数据加密服务

最佳实践

文档版本 14

发布日期 2024-12-13

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

安全声明

漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：
https://www.huawei.com/cn/psirt/vul-response-process
如企业客户须获取漏洞信息，请参见如下网址：
https://securitybulletin.huawei.com/enterprise/cn/security-advisory

目 录

1 密钥管理

1.1 如何使用 KMS 加密保护线下数据

1.1.1 加解密小量数据

场景说明

当有少量数据（例如：口令、证书、电话号码等）需要加解密时，用户可以通过密钥管理服务（Key Management Service，KMS）界面使用在线工具加解密数据，或者调用KMS的API接口使用指定的用户主密钥直接加密、解密数据。

约束条件

当前支持不大于4KB的小数据加解密。

在线工具加解密

- **加密数据**

步骤1 单击目标自定义密钥的名称，进入密钥信息页面后，单击“工具”页签。

步骤2 在“加密”文本框中输入待加密的数据，如图1-1所示。

图 1-1 加密数据

步骤3 单击“执行”，右侧文本框显示加密后的密文数据。

说明

- 加密数据时，使用当前指定的密钥加密数据。
- 用户可单击“清除”，清除已输入的数据。
- 用户可单击“复制到剪切板”拷贝加密后的密文数据，并保存到本地文件中。

- **解密数据**

步骤4 解密数据时，可单击任意“启用”状态的非默认密钥别名，进入该密钥的在线工具页面。

步骤5 单击“解密”，在左侧文本框中输入待解密的密文数据，如图1-2所示。

说明

- 在线工具自动识别并使用数据被加密时使用的密钥解密数据。
- 如果该密钥已被删除，会导致解密失败。

图 1-2 解密数据

步骤6 单击“执行”，右侧文本框中显示解密后的明文数据。

说明

- 用户可直接单击“复制到剪切板”拷贝解密后的明文数据，并保存到本地文件中。
- 在控制台输入的明文，会进行base64编码得到加密后的字符。

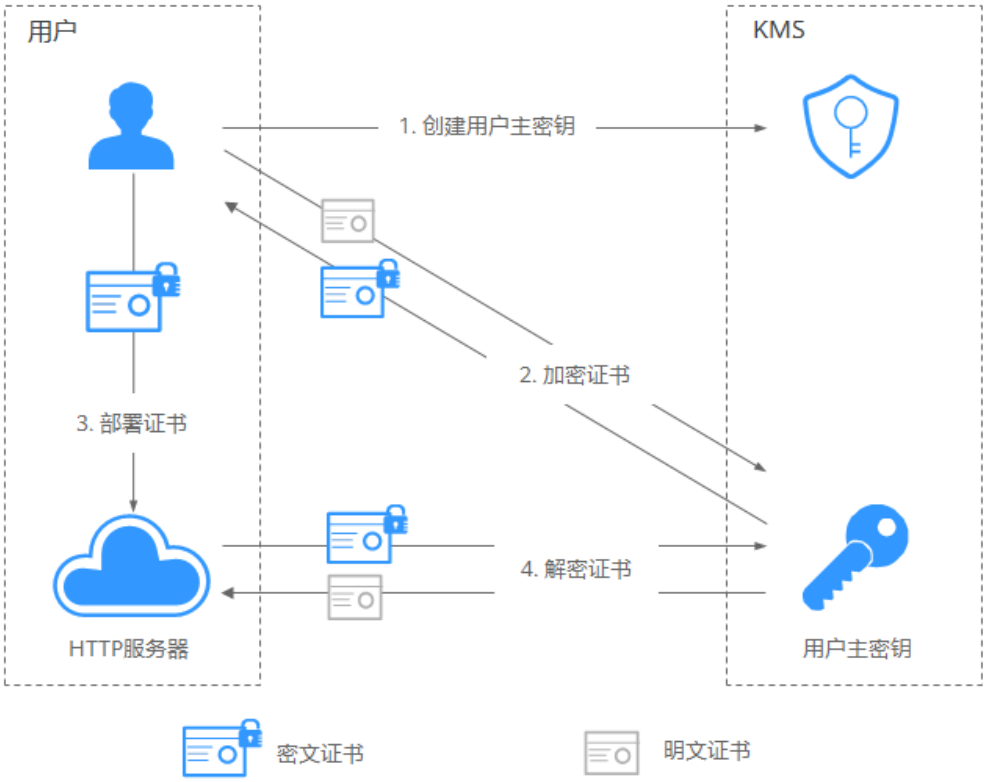
  如果直接调用API接口进行解密，得到的解密结果是进行了base64编码的内容。需再进行一次base64解码才能得到与控制台输入明文一致的内容。

----结束

调用 API 接口加解密

以保护服务器HTTPS证书为例，采用调用KMS的API接口方式进行说明，如图1-3所示。

图 1-3 保护服务器 HTTPS 证书



流程说明如下：

1. 用户需要在KMS中创建一个用户主密钥。
2. 用户调用KMS的**加密数据密钥**接口，使用指定的用户主密钥将明文证书加密为密文证书。
3. 用户在服务器上部署密文证书。
4. 当服务器需要使用证书时，调用KMS的**解密数据密钥**接口，将密文证书解密为明文证书。

说明

由于控制台输入的加密原文会经过一次Base64转码后才传至后端，所以当调用API接口解密密文的时候，返回的明文就是加密原文经过Base64转码得到的字符串，故API加密密文后需要调用API进行解密，如果使用控制台解密API加密密文则会产生乱码。

1.1.2 加解密大量数据

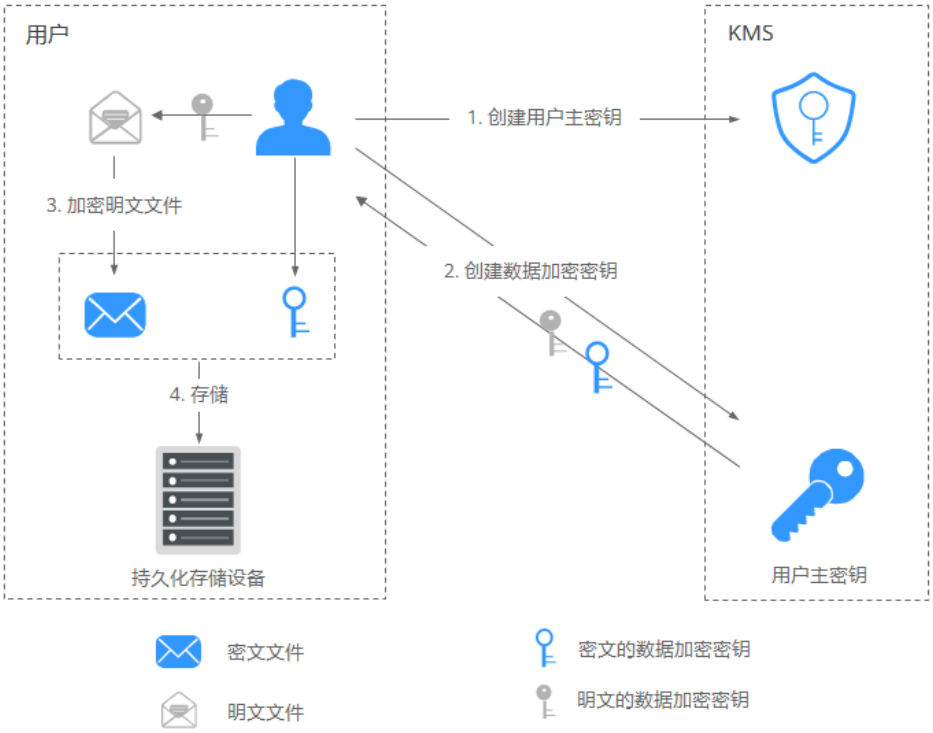
场景说明

当有大量数据（例如：照片、视频或者数据库文件等）需要加解密时，用户可采用信封加密方式加解密数据，无需通过网络传输大量数据即可完成数据加解密。

加密和解密原理

- 大量数据加密

图 1-4 加密本地文件



说明如下：

a. 用户需要在KMS中创建一个用户主密钥。

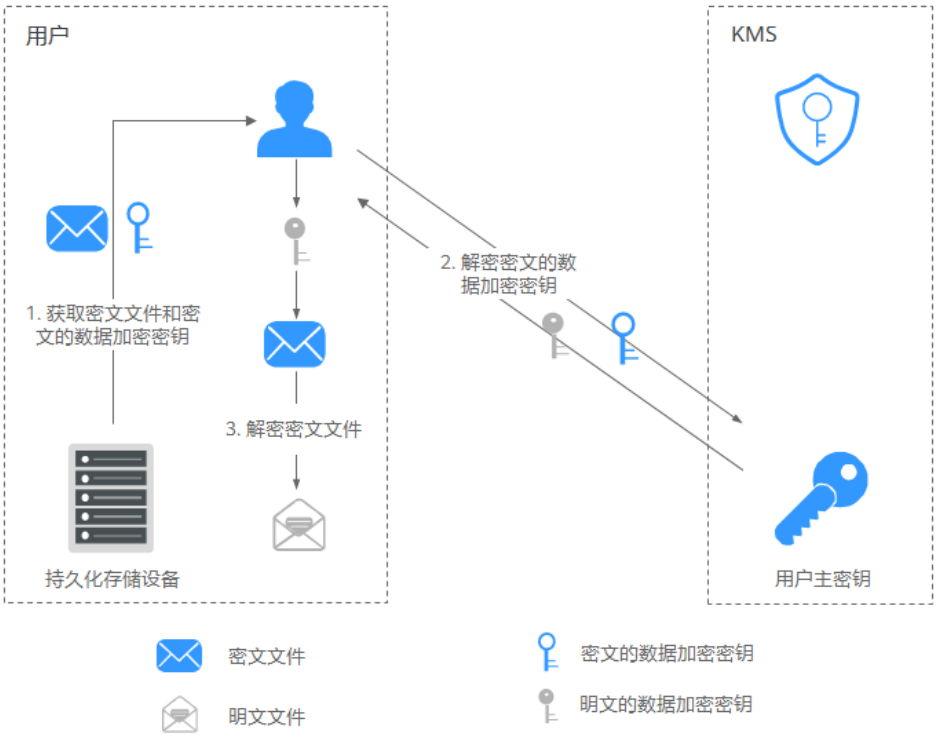
b. 用户调用KMS的“create-datakey”接口创建数据加密密钥。用户得到一个明文的数据加密密钥和一个密文的数据加密密钥。其中**密文的数据加密密钥**是由指定的**用户主密钥**加密**明文的数据加密密钥**生成的。

c. 用户使用明文的数据加密密钥来加密明文文件，生成密文文件。

d. 用户将密文的数据加密密钥和密文文件一同存储到持久化存储设备或服务中。

- 大量数据解密

图 1-5 解密本地文件



说明如下：

a. 用户从持久化存储设备或服务中读取密文的数据加密密钥和密文文件。

b. 用户调用KMS的“decrypt-datakey”接口，使用对应的用户主密钥（即生成密文的数据加密密钥时所使用的用户主密钥）来解密密文的数据加密密钥，取得明文的数据加密密钥。

   如果对应的用户主密钥被误删除，会导致解密失败。因此，需要妥善管理好用户主密钥。

c. 用户使用明文的数据加密密钥来解密密文文件。

信封加密使用的相关 API

您可以调用以下API，在本地对数据进行加解密。

API名称	说明
创建数据密钥	创建数据密钥。
加密数据密钥	用指定的主密钥加密数据密钥。
解密数据密钥	用指定的主密钥解密数据密钥。

加密本地文件

1. 通过华为云控制台，创建用户主密钥，请参见创建密钥。
2. 请准备基础认证信息。

- ACCESS_KEY: 华为账号Access Key
- SECRET_ACCESS_KEY: 华为账号Secret Access Key
- PROJECT_ID: 华为云局点项目ID，请参见**华为云局点项目**。
- KMS_ENDPOINT: 华为云KMS服务访问终端地址，请参见**终端节点**。
- 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
- 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。

说明

华为云SDK，提供统一的SDK使用方式。通过添加依赖或下载的方式调用华为云API，访问华为云应用、资源和数据。如果您需要，请参见**华为云SDK**。

3. 加密本地文件。

   示例代码中：

   - 用户主密钥：华为云控制台创建的密钥ID。
   - 明文数据文件：FirstPlainFile.jpg。
   - 输出的密文数据文件：SecondEncryptFile.jpg。

```
import com.huaweicloud.sdk.core.auth.BasicCredentials;
import com.huaweicloud.sdk.kms.v1.KmsClient;
import com.huaweicloud.sdk.kms.v1.model.CreateDatakeyRequest;
import com.huaweicloud.sdk.kms.v1.model.CreateDatakeyRequestBody;
import com.huaweicloud.sdk.kms.v1.model.CreateDatakeyResponse;
import com.huaweicloud.sdk.kms.v1.model.DecryptDatakeyRequest;
import com.huaweicloud.sdk.kms.v1.model.DecryptDatakeyRequestBody;

import javax.crypto.Cipher;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.file.Files;
import java.security.SecureRandom;

/**
 * 使用数据密钥（DEK）进行文件加解密
 * 激活assert语法，请在VM_OPTIONS中添加-ea
 */
public class FileStreamEncryptionExample {

    private static final String ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_AK");
    private static final String SECRET_ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_SK");
    private static final String PROJECT_ID = "<ProjectID>";
    private static final String KMS_ENDPOINT = "<KmsEndpoint>";


    /**
     * AES算法相关标识：
     * - AES_KEY_BIT_LENGTH: AES256密钥比特长度
     * - AES_KEY_BYTE_LENGTH: AES256密钥字节长度
     * - AES_ALG: AES256算法，本例分组模式使用GCM，填充使用PKCS5Padding
     * - AES_FLAG: AES算法标识
     * - GCM_TAG_LENGTH: GCM TAG长度
     * - GCM_IV_LENGTH: GCM 初始向量长度
     */
    private static final String AES_KEY_BIT_LENGTH = "256";
```



```
    private static final String AES_KEY_BYTE_LENGTH = "32";
    private static final String AES_ALG = "AES/GCM/PKCS5Padding";
    private static final String AES_FLAG = "AES";
    private static final int GCM_TAG_LENGTH = 16;
    private static final int GCM_IV_LENGTH = 12;

    public static void main(final String[] args) {
        // 您在华为云控制台创建的用户主密钥ID
        final String keyId = args[0];

        encryptFile(keyId);
    }

    /**
     * 使用数据密钥加解密文件实例
     *
     * @param keyId 用户主密钥ID
     */
    static void encryptFile(String keyId) {
        // 1.准备访问华为云的认证信息
        final BasicCredentials auth = new
BasicCredentials().withAk(ACCESS_KEY).withSk(SECRET_ACCESS_KEY)
                .withProjectId(PROJECT_ID);

        // 2.初始化SDK，传入认证信息及KMS访问终端地址
        final KmsClient kmsClient =
KmsClient.newBuilder().withCredential(auth).withEndpoint(KMS_ENDPOINT).build();

        // 3.组装创建数据密钥请求信息
        final CreateDatakeyRequest createDatakeyRequest = new CreateDatakeyRequest()
                .withBody(new
CreateDatakeyRequestBody().withKeyId(keyId).withDatakeyLength(AES_KEY_BIT_LENGTH));

        // 4.创建数据密钥
        final CreateDatakeyResponse createDatakeyResponse =
kmsClient.createDatakey(createDatakeyRequest);

        // 5.接收创建的数据密钥信息
        // 密文密钥与KeyId建议保存在本地，方便解密数据时获取明文密钥
        // 明文密钥在创建后立即使用，使用前需要将16进制明文密钥转换成byte数组
        final String cipherText = createDatakeyResponse.getCipherText();
        final byte[] plainKey = hexToBytes(createDatakeyResponse.getPlainText());

        // 6.准备待加密的文件
        // inFile 待加密的原文件
        // outEncryptFile 加密后的文件

        final File inFile = new File("FirstPlainFile.jpg");
        final File outEncryptFile = new File("SecondEncryptFile.jpg");

        // 7.使用AES算法进行加密时，可以创建初始向量
        final byte[] iv = new byte[GCM_IV_LENGTH];
        final SecureRandom secureRandom = new SecureRandom();
        secureRandom.nextBytes(iv);

        // 8.对文件进行加密，并存储加密后的文件
        doFileFinal(Cipher.ENCRYPT_MODE, inFile, outEncryptFile, plainKey, iv);

    }

    /**
     * 对文件进行加解密
     *
     * @param cipherMode 加密模式，可选值为Cipher.ENCRYPT_MODE或者
Cipher.DECRYPT_MODE
     * @param infile     待加解密的文件
     * @param outFile    加解密后的文件
     * @param keyPlain   明文密钥
     * @param iv         初始化向量
```

```
    */
    static void doFileFinal(int cipherMode, File infile, File outFile, byte[] keyPlain, byte[] iv) {

        try (BufferedInputStream bis = new BufferedInputStream(new FileInputStream(infile));
             BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream(outFile))) {
            final byte[] bytIn = new byte[(int) infile.length()];
            final int fileLength = bis.read(bytIn);

            assert fileLength > 0;

            final SecretKeySpec secretKeySpec = new SecretKeySpec(keyPlain, AES_FLAG);
            final Cipher cipher = Cipher.getInstance(AES_ALG);
            final GCMParameterSpec gcmParameterSpec = new
GCMParameterSpec(GCM_TAG_LENGTH * Byte.SIZE, iv);
            cipher.init(cipherMode, secretKeySpec, gcmParameterSpec);
            final byte[] bytOut = cipher.doFinal(bytIn);
            bos.write(bytOut);
        } catch (Exception e) {
            throw new RuntimeException(e.getMessage());
        }
    }

}
```

解密本地文件

1. 请准备基础认证信息。
   - ACCESS_KEY: 华为账号Access Key
   - SECRET_ACCESS_KEY: 华为账号Secret Access Key
   - PROJECT_ID: 华为云局点项目ID，请参见**华为云局点项目**。
   - KMS_ENDPOINT: 华为云KMS服务访问终端地址，请参见**终端节点**。
   - 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
   - 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。

   说明

   华为云SDK，提供统一的SDK使用方式。通过添加依赖或下载的方式调用华为云API，访问华为云应用、资源和数据。如果您需要，请参见**华为云SDK**。

2. 解密本地文件。

   示例代码中：
   - 用户主密钥：华为云控制台创建的密钥ID。
   - 输出的密文数据文件：SecondEncryptFile.jpg。
   - 加密后再解密的数据文件：ThirdDecryptFile.jpg。

```
import com.huaweicloud.sdk.core.auth.BasicCredentials;
import com.huaweicloud.sdk.kms.v1.KmsClient;
import com.huaweicloud.sdk.kms.v1.model.CreateDatakeyRequest;
import com.huaweicloud.sdk.kms.v1.model.CreateDatakeyRequestBody;
import com.huaweicloud.sdk.kms.v1.model.CreateDatakeyResponse;
import com.huaweicloud.sdk.kms.v1.model.DecryptDatakeyRequest;
import com.huaweicloud.sdk.kms.v1.model.DecryptDatakeyRequestBody;

import javax.crypto.Cipher;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.io.BufferedInputStream;
```



```
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.file.Files;
import java.security.SecureRandom;

/**
 * 使用数据密钥（DEK）进行文件加解密
 * 激活assert语法，请在VM_OPTIONS中添加-ea
 */
public class FileStreamEncryptionExample {

    private static final String ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_AK");
    private static final String SECRET_ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_SK");
    private static final String PROJECT_ID = "<ProjectID>";
    private static final String KMS_ENDPOINT = "<KmsEndpoint>";

    /**
     * AES算法相关标识：
     * - AES_KEY_BIT_LENGTH: AES256密钥比特长度
     * - AES_KEY_BYTE_LENGTH: AES256密钥字节长度
     * - AES_ALG: AES256算法，本例分组模式使用GCM，填充使用PKCS5Padding
     * - AES_FLAG: AES算法标识
     * - GCM_TAG_LENGTH: GCM TAG长度
     * - GCM_IV_LENGTH: GCM 初始向量长度
     */
    private static final String AES_KEY_BIT_LENGTH = "256";
    private static final String AES_KEY_BYTE_LENGTH = "32";
    private static final String AES_ALG = "AES/GCM/PKCS5Padding";
    private static final String AES_FLAG = "AES";
    private static final int GCM_TAG_LENGTH = 16;
    private static final int GCM_IV_LENGTH = 12;

    public static void main(final String[] args) {
        // 您在华为云控制台创建的用户主密钥ID
        final String keyId = args[0];
        // 创建数据密钥时，响应的密文数据密钥
        final String cipherText = args[1];

        decryptFile(keyId, cipherText);
    }

    /**
     * 使用数据密钥加解密文件实例
     *
     * @param keyId 用户主密钥ID
     * @param cipherText 密文数据密钥
     */
    static void decryptFile(String keyId，String cipherText) {
        // 1.准备访问华为云的认证信息
        final BasicCredentials auth = new
BasicCredentials().withAk(ACCESS_KEY).withSk(SECRET_ACCESS_KEY)
                .withProjectId(PROJECT_ID);

        // 2.初始化SDK，传入认证信息及KMS访问终端地址
        final KmsClient kmsClient =
KmsClient.newBuilder().withCredential(auth).withEndpoint(KMS_ENDPOINT).build();

        // 3.准备待加密的文件
        // inFile 待加密的文件
        // outEncryptFile 加密后的文件
        // outDecryptFile 加密后再解密的文件
        final File inFile = new File("FirstPlainFile.jpg");
        final File outEncryptFile = new File("SecondEncryptFile.jpg");
        final File outDecryptFile = new File("ThirdDecryptFile.jpg");

        // 4.使用AES算法进行解密时，初始向量需要与加密时保持一致，此处仅为占位。
```

```
        final byte[] iv = new byte[GCM_IV_LENGTH];

        // 5.组装解密数据密钥的请求，其中cipherText为创建数据密钥时返回的密文数据密钥。
        final DecryptDatakeyRequest decryptDatakeyRequest = new DecryptDatakeyRequest()
                .withBody(new DecryptDatakeyRequestBody()
                        .withKeyId(keyId).withCipherText(cipherText).withDatakeyCipherLength(AES_KEY
_BYTE_LENGTH));

        // 6.解密数据密钥，并对返回的16进制明文密钥换成byte数组
        final byte[] decryptDataKey =
hexToBytes(kmsClient.decryptDatakey(decryptDatakeyRequest).getDataKey());

        // 7.对文件进行解密，并存储解密后的文件
        // 句末的iv为加密示例中创建的初始向量
        doFileFinal(Cipher.DECRYPT_MODE, outEncryptFile, outDecryptFile, decryptDataKey, iv);

        // 8.比对原文件和加密后再解密的文件
        assert getFileSha256Sum(inFile).equals(getFileSha256Sum(outDecryptFile));

    }

    /**
     * 对文件进行加解密
     *
     * @param cipherMode 加密模式，可选值为Cipher.ENCRYPT_MODE或者
Cipher.DECRYPT_MODE
     * @param infile     待加解密的文件
     * @param outFile    加解密后的文件
     * @param keyPlain   明文密钥
     * @param iv         初始化向量
     */
    static void doFileFinal(int cipherMode, File infile, File outFile, byte[] keyPlain, byte[] iv) {

        try (BufferedInputStream bis = new BufferedInputStream(new FileInputStream(infile));
             BufferedOutputStream bos = new BufferedOutputStream(new
FileOutputStream(outFile))) {
            final byte[] bytIn = new byte[(int) infile.length()];
            final int fileLength = bis.read(bytIn);

            assert fileLength > 0;

            final SecretKeySpec secretKeySpec = new SecretKeySpec(keyPlain, AES_FLAG);
            final Cipher cipher = Cipher.getInstance(AES_ALG);
            final GCMParameterSpec gcmParameterSpec = new
GCMParameterSpec(GCM_TAG_LENGTH * Byte.SIZE, iv);
            cipher.init(cipherMode, secretKeySpec, gcmParameterSpec);
            final byte[] bytOut = cipher.doFinal(bytIn);
            bos.write(bytOut);
        } catch (Exception e) {
            throw new RuntimeException(e.getMessage());
        }
    }

    /**
     * 十六进制字符串转byte数组
     *
     * @param hexString 十六进制字符串
     * @return byte数组
     */
    static byte[] hexToBytes(String hexString) {
        final int stringLength = hexString.length();
        assert stringLength > 0;
        final byte[] result = new byte[stringLength / 2];
        int j = 0;
        for (int i = 0; i < stringLength; i += 2) {
            result[j++] = (byte) Integer.parseInt(hexString.substring(i, i + 2), 16);
        }
        return result;
    }
```



```
    /**
     * 计算文件SHA256摘要
     *
     * @param file 文件
     * @return SHA256摘要
     */
    static String getFileSha256Sum(File file) {
        int length;
        MessageDigest sha256;
        byte[] buffer = new byte[1024];
        try {
            sha256 = MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e.getMessage());
        }
        try (FileInputStream inputStream = new FileInputStream(file)) {
            while ((length = inputStream.read(buffer)) != -1) {
                sha256.update(buffer, 0, length);
            }
            return new BigInteger(1, sha256.digest()).toString(16);
        } catch (IOException e) {
            throw new RuntimeException(e.getMessage());
        }
    }
}
```

1.2 云服务使用 KMS 加解密数据

1.2.1 概述

云服务与KMS集成后，您只需在决定加密云服务数据时，选择一个KMS管理的用户主密钥，就可以轻松使用您选择的用户主密钥加解密您存储在这些云服务内的数据。

您可以选择云服务自动通过KMS创建的默认密钥，也可以选择您通过KMS自行创建或导入的自定义密钥，详细请参见**默认密钥与自定义密钥的区别**。

表 1-1 使用 KMS 加密的云服务列表

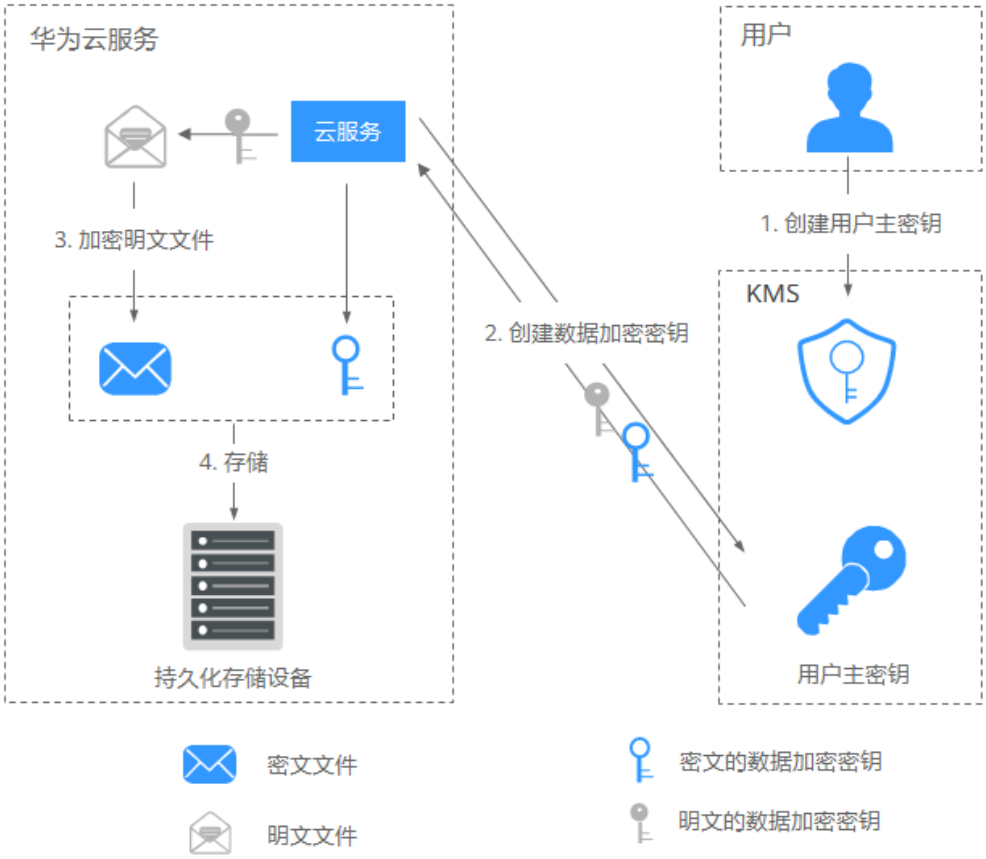
类型	服务	加密方式说明
计算	弹性云服务器ECS	弹性云服务器资源加密包括镜像加密和云硬盘加密。 • 在创建弹性云服务器时，您如果选择加密镜像，弹性云服务器的系统盘会自动开启加密功能，加密方式与镜像保持一致。 • 在创建弹性云服务器时，您也可以对添加的数据盘进行加密。
	镜像服务IMS	**IMS服务端加密**
存储	对象存储服务OBS	**OBS服务端加密**
	云硬盘EVS	**EVS服务端加密**

类型	服务	加密方式说明
	云硬盘备份VBS	云硬盘备份主要对服务器中单个的云硬盘（系统盘和数据盘）创建在线备份，加密云硬盘的备份数据会以加密方式存放。
	云服务器备份CSBS	云服务器备份主要对服务器下所有云硬盘创建一致性在线备份，云服务器备份产生的备份，会显示在云硬盘备份中。加密云硬盘的备份数据会以加密方式存放。
	弹性文件服务SFS	**SFS服务端加密**
数据库	云数据库MySQL	**RDS数据库加密**
	云数据库Postgre SQL	
	云数据库SQL Server	
	文档数据库服务DDS	**DDS数据库加密**
EI企业智能	数据仓库服务DWS	**DWS数据库加密**

原理介绍

华为云服务基于信封加密技术，通过调用KMS接口来加密云服务资源。由用户管理自己的用户主密钥，华为云服务在拥有用户授权的情况下，使用用户指定的用户主密钥对数据进行加密。

图 1-6 华为云服务使用 KMS 加密原理



加密流程说明如下：

1. 用户需要在KMS中创建一个用户主密钥。
2. 华为云服务调用KMS的“create-datakey”接口创建数据加密密钥。得到一个明文的数据加密密钥和一个密文的数据加密密钥。

   说明

   密文的数据加密密钥是由指定的用户主密钥加密明文的数据加密密钥生成的。
3. 华为云服务使用明文的数据加密密钥来加密明文文件，得到密文文件。
4. 华为云服务将密文的数据加密密钥和密文文件一同存储到持久化存储设备或服务中。

说明

用户通过华为云服务下载数据时，华为云服务通过KMS指定的用户主密钥对密文的数据加密密钥进行解密，并使用解密得到的明文的数据加密密钥来解密密文数据，然后将解密后的明文数据提供给用户下载。

1.2.2 ECS 服务端加密

简介

KMS支持对ECS进行一键加密，弹性云服务器资源加密包括镜像加密和数据盘加密。

- 在创建弹性云服务器时，您如果选择加密镜像，弹性云服务器的系统盘会自动开启加密功能，加密方式与镜像保持一致。

- 在创建弹性云服务器时，您也可以对添加的数据盘进行加密。

镜像加密请参见**IMS服务端加密**。

数据盘加密请参见**EVS服务端加密**。

1.2.3 OBS 服务端加密

应用场景

KMS可以对OBS桶中的对象进行全量加密或者部分加密，在OBS服务使用KMS加密过程中，KMS提供的信封加密能力使数据加解密操作无需通过网络传输大量数据即可完成。信封加密方式有效保障了数据传输的加密性、数据解密的效率和便捷性，在对象上传和下载过程中，保证信息安全。

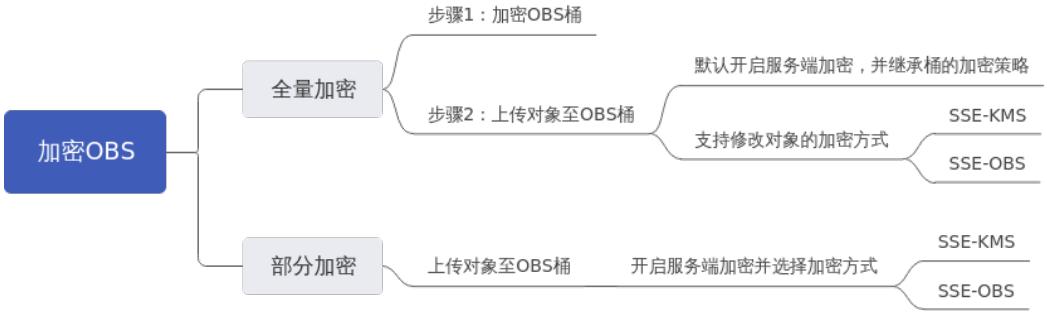
- 全量加密：指对OBS桶内上传的所有对象进行加密。

  此时，您只需要对OBS桶加密，桶中上传的对象会默认继承OBS桶的加密配置。具体操作请参见**加密OBS桶：创建OBS桶时开启服务端加密功能**或者**加密OBS桶：为已创建的OBS桶开启加密**。

  开启OBS桶加密后，上传对象时默认开启“继承桶加密配置”加密方式，此时桶中的对象和OBS桶采用相同的加密方式；如需修改桶中对象的加密方式，需要在上传对象时手动关闭“继承桶加密配置”开关，然后修改。具体操作请参见**上传对象至OBS桶**。
- 部分加密：指对OBS桶内上传的部分对象进行加密。

  此时不需要对OBS桶进行加密，直接上传对象到OBS桶并进行加密配置。具体操作请参见**上传对象至OBS桶**。

图 1-7 加密 OBS



约束与限制

使用中的密钥不可以删除，如果删除将导致加密对象不能下载。

加密 OBS 桶：创建 OBS 桶时开启服务端加密功能

步骤1 **登录管理控制台**。

步骤2 单击页面左侧 ☰，选择“存储 > 对象存储服务 OBS”。

步骤3 在左侧导航栏选择“桶列表”，在页面右上角单击“创建桶”。

步骤4 在服务端加密选择“SSE-KMS”方式后，选择加密密钥类型。

图 1-8 OBS 服务端加密

说明

OBS使用KMS提供的加密密钥，支持选择的密钥：

- 默认密钥“obs/default”，如果您没有默认密钥，系统将会在首次上传对象时为您自动创建。
- 用户使用KMS创建的自定义密钥，具体操作请参见**创建密钥**。
- 使用SM4加密算法类型密钥加密OBS服务，仅支持在华北-乌兰察布一区域。

步骤5 完成其他参数配置后，单击“立即创建”，完成OBS桶加密配置。

----结束

加密 OBS 桶：为已创建的 OBS 桶开启加密

步骤1 **登录管理控制台**。

步骤2 单击页面左侧 ☰，选择“存储 > 对象存储服务 OBS”。

步骤3 在左侧导航栏选择“桶列表”，在桶列表中，单击待操作的桶，进入“对象”页面。

步骤4 在左侧导航栏单击“概览”，进入“概览”页面。

步骤5 在概览页的“基础配置”区域下，单击“服务端加密”卡片，系统弹出“服务端加密”对话框。

步骤6 选择“SSE-KMS”方式后，选择加密密钥类型。

图 1-9 开启服务端加密

说明

OBS使用KMS提供的加密密钥，支持选择的密钥：

- 默认密钥“obs/default”，如果您没有默认密钥，系统将会在首次上传对象时为您自动创建。
- 用户使用KMS创建的自定义密钥，具体操作请参见**创建密钥**。
- 使用SM4加密算法类型密钥加密OBS服务，仅支持在华北-乌兰察布一区域。

步骤7 完成设置后，单击“确定”，服务端加密配置生效。

----结束

上传对象至 OBS 桶

步骤1 在OBS管理控制台桶列表中，单击待操作的桶，进入“概览”页面。

步骤2 在左侧导航栏，单击“对象”。

步骤3 单击“上传对象”，系统弹出“上传对象”对话框。

步骤4 单击“添加文件”，选择待上传的文件后，单击“打开”。

步骤5 在服务端加密行，选择目标加密方式，选择完成后，在下方的选择框中选择默认密钥或者自定义密钥，如**图1-10**所示。

图 1-10 加密上传对象（已开启 OBS 桶加密）

说明

- 开启OBS桶加密后，上传对象时默认开启继承桶的加密配置。
- 如果需要修改加密配置，需要手动关闭“继承桶的加密配置”选项，根据使用需求选择SSE-KMS或SSE-OBS加密方式。

图 1-11 加密上传对象（未开启 OBS 桶加密）

说明

未开启OBS桶加密在上传对象时需要手动开启服务端加密。

步骤6 对象上传成功后，可在对象列表中查看对象的加密状态。

说明

- 对象的加密状态不可以修改。
- 使用中的密钥不可以删除，如果删除将导致加密对象不能下载。

----结束

相关操作

用户也可以通过调用OBS API接口，选择服务端加密SSE-KMS方式（SSE-KMS方式是指OBS使用KMS提供的密钥进行服务端加密）上传文件，详情请参考**《对象存储服务API参考》**。

1.2.4 EVS 服务端加密

简介

当您由于业务需求需要对存储在云硬盘的数据进行加密时，EVS为您提供加密功能，可以对新创建的云硬盘进行加密。加密云硬盘使用的密钥由数据加密服务（DEW，Data

Encryption Workshop）中的密钥管理（KMS，Key Management Service）功能提供，无需您自行构建和维护密钥管理基础设施，安全便捷。

磁盘加密针对数据盘加密。系统盘的加密依赖于镜像，具体请参见**IMS服务端加密**。

用户权限说明

- 安全管理员（拥有“Security Administrator”权限）可以直接授权EVS访问KMS，使用加密功能。
- 普通用户（没有“Security Administrator”权限）使用加密功能时，根据该普通用户是否为当前区域或者项目内第一个使用加密特性的用户，作如下区分：
  - 是，即该普通用户是当前区域或者项目内第一个使用加密功能的，需先联系安全管理员进行授权，然后再使用加密功能。
  - 否，即区域或者项目内的其他用户已经使用过加密功能，该普通用户可以直接使用加密功能。

对于一个租户而言，同一个区域内只要安全管理员成功授权EVS访问KMS，则该区域内的普通用户都可以直接使用加密功能。

如果当前区域内存在多个项目，则每个项目下都需要安全管理员执行授权操作。

云硬盘加密的密钥

加密云硬盘使用KMS提供的密钥，包括默认主密钥和用户主密钥 (CMK，Customer Master Key)：

- 默认主密钥：由EVS通过KMS自动创建的密钥，名称为“evs/default”。

  默认主密钥不支持禁用、计划删除等操作。
- 用户主密钥：由用户自己创建的密钥，您可以选择已有的密钥或者新创建密钥，具体请参见**创建密钥**。

使用用户主密钥加密云硬盘，如果对用户主密钥执行禁用、计划删除等操作，将会导致云硬盘不可读写，甚至数据永远无法恢复，具体请参见**表1-2**。

表 1-2 用户主密钥不可用对加密云硬盘的影响

用户主密钥的状态	对加密云硬盘的影响	恢复方法
禁用	• 如果加密云硬盘此时挂载至云服务器，则该云硬盘仍可以正常使用，但不保证一直可以正常读写。 • 如果卸载加密云硬盘后，再重新挂载至云服务器将会失败。	启用用户主密钥，具体请参见**启用密钥**。
计划删除		取消删除用户主密钥，具体请参见**取消删除密钥**。
已经被删除		云硬盘数据永远无法恢复。

须知

用户主密钥为付费使用，如果为按需计费的密钥，请及时充值确保账户余额充足，如果为包年/包月的密钥，请及时续费，以避免加密云硬盘不可读写导致业务中断，甚至数据永远无法恢复。

使用 KMS 加密云硬盘（控制台）

步骤1 登录EVS管理控制台。

步骤2 单击页面右上角“购买磁盘”，进入“购买磁盘”页面。

步骤3 配置“加密”参数。

1. 展开“更多”，出现“加密”勾选框。

图 1-12 展开更多

2. 创建委托。

   勾选“加密”，如果当前未授权EVS访问KMS，则会弹出“创建委托”对话框，单击“是”，授权EVS访问KMS，当授权成功后，EVS可以获取KMS密钥用来加解密云硬盘。

   说明

   当您需要使用云硬盘加密功能时，需要授权EVS访问KMS。如果您有授权资格，则可直接授权。如果权限不足，需先联系拥有“Security Administrator”权限的用户授权，然后再重新操作。

3. 设置加密参数。

   勾选“加密”，如果已经授权，系统弹出“加密设置”对话框。

图 1-13 加密设置

密钥名称是密钥的标识，您可以通过“密钥名称”下拉框选择需要使用的密钥。您可以选择使用的密钥如下：

- 默认主密钥：成功授权EVS访问KMS，系统会创建默认主密钥“evs/default”。
- 用户主密钥：即您已有的密钥或者新创建密钥，具体请参见**创建密钥**。

步骤4 根据界面提示，配置云硬盘的其他基本信息。详细参数说明请参见**购买云硬盘**。

----结束

使用 KMS 加密云硬盘（API）

用户也可以通过调用EVS API接口购买加密磁盘，详情请参考《云硬盘API参考》。

1.2.5 IMS 服务端加密

用户可以采用加密方式创建私有镜像，确保镜像数据安全性。

约束条件

- 用户已启用数据加密服务。
- 加密镜像不能共享给其他用户。
- 加密镜像不能发布到应用超市。
- 如果云服务器的系统盘已加密，那么使用该云服务器创建的私有镜像也是加密的。
- 不能修改加密镜像使用的密钥。
- 加密镜像使用的密钥为禁用状态或者被删除时，该镜像无法使用。
- 对于加密镜像创建的弹性云服务器，其系统盘只能为加密状态，且磁盘密钥与镜像密钥一致。

使用 KMS 加密私有镜像（控制台）

创建加密镜像分为通过加密弹性云服务器创建加密镜像和通过外部镜像文件创建加密镜像。

- 通过加密弹性云服务器创建加密镜像

  用户选择弹性云服务器创建私有镜像时，如果该云服务器的系统盘已加密，那么使用该云服务器创建的私有镜像也是加密的。镜像加密使用的密钥为创建该系统盘时使用的密钥。

- 通过外部镜像文件创建加密镜像

  用户使用OBS桶中已上传的外部镜像文件创建私有镜像过程中，可以在注册镜像时勾选KMS加密完成镜像加密。

  用户上传镜像文件时，可以选择“KMS加密”，使用KMS提供的密钥来加密上传的文件，如**图1-14**所示。

  a. 在IMS管理控制台，单击“创建私有镜像”。
  b. “创建方式”选择“系统盘镜像”。
  c. “选择镜像源”为“镜像文件”。
  d. 勾选“KMS加密”。

图 1-14 IMS 服务端加密

密钥名称是密钥的标识，您可以通过“密钥名称”下拉框选择需要使用的密钥。您可以选择使用的密钥如下：

- 默认主密钥：KMS为使用IMS（Image Management Service，IMS）的用户创建一个默认主密钥“ims/default”。
- 用户主密钥：即您已有的密钥或者新创建密钥，具体请参见**创建密钥**。

e. 根据界面提示，配置其他信息。详细参数说明请参见**注册镜像**。

使用 KMS 加密私有镜像（API）

用户也可以通过调用IMS API接口创建加密镜像，详情请参考《镜像服务API参考》。

1.2.6 SFS 服务端加密

简介

当您由于业务需求需要对存储在文件系统的数据进行加密时，弹性文件服务为您提供加密功能，可以对新创建的文件系统进行加密。

加密文件系统使用的是密钥管理服务（KMS）提供的密钥，无需您自行构建和维护密钥管理基础设施，安全便捷。当用户希望使用自己的密钥材料时，可通过KMS管理控制台的导入密钥功能创建密钥材料为空的用户主密钥，并将自己的密钥材料导入该用户主密钥中。具体操作请参见**导入密钥材料**。

当您需要使用文件系统加密功能时，创建SFS文件系统需要授权SFS访问KMS。如果创建SFS Turbo文件系统，则不需要授权。

哪些用户有权限使用文件系统加密

- 安全管理员（拥有“Security Administrator”权限）可以直接授权SFS访问KMS，使用加密功能。
- 普通用户（没有“Security Administrator”权限）使用加密功能时，需要联系系统管理员获取安全管理员权限。

同一个区域内只要安全管理员成功授权SFS访问KMS，则该区域内的普通用户都可以直接使用加密功能。

如果当前区域内存在多个项目，则每个项目下都需要安全管理员执行授权操作。

文件系统加密的密钥

SFS加密文件系统使用KMS提供的密钥，包括默认主密钥和用户主密钥 (CMK，Customer Master Key)：

- 默认主密钥：系统会为您创建默认主密钥，名称为“sfs/default”。
  默认主密钥不支持禁用、计划删除等操作。
- 用户主密钥：即您已有的密钥或者新创建密钥，具体请参见**创建密钥**。
  如果加密文件系统使用的用户主密钥被执行禁用或计划删除操作，当操作生效后，使用该用户主密钥加密的文件系统仅可以在一段时间内（默认为60s）正常使用。请谨慎操作。

SFS Turbo文件系统无默认主密钥，可以使用您已有的密钥或者创建新的密钥，具体请参见**创建密钥**。

使用 KMS 加密文件系统（控制台）

用户通过弹性文件服务（Scalable File Service，SFS）创建文件系统时，可以选择“启用静态数据加密”，使用KMS提供的密钥来加密文件系统。

步骤1 在SFS管理控制台，单击“创建文件系统”。

步骤2 配置“加密”参数。

1. 创建委托。
   勾选“启用静态数据加密”，如果当前未授权SFS访问KMS，则会弹出“创建委托”对话框，单击“是”，授权SFS访问KMS，当授权成功后，SFS可以获取KMS密钥用来加解密文件系统。

   说明

   当您需要使用文件系统加密功能时，需要授权SFS访问KMS。如果您有授权资格，则可直接授权。如果权限不足，需先联系拥有“Security Administrator”权限的用户授权，然后再重新操作。

2. 设置加密参数。
   勾选“加密”，如果已经授权，会弹出“加密设置”对话框。

图 1-15 加密设置

密钥名称是密钥的标识，您可以通过“密钥名称”下拉框选择需要使用的密钥。您可以选择使用的密钥如下：

- 默认主密钥：成功授权SFS访问KMS，系统会创建默认主密钥“sfs/default”。

- 自定义密钥：即您已有的密钥或者新创建密钥，具体请参见**创建密钥**。

步骤3 根据界面提示，配置云硬盘的其他基本信息。详细参数说明请参见**创建文件系统**。

----结束

使用 KMS 加密文件系统（API）

用户也可以通过调用SFS API接口创建加密的文件系统，详情请参考《弹性文件服务API参考》。

1.2.7 RDS 数据库加密

简介

关系型数据库支持MySQL、PostgreSQL、SQL Server引擎。

当启用加密功能后，用户创建数据库实例和扩容磁盘时，磁盘数据会在服务端加密成密文后存储。用户下载加密对象时，存储的密文会先在服务端解密为明文，再提供给用户。

约束条件

- 当前登录用户已通过统一身份认证服务添加华为云关系型数据库所在区域的KMS Administrator权限。权限添加方法请参见《统一身份认证服务用户指南》的“如何管理用户组并授权？”章节。
- 如果用户需要使用自定义密钥加密上传对象，则需要先通过数据加密服务创建密钥。使用数据加密服务创建密钥详情请参见**创建密钥**。
- 实例创建成功后，不可修改磁盘加密状态，且无法更改密钥。存放在对象存储服务上的备份数据不会被加密。
- 华为云关系型数据库实例创建成功后，请勿禁用或删除正在使用的密钥，否则会导致服务不可用，数据无法恢复。
- 选择磁盘加密的实例，新扩容的磁盘空间依然会使用原加密密钥进行加密。

使用 KMS 加密数据库实例（控制台）

用户在通过关系型数据库（Relational Database Service，RDS）购买数据库实例时，可以选择“磁盘加密”，使用KMS提供的密钥来加密数据库实例的磁盘，更多信息请参见**购买MySQL实例**、**购买PostgreSQL实例**、**购买SQL Server实例**。

图 1-16 RDS 服务端加密

使用 KMS 加密数据库实例（API）

用户也可以通过调用RDS API接口购买加密数据库实例，详情请参考《关系型数据库API参考》。

1.2.8 DDS 数据库加密

简介

当启用加密功能后，用户创建数据库实例和扩容磁盘时，磁盘数据会在服务端加密成密文后存储。用户下载加密对象时，存储的密文会先在服务端解密为明文，再提供给用户。

约束条件

- 已通过统一身份认证服务添加华为云文档数据库服务所在区域的KMS Administrator权限。权限添加方法请参见《统一身份认证服务用户指南》的“如何管理用户组并授权？”章节。
- 如果用户需要使用自定义密钥加密上传对象，则需要先通过数据加密服务创建密钥。使用数据加密服务创建密钥详情请参见**创建密钥**。
- 实例创建成功后，不可修改磁盘加密状态，且无法更改密钥。存放在对象存储服务上的备份数据不会被加密。
- 华为云文档数据库服务实例创建成功后，请勿禁用或删除正在使用的密钥，否则会导致数据库不可用，数据无法恢复。
- 选择磁盘加密的实例，新扩容的磁盘空间依然会使用原加密密钥进行加密。

使用 KMS 加密数据库实例（控制台）

用户在通过文档数据库服务（Document Database Service，DDS）购买数据库实例时，可以选择“磁盘加密”，使用KMS提供的密钥来加密数据库实例的磁盘，更多信息请参见**购买实例**。

图 1-17 DDS 服务端加密

使用 KMS 加密数据库实例（API）

用户也可以通过调用DDS API接口购买加密数据库实例，详情请参考《文档数据库服务API参考》。

1.2.9 DWS 数据库加密

简介

在DWS中，您可以为集群启用数据库加密，以保护静态数据。当您为集群启用加密时，该集群及其快照的数据都会得到加密处理。您可以在创建集群时启用加密。加密是集群的一项可选且不可变的设置。要从未加密的集群更改为加密集群(或反之)，必须从现有集群导出数据，然后在已启用数据库加密的新集群中重新导入这些数据。

如果希望加密，可以在集群创建时启用加密。加密是DWS集群中的一项可选设置，建议您为包含敏感数据的集群启用该设置。

哪些用户有权限使用 DWS 数据库加密

- 安全管理员（拥有“Security Administrator”权限）可以直接授权DWS访问KMS，使用加密功能。
- 普通用户（没有“Security Administrator”权限）使用加密功能时，根据该普通用户是否为当前区域或者项目内第一个使用加密特性的用户，作如下区分：
  - 是，即该普通用户是当前区域或者项目内第一个使用加密功能的，需先联系安全管理员进行授权，然后再使用加密功能。
  - 否，即区域或者项目内的其他用户已经使用过加密功能，该普通用户可以直接使用加密功能。

对于一个租户而言，同一个区域内只要安全管理员成功授权DWS访问KMS，则该区域内的普通用户都可以直接使用加密功能。

如果当前区域内存在多个项目，则每个项目下都需要安全管理员执行授权操作。

使用 KMS 加密 DWS 数据库流程

当选择KMS对DWS进行密钥管理时，加密密钥层次结构有三层。按层次结构顺序排列，这些密钥为主密钥（CMK）、集群加密密钥 (CEK)、数据库加密密钥 (DEK)。

主密钥用于给CEK加密，保存在KMS中。

CEK用于加密DEK，CEK明文保存在DWS集群内存中，密文保存在DWS服务中。

DEK用于加密数据库中的数据，DEK明文保存在DWS集群内存中，密文保存在DWS服务中。

密钥使用流程如下：

1. 用户选择主密钥。
2. DWS随机生成CEK和DEK明文。
3. KMS使用用户所选的主密钥加密CEK明文并将加密后的CEK密文导入到DWS服务中。
4. DWS使用CEK明文加密DEK明文并将加密后的DEK密文保存到DWS服务中。
5. DWS将DEK明文传递到集群中并加载到集群内存中。

当该集群重启时，集群会自动通过API向DWS请求DEK明文，DWS将CEK、DEK密文加载到集群内存中，再调用KMS使用主密钥CMK来解密CEK，并加载到集群内存中，最后用CEK明文解密DEK，并加载到集群内存中，返回给集群。

使用 KMS 加密 DWS 数据库（控制台）

步骤1 在DWS管理控制台，单击“购买数据仓库集群”。

步骤2 打开“加密数据库”开关。

1. 在“高级配置”中选择“自定义”，出现“加密数据库”开关。

图 1-18 加密数据库

表示打开“加密数据库”开关，启用数据库加密。每个区域的每个项目首次启用数据库加密时，系统会弹出一个“创建委托”的对话框，单击“是”创建委托以授权DWS访问KMS，如果单击“否”将不会启用加密功能。然后在“密钥名称”的下拉列表中选择已创建的密钥。如果没有密钥，可以登录KMS服务进行创建，详细操作请参见**《数据加密服务用户指南》**。

表示关闭“加密数据库”开关，不启用数据库加密。

2. 创建委托。

   打开“加密数据库”开关，如果当前未授权DWS访问KMS，则会弹出“创建委托”对话框，单击“是”，授权DWS访问KMS，当授权成功后，DWS可以获取KMS密钥用来加解密云硬盘。

   说明

   当您需要使用数据库加密功能时，需要授权DWS访问KMS。如果您有授权资格，则可直接授权。如果权限不足，需先联系拥有“Security Administrator”权限的用户授权，然后再重新操作。

3. 加密设置。

   打开“加密数据库”开关，如果已经授权，会弹出“加密设置”对话框。

图 1-19 加密设置

密钥名称是密钥的标识，您可以通过“密钥名称”下拉框选择需要使用的密钥。

步骤3 根据界面提示，配置其他基本信息。详细参数说明请参见**创建集群**。

----结束

1.3 使用加密 SDK 进行本地文件加解密

文件加密，指通过指定算法对文本信息进行加密，使其无法被窃取或修改。

加密SDK（Encryption SDK）是一个客户端密码库，提供了数据的加解密、文件流加解密等功能，旨在帮助客户专注于应用程序的核心功能，而不用关心数据加密和解密的实现过程，用户只需调用加解密接口即可轻松实现海量数据加解密。

说明

更多详情请访问：**详情参考**。

应用场景

对大型文件、图片等数据通过HTTPS请求到KMS服务进行保护时会消耗大量网络资源，降低加密效率。

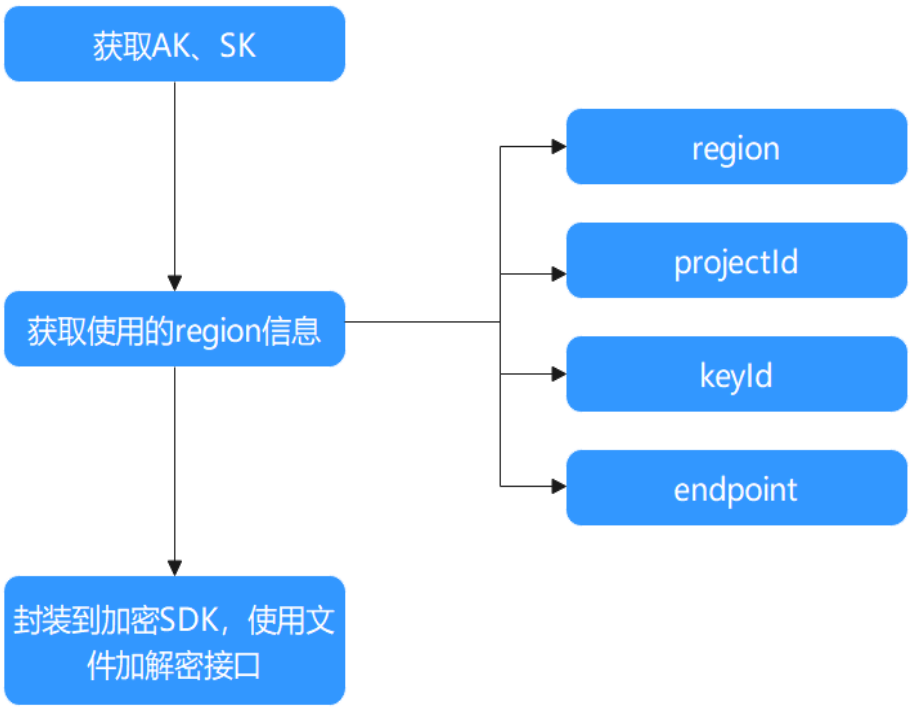
解决方案

加密SDK通过文件流分段信封加密的原理进行加密。

数据加密通过KMS生成的数据密钥在SDK内部进行，内存中分段加解密文件无需将数据通过网络传输后再进行加解密，保证了文件加密的安全性和正确性。

对于大型文件，SDK执行加密过程中，将文件分段读取到内存中，通过加解密写到目标文件后再进行下一段文件读取和加解密，直至完成整个文件的加解密。

操作流程



操作步骤

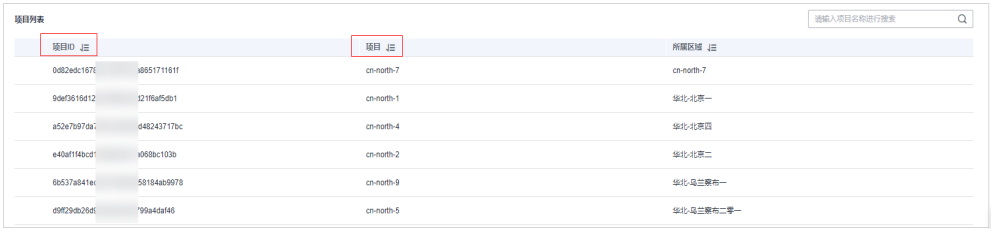
步骤1 获取AK/SK：

- ACCESS_KEY: 华为账号Access Key，获取方式请参见**获取AK/SK**。
- SECRET_ACCESS_KEY: 华为账号Secret Access Key，获取方式请参见**获取AK/SK**。
- 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
- 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。

步骤2 获取region相关信息。

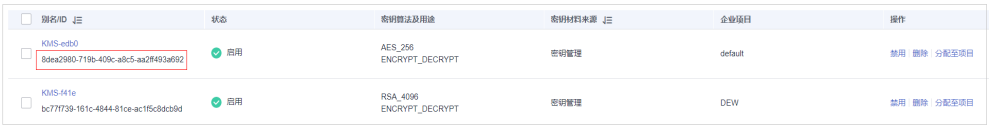
1. **登录管理控制台**。
2. 鼠标移动至右上方的用户名，在下拉列表中选择“我的凭证”。
3. 获取“项目ID”(PROJECT_ID)和“项目”(REGION)。

图 1-20 获取项目 ID 和项目



4. 单击页面左侧 ☰，选择“安全与合规 > 数据加密服务”，默认进入“密钥管理”界面。
5. 获取当前Region需要使用的主密钥ID（KEYID）。

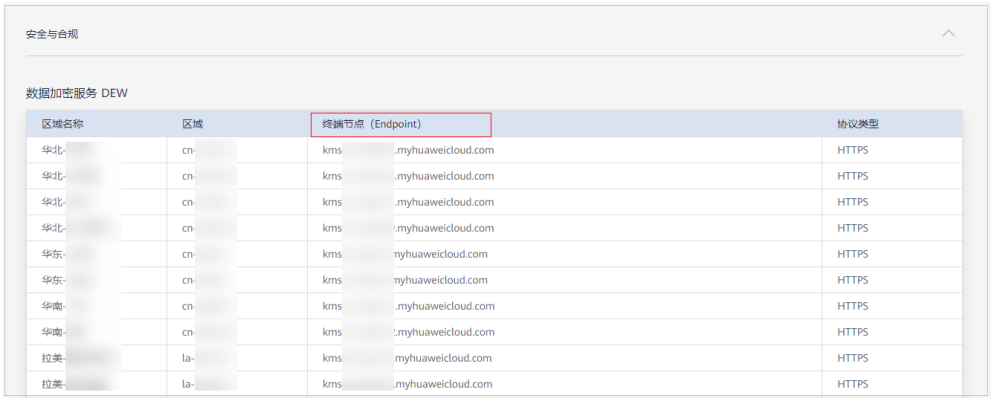
图 1-21 获取主密钥 ID



6. 获取当前Region需要使用的终端节点（ENDPOINT）。

   终端节点（Endpoint）即调用API的**请求地址**，不同服务不同区域的终端节点不同，您可以从**地区和终端节点**中查询服务的终端节点。

图 1-22 获取终端节点



步骤3 文件加解密。

```
public class KmsEncryptFileExample {

    private static final String ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_AK");
    private static final String SECRET_ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_SK");
    private static final String PROJECT_ID = "<projectId>";
    private static final String REGION = "<region>";
    private static final String KEYID = "<keyId>";
    public static final String ENDPOINT = "<endpoint>";

    public static void main(String[] args) throws IOException {
        // 源文件路径
        String encryptFileInPutPath = args[0];
        // 加密后密文文件路径
        String encryptFileOutPutPath = args[1];
        // 解密后文件路径
        String decryptFileOutPutPath = args[2];
        // 加密上下文
        Map<String, String> encryptContextMap = new HashMap<>();
        encryptContextMap.put("encryption", "context");
        encryptContextMap.put("simple", "test");
        encryptContextMap.put("caching", "encrypt");
        // 构建加密配置
        HuaweiConfig config = HuaweiConfig.builder().buildSk(SECRET_ACCESS_KEY)
                .buildAk(ACCESS_KEY)
                .buildKmsConfig(Collections.singletonList(new KMSConfig(REGION, KEYID, PROJECT_ID,
ENDPOINT)))
                .buildCryptoAlgorithm(CryptoAlgorithm.AES_256_GCM_NOPADDING)
                .build();
        HuaweiCrypto huaweiCrypto = new HuaweiCrypto(config);
        // 设置密钥环
        huaweiCrypto.withKeyring(new
KmsKeyringFactory().getKeyring(KeyringTypeEnum.KMS_MULTI_REGION.getType()));
        // 加密文件
        encryptFile(encryptContextMap, huaweiCrypto, encryptFileInPutPath, encryptFileOutPutPath);
        // 解密文件
        decryptFile(huaweiCrypto, encryptFileOutPutPath, decryptFileOutPutPath);
    }

    private static void encryptFile(Map<String, String> encryptContextMap, HuaweiCrypto huaweiCrypto,
                                    String encryptFileInPutPath, String encryptFileOutPutPath) throws IOException {
        // fileInputStream 加密后文件对应的输入流
        FileInputStream fileInputStream = new FileInputStream(encryptFileInPutPath);
        // fileOutputStream 源文件对应的输出流
        FileOutputStream fileOutputStream = new FileOutputStream(encryptFileOutPutPath);
        // 加密
        huaweiCrypto.encrypt(fileInputStream, fileOutputStream, encryptContextMap);
        fileInputStream.close();
        fileOutputStream.close();
    }
```

```
    private static void decryptFile(HuaweiCrypto huaweiCrypto, String decryptFileInPutPath, String
decryptFileOutPutPath) throws IOException {
        // in 源文件对应的输入流
        FileInputStream fileInputStream = new FileInputStream(decryptFileInPutPath);
        // out 加密后文件对应的输出流
        FileOutputStream fileOutputStream = new FileOutputStream(decryptFileOutPutPath);
        // 解密
        huaweiCrypto.decrypt(fileInputStream, fileOutputStream);
        fileInputStream.close();
        fileOutputStream.close();
    }
}
```

----结束

1.4 跨 Region 容灾加解密

应用场景

当单Region加解密出现服务侧故障时，无法再对数据进行加解密操作，用户可以通过密钥管理服务（Key Management Service，KMS）实现跨Region容灾加解密，保证业务不中断。

解决方案

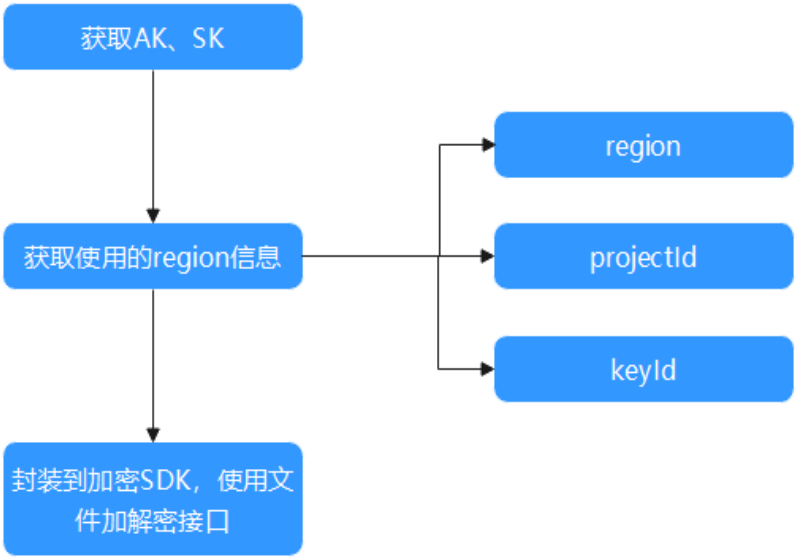
当一个或多个Region的KMS服务出现故障，只要密钥环中存在一个可用的Region即可完成加解密。

跨Region密钥支持同时使用多个Region的主密钥对一个数据加密，生成唯一的数据密文。解密使用的密钥环，可以使用加密数据时相同的密钥环，也可以使用不同的密钥环，此时解密密钥环只需包含一个或多个加密密钥环中使用的可用的用户主密钥。

说明

更多详情请访问：**详情参考**。

操作流程



操作步骤

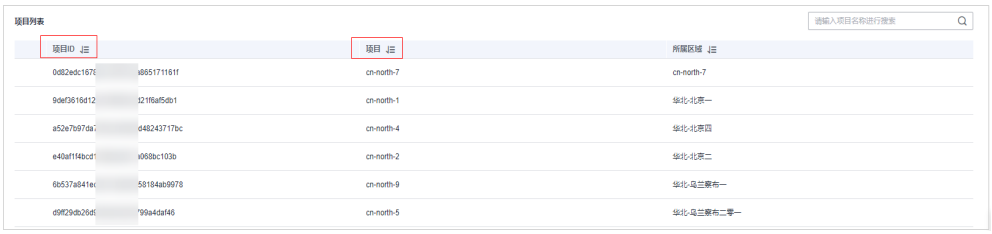
步骤1 获取AK/SK：

- ACCESS_KEY: 华为账号Access Key，获取方式请参见**获取AK/SK**。
- SECRET_ACCESS_KEY: 华为账号Secret Access Key，获取方式请参见**获取AK/SK**。
- 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
- 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。

步骤2 获取region相关信息。

1. **登录管理控制台**。
2. 鼠标移动至右上方的用户名，在下拉列表中选择“我的凭证”。
3. 获取“项目ID”(PROJECT_ID)和“项目”(REGION)。

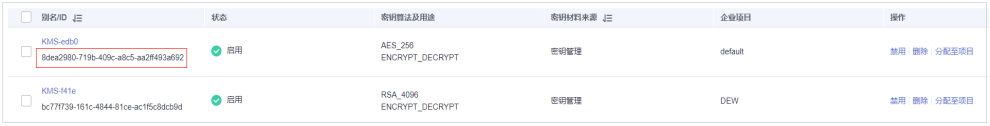
图 1-23 获取项目 ID 和项目



4. 单击页面左侧 ☰，选择“安全与合规 > 数据加密服务”，默认进入“密钥管理”界面。

5. 获取当前Region需要使用的主密钥ID（KEYID）。

图 1-24 获取主密钥 ID



步骤3 使用密钥环加解密。

```
public class KmsEncryptionExample {
    private static final String ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_AK");
    private static final String SECRET_ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_SK");

    private static final String PROJECT_ID_1 = "<projectId1>";
    private static final String REGION_1 = "<region1>";
    private static final String KEYID_1 = "<keyId1>";

    public static final String PROJECT_ID_2 = "<projectId2>";
    public static final String REGION_2 = "<region2>";
    public static final String KEYID_2 = "<keyId2>";

    // 需要加密的数据
    private static final String PLAIN_TEXT = "Hello World!";

    public static void main(String[] args) {
        // 主密钥列表
        List<KMSConfig> kmsConfigList = new ArrayList<>();
        kmsConfigList.add(new KMSConfig(REGION_1, KEYID_1, PROJECT_ID_1));
        kmsConfigList.add(new KMSConfig(REGION_2, KEYID_2, PROJECT_ID_2));
        // 构建加密相关信息
        HuaweiConfig multiConfig = HuaweiConfig.builder().buildSk(SECRET_ACCESS_KEY)
                .buildAk(ACCESS_KEY)
                .buildKmsConfig(kmsConfigList)
                .buildCryptoAlgorithm(CryptoAlgorithm.AES_256_GCM_NOPADDING)
                .build();
        // 选择密钥环
        KMSKeyring keyring = new
KmsKeyringFactory().getKeyring(KeyringTypeEnum.KMS_MULTI_REGION.getType());
        HuaweiCrypto huaweiCrypto = new HuaweiCrypto(multiConfig).withKeyring(keyring);
        // 加密上下文
        Map<String, String> encryptContextMap = new HashMap<>();
        encryptContextMap.put("key", "value");
        encryptContextMap.put("context", "encrypt");
        // 加密
        CryptoResult<byte[]> encryptResult = huaweiCrypto.encrypt(new EncryptRequest(encryptContextMap,
PLAIN_TEXT.getBytes(StandardCharsets.UTF_8)));
        // 解密
        CryptoResult<byte[]> decryptResult = huaweiCrypto.decrypt(encryptResult.getResult());
        Assert.assertEquals(PLAIN_TEXT, new String(decryptResult.getResult()));
    }
}
```

----结束

1.5 如何使用 KMS 对文件进行完整性保护

应用场景

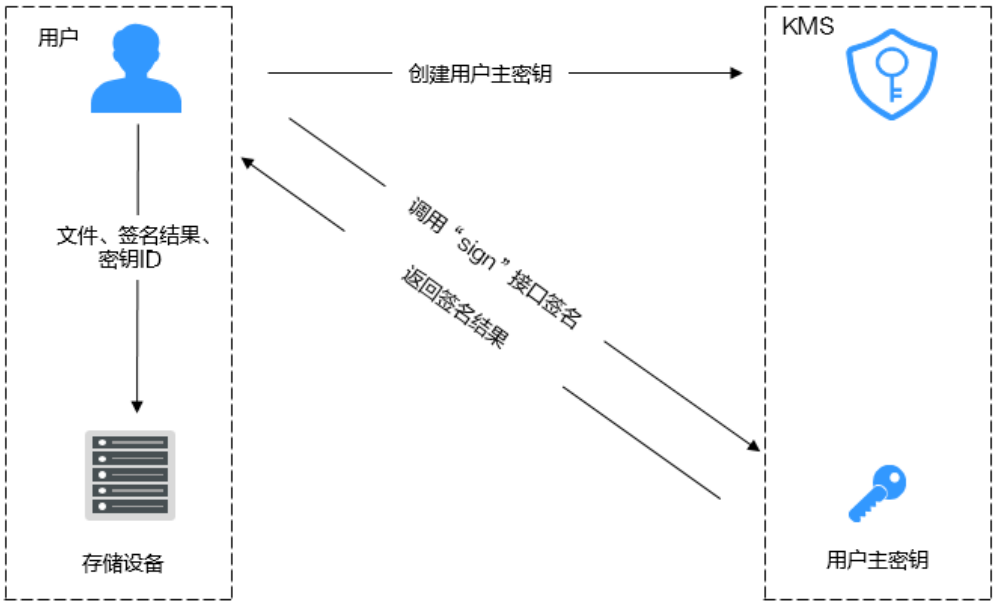
当有大量文件（例如：镜像、电子保单或者重要文件等）需要在传输或者存储时确保安全性，用户可以使用KMS对文件摘要进行签名，再次使用时可以重新计算摘要进行验签。确保文件在传输或者存储过程中没有被篡改。

解决方案

用户需要在KMS中创建一个用户主密钥。

用户计算文件的摘要，调用KMS的“sign”接口对摘要进行签名。用户得到摘要的签名结果。将摘要签名结果和密钥ID与文件一同传输或者存储。签名流程如图 签名流程所示。

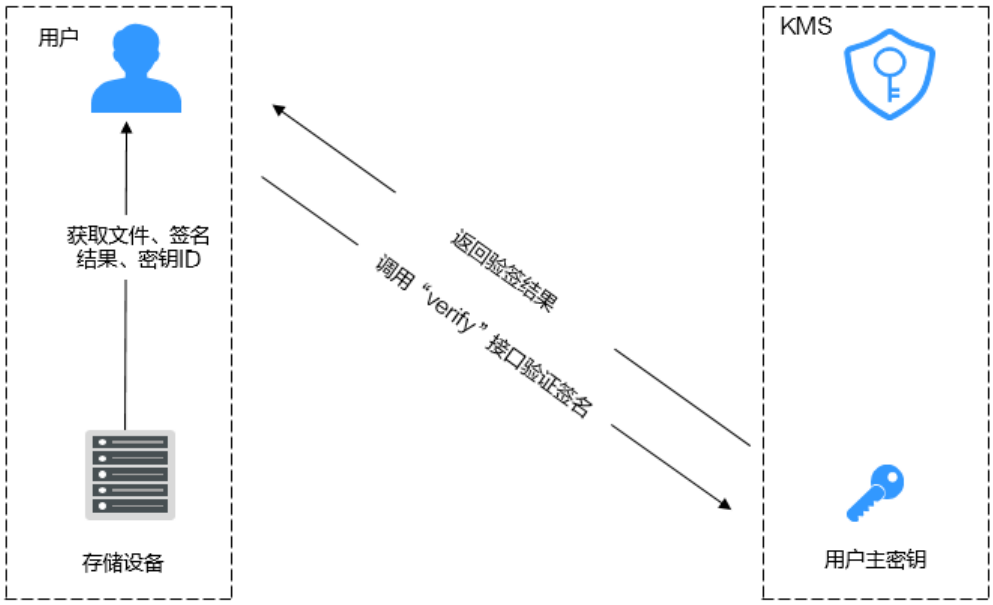
图 1-25 签名流程



用户需要使用文件时，先进行完整性校验，确保文件未被篡改。

用户重新计算文件的摘要，连同签名值调用KMS的“verify”接口对摘要进行验签。用户得到验签结果。如果能正常验签，则表明文件未被篡改。验签流程如图 验签流程所示。

图 1-26 验签流程



操作步骤

步骤1 获取AK/SK：

- ACCESS_KEY: 华为账号Access Key，获取方式请参见**获取AK/SK**。
- SECRET_ACCESS_KEY: 华为账号Secret Access Key，获取方式请参见**获取AK/SK**。
- 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
- 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。

步骤2 获取region相关信息：

- **登录管理控制台**。

步骤3 使用KMS对文件进行签名并验签。

```
public class FileStreamSignVerifyExample {

    /**
     * 基础认证信息：
     * - ACCESS_KEY: 华为云账号Access Key
     * - SECRET_ACCESS_KEY: 华为云账号Secret Access Key, 敏感信息，建议密文存储
     * - IAM_ENDPOINT: 华为云IAM服务访问终端地址,详情见终端节点
     * - KMS_REGION_ID: 华为云KMS支持的地域,详情见KMS地域KMS支持地域
     * - KMS_ENDPOINT: 华为云KMS服务访问终端地址,详情见终端地址终端地址
     */
    private static final String ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_AK");
    private static final String SECRET_ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_SK");
    private static final String IAM_ENDPOINT = "https://<IamEndpoint>";
    private static final String KMS_REGION_ID = "<RegionId>";
    private static final String KMS_ENDPOINT = "https://<KmsEndpoint>";

    public static void main(String[] args) {
        // 用户主密钥ID,需要选择密钥及用途包含SIGN_VERIFY的密钥
        final String keyId = args[0];
```



```
        signAndVerifyFile(keyId);
    }

    /**
     * 使用KMS对文件进行签名和验签
     *
     * @param keyId 用户主密钥ID
     */
    static void signAndVerifyFile(String keyId) {
        // 1.准备访问华为云的认证信息
        final BasicCredentials auth = new BasicCredentials()
                .withIamEndpoint(IAM_ENDPOINT).withAk(ACCESS_KEY).withSk(SECRET_ACCESS_KEY);

        // 2.初始化SDK，传入认证信息及KMS访问终端地址
        final KmsClient kmsClient = KmsClient.newBuilder()
                .withRegion(new Region(KMS_REGION_ID, KMS_ENDPOINT)).withCredential(auth).build();

        // 3.准备待签名的文件
        // inFile 待签名的文件
        final File inFile = new File("FirstSignFile.iso");
        final String fileSha256Sum = getFileSha256Sum(inFile);

        // 4.计算摘要，需要根据密钥类型，选择合适的签名算法
        final SignRequest signRequest = new SignRequest().withBody(
                new
SignRequestBody().withKeyId(keyId).withSigningAlgorithm(SignRequestBody.SigningAlgorithmEnum.RSASSA
_PSS_SHA_256)
                        .withMessageType(SignRequestBody.MessageTypeEnum.DIGEST).withMessage(fileSha256Su
m));

        final SignResponse signResponse = kmsClient.sign(signRequest);

        // 5.验证摘要
        final ValidateSignatureRequest validateSignatureRequest = new ValidateSignatureRequest().withBody(
                new
VerifyRequestBody().withKeyId(keyId).withMessage(fileSha256Sum).withSignature(signResponse.getSignatur
e())
                        .withSigningAlgorithm(VerifyRequestBody.SigningAlgorithmEnum.RSASSA_PSS_SHA_256)
                        .withMessageType(VerifyRequestBody.MessageTypeEnum.DIGEST));
        final ValidateSignatureResponse validateSignatureResponse =
kmsClient.validateSignature(validateSignatureRequest);

        // 6.比对摘要结果
        assert validateSignatureResponse.getSignatureValid().equalsIgnoreCase("true");

    }

    /**
     * 计算文件SHA256摘要
     *
     * @param file 文件
     * @return SHA256摘要, Base64格式
     */
    static String getFileSha256Sum(File file) {
        int length;
        MessageDigest sha256;
        byte[] buffer = new byte[1024];
        try {
            sha256 = MessageDigest.getInstance("SHA-256");
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e.getMessage());
        }
        try (FileInputStream inputStream = new FileInputStream(file)) {
            while ((length = inputStream.read(buffer)) != -1) {
                sha256.update(buffer, 0, length);
            }
            return Base64.getEncoder().encodeToString(sha256.digest());
        } catch (IOException e) {
            throw new RuntimeException(e.getMessage());
```

```
    }
  }

}
```

----结束

2 凭据管理

2.1 如何使用凭据管理服务替换硬编码的数据库账号密码

您在日常访问应用程序的过程中，通常会嵌入凭据直接访问程序。在需要更新凭据时，您除了创建新的凭据以外，还需要执行一些其他操作。您还需要花费一些时间更新应用程序以使用新的凭据。如果您有多个应用程序同一凭据，而您错过更新其中一个，则该应用程序就无法使用凭据登录。

因此使用方便有效且安全性高的凭据管理工具十分关键。

华为云凭据管理服务（Cloud Secret Management Service，CSMS）可以帮助您拥有以下优势：

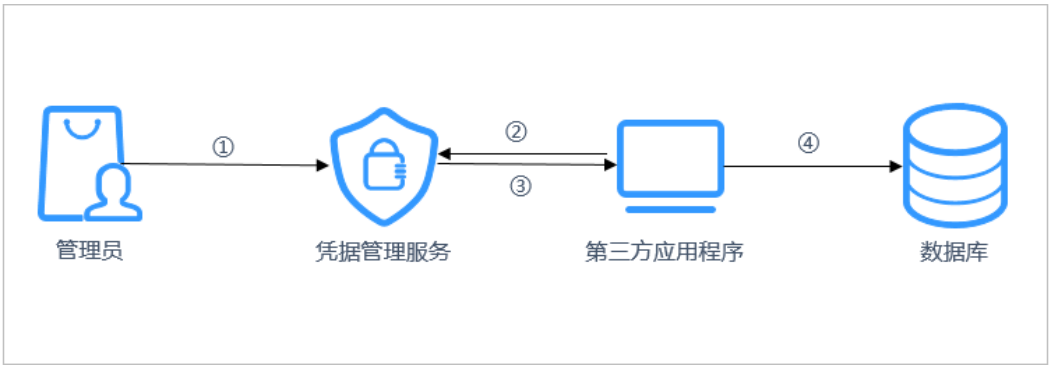
- 通过凭据托管功能，降低通过硬编码方式带来的凭据泄露风险，提高数据及资产的安全性。
- 减少对业务的影响：当您进行**人工轮换**的方式更新凭据时，您的业务不会受到影响。
- 提供安全的SDK接入方式，动态调用您的凭据。
- 凭据存储类型多样化。您不仅可以存储业务相关的账号密码，也可以存储业务数据库详细信息，包括但不限于：数据库名称、IP地址和端口等。

使用凭据登录数据库

下文为您介绍如何创建凭据，并且通过API调用凭据来登录到您的数据库。

您首先需要确保您的账号拥有KMS Administrator或者KMS CMKFullAccess权限，详情见**DEW权限管理**。

图 2-1 凭据登录流程



流程说明如下：

步骤1 您首先需要在凭据管理服务中使用**控制台**或者**API**创建一个凭据，用来存储数据库的相关信息（例如：数据库地址、端口、密码）。

步骤2 当您使用应用程序访问数据库时，凭据管理服务会去查询**步骤1**所创建的凭据存储的内容。

步骤3 凭据管理服务检索并解密凭据密文，将凭据中保存的信息通过凭据管理API安全地返回到应用程序中。

步骤4 应用程序获取到解密后的凭据明文信息，使用这些安全的信息访问数据库。

----结束

创建凭据和查询凭据 API

您可以调用以下API，通过API创建凭据保存相关内容并且查询相关凭据信息。

API名称	说明
创建凭据	创建新的凭据，并且将凭据值存入凭据的初始版本
查询凭据	查询指定凭据的信息

通过 API 接口创建凭据和查询凭据

1. 请准备基础认证信息：
   - ACCESS_KEY: 华为账号Access Key
   - SECRET_ACCESS_KEY: 华为账号Secret Access Key
   - PROJECT_ID: 华为云局点项目ID，请参见**华为云局点项目**。
   - CSMS_ENDPOINT: 华为云CSMS服务访问终端地址，请参见**终端节点**。
   - 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
   - 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。

2. 创建和查询凭据信息：
   凭据名称：secretName
   凭据值：secretString
   凭据版本值：LATEST_SECRET
   凭据版本：versionId

```
import com.huaweicloud.sdk.core.auth.BasicCredentials;
import com.huaweicloud.sdk.csms.v1.CsmsClient;
import com.huaweicloud.sdk.csms.v1.model.CreateSecretRequest;
import com.huaweicloud.sdk.csms.v1.model.CreateSecretRequestBody;
import com.huaweicloud.sdk.csms.v1.model.CreateSecretResponse;
import com.huaweicloud.sdk.csms.v1.model.ShowSecretVersionRequest;
import com.huaweicloud.sdk.csms.v1.model.ShowSecretVersionResponse;

public class CsmsCreateSecretExample {
    /**
     * 基础认证信息：
     * - ACCESS_KEY: 华为账号Access Key
     * - SECRET_ACCESS_KEY: 华为账号Secret Access Key
     * - PROJECT_ID: 华为云局点项目ID 详情见https://support.huaweicloud.com/productdesc-iam/
iam_01_0023.html
     * - CSMS_ENDPOINT: 华为云CSMS服务访问终端地址 详情见https://support.huaweicloud.com/api-dew/
dew_02_0052.html
     * - 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解
密，确保安全。
     * - 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量
HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
     */
    private static final String ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_AK");
    private static final String SECRET_ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_SK");
    private static final String PROJECT_ID = "<ProjectID>";
    private static final String CSMS_ENDPOINT = "<CsmsEndpoint>";

    // 用来查询凭据最新版本详情的版本Id
    private static final String LATEST_SECRET = "latest";

    public static void main(String[] args) {
        String secretName = args[0];
        String secretString = args[1];

        // 创建凭据
        createSecret(secretName, secretString);

        // 通过凭据版本值“latest”或者版本值“v1"查询到新创建的凭据内容
        ShowSecretVersionResponse latestVersion = showSecretVersion(secretName, LATEST_SECRET);
        ShowSecretVersionResponse firstVersion = showSecretVersion(secretName, "v1");

        assert latestVersion.equals(firstVersion);
        assert latestVersion.getVersion().getSecretString().equalsIgnoreCase(secretString);
    }

    /**
     * 创建凭据
     * @param secretName
     * @param secretString
     */
    private static void createSecret(String secretName, String secretString) {
        CreateSecretRequest secret = new CreateSecretRequest().withBody(
                new CreateSecretRequestBody().withName(secretName).withSecretString(secretString));

        CsmsClient csmsClient = getCsmsClient();

        CreateSecretResponse createdSecret = csmsClient.createSecret(secret);

        System.out.printf("Created secret success, secret detail:%s", createdSecret);
    }
    /**
```



```
    * 根据凭据版本id查询凭据版本详情
    * @param secretName
    * @param versionId
    * @return
    */
   private static ShowSecretVersionResponse showSecretVersion(String secretName, String versionId) {
       ShowSecretVersionRequest showSecretVersionRequest = new
ShowSecretVersionRequest().withSecretName(secretName)
               .withVersionId(versionId);

       CsmsClient csmsClient = getCsmsClient();

       ShowSecretVersionResponse version = csmsClient.showSecretVersion(showSecretVersionRequest);

       System.out.printf("Query secret success. version id:%s",
version.getVersion().getVersionMetadata().getId());

       return version;
   }

   /**
    * 获取CSMS服务客户端
    * @return
    */
   private static CsmsClient getCsmsClient() {
       BasicCredentials auth = new BasicCredentials()
               .withAk(ACCESS_KEY)
               .withSk(SECRET_ACCESS_KEY)
               .withProjectId(PROJECT_ID);

       return CsmsClient.newBuilder().withCredential(auth).withEndpoint(CSMS_ENDPOINT).build();
   }
}
```

使用应用程序获取数据库账号和密码

1. 获取凭据管理服务的CSMS SDK的依赖声明。

   示例如下：

   ```
   <dependency>
               <groupId>mysql</groupId>
               <artifactId>mysql-connector-java</artifactId>
               <version>XXX</version>
           </dependency>
           <dependency>
               <groupId>com.google.code.gson</groupId>
               <artifactId>gson</artifactId>
               <version>2.8.9</version>
           </dependency>
           <dependency>
               <groupId>com.huaweicloud.sdk</groupId>
               <artifactId>huaweicloud-sdk-csms</artifactId>
               <version>3.0.79</version>
           </dependency>
   ```

2. 建立数据库链接并且获取账号密码：

   示例代码如下

   ```
   import com.google.gson.Gson;
   import com.google.gson.JsonObject;
   import com.huaweicloud.sdk.csms.v1.model.ShowSecretVersionRequest;
   import com.huaweicloud.sdk.csms.v1.model.ShowSecretVersionResponse;

   import java.sql.Connection;
   import java.sql.DriverManager;
   import java.sql.SQLException;

    // 通过凭据信息获取指定的数据库账号和口令
      public static Connection getMySQLConnectionBySecret(String secretName, String jdbcUrl) throws
   ```

```
ClassNotFoundException, SQLException{
        Class.forName(MYSQL_JDBC_DRIVER);
        ShowSecretVersionResponse latestVersionValue = getCsmsClient().showSecretVersion(new
ShowSecretVersionRequest().withSecretName(secretName).withVersionId("latest"));
        String secretString = latestVersionValue.getVersion().getSecretString();
        JsonObject jsonObject = new Gson().fromJson(secretString, JsonObject.class);
        return DriverManager.getConnection(jdbcUrl, jsonObject.get("username").getAsString(),
jsonObject.get("password").getAsString());
    }
```

2.2 如何使用凭据管理服务解决 AK&SK 泄露问题

凭据管理，即云凭据管理服务（Cloud Secret Management Service，CSMS），是一种安全、可靠、简单易用的凭据托管服务。用户或应用程序通过凭据管理服务，创建、检索、更新、删除凭据，轻松实现对敏感凭据的全生命周期的统一管理，有效避免程序硬编码或明文配置等问题导致的敏感信息泄露以及权限失控带来的业务风险。

应用场景

保存应用凭据，通过临时访问的方式防止AK&SK泄露。

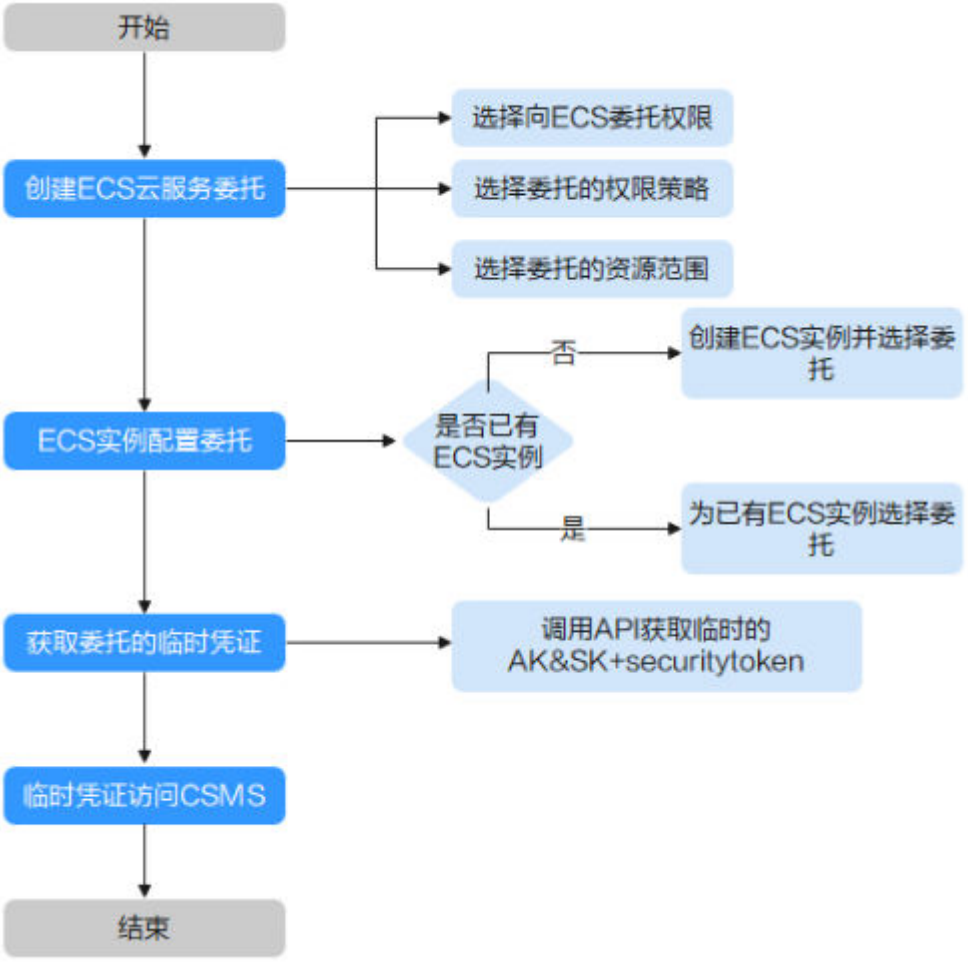
应用原理

通过统一身份认证服务（Identity and Access Management，IAM ）对弹性云服务器（Elastic Cloud Server，ECS）的委托获取临时访问密钥来保护AK&SK。

访问凭证按照时效性可分为永久凭证和临时凭证，相较于永久性访问凭证，例如用户名和密码，临时访问密钥因为有效期短且刷新频率高，所以安全性更高。因此，您可以为ECS实例授予IAM委托，使ECS实例内的应用程序可以使用临时AK&SK +SecurityToken访问CSMS，不需要保存临时访问密钥，每次需要时动态获取，也可以缓存在内存里定时更新。

操作流程

图 2-2 ECS 委托操作流程



约束限制

管理员账号或拥有ECS权限的IAM用户才能为ECS实例配置委托。

操作步骤

步骤1 IAM创建ECS委托。

1. 登录管理控制台。
2. 单击页面左侧 ☰，选择“管理与监督 > 统一身份认证服务”，默认进入“用户”界面。
3. 在左侧导航树中，选择“委托”，进入“委托”页面。
4. 单击右上角的“创建委托”。
5. 在弹出的“创建委托”对话框中，填写对应参数。参数说明如表 创建委托参数说明所示。

图 2-3 创建委托

表 2-1 创建委托参数说明

参数名称	参数说明
委托名称	填写自定义的委托名称。下文以“ECS_TO_CSMS”为例。
委托类型	选择“云服务”。
云服务	选择“弹性云服务器 ECS 裸金属服务器 BMS”。
持续时间	选择持续时间，可选择“永久”、“一天”、“自定义”。
描述	（可选）填写委托信息。

6. 单击“下一步”，进入“授权”页面。
7. 单击页面右上角“新建策略”，如果已存在需使用的策略忽略此步骤。
   a. 在“新建策略”页面配置参数，参数详情如**表 新建策略参数说明**所示。

图 2-4 新建策略

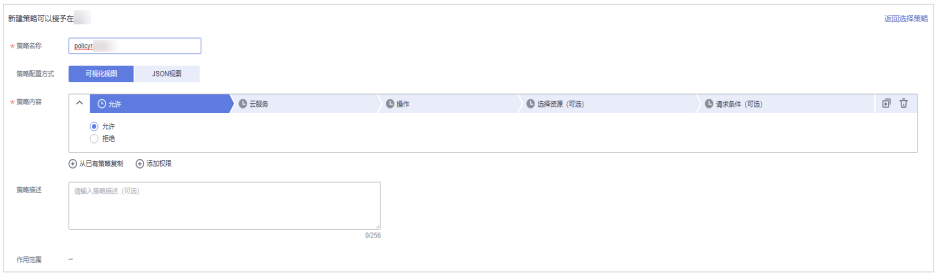


表 2-2 新建策略参数说明

参数名称	参数说明
策略名称	输入策略名称。
策略配置方式	选择“可视化视图”。
策略内容	■ 允许：选择“允许”。 ■ 云服务：选择“凭据管理服务CSMS”以及“数据加密服务KMS”。 **说明** 仅添加CSMS服务权限时，可能会导致接口调用KMS失败。 ■ 操作：选择您的读写权限。 ■ 选择资源（可选）：选择访问的资源范围。 ○ 特定资源：访问特定的凭据。 **说明** 可以选择“通过资源路径指定”访问特定的凭据，通过“添加资源路径”加入可以访问的凭据名称。 ○ 所有资源：访问所有凭据。 ■ 请求条件（可选）：单击“添加条件”，选择条件键、运算符，填写相应的值。
策略描述	（可选）输入策略描述。

8. 为委托选择策略。勾选策略后，单击“下一步”。
9. 选择授权范围方案，单击“确定”。
   - 所有资源：授权后，IAM用户可以根据权限使用账号中所有资源，包括企业项目、区域项目和全局服务资源。
   - 指定企业项目资源：授权后，用户根据权限使用已选企业项目中的资源。
   - 指定区域项目资源：授权后，用户根据权限使用已选区域项目中的资源。

步骤2 将委托（如ECS_TO_CSMS）赋予ECS实例。

- 如果ECS实例未创建，请参考**自定义购买弹性云服务器**高级配置的“委托”选择新建的委托（如ECS_TO_CSMS）。
- 如果ECS实例已创建，请按照如下流程：

a. 单击页面左侧 ☰，选择“计算 > 弹性云服务器 ECS”，进入“弹性云服务器”界面。

b. 单击需要配置委托的ECS实例的“名称”，进入“基本信息”界面。

c. 在“管理信息”中，单击 选择委托（如ECS_TO_CSMS）。

图 2-5 选择委托

步骤3 运行在ECS实例上的应用程序调用获取委托临时凭证的API，即可获得用于访问CSMS的临时AK&SK+SecrityToken。

1. 获取临时AK&SK(Security Key 目录下)，详情请参见**元数据获取**。
   - URI

     /openstack/latest/securitykey
   - 方法

     支持 GET 请求
   - 返回数据内容：

```
{
    "credential":{
       "access": "LDHZK30XXXXXXXXXXXV",
       "secret":"gyqcdzVXXXXXXXXXXXXXXXXXXXXXXXXXXMl6",
       "securitytoken": "El9FI2C65qXXXXXXXXXXXXXXXXXXXXXXnkcaoV",
       "expires_at": "2022-07-14T12:09:24.147000Z"
            }
}
```

   说明

   - 提取返回数据中的“access”、“secret”、“securitytoken”用于访问CSMS。
   - ECS会为您自动轮换临时凭证，从而确保每次申请的临时凭证安全、有效。

2. 使用临时AK&SK+SecurityToken访问CSMS。
   - 查询凭证列表示例如下，详情请参见**凭据管理服务**。
   - 请准备基础认证信息：
     - ACCESS_KEY: 华为账号Access Key，获取方式请参见**获取AK/SK**。
     - SECRET_ACCESS_KEY: 华为账号Secret Access Key，获取方式请参见**获取AK/SK**。
     - 认证用的ak和sk直接写到代码中有很大的安全风险，通过**步骤3**获取的临时access、secret、securitytoken无需进行本地存储；在使用过程中，直接通过接口调用后访问CSMS即可。

- 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。

```
package com.huaweicloud.sdk.test;

import com.huaweicloud.sdk.core.auth.ICredential;
import com.huaweicloud.sdk.core.auth.BasicCredentials;
import com.huaweicloud.sdk.core.exception.ConnectionException;
import com.huaweicloud.sdk.core.exception.RequestTimeoutException;
import com.huaweicloud.sdk.core.exception.ServiceResponseException;
import com.huaweicloud.sdk.csms.v1.region.CsmsRegion;
import com.huaweicloud.sdk.csms.v1.*;
import com.huaweicloud.sdk.csms.v1.model.*;

public class ListSecretsSolution {
    public static void main(String[] args) {
    * 基础认证信息：
    * - ACCESS_KEY: 华为账号Access Key
    * - SECRET_ACCESS_KEY: 华为账号Secret Access Key
    * - PROJECT_ID: 华为云局点项目ID 详情见https://support.huaweicloud.com/productdesc-iam/iam_01_0023.html
    * - CSMS_ENDPOINT: 华为云CSMS服务访问终端地址 详情见https://support.huaweicloud.com/api-dew/dew_02_0052.html
    * - 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
    * - 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
    */
    private static final String ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_AK");
    private static final String SECRET_ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_SK");
    String securitytoken = "<YOUR SecurityToken>";

        String ak = System.getenv("CLOUD_SDK_AK");
        String sk = System.getenv("CLOUD_SDK_SK");

        ICredential auth = new BasicCredentials()
                .withAk(ak)
                .withSk(sk)
                .withSecurityToken(securitytoken);

        CsmsClient client = CsmsClient.newBuilder()
                .withCredential(auth)
                .withRegion(CsmsRegion.valueOf("cn-north-1"))
                .build();
        ListSecretsRequest request = new ListSecretsRequest();
        try {
            ListSecretsResponse response = client.listSecrets(request);
            System.out.println(response.toString());
        } catch (ConnectionException e) {
            e.getMessage();
        } catch (RequestTimeoutException e) {
            e.getMessage();
        } catch (ServiceResponseException e) {
            e.getMessage();
            System.out.println(e.getHttpStatusCode());
            System.out.println(e.getErrorCode());
            System.out.println(e.getErrorMsg());
        }
    }
}
```

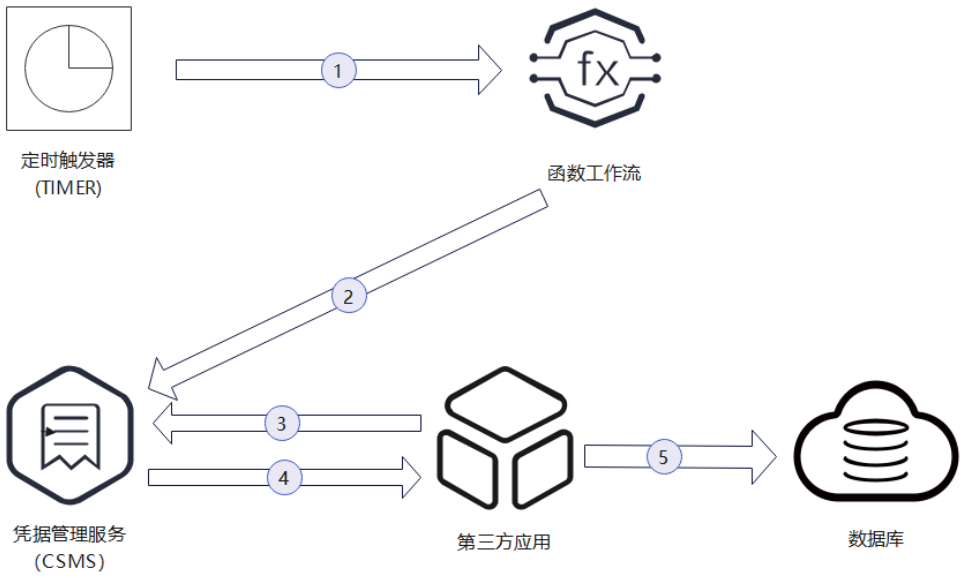
----结束

2.3 如何使用凭据管理服务自动轮转安全密码

本文介绍如何通过函数工作流和凭据管理服务，定期生成和轮转强安全密码，以满足用户安全合规的密码生成、托管、以及定期自动轮换的要求。

使用流程

图 2-6 轮转密码流程



流程说明如下：

1. 定时触发器到期后，会发布定时触发事件。
2. 函数工作流接收到事件后，会生成新的随机密码，替换凭据模板内容中的占位符，随后将替换后的内容作为新版本存入凭据中。
3. 应用程序定期通过调用API/SDK获取最新凭据版本。
4. 凭据管理服务检索并解密凭据密文，将凭据中保存的信息通过凭据管理API安全地返回到应用程序中。
5. 应用程序获取到解密后的凭据明文信息，使用新密码更新目标对象（数据库、服务器等），使新密码生效，后续使用新密码对目标对象进行访问。

约束条件

- 区域已上线凭据管理服务（CSMS）。
- 区域已上线函数工作流服务（FunctionGraph）。

创建委托

步骤1 登录管理控制台。

步骤2 单击页面左侧的 ☰，选择“管理与监管 > 统一身份认证服务”，进入“用户”界面。

步骤3 在左侧导航栏选择“委托”， 进入委托页面。

步骤4 单击“创建委托”，进入“创建委托”页面，如**图 创建委托**所示，填写参数，参数说明如**表 创建委托参数说明**所示。

图 2-7 创建委托

表 2-3 创建委托参数说明

参数	配置说明
委托名称	可自定义。
委托类型	选择“云服务”。
云服务	选择“FunctionGraph”。
持续时间	由函数使用场景决定。如函数需要长时间执行，推荐选择“永久”。
描述	自定义。

步骤5 单击“下一步”，进入委托授权界面。

步骤6 勾选需要授权函数工作流的“CSMS FullAccess”、“KMS CMKFullAccess”权限。

图 2-8 选择授权权限

步骤7 单击“下一步”，根据业务需要选择授权范围。

图 2-9 选择授权范围

步骤8 单击“确定”，委托创建成功。

----结束

创建密码轮转函数

步骤1 登录管理控制台。

步骤2 单击页面左侧的 ☰，选择“计算 > 函数工作流”，进入“函数工作流”界面。

步骤3 单击页面右上角的“创建函数”，进入创建函数页面，如图 创建函数所示，填写参数，参数说明如表 基本信息参数配置所示。

图 2-10 创建函数

表 2-4 基本信息参数配置

参数	配置说明
区域	选择函数部署的区域。
项目	选择函数部署的项目。
函数名称	自定义函数名称。
委托名称	选择**创建委托**的委托名称。
企业项目	如果您已开通企业项目，选择需要添加函数的企业项目即可。 如果您未开通企业项目，控制台默认不显示“企业项目”，可直接跳过该参数。如需开通企业项目，请参考**如何开通企业项目/企业多账号**。
运行时	选择用来编写函数的语言，当前支持使用Python进行代码配置。 **说明** 仅支持使用Python 3.6、3.9、3.10版本。

步骤4 单击“创建函数”，进入函数配置界面。

步骤5 单击“设置”页签，在左侧导航栏单击“环境变量”。单击“添加环境变量”，在变量配置行添加环境变量，参数说明如**表 基本信息参数配置**所示。添加完成后单击“保存”。

表 2-5 环境变量配置

参数	配置说明	示例
region	项目名称，比如北京四对应：cn-north-4。单击页面右上角用户名，在下拉框中选择我的凭证，即可查看region信息。	cn-north-4
secret_name	待轮转的凭据名称。 **说明** 需提前完成创建凭据操作。具体请参见**创建凭据**。	rds-functionGraph-rotate
secret_content	指定的凭据模板内容，需用{}指定，比如：{"password":"password_placeholder"}其中password_placeholder为占位符，函数执行后会将password_placeholder替换为生成的安全密码，并将替换后的内容整体存入凭据中。 **说明** 当凭据模板内容中存在多个占位符时，会生成多个密码并依次进行替换。	{"password":"password_placeholder"}
password_length	密码长度，可选范围8-128，默认为16。	16
password_format	密码格式，共支持四种，可选，默认为2： 1. 包含数字和字母； 2. 包含数字、字母和特殊字符（~!@#%^*-_=+?）； 3. 只包含数字； 4. 只包含字母。	2

步骤6 单击“代码”页签，并在编辑窗口中添加如下密码轮转函数，完成后单击“部署”。

```
# -*- coding:utf-8 -*-
import json
import secrets
import string
import requests
import inspect

def handler (event, context):
    global secret_content
    global password_length
    global password_format
    global kms_endpoint
    global region
    global secret_name
    global headers
    region = context.getUserData('region')
    secret_name = context.getUserData('secret_name')
    password_length = 16 if context.getUserData('password_length') is None else
int(context.getUserData('password_length'))
    password_format = 2 if context.getUserData('password_format') is None else
int(context.getUserData('password_format'))
```

```
    secret_content = context.getUserData('secret_content')
    headers = {
        'Content-Type': 'application/json',
        'x-Auth-Token': context.getToken()
    }
    try:
        new_content = replace_old_content(secret_content)
        # check region, if pass, return kms endpoint
        kms_endpoint = check_region(region)
        return update(context, new_content)
    except Exception as e:
        print("ERROR: %s" % e)
        return 'FAILED'

# replace "password_placeholder" in secret_content by new password
def replace_old_content(content):
    while content.find("password_placeholder") != -1:
        password = generate_password()
        while password.find("password_placeholder") != -1:
            password = generate_password()
        content = content.replace("password_placeholder", password, 1)
    return content

def generate_password():
    special_chars = "~!@#%^*-_=+?"
    # password format(default is 2):
    # 1.support letters and digits; 2.support letters, digits and special chars(~!@#%^*-_=+?);
    # 3.only support digits; 4.only support letters
    format_mapping = {
        1: string.ascii_letters + string.digits,
        2: string.ascii_letters + string.digits + special_chars,
        3: string.digits,
        4: string.ascii_letters
    }
    if password_length < 8 or password_length > 128:
        raise Exception("invalid password_length: %s, the password length range must be between 8-128." %
password_length)
    try:
        support_chars = format_mapping[password_format]
        password = ''.join([secrets.choice(support_chars) for _ in range(password_length)])
        return password
    except:
        raise Exception("invalid password_format: %s." % password_format)

def check_region(region):
    endpoint_mapping = {
        'cn-north-1': 'cn-north-1.myhuaweicloud.com',
        'cn-north-2': 'cn-north-2.myhuaweicloud.com',
        'cn-north-4': 'cn-north-4.myhuaweicloud.com',
        'cn-north-7': 'cn-north-7.myhuaweicloud.com',
        'cn-north-9': 'cn-north-9.myhuaweicloud.com',
        'cn-east-2': 'cn-east-2.myhuaweicloud.com',
        'cn-east-3': 'cn-east-3.myhuaweicloud.com',
        'cn-south-1': 'cn-south-1.myhuaweicloud.com',
        'cn-south-2': 'cn-south-2.myhuaweicloud.com',
        'cn-southwest-2': 'cn-southwest-2.myhuaweicloud.com',
        'ap-southeast-1': 'ap-southeast-1.myhuaweicloud.com',
        'ap-southeast-2': 'ap-southeast-2.myhuaweicloud.com',
        'ap-southeast-3': 'ap-southeast-3.myhuaweicloud.com',
        'af-south-1': 'af-south-1.myhuaweicloud.com',
        'la-north-2': 'la-north-2.myhuaweicloud.com',
        'la-south-2': 'la-south-2.myhuaweicloud.com',
        'na-mexico-1': 'na-mexico-1.myhuaweicloud.com',
        'sa-brazil-1': 'sa-brazil-1.myhuaweicloud.com'
    }
    try:
        endpoint = endpoint_mapping[region]
        kms_endpoint = '%s.%s' % ('kms', endpoint)
        return kms_endpoint
```

```
    except:
        raise Exception("invalid region: %s" % region)

def check_csms_resp(resp):
    if resp.status_code in (200, 201, 204):
        return
    caller_function_name = inspect.stack()[1].function
    json_resp = json.loads(resp.text)
    if 'error_msg' in json_resp:
        error_message = 'function:%s , reason: %s' % (
            caller_function_name, json_resp['error_msg'])
        raise Exception(error_message)
    error_message = 'function:%s , reason: %s' % (
        caller_function_name, resp.text)
    raise Exception(error_message)

def update(context, new_content):
    project_id = context.getProjectID()
    url = 'https://%s/v1/%s/secrets/%s/versions' % (kms_endpoint, project_id, secret_name)
    payload = {'secret_string': new_content}
    payload = json.dumps(payload)
    resp = requests.post(url, headers=headers, data=payload)
    check_csms_resp(resp)
    return 'SUCCESS'
```

----结束

调试

需对创建的函数工作流进行调试，具体请参见**在线调试**。

创建触发器

需创建触发器，具体请参见**使用定时触发器**。

查看凭据

步骤1 **登录管理控制台**。

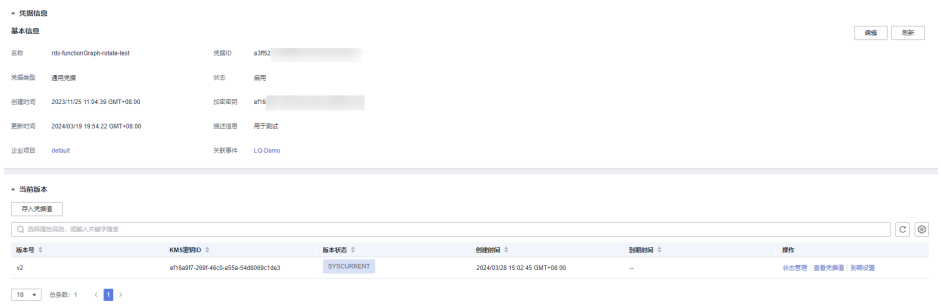
步骤2 单击页面左侧 ☰，选择“安全与合规 > 数据加密服务”，默认进入“密钥管理”界面。

步骤3 在左侧导航树中，选择“凭据管理 > 凭据列表”，进入“凭据管理”页面。

步骤4 在凭据列表查找指定凭据，即**创建密码轮转函数**中secret_name对应凭据。

步骤5 单击凭据名称，进入凭据详情页面，即可查看凭据当前版本以及历史版本等信息。

图 2-11 查看凭据信息



步骤6 在当前版本区域，单击“查看凭据值”，即可查看当前生效的安全密码。

图 2-12 查看凭据值

----结束

2.4 如何轮换凭据

2.4.1 概述

您如果长时间不更新凭据，凭据内保护的重要信息（例如：重要密码、令牌、证书、SSH密钥、API密钥等）的泄露风险也会增加，定期轮换凭据会增加所保护的明文信息安全性。

华为云为您提供了两种凭据轮换策略：

- **单用户凭据轮换**
- **双用户凭据轮换**

您可以根据您自己的实际需求，选择对应的轮换策略以进一步提高凭据的安全性。

轮换步骤

1. 管理员通过凭据管理**控制台**或**API**接口新增凭据版本，更新凭据内容。
2. 应用程序通过调用凭据管理的API获取最新凭据版本，或指定版本状态的凭据，实现凭据轮换。
3. 定期重复**步骤1**和**步骤2**实现凭据定期轮转。

2.4.2 单用户凭据轮换策略

简介

单用户凭据轮换是指一个凭据中更新一个用户所保存的信息。

这是最基础的凭据轮换策略，适用于大多数日常使用场景。

例如：

- 访问数据库。凭据轮换时不会删除数据库连接，轮换后的新连接使用新凭据。
- 访问允许用户创建一个用户账户的服务，例如以电子邮件地址作为用户名。服务通常允许用户根据需要经常更改密码，但用户无法创建其他用户或更改用户名。
- 根据需要创建的用户，称为临时用户。
- 以交互方式输入密码的用户，而不是让应用程序以编程方式从凭据管理服务中检索密码。这种类型的用户不需要更改用户名和密码。

约束条件

您首先需要确保您的账号拥有KMS Administrator或者KMS CMKFullAccess权限，详情见**DEW权限管理**。

轮换凭据的 API

您可以调用以下API，在本地进行凭据轮换。

API名称	说明
创建凭据版本	创建新的凭据版本。
查询凭据版本与凭据值	查询指定凭据版本的信息和版本中的明文凭据值。

单账号凭据轮换代码示例

1. 通过华为云控制台，创建一个凭据，详情见**创建凭据**。
2. 请准备基础认证信息。
   - ACCESS_KEY: 华为账号Access Key
   - SECRET_ACCESS_KEY: 华为账号Secret Access Key
   - PROJECT_ID: 华为云局点项目ID，请参见**华为云局点项目**。
   - CSMS_ENDPOINT: 华为云CSMS服务访问终端地址，请参见**终端节点**。
   - 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
   - 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
3. 进行单用户凭据轮换。

   示例代码中：
   - 凭据名称：华为云控制台创建的凭据名称。
   - 凭据内容：华为云控制台创建的凭据中所保存的值。
   - 凭据版本Id：华为云控制台创建凭据后自行生成凭据ID。
   - 凭据版本：LATEST_VERSION

```
import com.huaweicloud.sdk.core.auth.BasicCredentials;
import com.huaweicloud.sdk.csms.v1.CsmsClient;
```

```
import com.huaweicloud.sdk.csms.v1.model.CreateSecretVersionRequest;
import com.huaweicloud.sdk.csms.v1.model.CreateSecretVersionRequestBody;
import com.huaweicloud.sdk.csms.v1.model.CreateSecretVersionResponse;
import com.huaweicloud.sdk.csms.v1.model.ShowSecretVersionRequest;
import com.huaweicloud.sdk.csms.v1.model.ShowSecretVersionResponse;

public class CsmsSingleAccountExample {
    /**
     * 基础认证信息：
     * - ACCESS_KEY: 华为账号Access Key
     * - SECRET_ACCESS_KEY: 华为账号Secret Access Key
     * - PROJECT_ID: 华为云局点项目ID 详情见https://support.huaweicloud.com/productdesc-iam/iam_01_0023.html
     * - CSMS_ENDPOINT: 华为云CSMS服务访问终端地址 详情见https://support.huaweicloud.com/api-dew/dew_02_0052.html
     * - 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
     * - 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
     */
    private static final String ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_AK");
    private static final String SECRET_ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_SK");
    private static final String PROJECT_ID = "<ProjectID>";
    private static final String CSMS_ENDPOINT = "<CsmsEndpoint>";
    // 用来查询凭据最新版本详情的版本Id
    private static final String LATEST_VERSION = "latest";
    public static void main(String[] args) {
        String secretName = args[0];
        String secretString = args[1];
        singleAccountRotation(secretName, secretString);
    }

    /**
     * 凭据轮换-单账号模式代码示例
     *
     * @param secretName   凭据名称
     * @param secretString 凭据内容
     */
    private static void singleAccountRotation(String secretName, String secretString) {
        // 将新版凭据托管到凭据管理服务
        createNewSecretVersion(secretName, secretString);
        // 通过凭据版本“latest”查询到最新版本凭据
        ShowSecretVersionResponse secretResponseByLatest = showSecretVersionDetail(secretName, LATEST_VERSION);
        assert secretResponseByLatest.getVersion().getSecretString().equals(secretString);
    }

    /**
     * 创建凭据示例
     * 使用不附带版本状态的方式添加凭据版本，则程序默认将SYSCURRENT版本状态指向当前新创建的版本
     *
     * @param secretName   凭据名称
     * @param secretString 凭据内容
     * @return
     */
    private static CreateSecretVersionResponse createNewSecretVersion(String secretName, String secretString) {
        CsmsClient csmsClient = getCsmsClient();

        CreateSecretVersionRequest createSecretVersionRequest = new CreateSecretVersionRequest()
                .withSecretName(secretName)
                .withBody(new CreateSecretVersionRequestBody().withSecretString(secretString));

        CreateSecretVersionResponse secretVersion = csmsClient.createSecretVersion(createSecretVersionRequest);

        System.out.printf("Created new version success, version id:%s",
```

```
secretVersion.getVersionMetadata().getId());
        return secretVersion;
    }
    /**
     * 查询指定版本凭据
     *
     * @param secretName 查询凭据名称
     * @param versionId  查询凭据版本Id
     * @return
     */
    private static ShowSecretVersionResponse showSecretVersionDetail(String secretName, String
versionId) {
        ShowSecretVersionRequest showSecretVersionRequest = new
ShowSecretVersionRequest().withSecretName(secretName)
                .withVersionId(versionId);
        CsmsClient csmsClient = getCsmsClient();
        ShowSecretVersionResponse secretDetail =
csmsClient.showSecretVersion(showSecretVersionRequest);
        System.out.printf("Query latest version success. version id:%s",
secretDetail.getVersion().getVersionMetadata().getId());
        return secretDetail;
    }
    /**
     * 获取CSMS服务客户端
     *
     * @return
     */
    private static CsmsClient getCsmsClient() {
        BasicCredentials auth = new BasicCredentials()
                .withAk(ACCESS_KEY)
                .withSk(SECRET_ACCESS_KEY)
                .withProjectId(PROJECT_ID);
        return
CsmsClient.newBuilder().withCredential(auth).withEndpoint(CSMS_ENDPOINT).build();
    }
}
```

2.4.3 双用户凭据轮换策略

简介

双用户轮换策略是指在一个凭据中更新两个用户所保存的信息。

例如：您的应用程序需要保持高可用性，如果您使用单用户轮换，那么您在修改用户密码和更新凭据内容的过程中会出现访问应用失败的情况，这样您就需要使用双用户凭据轮换策略。

您需要有一个账号例如Admin有权限创建并修改user1、user2用户的账号密码，首先您创建一个凭据创建并保存user1的账号与密码，记为user1/password1。通过新增凭据版本v2创建user2，并且保存user2的账号密码，记为user2/password2。修改user1的密码后，您会需要新增凭据版本v3，记为user1/password3。比如在您轮换创建新的凭据版本v3时，应用程序会继续使用现有的凭据版本v2获取信息。一旦凭据版本v3准备就绪，轮换会切换暂存标签，以便应用程序使用凭据版本v3获取信息。

约束条件

您首先需要确保您的账号拥有KMS Administrator或者KMS CMKFullAccess权限，详情见**DEW权限管理**。

轮换凭据的 API

您可以调用以下API，在本地进行凭据轮换。

API名称	说明
创建凭据版本	创建新的凭据版本。
查询凭据版本与凭据值	查询指定凭据版本的信息和版本中的明文凭据值。
更新凭据版本状态	更新凭据的版本状态。
查询凭据版本状态	查询指定的凭据版本状态标记的版本信息。

双账号凭据轮换代码示例

1. 通过华为云控制台，使用管理员账号创建一个凭据，详情见**创建凭据**。
2. 请准备基础认证信息。
   - ACCESS_KEY: 华为账号Access Key
   - SECRET_ACCESS_KEY: 华为账号Secret Access Key
   - PROJECT_ID: 华为云局点项目ID，请参见**华为云局点项目**。
   - CSMS_ENDPOINT: 华为云CSMS服务访问终端地址，请参见**终端节点**。
   - 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
   - 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
3. 进行双用户凭据轮换。

   示例代码中：
   - 凭据名称：华为云控制台创建的凭据名称。
   - 凭据内容：华为云控制台创建的凭据中所保存的值。
   - 凭据版本Id：华为云控制台创建凭据后自行生成凭据ID。
   - 凭据版本：LATEST_VERSION

```
import com.huaweicloud.sdk.core.auth.BasicCredentials;
import com.huaweicloud.sdk.csms.v1.CsmsClient;
import com.huaweicloud.sdk.csms.v1.model.CreateSecretVersionRequest;
import com.huaweicloud.sdk.csms.v1.model.CreateSecretVersionRequestBody;
import com.huaweicloud.sdk.csms.v1.model.CreateSecretVersionResponse;
import com.huaweicloud.sdk.csms.v1.model.ListSecretStageRequest;
import com.huaweicloud.sdk.csms.v1.model.ListSecretStageResponse;
import com.huaweicloud.sdk.csms.v1.model.ShowSecretVersionRequest;
import com.huaweicloud.sdk.csms.v1.model.ShowSecretVersionResponse;
import com.huaweicloud.sdk.csms.v1.model.UpdateSecretStageRequest;
import com.huaweicloud.sdk.csms.v1.model.UpdateSecretStageRequestBody;

import java.util.Collections;
import java.util.List;

public class CsmsDualAccountExample {
    /**
     * 基础认证信息：
     * - ACCESS_KEY: 华为账号Access Key
     * - SECRET_ACCESS_KEY: 华为账号Secret Access Key
     * - PROJECT_ID: 华为云局点项目ID 详情见https://support.huaweicloud.com/productdesc-iam/
iam_01_0023.html
     * - CSMS_ENDPOINT: 华为云CSMS服务访问终端地址 详情见https://support.huaweicloud.com/
```

```
api-dew/dew_02_0052.html
    * - 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全；
    * - 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。
    */
   private static final String ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_AK");
   private static final String SECRET_ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_SK");
   private static final String PROJECT_ID = "<ProjectID>";
   private static final String CSMS_ENDPOINT = "<CsmsEndpoint>";

   // 用来查询凭据最新版本详情的版本Id
   private static final String LATEST_VERSION = "latest";
   private static final String HW_CURRENT_STAGE = "SYSCURRENT";
   private static final String HW_PENDING_STAGE = "HW_PENDING";

   public static void main(String[] args) {
      String secretName = args[0];
      String secretString = args[1];

      dualAccountRotation(secretName, secretString);
   }

   /**
    * 凭据轮换-双账号模式代码示例
    *
    * @param secretName
    * @param secretString
    */
   private static void dualAccountRotation(String secretName, String secretString) {
      // 创建带自定义版本状态的新版本凭据，假如凭据内容为服务A的账号密码。
      CreateSecretVersionResponse newSecretVersion =
createNewSecretVersionWithStage(secretName,
            secretString, Collections.singletonList(HW_PENDING_STAGE));
      String versionId = newSecretVersion.getVersionMetadata().getId();

      //  轮换前检查pending是否被修改
      ListSecretStageResponse listSecretStageResponse = showSecretStage(secretName,
HW_PENDING_STAGE);
      assert listSecretStageResponse.getStage().getVersionId().equals(versionId);

      // 在更新服务A的账号密码后，同步将新版本凭据的版本状态更新为SYSCURRENT，之前旧的凭据仍存储在服务中，已无法通过latest访问。
      updateSecretStage(secretName, versionId, HW_CURRENT_STAGE);

      // 此时通过"latest"版本状态查询最新版本凭据，完成双账号凭据轮转。
      ShowSecretVersionResponse secretResponse = showVersionDetail(secretName,
LATEST_VERSION);

      assert secretResponse.getVersion().getSecretString().equals(secretString);
   }

   /**
    * 创建凭据示例
    * 使用自定义版本状态的方式添加凭据版本，程序不会将SYSCURRENT版本状态指向新版本凭据。
    *
    * @param secretName
    * @param newSecretVersionText
    * @param stageList
    * @return
    */
   private static CreateSecretVersionResponse createNewSecretVersionWithStage(String
secretName,
                                                               String newSecretVersionText,
                                                               List<String> stageList) {
      CsmsClient csmsClient = getCsmsClient();

      // 将新版凭据托管到凭据管理服务
      CreateSecretVersionRequest createSecretVersionRequest = new
```



```
CreateSecretVersionRequest()
                .withSecretName(secretName)
                .withBody(new CreateSecretVersionRequestBody()
                        .withSecretString(newSecretVersionText)
                        .withVersionStages(stageList));

        CreateSecretVersionResponse secretVersion =
csmsClient.createSecretVersion(createSecretVersionRequest);

        System.out.printf("Created new version success, version id: %s",
secretVersion.getVersionMetadata().getId());

        return secretVersion;
    }

    /**
     * 将传入的版本状态指向指定凭据版本
     * @param secretName
     * @param versionId
     * @param newStageName
     */
    private static void updateSecretStage(String secretName, String versionId, String
newStageName) {
        UpdateSecretStageRequest updateSecretStageRequest = new
UpdateSecretStageRequest().withSecretName(secretName)
                .withStageName(newStageName).withBody(new
UpdateSecretStageRequestBody().withVersionId(versionId));

        CsmsClient csmsClient = getCsmsClient();

        csmsClient.updateSecretStage(updateSecretStageRequest);

        System.out.printf("Version stage update success. version id:%s, new stage name:%s",
versionId, newStageName);
    }

    /**
     * 查询指定版本凭据
     * @param secretName
     * @param versionId
     * @return
     */
    private static ShowSecretVersionResponse showVersionDetail(String secretName, String
versionId) {
        ShowSecretVersionRequest showSecretVersionRequest = new
ShowSecretVersionRequest().withSecretName(secretName)
                .withVersionId(versionId);

        CsmsClient csmsClient = getCsmsClient();
        ShowSecretVersionResponse secretDetail =
csmsClient.showSecretVersion(showSecretVersionRequest);

        System.out.printf("Query latest version success. version id:%s",
 secretDetail.getVersion().getVersionMetadata().getId());
        return secretDetail;
    }

    /**
     * 查询指定版本状态详情
     * @param secretName
     * @param stageName
     * @return
     */
    private static ListSecretStageResponse showSecretStage(String secretName, String
stageName) {
        ShowSecretStageRequest showSecretStageRequest = new ShowSecretStageRequest()
                .withSecretName(secretName).withStageName(stageName);
        CsmsClient csmsClient = getCsmsClient();
        return csmsClient.showSecretStage(showSecretStageRequest);
```

```
    }

    /**
     * 获取CSMS服务客户端
     *
     * @return
     */
    private static CsmsClient getCsmsClient() {
        BasicCredentials auth = new BasicCredentials()
                .withAk(ACCESS_KEY)
                .withSk(SECRET_ACCESS_KEY)
                .withProjectId(PROJECT_ID);
        return
CsmsClient.newBuilder().withCredential(auth).withEndpoint(CSMS_ENDPOINT).build();
    }
}
```

2.4.4 通过函数工作流轮转 IAM 凭证

应用场景

本文介绍使用函数工作流模板，通过凭据管理服务轮转IAM凭证（Access Key and Secret Key）。

约束条件

- 仅支持轮转IAM子账号，不支持IAM主账号轮转。
- 待轮转的IAM子账号下至少需要存在一组凭证。
- 区域已上线凭据管理服务（CSMS）。

操作步骤

创建委托

步骤1 登录管理控制台。

步骤2 单击管理控制台左上角，选择区域或项目。

步骤3 单击页面左上方的，选择“管理与监管 > 统一身份认证服务”，进入统一身份认证服务界面。

步骤4 在左侧导航栏单击“委托”，进入委托页面。在委托界面右上角单击“创建委托”，进入创建委托界面。

步骤5 在创建委托界面，设置委托信息，如图 创建委托所示，参数描述参见表 创建委托参数说明所示

图 2-13 创建委托

表 2-6 创建委托参数说明

参数	配置说明
委托名称	自定义委托名称，例如“test”。
委托类型	选择“云服务”。
云服务	选择“FunctionGraph”。
持续时间	由函数使用场景决定，如果函数需要长时间执行，推荐选择永久。
描述	自定义。

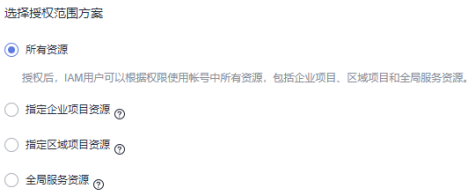
步骤6 填写完成后，单击“下一步”，进入委托授权页面。

步骤7 勾选需要授权给函数工作流的“Security Administrator”、“CSMS FullAccess”、“KMS CMKFullAccess”权限，如图 授权权限所示。

图 2-14 授权权限

步骤8 单击“下一步”，根据业务需要选择授权范围，如图 业务授权范围所示。

图 2-15 业务授权范围



步骤9 单击“确定”，委托创建成功。

----结束

创建函数模板、创建函数

步骤1 登录管理控制台。

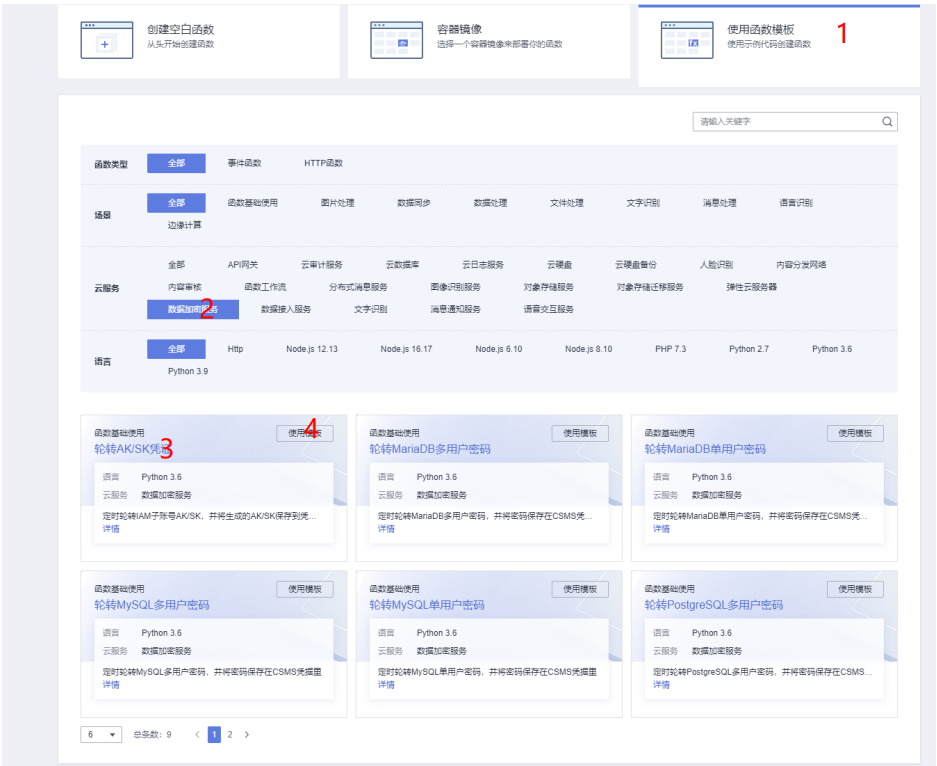
步骤2 单击管理控制台左上角，选择区域或项目。

步骤3 单击页面左上方的，选择“计算 > 函数工作流”，进入函数工作流界面。

步骤4 单击页面右上角的“创建函数”，进入创建函数界面。

步骤5 单击“使用函数模板”，在“云服务”行选择“数据加密服务”，选择“轮转AK/SK凭证”，在对应模板右上角单击“使用模板”，具体如图 创建函数所示。

图 2-16 创建函数



步骤6 配置函数基本信息，参数配置说明参见**表 基本信息参数配置**所示。

图 2-17 函数基本信息

表 2-7 基本信息参数配置

参数	配置说明
区域	选择函数部署的区域
项目	选择函数部署的项目
函数名称	自定义函数名称
委托名称	选择创建委托步骤创建的委托名称。
企业项目	如果您已开通企业项目，选择需要添加函数的企业项目即可。 如果您未开通企业项目，将无法看到企业项目的选项，可直接跳过该参数。

步骤7 配置函数环境变量，变量参数说明参见**表 环境变量参数说明**所示。

图 2-18 函数环境变量

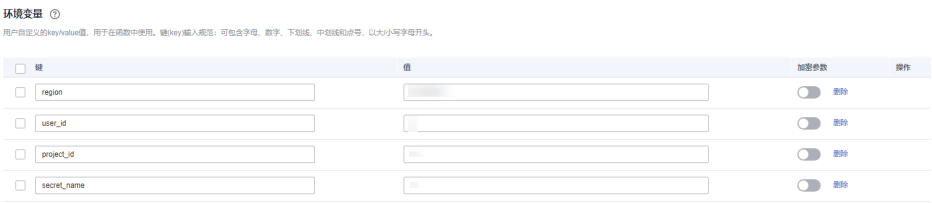


表 2-8 环境变量参数说明

参数	配置说明
region	项目。例如北京四对应：cn-north-4 **说明** 查找路径：单击“用户名称 > 我的凭证”，API凭证页面的项目，就是对应region。
user_id	待轮转的IAM用户ID。 **说明** 查找路径：单击“用户名称 > 我的凭证”，API凭证页面的IAM用户ID就是对应user_id。
project_id	项目ID。 **说明** 查找路径：单击“用户名称 > 我的凭证”，API凭证页面的项目ID，就是对应project_id。
secret_name	待创建凭据名称，不能和已有的凭据名称重复

步骤8 配置触发器变量，变量参数说明参见**表 函数触发器配置参数说明**所示。

图 2-19 触发器变量

表 2-9 函数触发器配置参数说明

参数	配置说明
定时器名称	自定义
触发规则	按需配置
是否开启	按需配置

步骤9 单击“创建函数”，完成创建操作。

----结束

调试

具体函数工作流调试请参见**在线调试**。

查看凭证

步骤1 **登录管理控制台**。

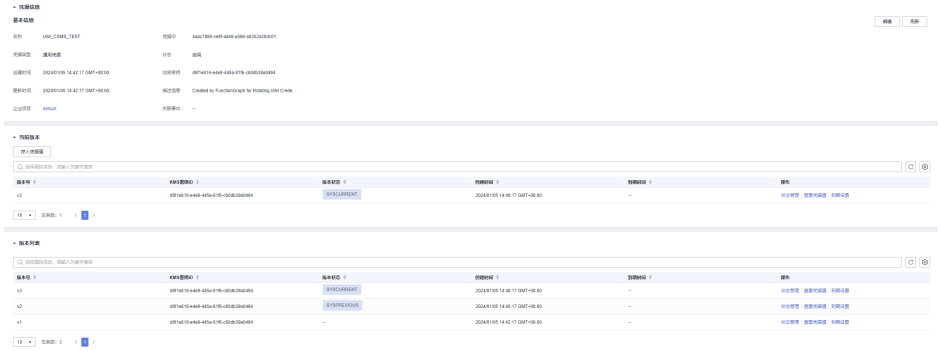
步骤2 单击管理控制台左上角，选择区域或项目。

步骤3 单击页面左侧，选择“安全与合规 > 数据加密服务”，默认进入“密钥管理”界面。

步骤4 在左侧导航树中，选择“凭据管理 > 凭据列表”，进入“凭据管理”页面。

步骤5 查找**步骤7**页面配置的指定凭据，单击凭据名称，进入凭据详情页面，查看当前生效凭据、历史凭据以及其他信息。

图 2-20 凭据详情



步骤6 选择最新凭据版本号所在行，单击“查看凭据值”，即可查看当前生效的凭据AK/SK。

----结束

2.5 云服务使用凭据管理服务

2.5.1 CCE 服务端使用凭据管理服务

简介

CCE提供了多种类型的插件，用于管理集群的扩展功能，以支持选择性扩展满足特性需求的功能。CCE服务的dew-provider插件对接了凭据管理服务，通过该插件将凭据挂载至业务Pod内，从而将敏感信息与集群环境解耦，有效避免程序硬编码或明文配置等问题导致的敏感信息泄密。

约束条件

- 支持的集群版本：v1.19+。
- 支持的集群类型：CCE Standard集群和CCE Turbo集群。

组件说明

表 2-10 dew-provider 组件

容器组件	说明	资源类型
dew-provider	dew-provider负责与凭据管理服务交互，从凭据管理服务中获取指定的凭据，并挂载到业务Pod内。	DaemonSet
secrets-store-csi-driver	secrets-store-csi-driver负责维护两个CRD资源，即SecretProviderClass（以下简称为SPC）和SecretProviderClassPodStatus（以下简称为spcPodStatus），其中SPC用于描述用户感兴趣的凭据信息（比如指定凭据的版本、凭据的名称等），由用户创建，并在业务Pod中进行引用；spcPodStatus用于跟踪Pod与凭据的绑定关系，由csi-driver自动创建，用户无需关心。一个Pod对应一个spcPodStatus，当Pod正常启动后，会生成一个与之对应的spcPodStatus；当Pod生命周期结束时，相应的spcPodStatus也会被删除。	DaemonSet

使用控制台安装插件

步骤1 在CCE服务控制台，单击集群名称进入对应集群，单击左侧导航栏的“插件中心”，在右侧找到dew-provider插件，单击“安装”。

步骤2 在安装插件页面，在参数配置栏进行参数配置。参数配置说明如**表 参数配置表**所示。

表 2-11 参数配置表

参数	参数说明
rotation_poll_interval	轮转时间间隔。单位：分钟，即m（注意不是min）。 轮转时间间隔表示向云凭据管理服务发起请求并获取最新的凭据的周期，合理的时间间隔范围为[1m, 1440m]，默认值为2m。

步骤3 单击“安装”。待插件安装完成后，选择对应的集群，然后单击左侧导航栏的“插件中心”，可在“已安装插件”页签中查看相应的插件。

步骤4 插件成功使用需使用已在数据加密服务中创建好的凭据，否则挂载失败会导致Pod无法运行。具体创建凭据操作请参见**创建通用凭据**。

步骤5 插件安装完成后即可进行使用，具体操作步骤参见**CCE密钥管理插件使用说明**。

----结束

2.5.2 通过凭据管理服务免 AKSK 硬编码访问 OBS 服务

应用场景

在代码中将认证所需的 AK 和 SK 硬编码或以明文方式存储，会带来较大的安全风险。针对初始化Java OBS SDK客户端的场景，介绍一种通过动态获取托管于凭据管理服务（CSMS）的凭证，避免硬编码 AK/SK，从而安全地创建和配置 OBS 客户端的解决方案。

实现原理

本示例基于AK和SK已托管于凭据管理服务（CSMS）的场景，演示如何通过实现OBS SDK提供的 IObsCredentialsProvider接口来创建ObsClient实例。

CsmsObsCredentialsProvider 类通过从ECS服务器自动获取临时访问凭证，进一步访问CSMS获取托管的AK和SK，并将其用作OBS客户端的访问凭证。

前提条件

1. IAM创建ECS云服务委托并获取临时凭证。具体操作请参见**IAM创建ECS委托**。
2. 已将目标AK/SK存入凭据管理服务，可以通过以下两种方式：
   a. AK/SK按照存入凭据值的方式手动托管在凭据管理服务中，具体操作可参见**存入凭据值**。
   b. 通过函数工作流轮转IAM凭据的方式自动化托管AK/SK，具体操作可参见**通过函数工作流轮转IAM凭证**。

代码示例

1. 获取CSMS SDK和OBS SDK的依赖声明

```
<dependencies>
   <dependency>
      <groupId>com.huaweicloud</groupId>
```



```
        <artifactId>esdk-obs-java-bundle</artifactId>
        <!-- obs java SDK 版本号，以3.24.9为例，其他版本替换成相应的版本号-->
        <version>3.24.9</version>
    </dependency>
    <dependency>
        <groupId>com.huaweicloud.sdk</groupId>
        <artifactId>huaweicloud-sdk-csms</artifactId>
        <!-- csms java SDK 版本号，以3.1.94为例，其他版本替换成相应的版本号-->
        <version>3.1.94</version>
    </dependency>
    <dependency>
        <groupId>org.json</groupId>
        <artifactId>json</artifactId>
        <!-- json 版本号，以20231013为例，其他版本替换成相应的版本号-->
        <version>20231013</version>
    </dependency>
</dependencies>
```

2. 创建并配置OBS客户端示例代码如下：

```
// Endpoint以北京四为例，其他地区请按实际情况填写。
String endPoint = "https://obs.cn-north-4.myhuaweicloud.com";
// Secret name为托管AKSK凭据的凭据名称
String secretName = "<Your CSMS_SECRET_NAME>";
// ECS场景从凭据管理服务获取认证信息，并初始化OBS客户端
ObsClient obsClient = new ObsClient(new CsmsObsCredentialsProvider(secretName), endPoint);
// 使用访问OBS
// 关闭obsClient
obsClient.close();
```

3. 实现OBS SDK中的IObsCredentialsProvider的示例代码如下：

```
import static com.obs.services.internal.security.LimitedTimeSecurityKey.getUtcTime;
import com.huaweicloud.sdk.core.auth.BasicCredentials;
import com.huaweicloud.sdk.core.auth.ICredential;
import com.huaweicloud.sdk.core.region.Region;
import com.huaweicloud.sdk.csms.v1.CsmsClient;
import com.huaweicloud.sdk.csms.v1.model.ShowSecretVersionRequest;
import com.huaweicloud.sdk.csms.v1.model.ShowSecretVersionResponse;
import com.obs.log.ILogger;
import com.obs.log.LoggerBuilder;
import com.obs.services.IObsCredentialsProvider;
import com.obs.services.internal.security.EcsSecurityUtils;
import com.obs.services.internal.security.LimitedTimeSecurityKey;
import com.obs.services.internal.security.SecurityKey;
import com.obs.services.internal.security.SecurityKeyBean;
import com.obs.services.internal.utils.JSONChange;
import com.obs.services.model.ISecurityKey;
import kotlin.Pair;
import org.json.JSONObject;
import java.io.IOException;
import java.util.Calendar;
import java.util.Date;
import java.util.concurrent.atomic.AtomicBoolean;

public class CsmsObsCredentialsProvider implements IObsCredentialsProvider {
    private volatile LimitedTimeSecurityKey securityKey;
    private AtomicBoolean getNewKeyFlag = new AtomicBoolean(false);
    private static final ILogger ILOG = LoggerBuilder.getLogger(CsmsObsCredentialsProvider.class);
    // default is -1, not retry
    private int maxRetryTimes = -1;
    // csms secret name
    private final String secretName;
    // region code 以北京四为例，其他地区请按实际情况填写。
    private final String regionCode = "cn-north-4";
    // Csms endpoint以北京四为例，其他地区请按实际情况填写。
    private final String csmsEndpoint = "https://kms.cn-north-4.myhuaweicloud.com";
    // Iam endpoint以北京四为例，其他地区请按实际情况填写。
    private final String iamEndpoint = "https://iam.cn-north-4.myhuaweicloud.com";
```



```
    // Rotation time轮转刷新时间。注意：该值要小于凭据轮转时间，此处以15分钟为例。
    private final String rotationTime = "15m";
    public CsmsObsCredentialsProvider(String secretName) {
        this.maxRetryTimes = 3;
        this.secretName = secretName;
    }
    public CsmsObsCredentialsProvider(int maxRetryTimes, String secretName) {
        this.maxRetryTimes = maxRetryTimes;
        this.secretName = secretName;
    }
    @Override
    public void setSecurityKey(ISecurityKey securityKey) {
        throw new UnsupportedOperationException("CsmsObsCredentialsProvider class does not support this
method");
    }
    @Override
    public ISecurityKey getSecurityKey() {
        if (getNewKeyFlag.compareAndSet(false, true)) {
            try {
                if (securityKey == null || securityKey.willSoonExpire()) {
                    refresh(false);
                } else if (securityKey.aboutToExpire()) {
                    refresh(true);
                }
            } finally {
                getNewKeyFlag.set(false);
            }
        } else {
            if (ILOG.isDebugEnabled()) {
                ILOG.debug("some other thread is refreshing.");
            }
        }
        return securityKey;
    }
    /**
     * refresh
     *
     * @param ignoreException ignore exception
     */
    private void refresh(boolean ignoreException) {
        int times = 0;
        do {
            try {
                securityKey = getNewSecurityKey();
                break;
            } catch (IOException | RuntimeException e) {
                ILOG.warn("refresh new security key failed. times : " + times + "; maxRetryTimes is : " +
maxRetryTimes
                        + "; ignoreException : " + ignoreException, e);
                if (times >= this.maxRetryTimes) {
                    ILOG.error("refresh new security key failed.", e);
                    if (!ignoreException) {
                        throw new IllegalArgumentException(e);
                    }
                }
            }
        } while (times++ < maxRetryTimes);
    }
    private LimitedTimeSecurityKey getNewSecurityKey() throws IOException, IllegalArgumentException {
        String content = EcsSecurityUtils.getSecurityKeyInfoWithDetail();
        SecurityKey securityInfo = (SecurityKey) JSONChange.jsonToObj(new SecurityKey(), content);
        if (securityInfo == null) {
            throw new IllegalArgumentException("Invalid securityKey : " + content);
        }
        SecurityKeyBean securityKeyBean = securityInfo.getBean();
        ICredential auth = new BasicCredentials().withIamEndpoint(iamEndpoint)
                .withAk(securityKeyBean.getAccessKey()).withSk(securityKeyBean.getSecretKey())
                .withSecurityToken(securityKeyBean.getSecurityToken());
        Pair<String, String> akSkFromCSMS = getAKSKFromCSMS(auth, secretName);
```

```
        // 当前时间加上轮转时间为此访问凭据的过期时间。
        Date expiryDate = getUtcTimeAfterMinuteAdd(rotationTime);
        StringBuilder strAccess = new StringBuilder();
        String accessKey = akSkFromCSMS.getFirst();
        int length = accessKey.length();
        strAccess.append(accessKey.substring(0, length / 3));
        strAccess.append("******");
        strAccess.append(accessKey.substring(2 * length / 3, length - 1));
        ILOG.warn("the AccessKey : " + strAccess.toString() + "will expiry at UTC time : " + expiryDate);
        return new LimitedTimeSecurityKey(akSkFromCSMS.getFirst(), akSkFromCSMS.getSecond(), null,
expiryDate);
    }
    /**
     * Pair
     * first is access key id
     * second is access key secret
     */
    private Pair<String, String> getAKSKFromCSMS(ICredential auth, String secretName) {
        ILOG.info("Get ak sk from csms secret name " + secretName + " begin.");
        CsmsClient client = CsmsClient.newBuilder().withCredential(auth)
                .withRegion(new Region(regionCode, csmsEndpoint)).build();
        try {
            ShowSecretVersionRequest showSecretVersionRequest = new ShowSecretVersionRequest();
            showSecretVersionRequest.withSecretName(secretName);
            showSecretVersionRequest.withVersionId("latest");
            ShowSecretVersionResponse showSecretVersionResponse =
client.showSecretVersion(showSecretVersionRequest);
            String secretString = showSecretVersionResponse.getVersion().getSecretString();
            JSONObject secretJsonObject = new JSONObject(secretString);
            return new Pair<>((String) secretJsonObject.get("access_key_id"),
                    (String) secretJsonObject.get("access_key_secret"));
        } catch (Exception e) {
            // 异常情况打印，此处可以替换成业务相关的异常，以打印日志和抛出RuntimeException为例。
            ILOG.info("error message:" + e.getMessage());
            throw new RuntimeException(e.getMessage());
        }
    }
    private static Date getUtcTimeAfterMinuteAdd(String afterMinute) {
        Calendar calendar = Calendar.getInstance();
        int offset = calendar.get(Calendar.ZONE_OFFSET);
        int dstOffset = calendar.get(Calendar.DST_OFFSET);
        // 使用Calendar类的add方法，将当前时间增加afterMinute分钟
        calendar.add(Calendar.MINUTE, convertToMinute(afterMinute));
        // 减去时区的偏移量和夏令时的偏移量，获取UTC时间
        calendar.add(Calendar.MILLISECOND, -(offset + dstOffset));
        return calendar.getTime();
    }
    private static int convertToMinute(String period) {
        String unit = period.substring(period.length() - 1);
        int time = Integer.parseInt(period.substring(0, period.length() - 1));
        switch (unit) {
            case "d":
                time = time * 24 * 60;
                break;
            case "h":
                time = time * 60;
                break;
            case "m":
                break;
            default:
                time = 0;
                break;
        }
        return time;
    }
}
```

3 通用

3.1 使用指数退避方法对 DEW 服务请求错误进行重试

应用场景

当您调用API时，收到返回的错误信息，可参照本文使用指数退避方法对请求错误进行重试。

须知

对接KMS时，必须有重试，包括不限于504、502、500、429等错误码，且推荐重试3~5次。针对502和504错误码，推荐超时时间5~8秒。不建议配置过长超时时间，否则会造成客户侧无法响应。

应用原理

当服务侧出现连续错误响应（如限流错误）时，持续的访问只会导致持续的发生冲突，使用指数退避方法，可以有效的规避该类错误。

约束条件

当前账号下有用于加密的密钥，密钥“状态”为“启用”。

指数退避代码示例

1. 请准备基础认证信息：
   - ACCESS_KEY: 华为账号Access Key，获取方式请参见**获取AK/SK**。
   - SECRET_ACCESS_KEY: 华为账号Secret Access Key，获取方式请参见**获取AK/SK**。
   - PROJECT_ID: 局点项目ID，获取方式请参见**获取项目ID**。
   - KMS_ENDPOINT: 华为云KMS服务访问终端地址，获取方式请参见**终端节点**。

- 认证用的ak和sk直接写到代码中有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
- 本示例以ak和sk保存在环境变量中来实现身份验证为例，运行本示例前请先在本地环境中设置环境变量HUAWEICLOUD_SDK_AK和HUAWEICLOUD_SDK_SK。

2. 指数退避代码

```
import com.huaweicloud.sdk.core.auth.BasicCredentials;
import com.huaweicloud.sdk.core.auth.ICredential;
import com.huaweicloud.sdk.core.exception.ClientRequestException;
import com.huaweicloud.sdk.kms.v2.model.EncryptDataRequest;
import com.huaweicloud.sdk.kms.v2.model.EncryptDataRequestBody;
import com.huaweicloud.sdk.kms.v2.KmsClient;

public class KmsEncryptExample {

    private static final String ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_AK");

    private static final String SECRET_ACCESS_KEY = System.getenv("HUAWEICLOUD_SDK_SK");

    private static final String KMS_ENDPOINT = "xxxx";

    private static final String KEY_ID = "xxxx";

    private static final String PROJECT_ID = "xxxx";


    private static KmsClient KmsClientInit() {
        ICredential auth = new BasicCredentials()
                .withAk(ACCESS_KEY)
                .withSk(SECRET_ACCESS_KEY)
                .withProjectId(PROJECT_ID);
        return KmsClient.newBuilder()
                .withCredential(auth)
                .withEndpoint(KMS_ENDPOINT)
                .build();
    }

    public static long getWaitTime(int retryCount) {
        long initialDelay = 200L;
        return (long) (Math.pow(2, retryCount) * initialDelay);
    }

    public static void encryptData(KmsClient client, String plaintext) {
        EncryptDataRequest request = new EncryptDataRequest().withBody(
                new EncryptDataRequestBody()
                        .withKeyId(KEY_ID)
                        .withPlainText(plaintext));
        client.encryptData(request);
    }

    public static void main(String[] args) {
        int maxRetryTimes = 6;
        String plaintext = "plaintext";
        String errorMsg = "The throttling threshold has been reached";

        KmsClient client = KmsClientInit();
        for (int i = 0; i < maxRetryTimes; i++) {
            try {
                encryptData(client, plaintext);
                return;
            } catch (ClientRequestException e) {
                if (e.getErrorMsg().contains(errorMsg)) {
                    try {
                        Thread.sleep(getWaitTime(i));
                    } catch (InterruptedException ex) {
                        throw new RuntimeException(ex);
                    }
```



```
        }
      }
    }
  }
}
```

