阿里云
密钥管理服务

最佳实践

文档版本：20190911

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
	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。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能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，恢复业务所需时间约10分钟。
	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。	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。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	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。	`swich` *`{stand | slave}`*

目录

1 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

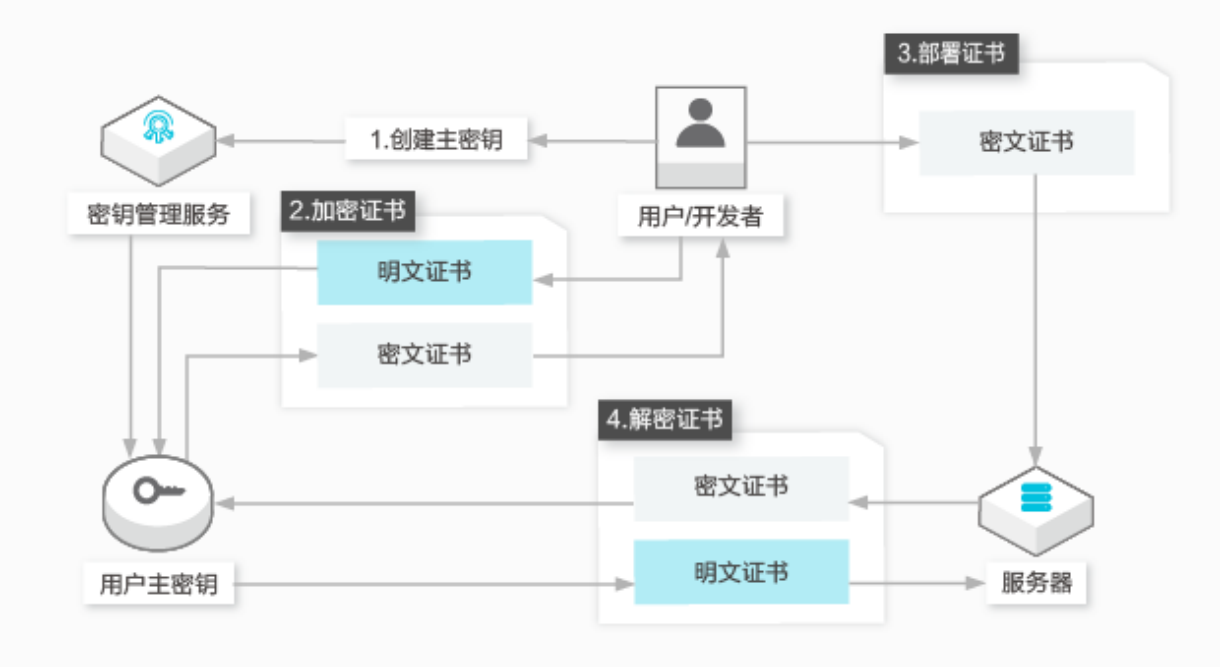
典型的使用场景包括（但不限于）：

- 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
#unique_4	创建用户主密钥（CMK）。
#unique_5	为指定用户主密钥创建一个别名。
#unique_6	指定CMK，由KMS加密数据。
#unique_7	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):
  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)
```

2 使用KMS信封加密在本地加密、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）：

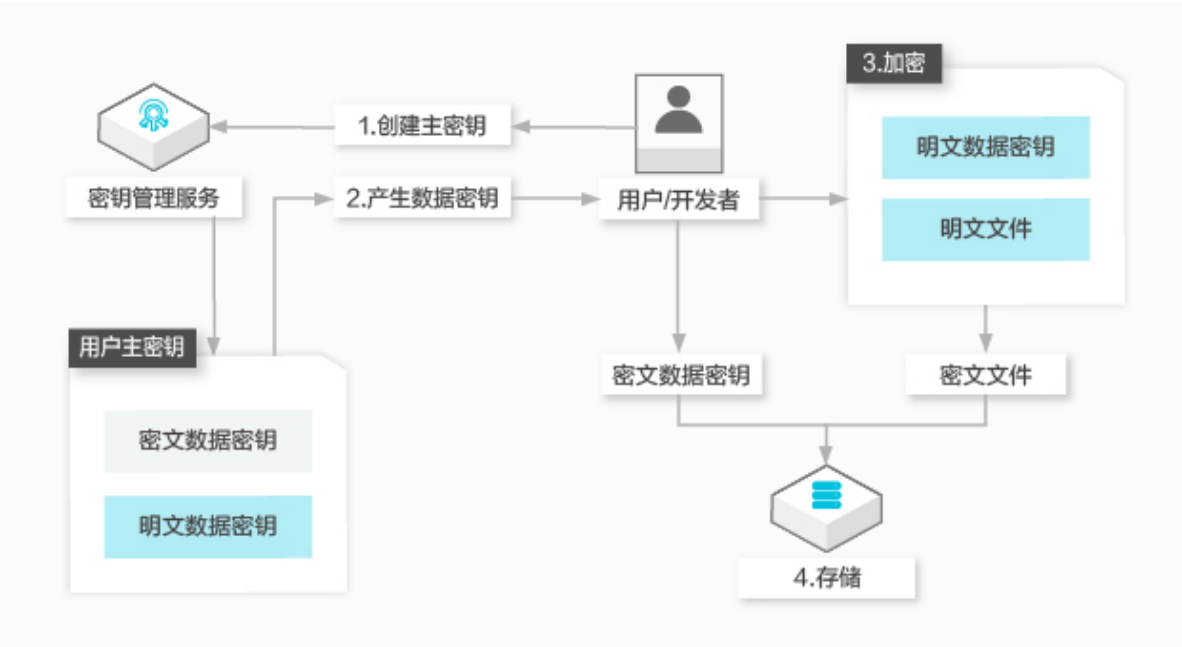
- 对业务数据文件的加密
- 对全磁盘数据加密

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。

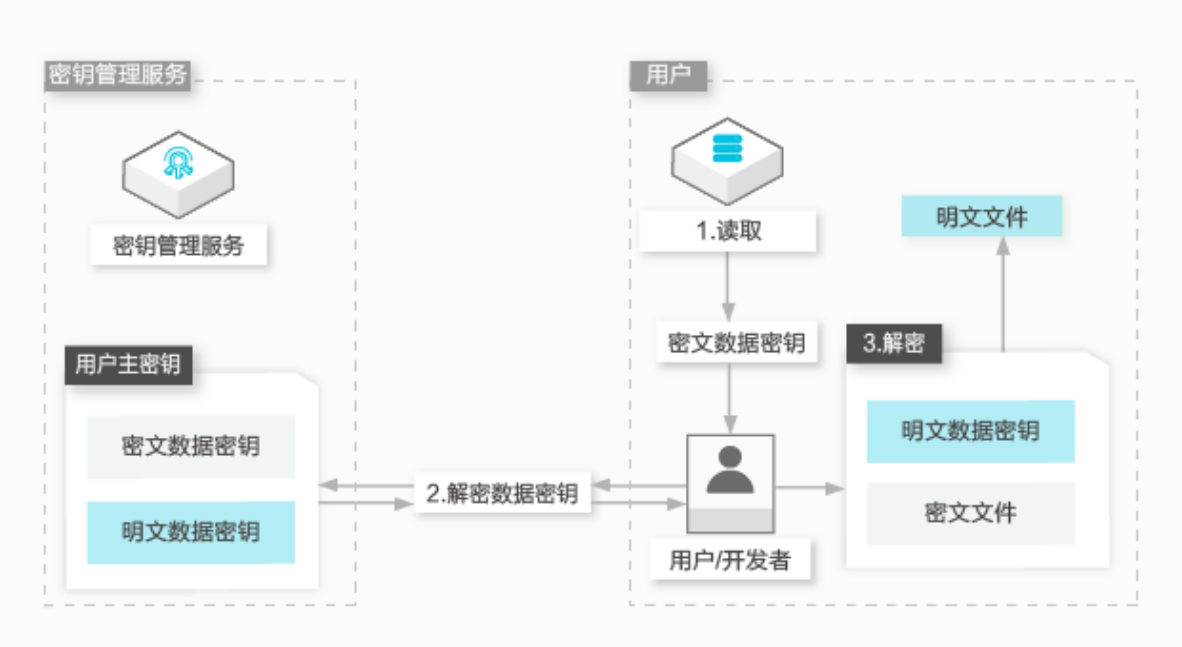
- 信封加密



操作流程如下：

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。
3. 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。
4. 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：

1. 从本地文件中读取密文数据密钥。
2. 调用KMS服务的Decrypt接口，将加密过的密钥解密为明文密钥。
3. 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
#unique_4	创建用户主密钥（CMK）。
#unique_5	为指定用户主密钥创建一个别名。
#unique_9	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。
#unique_7	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/解密证书密钥

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（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
#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_CiphertextBlob(ciphertext)
  response = json.loads(clt.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
  )
```