密钥管理系统

最佳实践

产品文档

腾讯云

文档目录

最佳实践
对称加解密
敏感信息加解密
概述

最近更新时间：2019-11-15 18:36:38

敏感信息加密是密钥管理服务 KMS 核心的能力，适用于保护小型敏感数据（小于4KB），如密钥、证书、配置文件等。使用 CMK 加密敏感数据信息，而非直接将明文存储。使用时，再将密文解密到内存，保证明文不落盘。整个交互传输过程都使用 HTTPS 请求，确保敏感数据安全。

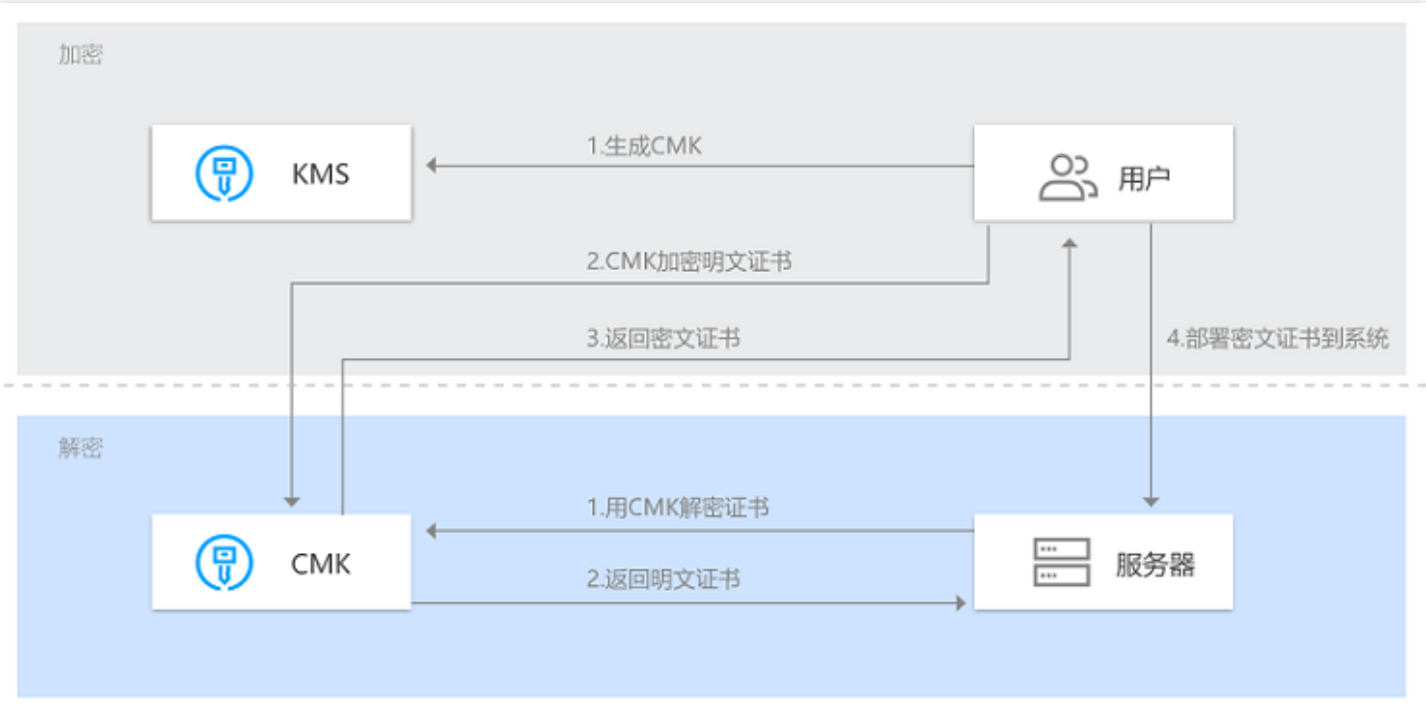
若需要通过 KMS 对海量数据进行高性能的加解密服务，请参见 信封加密 场景。

敏感信息举例

-	密钥，证书	后台配置文件
用途	加密业务数据，通信通道，数字签名	保存系统架构和其他业务信息，例如数据库 IP、密码
丢失风险	保密信息被盗、加密通道遭监听、签名被伪造	业务数据被拖库、成为攻击其他系统的跳板

示意图

本场景中，敏感数据通过主密钥 CMK 进行加解密保护，主密钥 CMK 受到经过第三方认证硬件安全模块（HSM）的保护。主密钥 CMK 在 HSM 内部实现加解密，包括腾讯云在内的任何无权限人员都无法看到 CMK 明文。



功能特点

- 权限控制：与腾讯云访问管理（CAM）集成，通过身份管理和策略管理控制账号对CMK的访问权限。
- 内置审计：与腾讯云审计集成，可记录所有 API 请求，包括密钥管理操作和密钥使用情况。保证数据操作可溯源可审计。
- 集中化密钥管理：通过腾讯云 KMS 服务实现对各类应用程序的密钥的集中管理。
- 安全合规：密钥管理服务底层使用国家密码局或 FIPS-140-2 认证的硬件安全模块（HSM）来保护密钥的安全，确保密钥的保密性、完整性和可用性。
- 敏感数据加密：支持敏感数据（小于4KB）的加密和解密操作。如密钥、证书、配置文件等。

注意事项

- 需注意 SecretId 和 SecretKey 的保密存储：
  - 腾讯云接口认证主要依靠 SecretID 和 SecretKey，SecretID 和 SecretKey 是用户的唯一认证凭证。业务系统需要该凭证调用腾讯云接口。
- 需注意 SecretID 和 SecretKey 的权限控制：
  - 建议使用子账号，根据业务需要进行接口授权的方式管控风险。
- 需注意明文数据的存储：
  - 敏感数据加密已将数据进行加密处理，为保障数据的安全性，需确保原始明文数据的删除。

操作指南

最近更新时间：2020-04-13 16:04:01

该操作指南以 Python 为例，其它编程语言类似。

前期准备

- 示例代码依赖环境：Python 2.7。
- KMS 产品服务开通：从 腾讯云控制台 开通 KMS 产品。
- 云 API 密钥服务开通：获取 SecretID、SecretKey 以及调用地址（endpoint），endpoint 一般形式为 `*.tencentcloudapi.com`，例如 KMS 的调用地址为 `kms.tencentcloudapi.com`，具体参考各产品说明。
- SDK 安装，执行以下命令，详细可参见 tencentcloud-sdk-python github 项目。

```
pip install tencentcloud-sdk-python
```

操作流程

您可以通过以下4个步骤完成敏感数据加密的操作。

1. 通过控制台 Console 或 API（CreateKey）创建一个用户主密钥 CMK，即创建用户主密钥 CMK。
2. 通过 API 调用 KMS 加密接口（Encrypt）将用户敏感数据进行加密，获取密文。
3. 根据业务需求将密文数据存储。
4. 读取数据时，通过 API 调用 KMS 解密接口（Decrypt）解密成明文。

操作步骤

步骤1：创建用户主密钥 CMK

用户主密钥的创建方式请参见 创建密钥 文档。

步骤2：敏感信息加密

前提条件：确保步骤1创建的用户主密钥为启用状态。

控制台方式

在线工具适合处理单次或者非批量的加解密操作，例如首次生成密钥密文，开发者无需为非批量的加解密操作而去开发额外的工具，将精力集中在实现核心业务能力上，详情请参见 加密解密 文档。

Python SDK 方式

通过 Encrypt 来针对用户的数据进行加密，用于加密的数据大小最多为4KB任意数据，可用于加密数据库密码，RSA Key，或其它较小的敏感信息。本文示例使用腾讯云 Python SDK 实现，您也可以使用其它支持的编程语言。

该 API 操作的 KeyId 和 Plaintext 为必选参数，详情请参见 Encrypt 接口文档来查看其它参数说明。

加密 Python SDK 示例：

以下示例代码展示了如何使用指定 CMK 对数据进行加密操作。

Python 代码示例：

```
# -*- coding: utf-8 -*-
import base64

from tencentcloud.common import credential
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentClou
dSDKException
from tencentcloud.common.profile.client_profile import ClientProfile
from tencentcloud.common.profile.http_profile import HttpProfile
from tencentcloud.kms.v20190118 import kms_client, models

def KmsInit(region="ap-guangzhou", secretId="", secretKey=""):
try:
credProfile = credential.Credential(secretId, secretKey)
client = kms_client.KmsClient(credProfile, region)
return client
except TencentCloudSDKException as err:
print(err)
return None

def Encrypt(client, keyId="", plaintext=""):
try:
req = models.EncryptRequest()
req.KeyId = keyId
req.Plaintext = base64.b64encode(plaintext)
rsp = client.Encrypt(req) # 调用加密接口
return rsp
except TencentCloudSDKException as err:
print(err)
return None

if __name__ == '__main__':
```

```
# 用户自定义参数
secretId = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
secretKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
region = "ap-guangzhou"
keyId = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
plaintext = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

client = KmsInit(region, secretId, secretKey)
rsp = Encrypt(client, keyId, plaintext)
print "plaintext=", plaintext, ", cipher=", rsp.CiphertextBlob
```

步骤3：将加密后的数据存储

根据业务的应用场景，将密文进行存储。

步骤4：敏感数据解密

控制台方式

详情请参见 加密解密 文档。

Python SDK 方式

通过 Decrypt 来针对用户的数据进行解密。

该 API 操作的 CiphertextBlob 为必选参数，详情请参见 Decrypt 接口文档查看其它参数说明。

Python 代码示例

```
# -*- coding: utf-8 -*-
import base64
from tencentcloud.common import credential
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentClou
dSDKException
from tencentcloud.common.profile.client_profile import ClientProfile
from tencentcloud.common.profile.http_profile import HttpProfile
from tencentcloud.kms.v20190118 import kms_client, models

def KmsInit(region="ap-guangzhou", secretId="", secretKey=""):
try:
credProfile = credential.Credential(secretId, secretKey)
client = kms_client.KmsClient(credProfile, region)
return client
except TencentCloudSDKException as err:
print(err)
return None

def Decrypt(client, keyId="", ciphertextBlob=""):
```

```
try:
req = models.DecryptRequest()
req.CiphertextBlob = ciphertextBlob
rsp = client.Decrypt(req) # 调用解密接口
return rsp
except TencentCloudSDKException as err:
print(err)
return None

if __name__ == '__main__':
# 用户自定义参数
secretId = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
secretKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
region = "ap-guangzhou"
keyId = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
ciphertextBlob = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

client = KmsInit(region, secretId, secretKey)
rsp = Decrypt(client, keyId, ciphertextBlob)
print "cipher=", ciphertextBlob, ", base64 decoded plaintext=", base64.b64decode
(rsp.Plaintext)
```

信封加解密
概述

最近更新时间：2019-11-28 19:01:14

信封加密（Envelope Encryption）是一种应对海量数据的高性能加解密方案。对于较大的文件或者对性能敏感的数据加密，使用 GenerateDataKey 接口生成数据加密密钥 DEK，只需要传输数据加密密钥 DEK 到 KMS 服务端（通过 CMK 进行加解密），所有的业务数据都是采用高效的本地对称加密处理，对业务的访问体验影响很小。
在实际业务场景中，对数据加密性能要求较高，数据加密量大的场景下，可通过生成 DEK 来对本地数据进行加解密，保证了业务加密性能的要求，同时也由 KMS 确保了数据密钥的随机性和安全性。

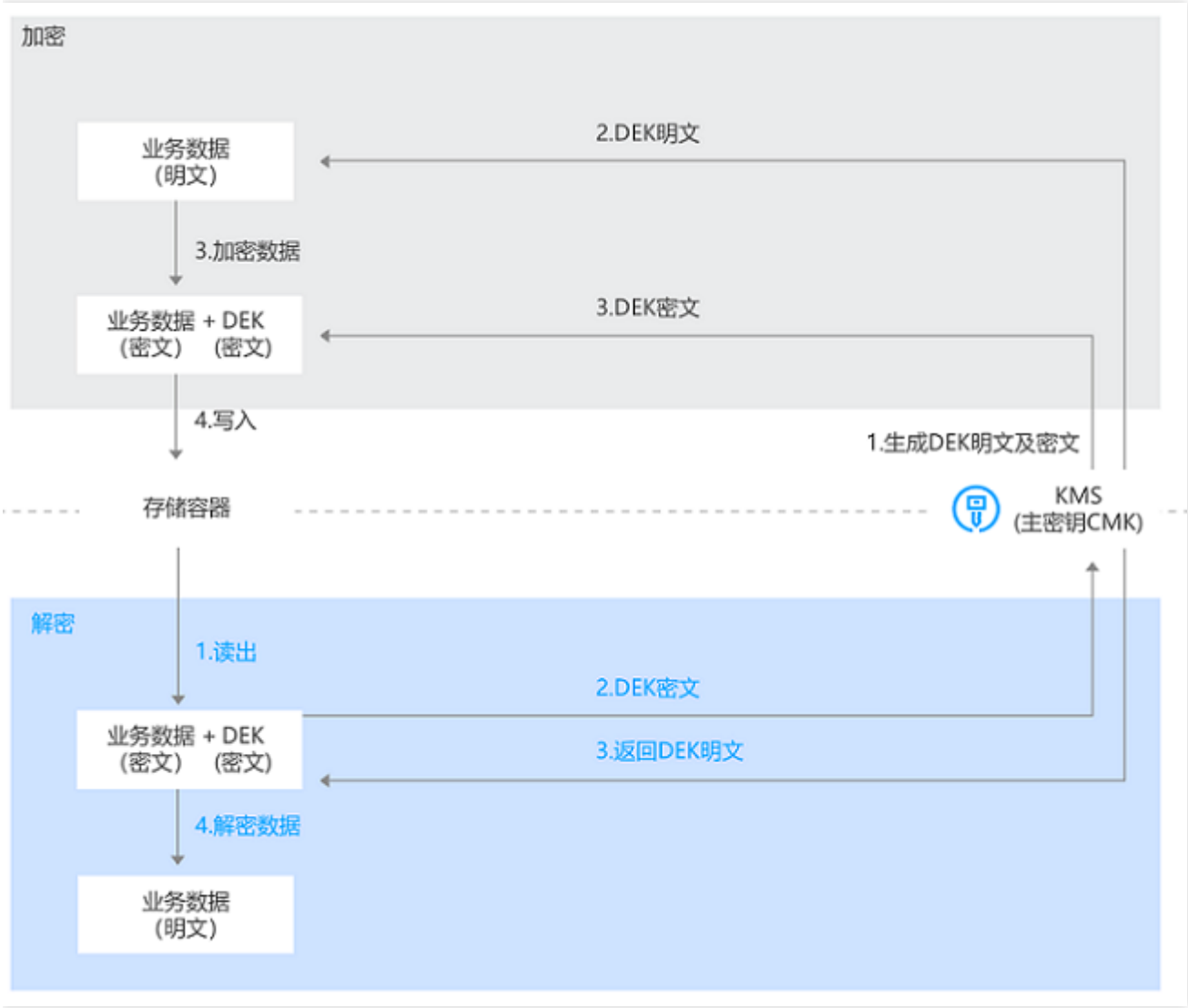
KMS 加密方案对比

对比项	敏感信息加密	信封加密
相关密钥	CMK	CMK、DEK
性能	对称加密，远程调用	少量远程对称加密，海量本地对称加密
主要场景	密钥、证书、小型数据，适用于调用频率较低的场景	海量大型数据，适用于对性能要求较高的场景

示意图

本场景中，KMS 生成的 CMK 作为重要资源，通过 CMK 生成和获取 DEK 的明文和密文。用户根据实际业务场景，首先在内存中通过 DEK 明文来对本地数据进行加密，然后将 DEK 密文和密文数据落盘，其次在业务解密场景中需

通过 KMS 来解密 DEK 密文，最后通过解密出来的 DEK 明文在内存中解密。



功能特点

- 高效：所有的业务数据都是采用高效的本地对称加密处理，对业务的访问体验影响很小。而对于 DEK 的创建和加解密开销，除了非常极端的情况下，您需要采用"一次一钥"的方案，大部分场景下可以在一段时间内复用一个 DEK 的明文和密文，所以大多数情况下这部分开销非常小。
- 安全易用：信封加密的安全性由 KMS 密钥安全提供保障。由DEK 保护业务数据，而腾讯云 KMS 则保护 DEK 并提供更好的可用性，您的主密钥主要用来生成 DEK，并且只有具备密钥访问权限的对象才能操作。

注意事项

- 需注意 SecretId 和 SecretKey 的保密存储：
  - 腾讯云接口认证主要依靠 SecretID 和 SecretKey，SecretID 和 SecretKey 是用户的唯一认证凭证。业务系统需要该凭证调用腾讯云接口。
- 需注意 SecretID 和 SecretKey 的权限控制：
  - 建议使用子账号，根据业务需要进行接口授权的方式管控风险。
- 需注意业务系统对明文密钥的处理：
  - 信封加密场景中采用的是对称加密，故明文密钥不可落盘，需在业务流程的内存中使用。
- 需注意后台系统对数据密钥的处理：
  - 信封加密场景中采用的是对称加密，可根据业务需求复用同一个数据密钥或针对不同用户、不同时间使用不同的数据密钥进行加密，避免 DEK 重复。

操作指南

最近更新时间：2020-04-13 16:04:01

本实践指南以 Python 为例，其它语言类似。

前期准备

- 示例代码依赖环境：Python 2.7。
- KMS 产品服务开通：从 腾讯云控制台 开通 KMS 产品。
- 云 API 密钥服务开通：获取 SecretID、SecretKey 以及调用地址（endpoint），KMS 的调用地址为 `kms.tencentcloudapi.com`，具体参考各产品说明。
- SDK 安装：执行以下命令，详细可见 tencentcloud-sdk-python github 开源项目。

```
pip install tencentcloud-sdk-python
```

操作流程

您可以通过以下3个步骤完成信封加密场景的操作。

1. 创建主密钥 CMK。
2. 数据信封加密，业务程序通过 API 调用 KMS GenerateDataKey 接口生成数据密钥，系统通过明文密钥将数据加密，并将密文密钥和密文落盘。
3. 数据读取解密，系统读取密文密钥和密文，通过 KMS 解密接口解密密文密钥，返回明文密钥，最后通过明文密钥解密密文数据。

实践步骤

步骤1：创建主密钥 CMK

主密钥 CMK 的创建指南，请参见 创建密钥 快速入门文档。

步骤2：数据信封加密

根据业务需求，在需要新的 DEK 时（例如针对新的用户需要进行加密，或者 DEK 复用超过一定时间，使用新的 DEK 等），可通过 KMS 接口创建新的数据密钥。生成数据密钥后在内存中使用明文密钥加密，最后将密文和密文密钥落盘。

生成数据密钥并对用户数据进行加密

通过 GenerateDataKey 来获取数据密钥 DEK，数据加密密钥是基于 CMK 生成的二级密钥，可用于用户本地数据加密解密。KMS 对 DEK 不做保存管理，需要调用方进行存储。

本文示例使用腾讯云 Python SDK 实现，您也可以使用其它支持的编程语言调用。

该 API 操作的 KeyId 为必选参数，您可以查看 GenerateDataKey 接口文档来查看其它参数说明。

Python SDK 示例

```
# -*- coding: utf-8 -*-
import base64
from Crypto.Cipher import AES
from tencentcloud.common import credential
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentClou
dSDKException
from tencentcloud.common.profile.client_profile import ClientProfile
from tencentcloud.common.profile.http_profile import HttpProfile
from tencentcloud.kms.v20190118 import kms_client, models

def KmsInit(region="ap-guangzhou", secretId="", secretKey=""):
try:
credProfile = credential.Credential(secretId, secretKey)
client = kms_client.KmsClient(credProfile, region)
return client
except TencentCloudSDKException as err:
print(err)
return None

def GenerateDatakey(client, keyId, keyspec='AES_128'):
try:
req = models.GenerateDataKeyRequest()
req.KeyId = keyId
req.KeySpec = keyspec
# 调用生成数据密钥接口
generatedatakeyResp = client.GenerateDataKey(req)
# 明文密钥需要在内存中使用，密文密钥用于持久化存储
print "DEK cipher=", generatedatakeyResp.CiphertextBlob
return generatedatakeyResp
except TencentCloudSDKException as err:
print(err)

def AddTo16(value):
while len(value) % 16 != 0:
value += '\0'
return str.encode(value)
```

```
# 用户自定义逻辑，此处仅作为参考
def LocalEncrypt(dataKey="", plaintext=""):
aes = AES.new(base64.b64decode(dataKey), AES.MODE_ECB)
encryptedData = aes.encrypt(AddTo16(plaintext))
ciphertext = base64.b64encode(encryptedData)
print "plaintext=", plaintext, ", cipher=", ciphertext

if __name__ == '__main__':
# 用户自定义参数
secretId = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
secretKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
region = "ap-guangzhou"
keyId = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
keySpec = "AES_256"
plaintext = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

client = KmsInit(region, secretId, secretKey)
rsp = GenerateDatakey(client, keyId, keySpec)

LocalEncrypt(rsp.Plaintext, plaintext)
```

步骤3：数据读取解密

先读取落盘的密文密钥，并通过调用解密接口，解密密文密钥。最后根据解密出的明文密钥解密数据。

解密（KMS Python SDK）

通过 Decrypt 来针对用户的数据进行解密。

本文示例使用腾讯云 Python SDK实现，后续您可以使用任何支持的编程语言调用。

该 API 操作的 CiphertextBlob 为必选参数，您可以查看 Decrypt 接口文档来查看其它参数说明。

Python SDK 示例

将 DEK 密文密钥通过调用 KMS 解密接口进行解密，然后用获取的 DEK 明文解密用户数据密文。

```
# -*- coding: utf-8 -*-
import base64
from Crypto.Cipher import AES
from tencentcloud.common import credential
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentClou
dSDKException
from tencentcloud.common.profile.client_profile import ClientProfile
from tencentcloud.common.profile.http_profile import HttpProfile
from tencentcloud.kms.v20190118 import kms_client, models
```

```
def KmsInit(region="ap-guangzhou", secretId="", secretKey=""):
try:
credProfile = credential.Credential(secretId, secretKey)
client = kms_client.KmsClient(credProfile, region)
return client
except TencentCloudSDKException as err:
print(err)
return None

def DecryptDataKey(client, ciphertextBlob):
try:
req = models.DecryptRequest()
req.CiphertextBlob = ciphertextBlob
rsp = client.Decrypt(req) #调用解密接口对 DEK 解密
return rsp
except TencentCloudSDKException as err:
print(err)

# 用户自定义逻辑，此处仅作为参考
def LocalDecrypt(dataKey="", ciphertext=""):
aes = AES.new(base64.b64decode(dataKey), AES.MODE_ECB)
decryptedData = aes.decrypt(base64.b64decode(ciphertext))
plaintext = str(decryptedData)
print "plaintext=", plaintext, ", cipher=", ciphertext

if __name__ == '__main__':
# 用户自定义参数
secretId = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
secretKey = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
region = "ap-guangzhou"
dekCipherBlob="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
ciphertext="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

client = KmsInit(region, secretId, secretKey)
rsp = DecryptDataKey(client, dekCipherBlob)

LocalDecrypt(rsp.Plaintext, ciphertext)
```

非对称加解密
概述

最近更新时间：2021-03-17 14:15:11

非对称加解密需要两个密钥：**公开密钥**和**私有密钥**，这两个密钥在密码学中是一对且具有双向性，即公钥和私钥中的任一个均可用作加密，但只能由另一个进行解密 。公开密钥可以交给任何人，即使对方是不可信任的，而私有密钥必须自行秘密保管。

相比对称加密，非对称加密无需考虑采用可靠的通道进行密钥分发，通常应用在信任等级不对等的系统之间 ，实现敏感数据加密传递或数字签名验签。

非对称密钥的类型

当前腾讯云 KMS 支持如下三种非对称密钥算法类型：

RSA

目前 KMS 支持模长为2048比特的 RSA 密钥，KeyUsage = ASYMMETRIC_DECRYPT_RSA_2048。

SM2

SM2 是国密标准的公钥密钥算法，在中国的商用密码体系中被用来替换 RSA 算法。对于有相应的国密要求的应用，可以考虑使用此类型的密钥，KeyUsage = ASYMMETRIC_DECRYPT_SM2。

ECC

ECC 是基于椭圆曲线的加密算法，KeyUsage = ASYMMETRIC_SIGN_VERIFY_ECC。

非对称加密的典型场景

非对称加解密在实际应用中包含**加密通信**和**数字签名**两种典型场景：

加密通信

加密通信是非对称加密算法的一种典型应用。加密通信的过程类似于对称加密，区别在于需要使用公钥进行加密，而且需要使用私钥进行解密。

加密通信场景的原理说明如下：

1. 信息接收者创建公钥私钥对，并将公钥给到一个或多个信息发送者。
2. 信息发送者使用公钥对敏感信息进行加密，并将加密后的密文通过传输介质发送给信息接收者。

3. 信息接收者从传输介质上获取到数据后，用自己持有的私钥对信息进行解密，还原出原文信息。

由于密文只有通过私钥才可以解密，而私钥是不公开的，所以即使由于传输介质的安全性比较低而导致信息泄露，拿到密文的人也无法将其破译，从而保证了敏感信息的安全。

腾讯云 KMS 提供加密通信的解决方案，具体操作详情请参见 非对称数据加解密。

数字签名

数字签名技术是非对称加密算法的另一种典型应用。数字签名分为签名和验证两个过程，在签名时使用私钥，验证时则使用公钥，这一实现过程正好与加密通信相反。

数字签名场景的原理说明如下：

1. 信息发送者创建公钥私钥对，并将公钥给到一个或多个信息接收者。
2. 信息发送者使用哈希函数从消息原文中生成消息摘要，然后使用私有密钥对这个摘要进行加密，即得到消息原文对应的数字签名。
3. 信息发送者将消息原文和数字签名一并传送给信息接收者。
4. 信息接收者得到消息原文和数字签名后，用同一个哈希函数从消息原文中生成摘要 A，另外，用发送者提供的公钥对数字签名进行解密，得到摘要 B，对比 A 和 B 是否相同，验证原文是否被篡改。

由于签名是使用私钥加密产生，而私钥不公开，这使得签名具有唯一的特征。所以数字签名既可以保证数据在传输过程中不会被篡改，又可以保证信息发送者的身份正确性，防止交易中的抵赖发生。

腾讯云 KMS 提供数字签名解决方案，具体操作详情请参见 非对称签名验签。

> ⚠ **注意：**
>
> 由于公私钥使用场景的特殊性，KMS 不支持对非对称的 CMK 进行自动轮转。如果您需要定期或不定期更新所使用的密钥，可以自行创建新的非对称密钥。

非对称数据加解密

最近更新时间：2021-02-08 15:44:23

操作流程

当您需要进行传递敏感信息时（例如密钥的交换），需要对敏感数据进行加密，使用非对称密钥加解密的方案从信息接收者的角度来说，您需要进行以下操作：

1. 在密钥管理系统 KMS 中创建非对称加密密钥，详情请参见 创建主密钥 。
2. 在密钥管理系统 KMS 中获取公钥，详情请参见 获取非对称密钥的公钥。
3. 信息接收者将公钥分发给信息发送者。
4. 信息发送者用得到的公钥对敏感数据进行本地加密后，将密文发送给信息接收者。
5. 信息接收者拿到密文后，调用 KMS 的解密功能对密文解密。API 详情请参见 非对称密钥 Sm2 解密 和 非对称密钥 RSA 解密 ，TCCLI 方式请参见 非对称密钥解密 。

在整个敏感数据的传输的过程中，使用密文进行传输，而唯一能解密该密文的密钥托管在密钥管理系统 KMS 中保护，包括腾讯云在内的任何人都无法获取到您的密钥，极大程度上提高了敏感数据加密传输安全性。

操作步骤

RSA 示例

1. 创建非对称加密密钥

   请求：

```
tccli kms CreateKey --Alias test --KeyUsage ASYMMETRIC_DECRYPT_RSA_2048
```

   返回结果：

```
{
"Response": {
"KeyId": "22d79428-61d9-11ea-a3c8-525400******",
"Alias": "test",
"CreateTime": 1583739580,
"Description": "",
"KeyState": "Enabled",
"KeyUsage": "ASYMMETRIC_DECRYPT_RSA_2048",
"RequestId": "0e3c62db-a408-406a-af27-dd5ced******"
}
}
```

2. 下载公钥

请求：

```
tccli kms GetPublicKey --KeyId 22d79428-61d9-11ea-a3c8-525400******
```

返回结果：

```
{
"Response": {
"RequestId": "408fa858-cd6d-4011-b8a0-653805******",
"KeyId": "22d79428-61d9-11ea-a3c8-525400******",
"PublicKey": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAzQk7x7ladgVFEEGYDbeUc
5aO9TfiDplIO4WovBOVpIFoDS31n46YiCGiqj67qmYslZ2KMGCd3Nt+a+jdzwFiTx3O87wdKWcF2vHL
9Ja+95VuCmKYeK1uhPyqqj4t9Ch/cyvxb0xaLBzztTQ9dXCxDhwj08b24T+/FYB9a4icuqQypCvjY1X
9j8ivAsPEdHZoc9Di7JXBTZdVeZC1igCVgl6mwzdHTJCRydE2976zyjC7l6QsRT6pRsMF3696N07Wna
KgGv3K/Zr/6RbxebLqtmNypNERIR7jTCt9L+fgYOX7anmuF5v7z0GfFsen9Tqb1LsZuQR0vgqCauOj*
***********",
"PublicKeyPem": "-----BEGIN PUBLIC KEY-----\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMI
IBCgKCAQEAzQk7x7ladgVFEEGYDbeU\nc5aO9TfiDplIO4WovBOVpIFoDS31n46YiCGiqj67qmYslZ2
KMGCd3Nt+a+jdzwFi\nTx3O87wdKWcF2vHL9Ja+95VuCmKYeK1uhPyqqj4t9Ch/cyvxb0xaLBzztTQ9
dXCx\nDhwj08b24T+/FYB9a4icuqQypCvjY1X9j8ivAsPEdHZoc9Di7JXBTZdVeZC1igCV\ngl6mwzd
HTJCRydE2976zyjC7l6QsRT6pRsMF3696N07WnaKgGv3K/Zr/6RbxebLq\ntmNypNERIR7jTCt9L+fg
YOX7anmuF5v7z0GfFsen9Tqb1LsZuQR0***********\n1QIDAQAB\n-----END PUBLIC KEY----
-\n"
}
}
```

3. 使用公钥加密

i. 将公钥 PublicKey 存入文件 public_key.base64，并进行 base64 解码。

存入文件：

```
echo "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAzQk7x7ladgVFEEGYDbeUc5aO9Tf
iDplIO4WovBOVpIFoDS31n46YiCGiqj67qmYslZ2KMGCd3Nt+a+jdzwFiTx3O87wdKWcF2vHL9Ja+
95VuCmKYeK1uhPyqqj4t9Ch/cyvxb0xaLBzztTQ9dXCxDhwj08b24T+/FYB9a4icuqQypCvjY1X9j
8ivAsPEdHZoc9Di7JXBTZdVeZC1igCVgl6mwzdHTJCRydE2976zyjC7l6QsRT6pRsMF3696N07Wna
KgGv3K/Zr/6RbxebLqtmNypNERIR7jTCt9L+fgYOX7anmuF5v7z0GfFsen9Tqb1LsZuQR0vgqCauO
j***********" > public_key.base64
```

base64 解码获取公钥实际内容：

```
openssl enc -d -base64 -A -in public_key.base64 -out public_key.bin
```

ii. 创建测试明文文件：

```
echo "test" > test_rsa.txt
```

iii. 使用 OPENSSL 进行公钥加密 test_rsa.txt 文件内容。

```
openssl pkeyutl -in test_rsa.txt -out encrypted.bin -inkey public_key.bin -keyform DER -pubin -encrypt -pkeyopt rsa_padding_mode:oaep -pkeyopt rsa_oaep_md:sha256
```

iv. 将公钥加密后的数据进行 base64 编码，方便传输。

```
openssl enc -e -base64 -A -in encrypted.bin -out encrypted.base64
```

4. 通过 KMS 使用私钥解密

将上述 encrypted.base64 base64 编码之后的密文作为 AsymmetricRsaDecrypt 的 Ciphertext 参数，进行私钥的解密。

请求：

```
tccli kms AsymmetricRsaDecrypt --KeyId 22d79428-61d9-11ea-a3c8-525400****** --Algorithm RSAES_OAEP_SHA_256 --Ciphertext "DEb/JBmuhVkYS34r0pR7Gv1WTc4khkxqf7S1WIr7/GXsAs/tfP/v/2+1SwsIG7BqW7kUZqr38/FGkaIEqYeewot37t3+Jx0t5w7/yXkUnyUfyfPpXlHXf94g3wFOjijEWWsjWWzaXTkTr8uWOfRBenq+bcaY783FIy03XjJW/Y0wKWjD3tULvKndCJO/3bkb65kn1Fbsfm20xrUUwqV/p2DVLXBdG1ymr0DjsbG7R0tb3ytc2LmH33YPAQE32eP27ciKzSml+w2tdUM3dw3nEZcTGMs1wFDGk0O1WB052jZ7TitUD9zCftFv2dKlZD3LRx1+vHqpNVgPhLmL******=="
```

返回结果：

```
{
"Response": {
"RequestId": "6758cbf5-5e21-4c37-a2cf-8d47f5******",
"KeyId": "22d79428-61d9-11ea-a3c8-525400******",
"Plaintext": "dGVzdAo="
}
}
```

说明：

使用 SM2 非对称密钥加解密流程类似，私钥解密接口详情请参见 非对称密钥Sm2解密。

非对称签名验签

概述

最近更新时间：2021-09-08 17:11:28

在传递敏感信息的场景中，信息发送者可以通过非对称签名验签的方式提供身份证明，具体操作流程如下：

1. 在密钥管理系统 KMS 中创建非对称密钥对，详情请参见 创建主密钥 。
2. 信息发送者使用创建的用户私钥对需要传输的数据生成签名，详情请参见 签名 。
3. 信息发送者将签名和数据传递给信息接收者。
4. 信息接收者拿到签名和数据之后，进行签名验证，有以下两种方式进行校验。
   i. 调用 KMS 的验证签名功能接口进行校验，详情请参见 验证签名 。
   ii. 下载 KMS 非对称密钥公钥，在本地通过 gmssl、openssl、密码库、KMS 的国密 Encryption SDK 等验签方法进行验证。

> 说明：
> 非对称签名验签目前支持 SM2/RSA/ECC 签名验签算法。

SM2 签名验签

最近更新时间：2021-03-17 14:15:10

本文将为您介绍如何使用 SM2 签名验签算法。

操作步骤

步骤1：创建非对称签名密钥

> **⚠ 注意：**
>
> 在密钥管理系统（KMS）中调用 创建主密钥 接口创建用户主密钥时，在 KMS 中创建密钥的时候，必须传入正确的密钥用途 KeyUsage= ASYMMETRIC_SIGN_VERIFY_SM2，才可以使用签名的功能。

请求：

```
tccli kms CreateKey --Alias test --KeyUsage ASYMMETRIC_SIGN_VERIFY_SM2
```

返回结果：

```
{
"Response": {
"KeyId": "22d79428-61d9-11ea-a3c8-525400******",
"Alias": "test",
"CreateTime": 1583739580,
"Description": "",
"KeyState": "Enabled",
"KeyUsage": "ASYMMETRIC_SIGN_VERIFY_SM2",
"TagCode": 0,
"TagMsg": "",
"RequestId": "0e3c62db-a408-406a-af27-dd5ced******"
}
}
```

步骤2：下载公钥

请求：

```
tccli kms GetPublicKey --KeyId 22d79428-61d9-11ea-a3c8-525400******
```

返回结果：

```
{
"Response": {
"RequestId": "408fa858-cd6d-4011-b8a0-653805******",
"KeyId": "22d79428-61d9-11ea-a3c8-525400******",
"PublicKey": "MFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQgAEFLlge0vtct949CwtadHODzisgXJahuj
q+PvM**************bBs/f3axWbvgvHx8Jmqw==",
"PublicKeyPem": "-----BEGIN PUBLIC KEY-----\nMFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQgAE
FLlge0vtct949CwtadHODzisgXJa\nhujq+PvM**************bBs/f3axWbvgvHx8Jmqw==\n----
-END PUBLIC KEY-----\n"
}
}
```

将公钥 PublicKeyPem 转成 pem 格式，并存入 public_key.pem 文件：

```
echo "-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQgAEFLlge0vtct949CwtadHODzisgXJa
hujq+PvM**************bBs/f3axWbvgvHx8Jmqw==
-----END PUBLIC KEY-----" > public_key.pem
```

说明：

另外，您可以登录 KMS 控制台，单击【用户密钥】>【密钥ID/密钥名称】进入密钥信息页面，直接下载非对称密钥公钥。

步骤3：创建信息的明文文件

创建测试明文文件：

```
echo "test" > test_verify.txt
```

注意：

当生成的文件内容中，存在不可见的字符情况下（如换行符等），需对文件进行truncate操作（**truncate -s -1 test_verify.txt**），从而保证签名准确。

步骤4：步骤计算消息摘要

- 如果待签名的消息的长度不超过4096字节，可以跳过本步骤，直接进入 步骤 5。
- 如果待签名的消息的长度超过4096字节，则需先在用户端本地计算消息摘要。
  使用 gmssl 对 test_verity.txt 文件内容进行摘要计算：

```
gmssl sm2utl -dgst -in ./test_verify.txt -pubin -inkey ./public_key.pem -id 123
4567812345678 > digest.bin
```

步骤5：通过 KMS 签名接口生成签名

调用 KMS 的 签名 API 接口计算信息的签名。

1. 消息原文或消息摘要进行计算签名之前，需先进行 base64 编码。

```
//消息摘要进行base64编码
gmssl enc -e -base64 -A -in digest.bin -out encoded.base64
//消息原文进行base64编码
gmssl enc -e -base64 -A -in test_verify.txt -out encoded.base64
```

2. 进行签名的计算。

   请求：

```
// 将上述 encoded.base64 的文件内容作为 SignByAsymmetricKey 的 Message 参数，以消息摘
要的形式进行签名
tccli kms SignByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400****** --Al
gorithm SM2DSA --Message "qJQj83hSyOuU7Tn0SRReGCk4yuuVWaeZ44BP******==" --Messa
geType DIGEST

// 以消息原文的形式进行签名（原文要进行Base64编码）
tccli kms SignByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400****** --Al
gorithm SM2DSA --Message "dG***Ao=" --MessageType RAW
```

   返回结果：

```
{
"Response": {
"Signature": "U7Tn0SRReGCk4yuuVWaeZ4******",
"RequestId": "408fa858-cd6d-4011-b8a0-653805******"
}
}
```

   将签名内容 Signature 存入 signContent.sign 文件：

```
echo "U7Tn0SRReGCk4yuuVWaeZ4******" | base64 -d > signContent.bin
```

步骤6：验证签名

- 通过 KMS 验证签名接口校验(**推荐使用该方法进行验签**)

  请求：

```
// 对消息原文进行验证（原文要进行Base64编码）
tccli kms VerifyByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400****** --SignatureValue "U7Tn0SRReGCk4yuuVWaeZ4******" --Message "dG***Ao=" --Algorithm SM2DSA --MessageType RAW
// 对消息摘要进行验证(将步骤4 encoded.base64 文件内容作为 VerifyByAsymmetricKey 的 Message 参数，以消息摘要的形式进行验签)
tccli kms VerifyByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400****** --SignatureValue "U7Tn0SRReGCk4yuuVWaeZ4******" --Message "QUuAcNFr1Jl5+3GDbCxU7te7Uekq+oTxZ**********=" --Algorithm SM2DSA --MessageType DIGEST
```

> **说明：**
>
> 签名接口和验签接口中使用的参数 Message 和 MessageType 的取值要保持一致。

  返回结果：

```
{
"Response": {
"SignatureValid": true,
"RequestId": "6758cbf5-5e21-4c37-a2cf-8d47f5******"
}
}
```

- 通过 KMS 公钥和签名内容在本地进行验证

  请求：

```
gmssl sm2utl -verify -in ./test_verify.txt -sigfile ./signContent.bin -pubin -inkey ./public_key.pem -id 1234567812345678
```

  返回结果：

```
Signature Verification Successful
```

RSA 签名验签

最近更新时间：2021-03-17 14:22:05

本文将为您介绍如何使用 RSA 签名验签算法。

操作步骤

步骤1：创建非对称签名密钥

> **⚠ 注意：**
>
> 在 KMS 中调用 创建主密钥 接口创建用户主密钥时，必须传入正确的密钥用途 KeyUsage=ASYMMETRIC_SIGN_VERIFY_RSA_2048，这样才可以使用签名功能。

- **请求**：

```
tccli kms CreateKey --Alias test_rsa --KeyUsage ASYMMETRIC_SIGN_VERIFY_RSA_2048
```

- **返回结果**：

```
{
"Response": {
"KeyId": "22d79428-61d9-11ea-a3c8-525400******",
"Alias": "test_rsa",
"CreateTime": 1583739580,
"Description": "",
"KeyState": "Enabled",
"KeyUsage": "ASYMMETRIC_SIGN_VERIFY_RSA_2048",
"TagCode": 0,
"TagMsg": "",
"RequestId": "0e3c62db-a408-406a-af27-dd5ced******"
}
}
```

步骤2：下载公钥

- **请求**：

```
tccli kms GetPublicKey --KeyId 22d79428-61d9-11ea-a3c8-525400******
```

- **返回结果**：

```
{
"Response": {
"RequestId": "408fa858-cd6d-4011-b8a0-653805******",
"KeyId": "22d79428-61d9-11ea-a3c8-525400******",
"PublicKey": "MFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQgAEFLlge0vtct949CwtadHODzisgXJah
ujq+PvM**************bBs/f3axWbvgvHx8Jmqw==",
"PublicKeyPem": "-----BEGIN PUBLIC KEY-----\nMFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQg
AEFLlge0vtct949CwtadHODzisgXJa\nhujq+PvM**************bBs/f3axWbvgvHx8Jmqw==\n
-----END PUBLIC KEY-----\n"
}
}
```

- **将公钥 PublicKeyPem 转成 pem 格式，并存入文件 public_key.pem。**

```
echo "-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQgAEFLlge0vtct949CwtadHODzisgXJa
hujq+PvM**************bBs/f3axWbvgvHx8Jmqw==
-----END PUBLIC KEY-----" > public_key.pem
```

⚠ 注意：

您可以登录 KMS 控制台，单击【用户密钥】>【密钥 ID/密钥名称】进入密钥信息页面，直接下载非对称密钥公钥。

步骤3：创建信息的明文文件

创建测试明文文件。

```
echo "test" > test_verify.txt
```

⚠ 注意：

当生成的文件内容中，存在不可见的字符情况下（如换行符等），需对文件进行 truncate 操作（如 truncate -s -1 test_verify.txt），从而保证签名准确。

步骤4：计算消息摘要

⚠ 注意：

- 如果待签名的消息长度不超过4096字节，可以跳过本步骤，直接进入 步骤5。
- 如果待签名的消息的长度超过4096字节，则需先在用户端本地计算消息摘要。

使用 openssl 对 test_verity.txt 文件内容进行摘要计算。

```
openssl dgst -sha256 -binary -out digest.bin test_verify.txt
```

步骤5：通过 KMS 签名接口生成签名

调用 KMS 的 签名 接口计算信息的签名。

1. 消息原文或消息摘要进行计算签名之前，需先进行 base64 编码。

```
//消息摘要进行base64编码
openssl enc -e -base64 -A -in digest.bin -out encoded.base64
//消息原文进行base64编码
openssl enc -e -base64 -A -in test_verify.txt -out encoded.base64
```

2. 进行签名的计算。
   - **请求：**
     - **RSA_PSS_SHA_256**

```
// 将上述 encoded.base64 的文件内容作为 SignByAsymmetricKey 的 Message 参数，以消息摘要的形式进行签名。
tccli kms SignByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400****** --Algorithm RSA_PSS_SHA_256 --Message "qJQj83hSyOuU7Tn0SRReGCk4yuuVWaeZ44BP******==" --MessageType DIGEST
// 以消息原文的形式进行签名（原文要进行Base64编码）
tccli kms SignByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400****** --Algorithm RSA_PSS_SHA_256 --Message "dG***Ao=" --MessageType RAW
```

     - **RSA_PKCS1_SHA_256**

```
// 将上述 encoded.base64 的文件内容作为 SignByAsymmetricKey 的 Message 参数，以消息摘要的形式进行签名。
tccli kms SignByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400****** --Algorithm RSA_PKCS1_SHA_256 --Message "qJQj83hSyOuU7Tn0SRReGCk4yuuVWaeZ44BP******==" --MessageType DIGEST
// 以消息原文的形式进行签名（原文要进行Base64编码）
tccli kms SignByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400****** --Algorithm RSA_PKCS1_SHA_256 --Message "dG***Ao=" --MessageType RAW
```

   - **返回结果：**

```
{
"Response": {
```

```
"Signature": "U7Tn0SRReGCk4yuuVWaeZ4******",
"RequestId": "408fa858-cd6d-4011-b8a0-653805******"
}
}
```

- **将签名内容 Signature 存入文件 signContent.sign**：

```
echo "U7Tn0SRReGCk4yuuVWaeZ4******" | base64 -d > signContent.bin
```

步骤6：验证签名

1. 通过 KMS 验证签名接口校验。(建议使用该方法进行验证签名)
   - **请求**：
     - **RSA_PSS_SHA_256**

```
// 对消息摘要进行验证(将步骤4 encoded.base64 文件内容作为 VerifyByAsymmetricKey 的 Message 参数，以消息摘要的形式进行验签)。
tccli kms VerifyByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400******** --SignatureValue "U7Tn0SRReGCk4yuuVWaeZ4******" --Message "QUuAcNFr1Jl5+3GDbCxU7te7Uekq+oTxZ**********=" --Algorithm RSA_PSS_SHA_256 --MessageType DIGEST
// 对消息原文进行验证（原文要进行Base64编码）。
tccli kms VerifyByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400******** --SignatureValue "U7Tn0SRReGCk4yuuVWaeZ4******" --Message "dG***Ao=" --Algorithm RSA_PSS_SHA_256 --MessageType RAW
```

     - **RSA_PKCS1_SHA_256**

```
// 对消息摘要进行验证(将步骤4 encoded.base64 文件内容作为 VerifyByAsymmetricKey 的 Message 参数，以消息摘要的形式进行验签)。
tccli kms VerifyByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400******** --SignatureValue "U7Tn0SRReGCk4yuuVWaeZ4******" --Message "QUuAcNFr1Jl5+3GDbCxU7te7Uekq+oTxZ**********=" --Algorithm RSA_PKCS1_SHA_256 --MessageType DIGEST
// 对消息原文进行验证（原文要进行 Base64 编码）。
tccli kms VerifyByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400******** --SignatureValue "U7Tn0SRReGCk4yuuVWaeZ4******" --Message "dG***Ao=" --Algorithm RSA_PKCS1_SHA_256 --MessageType RAW
```

   - **返回结果**：

```
{
"Response": {
"SignatureValid": true,
"RequestId": "6758cbf5-5e21-4c37-a2cf-8d47f5******"
```

```
}
}
```

说明：

签名接口和验签接口中使用的参数 Message 和 MessageType 的取值要保持一致。

2. 通过 KMS 公钥和签名内容在本地进行验证。
   - **请求**：

```
//采用 RSA_PSS_SHA_256 算法进行签名的验签。
openssl dgst -verify public_key.pem -sha256 -sigopt rsa_padding_mode:pss -sig
opt rsa_pss_saltlen:-1 -signature ./signContent.bin ./test_verify.txt
//采用 RSA_PKCS1_SHA_256 算法进行签名的验签。
openssl dgst -verify public_key.pem -sha256 -signature ./signContent.bin ./te
st_verify.txt
```

   - **返回结果**：

```
Verified OK
```

ECC 签名验签

最近更新时间：2021-03-17 14:26:10

本文将为您介绍如何使用 ECC 签名验签算法。

操作步骤

步骤1：创建非对称签名密钥

> **⚠ 注意：**
>
> 在密钥管理系统（KMS）中调用 创建主密钥 接口创建用户主密钥时，必须传入正确的密钥用途 ASYMMETRIC_SIGN_VERIFY_ECC，这样才可以使用签名功能。

- **请求**：

```
tccli kms CreateKey --Alias test_ecc --KeyUsage ASYMMETRIC_SIGN_VERIFY_ECC
```

- **返回结果**：

```
{
"Response": {
"KeyId": "22d79428-61d9-11ea-a3c8-525400******",
"Alias": "test_ecc",
"CreateTime": 1583739580,
"Description": "",
"KeyState": "Enabled",
"KeyUsage": "ASYMMETRIC_SIGN_VERIFY_ECC",
"TagCode": 0,
"TagMsg": "",
"RequestId": "0e3c62db-a408-406a-af27-dd5ced******"
}
}
```

步骤2：下载公钥

- **请求**：

```
tccli kms GetPublicKey --KeyId 22d79428-61d9-11ea-a3c8-525400******
```

- **返回结果**：

```
{
"Response": {
"RequestId": "408fa858-cd6d-4011-b8a0-653805******",
"KeyId": "22d79428-61d9-11ea-a3c8-525400******",
"PublicKey": "MFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQgAEFLlge0vtct949CwtadHODzisgXJah
ujq+PvM**************bBs/f3axWbvgvHx8Jmqw==",
"PublicKeyPem": "-----BEGIN PUBLIC KEY-----\nMFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQg
AEFLlge0vtct949CwtadHODzisgXJa\nhujq+PvM**************bBs/f3axWbvgvHx8Jmqw==\n
-----END PUBLIC KEY-----\n"
}
}
```

- **将公钥 PublicKeyPem 转成pem格式，并存入文件 public_key.pem**：

```
echo "-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQgAEFLlge0vtct949CwtadHODzisgXJa
hujq+PvM**************bBs/f3axWbvgvHx8Jmqw==
-----END PUBLIC KEY-----" > public_key.pem
```

> ⚠ **注意：**
>
> 您也可以登录 KMS 控制台，单击【用户密钥】>【密钥 ID/密钥名称】进入密钥信息页面，直接下载非对称密钥公钥。

步骤3：创建信息的明文文件

创建测试明文文件。

```
echo "test" > test_verify.txt
```

> ⚠ **注意：**
>
> 当生成的文件内容中，存在不可见的字符情况下（如换行符等），需对文件进行 truncate 操作（如 truncate -s -1 test_verify.txt），从而保证签名准确。

步骤4：计算消息摘要

> ⚠ **注意：**

- 如果待签名的消息的长度不超过4096字节，可以跳过本步骤，直接进入 步骤5。
- 如果待签名的消息的长度超过4096字节，则需先在用户端本地计算消息摘要。

使用 openssl 对 test_verity.txt 文件内容进行摘要计算。

```
openssl dgst -sha256 -binary -out digest.bin test_verify.txt
```

步骤5：通过 KMS 签名接口生成签名

调用 KMS 的 签名 接口计算信息的签名。

1. 消息原文或消息摘要进行计算签名之前，需先进行 base64 编码。

```
//消息摘要进行base64编码。
openssl enc -e -base64 -A -in digest.bin -out encoded.base64
//消息原文进行base64编码。
openssl enc -e -base64 -A -in test_verify.txt -out encoded.base64
```

2. 进行签名的计算。
   - **请求**：

```
// 将上述 encoded.base64 的文件内容作为 SignByAsymmetricKey 的 Message 参数，以消息摘要的形式进行签名。
tccli kms SignByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400****** --Algorithm ECC_P256_R1 --Message "qJQj83hSyOuU7Tn0SRReGCk4yuuVWaeZ44BP******==" --MessageType DIGEST
// 以消息原文的形式进行签名（原文要进行 Base64 编码）。
tccli kms SignByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400****** --Algorithm ECC_P256_R1 --Message "dG***Ao=" --MessageType RAW
```

   - **返回结果**：

```
{
"Response": {
"Signature": "U7Tn0SRReGCk4yuuVWaeZ4******",
"RequestId": "408fa858-cd6d-4011-b8a0-653805******"
}
}
```

   - **将签名内容 Signature 存入文件 signContent.sign**：

```
echo "U7Tn0SRReGCk4yuuVWaeZ4******" | base64 -d > signContent.bin
```

步骤6：验证签名

1. 通过 KMS 验证签名接口校验。(建议使用该方法进行验证签名)
   - **请求**：

```
// 对消息摘要进行验证(将步骤4 encoded.base64 文件内容作为 VerifyByAsymmetricKey 的 Message 参数，以消息摘要的形式进行验签)。
tccli kms VerifyByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400****** --SignatureValue "U7Tn0SRReGCk4yuuVWaeZ4******" --Message "QUuAcNFr1Jl5+3GDbCxU7te7Uekq+oTxZ**********=" --Algorithm ECC_P256_R1 --MessageType DIGEST
// 对消息原文进行验证（原文要进行Base64编码）。
tccli kms VerifyByAsymmetricKey --KeyId 22d79428-61d9-11ea-a3c8-525400****** --SignatureValue "U7Tn0SRReGCk4yuuVWaeZ4******" --Message "dG***Ao=" --Algorithm ECC_P256_R1 --MessageType RAW
```

   - **返回结果**：

```
{
"Response": {
"SignatureValid": true,
"RequestId": "6758cbf5-5e21-4c37-a2cf-8d47f5******"
}
}
```

> ⚠ **注意：**
>
> 签名接口和验签接口中使用的参数 Message 和 MessageType 的取值要保持一致。

2. 通过 KMS 公钥和签名内容在本地进行验证。
   - **请求**：

```
openssl dgst -verify public_key.pem -sha256 -signature ./signContent.bin ./test_verify.txt
```

   - **返回结果**：

```
Verified OK
```

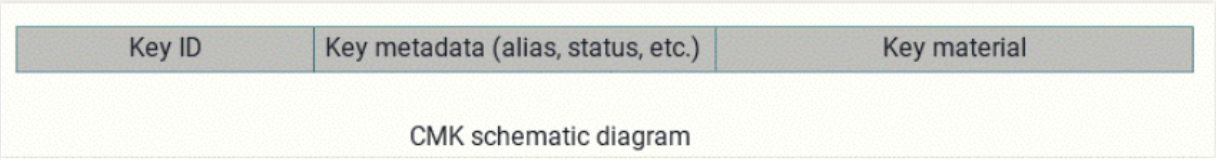
外部密钥导入

概述

最近更新时间：2020-04-27 14:33:31

用户主密钥（CMK，Customer Master Key）是 KMS 服务的基本元素，它包含密钥 ID、密钥元数据（别名、描述、状态等）和用于加解密数据的密钥材料。

默认情况下，通过 KMS 服务创建 CMK 密钥时，由 KMS 服务底层加密机生成安全的密钥材料。当您希望使用自己的密钥材料时，也就是希望实施 BYOK（Bring Your Own Key）方案时，可通过 KMS 服务生成一个密钥材料为空的 CMK，并将自己的密钥材料导入到该用户主密钥中，形成一个外部密钥 CMK（EXTERNAL CMK），再由 KMS 服务进行该外部密钥的分发管理。



CMK schematic diagram

功能特点

- 在腾讯云上实施 BYOK（Bring Your Own Key）方案，即允许您在腾讯云架构上使用您自有的密钥材料进行敏感数据加解密服务。
- 完全掌控并管理您在腾讯云上使用的密钥服务，包括按需导入或删除密钥材料。
- 您可以在本地密钥管理基础设施中备份一份密钥材料，作为腾讯云密钥管理系统的额外灾备措施。
- 通过支持在云上使用您自有的密钥材料进行加解密操作，满足相关行业合规要求。

注意事项

- 需要确保导入密钥材料的安全性：
  - 在使用密钥导入功能时，您需要确保自己生成密钥材料的随机源的安全可靠性。目前 KMS 国密版本仅允许导入128位对称密钥，FIPS 版本仅允许导入256位对称密钥。
- 需要确保导入密钥材料的可用性：
  - KMS 服务提供自身服务的高可用及备份恢复能力，但导入的密钥材料的可用性需由用户来管理。强烈建议您采用安全可靠的方式保存密钥材料的原始备份，以便在意外删除密钥材料或密钥材料过期时，能及时将备份的密钥材料重新导入KMS服务。
- 需注意密钥导入操作的规范性：

- 当您将密钥材料导入CMK 时，该 CMK 与该密钥材料永久关联，即不能将其他密钥材料导入该外部密钥 CMK 中。当使用该外部密钥 CMK 加密数据时，加密后的数据必须使用加密时采用的 CMK（即 CMK 的元数据及密钥材料与导入的密钥匹配）才能解密数据，否则解密将失败。请谨慎处理密钥材料、CMK 的删除操作。

- 需要注意密钥导入的状态：
  待导入状态的密钥属于启用状态的密钥，该启用状态密钥需付费使用。

操作指南

最近更新时间：2019-11-28 19:03:25

操作流程

创建外部密钥 CMK，您可以通过以下4个步骤完成操作。

1. 通过控制台或 API 创建一个密钥来源为“外部”的用户主密钥 CMK，即创建外部密钥 CMK。
2. 通过 API 操作获取密钥材料导入的参数，包括一个用于加密密钥材料的公钥，以及一个导入令牌。
3. 在本地通过加密机或其他安全的加密措施，利用步骤2获取的加密公钥对您的密钥材料进行加密。
4. 通过 API 操作，将加密后的密钥材料及步骤2获取的导入令牌，导入创建的外部密钥 CMK 中，至此导入外部密钥完成。

操作步骤

步骤1：创建外部密钥 CMK

创建外部密钥 CMK 有两种方式，控制台方式和调用 API 的方式。

- **控制台方式**

  （1）登录 密钥管理服务（合规） 控制台。

  （2）选择需要创建密钥的区域，单击【新建】开始创建密钥。

  （3）在新建密钥窗口，输入密钥名称，选择密钥材料来源为外部，阅读导入外部密钥材料的方法及注意事项并勾选确认框。

（4）单击【确定】，即可创建外部密钥 CMK。您可在控制台看到已创建的外部密钥 CMK，“密钥来源”显示为“外部”。

- **调用 API 方式**
  本文示例使用腾讯云 命令行工具 TCCLI，后续您可以使用任何受支持的编程语言调用。
  请求 CreateKey API 时指定参数 Type为2，执行命令如下。

```
tccli kms CreateKey --Alias <alias> --Type 2
```

CreateKey 函数源码示例：

```
def create_external_key(client, alias):
"""
生成 BYOK 密钥,
:param Type = 2
"""
try:
req = models.CreateKeyRequest()
req.Alias = alias
req.Type = 2
rsp = client.CreateKey(req)
return rsp, None
except TencentCloudSDKException as err:
return None, err
```

步骤2：获取导入密钥材料参数

为确保密钥材料的安全性，需要先对您的密钥材料进行加密后再导入。您可以通过 API 获取导入密钥材料的参数，其中包括一个用于加密密钥材料的公钥，以及一个导入令牌。

通过 TCCLI 执行命令如下：

```
tccli kms GetParametersForImport --KeyId <keyid> --WrappingAlgorithm RSAES_PKCS1_V1_5 --WrappingKeySpec RSA_2048
```

GetParametersForImport 函数源码示例：

```
def get_parameters_for_import(client, keyid):
"""
获取导入主密钥（CMK）材料的参数，
返回的Token作为执行ImportKeyMaterial的参数之一，
返回的PublicKey用于对自主导入密钥材料进行加密。
返回的Token和PublicKey 24小时后失效，失效后如需重新导入，需要再次调用该接口获取新的 Token 和 PublicKey。
WrappingAlgorithm 指定加密密钥材料的算法，目前支持 RSAES_PKCS1_V1_5、RSAES_OAEP_SHA_1、RSAES_OAEP_SHA_256。
WrappingKeySpec 指定加密密钥材料的类型，目前只支持 RSA_2048。
"""
try:
req = models.GetParametersForImportRequest()
req.KeyId = keyid
req.WrappingAlgorithm = 'RSAES_PKCS1_V1_5' # RSAES_PKCS1_V1_5 | RSAES_OAEP_SHA_1 | RSAES_OAEP_SHA_256
req.WrappingKeySpec = 'RSA_2048' # RSA_2048
rsp = self.client.GetParametersForImport(req)
return rsp, None
except TencentCloudSDKException as err:
return None, err
```

步骤3：本地加密您的密钥材料

在本地利用 步骤2 获取的加密公钥对您的密钥材料进行加密。加密公钥是一个2048比特的 RSA 公钥，使用的加密算法需要与获取导入密钥材料参数时指定的一致。 由于 API 返回的加密公钥经过 base64 编码，因此在使用时需要先进行 base64 解码。目前 KMS 支持的加密算法有 RSAES_OAEP_SHA_1、RSAES_OAEP_SHA_256 与 RSAES_PKCS1_V1_5。

以下为通过 openssl 加密密钥材料的测试示例。在实际使用中，建议您通过加密机或其他更为安全的加密措施对密钥材料进行加密。

（1）调用 GetParametersForImport 接口获取Token 和 PublicKey。将 PublicKey 写入文件 public_key.base64。
（2）使用 openssl 生成随机数。

```
openssl rand -out raw_material.bin 16
```

您也可以通过 GenerateRandom 生成随机数进行 base64 解码。

> ⚠ **注意：**
>
> 国密版本 key material 长度必须为 128， FIPS 版本为 256。

（3）Decode Public Key 获取公钥原文。

```
openssl enc -d -base64 -A -in public_key.base64 -out public_key.bin
```

（4）使用公钥对 key material 进行加密。

```
# RSAES_OAEP_SHA_1 对应的命令行如下
openssl pkeyutl -in raw_material.bin -out encrypted_key_material.bin -inkey publi
c_key.bin -keyform DER -pubin -encrypt -pkeyopt rsa_padding_mode:oaep -pkeyopt rs
a_oaep_md:sha1

# RSAES_PKCS1_V1_5 对应的命令行如下
openssl pkeyutl -in raw_material.bin -out encrypted_key_material.bin -inkey publi
c_key.bin -keyform DER -pubin -encrypt -pkeyopt rsa_padding_mode:pkcs1

# RSAES_OAEP_SHA_256 对应的命令行如下
openssl pkeyutl -in raw_material.bin -out encrypted_key_material.bin -inkey publi
c_key.bin -keyform DER -pubin -encrypt -pkeyopt rsa_padding_mode:oaep -pkeyopt rs
a_oaep_md:sha256
```

（5）将密文编码后作为参数可以导入 KMS。

```
openssl enc -e -base64 -A -in encrypted_key_material.bin -out encrypted_material.
base64
```

encrypted_material.base64 为最终输出，作为 EncryptedKeyMaterial 导入 KMS。

步骤4：导入密钥材料

最后，将加密后的密钥材料及 步骤2 获取的导入令牌，通过 API 操作，一起导入至 步骤1 创建的外部密钥 CMK中。

- 导入令牌与加密密钥材料的公钥具有绑定关系，同时一个令牌只能为其生成时指定的主密钥导入密钥材料。导入令牌的有效期为24小时，在有效期内可以重复使用，失效以后需要获取新的导入令牌和加密公钥。
- 如果多次调用 GetParametersForImport 获取导入材料，只有最后一次调用的 token 和 publicKey 有效，历史调用返回的将自动过期。

- 可以为从未导入过密钥材料的外部密钥导入密钥材料，也可以重新导入已经过期和已被删除的密钥材料，或者重置密钥材料的过期时间。

请求通过 ImportKeyMaterial 来导入，命令示例如下：

```
tccli kms ImportKeyMaterial --EncryptedKeyMaterial <material> --ImportToken <token> --KeyId <keyid>
```

ImportKeyMaterial 函数源码示例：

```
def import_key_material(client, material, token, keyid):
try:
req = models.ImportKeyMaterialRequest()
req.EncryptedKeyMaterial = material
req.ImportToken = token
req.KeyId = keyid
rsp = client.ImportKeyMaterial(req)
return rsp, None
except TencentCloudSDKException as err:
return None, err
```

至此，导入外部密钥操作完成。您可以像使用普通密钥一样使用外部密钥 CMK。

更多操作

删除外部密钥 CMK

外部密钥 CMK 的删除操作涉及到两类操作，一类操作为计划删除 CMK，一类操作为删除密钥材料，两类操作将会有不同的操作效果。

计划删除 CMK

通过计划删除功能，删除外部密钥 CMK。计划删除的操作强制要求有7 - 30天的等待期，当达到到期时间后，外部密钥CMK 将被彻底删除。已经删除的 CMK 将无法恢复，通过其加密的数据也将无法解密，请谨慎操作。

删除密钥材料

您可以通过两种方式删除密钥材料。当密钥材料过期或者被删除以后，外部密钥 CMK 将无法继续使用，由该 CMK 加密的密文也无法被解密，除非您重新导入相同的密钥材料。

- 通过 API 操作 DeleteImportedKeyMaterial 完成。当操作删除密钥材料后，密钥状态将变为等待导入（PendingImport）。
- 在导入密钥材料 API 操作中，将 ImportKeyMaterial 设置 ValidTo 输入参数设置过期时间完成，KMS 服务将自动删除到达过期时间的密钥材料。

说明：

等待密钥材料到期失效与手动删除密钥材料所达到的效果是一样的。

删除密钥材料，执行命令如下：

```
tccli DeleteImportedKeyMaterial --KeyId <keyid>
```

DeleteImportedKeyMaterial 函数源码示例：

```
def delete_key_material(client, keyid):
try:
req = models.DeleteImportedKeyMaterialRequest()
req.KeyId = keyid
rsp = client.DeleteImportedKeyMaterial(req)
return rsp, None
except TencentCloudSDKException as err:
return None, err
```

注意：

- 当您将密钥材料导入 CMK 时，该 CMK 与该密钥材料永久关联，即不能将其他密钥材料导入该外部密钥 CMK 中。当您删除密钥材料后，若需要重新导入密钥材料，导入的密钥材料必须与删除的密钥材料完全相同，才能导入成功。
- 当使用外部密钥 CMK 加密数据时，加密后的数据必须使用加密时采用的 CMK（即 CMK 的元数据及密钥材料与导入的密钥匹配）才能解密数据，否则解密会失败。请谨慎处理密钥材料、CMK 的删除操作。

指数回退策略应对服务限频

最近更新时间：2021-04-19 10:34:19

异常处理的策略建议

请求 KMS API 接口，即应用程序的请求发送到 KMS 远程服务器时，若出现异常报错，建议您可以采用以下策略进行处理：

- **取消**：当返回错误显示故障是非暂时性，或重新执行后也无法成功，则应当终止/取消程序调用并报告异常。
- **重试**：当返回错误是不常见或比较少见，如网络包在传输过程中损坏，但被发送，这种情况下可以立即采取重试。
- **延迟重试**：当返回错误是普通的连接或繁忙相关导致，则服务可能需要短时间的恢复，从而清除堆积的工作等，此类问题，等待一个合适时间后进行重试。

本文以延迟重试的策略展开说明，上述提到的等待时间（即延迟时间），可以采取逐步增加方式，或使用定时策略（如指数回退）来实现。因为调用 KMS API 接口服务会限制频率，所以当您调用的并发过高时，可以采用延迟重试的方法来避免限频所带来的问题。

指数回退

伪代码

```
//使用逐步增加的方式来延迟重试某个操作
InitDelayValue = 100
For(Retries = 0; Retries < MAX_RETRIES; Retries = Retries+1)
wait for (2^Retries * InitDelayValue) milliseconds
Status = KmsApiRequest()
IF Status == SUCCESS
BREAK // Succeeded, stop calling the API again.
ELSE IF Status = THROTTLED || Status == SERVER_NOT_READY
CONTINUE // Failed due to throttling or server busy, try again.
ELSE
BREAK // another error occurs, stop calling the API again.
END IF
```

策略运用

Python 示例：调用 KMS Encrypt 接口遇到限频错误时，如何使用指数回退方法处理

```
# -*- coding: utf-8 -*-
import base64
import math
import time
from tencentcloud.common import credential
from tencentcloud.common.exception.tencent_cloud_sdk_exception import TencentClou
dSDKException
from tencentcloud.common.profile.client_profile import ClientProfile
from tencentcloud.common.profile.http_profile import HttpProfile
from tencentcloud.kms.v20190118 import kms_client, models


def KmsInit(region="ap-guangzhou", secretId="", secretKey=""):
try:
credProfile = credential.Credential(secretId, secretKey)
client = kms_client.KmsClient(credProfile, region)
return client
except TencentCloudSDKException as err:
print(err)
return None

def BackoffFunction(RetryCount):
InitDelayValue = 100
DelayTime = math.pow(2, RetryCount) * InitDelayValue
return DelayTime

if __name__ == '__main__':
# 用户自定义参数
secretId = "replace-with-real-secretId"
secretKey = "replace-with-real-secretKey"
region = "ap-guangzhou"
keyId = "replace-with-realkeyid"
plaintext = "abcdefg123456789abcdefg123456789abcdefg"
Retries = 0
MaxRetries = 10
client = KmsInit(region, secretId, secretKey)
req = models.EncryptRequest()
req.KeyId = keyId
req.Plaintext = base64.b64encode(plaintext)
while Retries < MaxRetries:
try:
Retries += 1
rsp = client.Encrypt(req) # 调用加密接口
print 'plaintext: ',plaintext,'CiphertextBlob: ',rsp.CiphertextBlob
break
except TencentCloudSDKException as err:
```

```
if err.code == 'InternalError' or err.code == 'RequestLimitExceeded':
if Retries == MaxRetries:
break
time.sleep(BackoffFunction(Retries + 1))
continue
else:
print(err)
break
except Exception as err:
print(err)
break
```

⚠ 注意：

- 如需解决其他特定的错误，您可以直接对 except 语句的内容做更改、调整即可。
- 根据自己的代码逻辑、业务策略等，进行定时策略的规划与制定，从而设置最优的**初始延迟值**（InitDelayValue）及**重试次数**（Retries），避免阈值设置过低或过高，影响整体业务运转。