMS获取私钥证书请求CSR，到支付宝开放平台完成应用证书注册和签发。

  a. 登录密钥管理服务控制台。
  b. 在页面左上角的地域下拉列表，选择密钥所在的地域。
  c. 在左侧导航栏，单击**用户主密钥**。
  d. 找到已创建的RSA_2048类型密钥，单击别名进入详情页。
  e. 在**密钥版本**区域，单击**生成CSR**。

f. 在弹出的**生成CSR**对话框，根据控制台提示填写证书信息。

> **说明** **企业/单位名称**必须和支付宝开发者中心门户账号信息的公司名称保持一致，否则会导致后续步骤中上传CSR证书文件校验失败。

g. 登录支付宝开放平台。

h. 打开需要加密的应用，在左侧导航栏单击**开发设置**，并单击**开发设置**页签。

i. 在**开发设置**页签，单击接口加密方式右侧的**设置**，在弹出的对话框输入手机验证码进行验证。

j. 在**加密管理**对话框选择加签模式为**公钥证书**，在**上传证书文件**区域单击**上传CSR文件在线生成证书**。

k. 单击**上传CSR文件在线生成证书**上传公钥证书，完成公钥证书的设置。

使用支付宝SDK调用KMS为支付宝应用或小程序加签

支付宝开放平台新版SDK（EasySDK）集成了KMS作为加签提供器（Sign Provider），以简化加签操作。以Java SDK为例，您需要在支付宝应用中引用EasySDK 2.0.1以及之后的版本。

```
<dependency>
  <groupId>com.alipay.sdk</groupId>
  <artifactId>alipay-easysdk</artifactId>
  <version>2.0.1</version>
</dependency>
```

> **说明** 如果遵循支付宝开放API的签名规则，也可以不使用EasySDK，通过调用阿里云KMS的AsymmetricSign接口自行实现签名。

代码示例：普通公钥方式

```
package com.aliyun.kms.samples;

import com.alipay.easysdk.base.qrcode.models.AlipayOpenAppQrcodeCreateResponse;
import com.alipay.easysdk.factory.Factory;
import com.alipay.easysdk.kms.aliyun.AliyunKMSConfig;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;

import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Map;

/**
 * Alipay.easysdk使用KMS签名示例，本示例展示了公钥方式调用
```

```
 * Alipay-easysdk使用KMS签名示例，本示例展示了公钥方式调用。
 */
public class KmsAlipayEasySDKPublicKeyDemo {
  public static void main(String[] args) {
    Factory.setOptions(getOptions());
    try {
      AlipayOpenAppQrcodeCreateResponse response = Factory.Base.Qrcode().create("page/component/co
mponent-pages/view/view", "x=1", "二维码描述");
      if ("10000".equals(response.code)) {
        System.out.println("调用成功");
      } else {
        System.err.println("调用失败，原因：" + response.msg + "，" + response.subMsg);
      }
    } catch (Exception e) {
      System.err.println("调用遭遇异常，原因：" + e.getMessage());
      throw new RuntimeException(e.getMessage(), e);
    }
  }

  private static AliyunKMSConfig getOptions() {
    AliyunKMSConfig config = new AliyunKMSConfig();
    config.protocol = "https";
    config.gatewayHost = "openapi.alipay.com";
    config.signType = "RSA2";

    //请更换为您的AppID。
    config.appId = "202100****";
    //请修改如下的支付宝公钥字符串为自己的支付宝公钥。
    config.alipayPublicKey = "MIIBIjANB...";

    //如果使用阿里云KMS签名，则需要指定签名提供方名称，阿里云KMS的名称为"AliyunKMS"。
    config.signProvider = "AliyunKMS";

    //如果使用阿里云KMS签名，请更换为您的阿里云AccessKey ID。
    config.aliyunAccessKeyId = getAliyunAccessKey("AccessKeyId");
    //如果使用阿里云KMS签名，请更换为您的阿里云AccessKey Secret。
    config.aliyunAccessKeySecret = getAliyunAccessKey("AccessKeySecret");
    //如果使用阿里云KMS签名，请更换为您的KMS服务密钥ID。
    config.kmsKeyId = "4358f298-8e30-4849-9791-****";
    //如果使用阿里云KMS签名，请更换为您的KMS服务密钥版本ID。
    config.kmsKeyVersionId = "e71daa69-c321-4014-b0c4-****";
```

```
    //如果使用阿里云KMS签名，需要更换为您的KMS服务地址。
    //KMS服务地址列表详情，请参考：
    //https://help.aliyun.com/document_detail/69006.html
    config.kmsEndpoint = "kms.cn-hangzhou.aliyuncs.com";

    return config;
  }

  /**
   * 从文件中读取阿里云AccessKey配置信息。
   * 此处为了测试执行的环境普适性，AccessKey信息配置在resources资源下，实际过程中请不要这样配置。
   *
   * @param key AccessKey配置对应的key。
   * @return AccessKey配置字符串。
   */
  private static String getAliyunAccessKey(String key) {
    InputStream stream = KmsAlipayEasySDKPublicKeyDemo.class.getResourceAsStream("/fixture/aliyunAccessKey.json");
    Map<String, String> result = new Gson().fromJson(new InputStreamReader(stream), new TypeToken<Map<String, String>>() {
    }.getType());
    return result.get(key);
  }
}
```

代码示例：公钥证书方式

使用公钥证书方式时，EasySDK的使用方式和上述示例类似，区别主要在于配置了商户应用和支付宝平台的公钥证书。更多信息，请参见阿里云KMS Github代码样例库。

6.使用KMS一键加密Kubernetes集群Secret

阿里云容器服务Kubernetes版（简称ACK）通过您指定的KMS主密钥，对Kubernetes集群Secret进行落盘加密，您只需一键配置即可实现对Kubernetes集群的安全保护。

使用场景

Kubernetes拥有强大的运维编排管理能力，依赖大量的跨产品、跨服务、跨模块调用所必须使用的机密信息，例如：密码、证书、凭据、访问密钥等。Kubernetes集群使用Secret模型存储和管理集群系统和集群中业务应用的敏感信息，并且通过内部的etcd集群进行保存，同时在etcd集群的副本中进行分布式复制存储。

例如：部署一个没有任何业务负载的Kubernetes集群，初始情况下有大约50个Secret。其中任何一个Secret的泄露，都可能对Kubernetes集群、业务系统，甚至是企业的运行产生不可估量的损失。因此您在享受Kubernetes为您带来的便利时，也需要对Kubernetes集群中托管的大量凭据进行必要的保护，防止来自各方面的安全威胁。

加密机制

在ACK Pro托管集群中，您可以使用在KMS中创建的用户主密钥（CMK）加密Kubernetes Secret，加密过程基于Kubernetes提供的KMS Encryption Provider机制，使用信封加密的方式对存储在etcd中的Kubernetes Secret密钥进行自动加密和解密。关于信封加密的详情，请参见什么是信封加密？。Kubernetes Secret密钥加密和解密的过程如下：

- 当一个业务密钥需要通过Kubernetes Secret API存储时，数据会首先被API Server生成的一个随机的数据密钥加密，然后该数据密钥会被指定的KMS用户主密钥（CMK）加密为一个密文密钥存储在etcd中。
- 解密Kubernetes Secret密钥时，系统会首先调用KMS的解密API进行密文密钥的解密，然后使用解密后的明文密钥对Secret数据解密并最终返回给用户。

前提条件

- 您需要为使用Kubernetes的账号授予AliyunCSManagedSecurityRole角色的权限。如果您使用的账号未授权，在创建Pro集群或修改已有Pro集群过程中开启Secret落盘加密时，系统会提示您进行安全系统角色授权。
- 如果您使用RAM用户登录，请确保RAM用户具备AliyunKMSCryptoAdminAccess权限。具体操作，请参见为RAM用户授权。
- 请确保您已在KMS控制台创建用户主密钥（CMK）。具体操作，请参见创建密钥。

> **说明** 仅支持Aliyun_AES_256类型的用户主密钥（CMK）。

在新创建的ACK Pro集群中开启Secret落盘加密

1. 登录容器服务管理控制台。
2. 在左侧导航栏，单击**集群**。
3. 单击页面右上角的**创建集群**，在弹出的**选择集群模板**对话框，选择**Pro版集群**，并单击**创建**。
4. 在**ACK托管版**页签找到**Secret落盘加密**，选中**选择KMS密钥**，在下拉列表中选择KMS密钥ID。

5. 根据控制台提示完成其他参数设置。具体操作，请参见创建Kubernetes Pro版集群。

在已创建的ACK Pro集群中开启Secret落盘加密

1. 登录容器服务管理控制台。
2. 在左侧导航栏，单击**集群**。
3. 在**集群列表**页面单击目标Pro集群名称。
4. 在集群详情页面单击**基本信息**页签，在**基本信息**区域中打开**Secret落盘加密**开关。
   当集群状态由**更新中**变为**运行中**时，说明该集群Secret落盘加密的特性已变更完成。

执行结果

如果您在操作审计控制台的**详细事件查询**页面获取到使用AliyunCSManagedSecurityRole系统角色的加密和解密操作事件，则说明该集群已成功开启Secret落盘加密特性，此时您可以通过操作审计控制台查看对KMS的所有调用记录。