

Data Encryption Workshop

Best Practice

Issue 06
Date 2023-03-03



Copyright © Huawei Technologies Co., Ltd. 2024. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Security Declaration

Vulnerability

Huawei's regulations on product vulnerability management are subject to the *Vul. Response Process*. For details about this process, visit the following web page:

<https://www.huawei.com/en/psirt/vul-response-process>

For vulnerability information, enterprise customers can visit the following web page:

<https://securitybulletin.huawei.com/enterprise/en/security-advisory>

Contents

1 Key Management.....	1
1.1 Using KMS to Encrypt Offline Data.....	1
1.1.1 Encrypting or Decrypting Small Volumes of Data.....	1
1.1.2 Encrypting or Decrypting a Large Amount of Data.....	3
1.2 Using KMS to Encrypt and Decrypt Data for Cloud Services.....	11
1.2.1 Overview.....	11
1.2.2 Encrypting Data in ECS.....	13
1.2.3 Encrypting Data in OBS.....	14
1.2.4 Encrypting Data in EVS.....	15
1.2.5 Encrypting Data in IMS.....	18
1.2.6 Encrypting an RDS DB Instance.....	20
1.2.7 Encrypting a DDS DB Instance.....	21
1.3 Using the Encryption SDK to Encrypt and Decrypt Local Files.....	21
1.4 Encrypting and Decrypting Data Through Cross-region DR.....	25
2 Secret Management.....	28
2.1 Using CSMS to Change Hard-coded Database Account Passwords.....	28
2.2 Using CSMS to Prevent AK and SK Leakage.....	32
2.3 Using CSMS to Automatically Rotate Security Passwords.....	37
2.4 Rotating Secrets.....	45
2.4.1 Overview.....	45
2.4.2 Rotating a Secret for a User.....	46
2.4.3 Rotating a Secret for Two Users.....	49
3 General.....	54
3.1 Retrying Failed DEW Requests by Using Exponential Backoff.....	54
A Change History.....	57

1 Key Management

1.1 Using KMS to Encrypt Offline Data

1.1.1 Encrypting or Decrypting Small Volumes of Data

Scenario

You can use online tools on the Key Management Service (KMS) console or call the necessary KMS APIs to directly encrypt or decrypt small-size data with a CMK, such as passwords, certificates, or phone numbers.

Restrictions

Currently, a maximum of 4 KB of data can be encrypted or decrypted in this way.

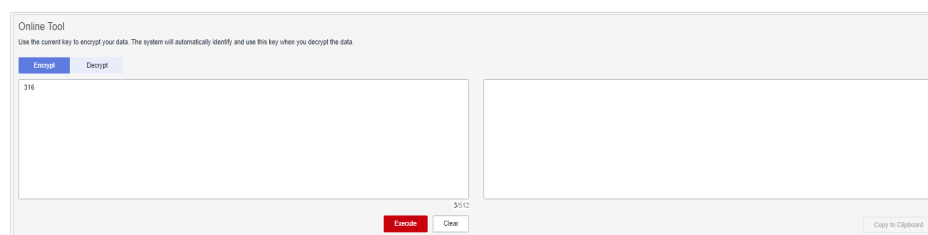
Encryption and Decryption Using Online Tools

- **Encrypting data**

Step 1 Click the alias of a custom key to view its details, and go to the online tool for data encryption and decryption.

Step 2 Click **Encrypt**. In the text box on the left, enter the data to be encrypted, as shown in [Figure 1-1](#).

Figure 1-1 Encrypting data



Step 3 Click **Execute**. Ciphertext of the data is displayed in the text box on the right.

 **NOTE**

- Use the current CMK to encrypt the data.
- You can click **Clear** to clear the entered data.
- You can click **Copy to Clipboard** to copy the ciphertext and save it in a local file.

- **Decrypting data**

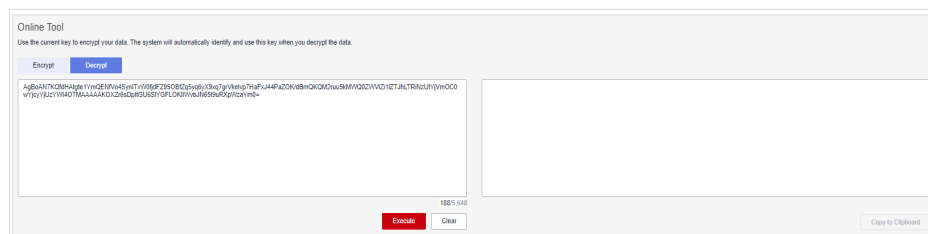
Step 4 You can click any non-default key in **Enabled** status to go to the encryption and decryption page of the online tool.

Step 5 Click **Decrypt**. In the text box on the left, enter the data to be decrypted. For details, see [Figure 1-2](#).

 **NOTE**

- The tool will identify the original encryption CMK and use it to decrypt the data.
- If the key has been deleted, the decryption will fail.

Figure 1-2 Decrypting data



Step 6 Click **Execute**. Plaintext of the data is displayed in the text box on the right.

 **NOTE**

- You can click **Copy to Clipboard** to copy the plaintext and save it in a local file.
- Enter the plaintext on the console, the text will be encoded to Base64 format before encryption.

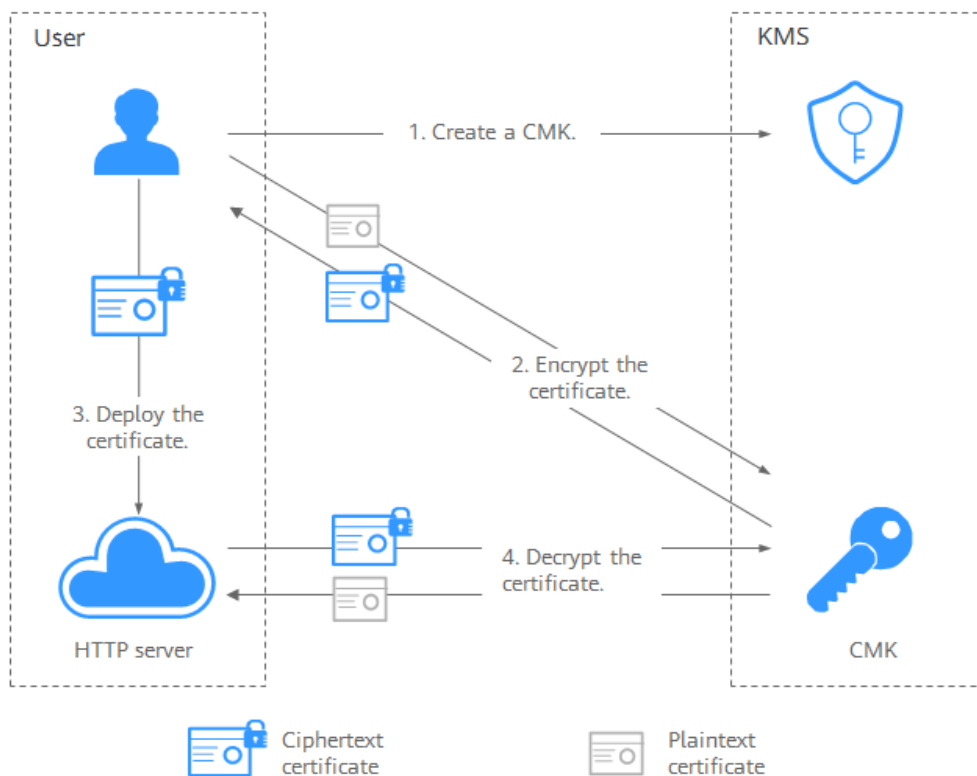
The decryption result returned via API will be in Base64 format. Perform Base64 decoding to obtain the plaintext entered on the console.

----End

Calling APIs for Encryption and Decryption

[Figure 1-3](#) shows an example about how to call KMS APIs to encrypt and decrypt an HTTPS certificate.

Figure 1-3 Encrypting and decrypting an HTTPS certificate



The procedure is as follows:

1. Create a CMK on KMS.
2. Call the **encrypt-data** interface of KMS and use the CMK to encrypt the plaintext certificate.
3. Deploy the certificate onto a server.
4. The server uses the **decrypt-data** interface of KMS to decrypt the ciphertext certificate.

NOTE

If you enter and encrypt text on the console, the text will be encoded to Base64 format before being transferred to the backend for encryption. The decryption result returned via API will be in Base64 format. Text encrypted via API cannot be decrypted on the console, or garbled characters will be returned.

1.1.2 Encrypting or Decrypting a Large Amount of Data

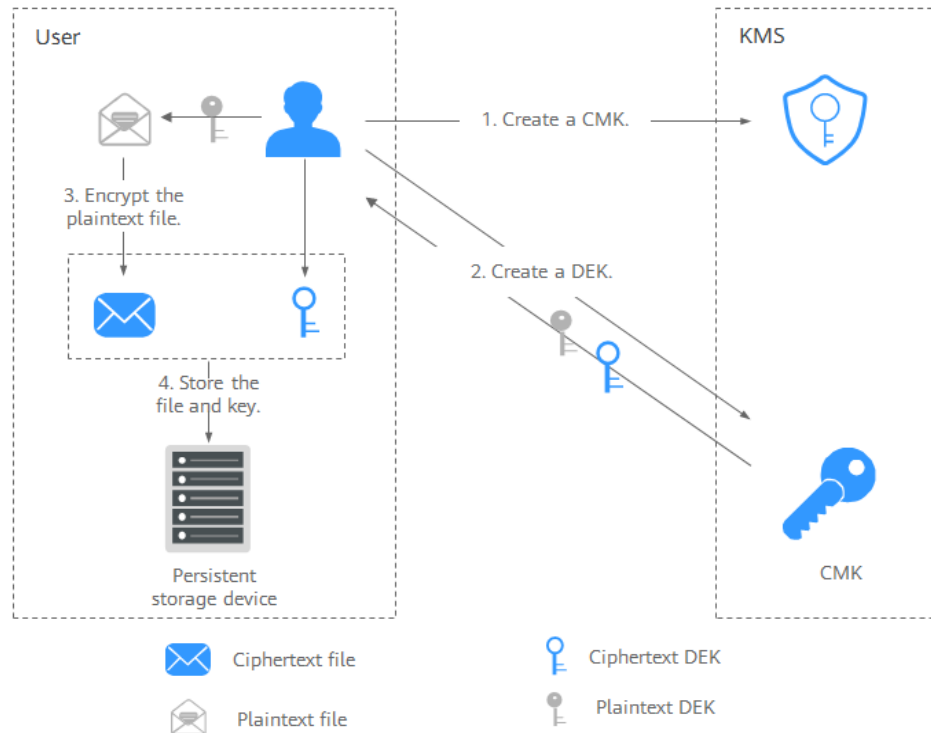
Scenario

If you want to encrypt or decrypt large volumes of data, such as pictures, videos, and database files, you can use envelope encryption, which allows you to encrypt and decrypt files without having to transfer a large amount of data over the network.

Encryption and Decryption Processes

- Large-size data encryption

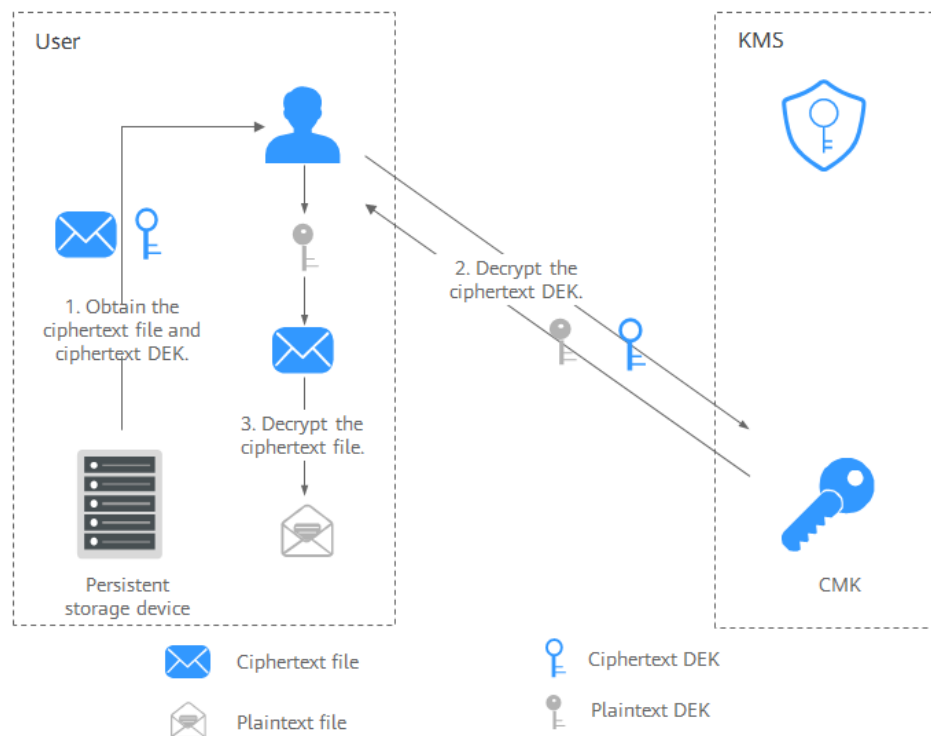
Figure 1-4 Encrypting a local file



The process is as follows:

- a. Create a CMK in KMS.
 - b. Call the **create-datakey** API of KMS to create a DEK. A plaintext DEK and a ciphertext DEK will be generated. The ciphertext DEK is generated when you use a CMK to encrypt the plaintext DEK.
 - c. Use the plaintext DEK to encrypt a plaintext file, generating a ciphertext file.
 - d. Store the ciphertext DEK and the ciphertext file together in a permanent storage device or a storage service.
- Large-size data decryption

Figure 1-5 Decrypting a local file



The process is as follows:

- a. Read the ciphertext DEK and the ciphertext file from the permanent storage device or storage service.
- b. Call the **decrypt-datakey** API of KMS and use the corresponding CMK (the one used for encrypting the DEK) to decrypt the ciphertext DEK. Then you get the plaintext DEK.
If the CMK is deleted, the decryption will fail. Properly keep your CMKs.
- c. Use the plaintext DEK to decrypt the ciphertext file.

Encryption and Decryption APIs

You can use the following APIs to encrypt and decrypt data.

API	Description
Creating a DEK	This API is used to create a DEK.
Decrypting a DEK	This API is used to decrypt a DEK with the specified CMK.

Encrypting a Local File

1. Create a CMK on the management console. For details, see [Creating a CMK](#).
2. Prepare basic authentication information.

- **ACCESS_KEY**: access key of the Huawei ID
- **SECRET_ACCESS_KEY**: secret access key of the Huawei ID
- **PROJECT_ID**: project ID of a HUAWEI CLOUD site. For details, see [Project](#).
- **KMS_ENDPOINT**: endpoint for accessing KMS. For details, see [Endpoints](#).
- There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt the AK/SK in the configuration file or environment variables for storage.
- In this example, the AK/SK stored in the environment variables are used for identity authentication. Configure the environment variables **HUAWEICLOUD_SDK_AK** and **HUAWEICLOUD_SDK_SK** in the local environment first.

3. Encrypt a local file.

Example code is as follows.

- CMK is the ID of the key created on the HUAWEI CLOUD management console.
- The plaintext data file is **FirstPlainFile.jpg**.
- The data file generated after encryption is **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;

/**
 * Use a DEK to encrypt and decrypt files.
 * To enable the assert syntax, add -ea to enable VM_OPTIONS.
 */
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>";

    //Version of the KMS interface. Currently, the value is fixed to v1.0.
    private static final String KMS_INTERFACE_VERSION = "v1.0";

    /**
     * AES algorithm flags:
     * - AES_KEY_BIT_LENGTH: bit length of the AES256 key
     * - AES_KEY_BYTE_LENGTH: byte length of the AES256 key
     * - AES_ALG: AES256 algorithm. In this example, the Group mode is GCM and the padding
     mode is PKCS5Padding.
     * - AES_FLAG: AES algorithm flag
     * - GCM_TAG_LENGTH: GCM tag length
     * - GCM_IV_LENGTH: length of the GCM initial vector
     */
}
```

```
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) {
    // ID of the CMK you created on the HUAWEI CLOUD management console
    final String keyId = args[0];

    encryptFile(keyId);
}

/**
 * Using a DEK to encrypt and decrypt a file
 *
 * @param keyId: user CMK ID
 */
static void encryptFile(String keyId) {
    // 1. Prepare the authentication information for accessing HUAWEI CLOUD.
    final BasicCredentials auth = new
BasicCredentials().withAk(AccessKey).withSk(SecretAccessKey)
.withProjectId(ProjectId);

    // 2. Initialize the SDK and transfer the authentication information and the address for the
KMS to access the client.
    final KmsClient kmsClient =
KmsClient.newBuilder().withCredential(auth).withEndpoint(KmsEndpoint).build();

    // 3. Assemble the request message for creating a DEK.
    final CreateDatakeyRequest createDatakeyRequest = new
CreateDatakeyRequest().withVersionId(KmsInterfaceVersion)
.withBody(new
CreateDatakeyRequestBody().withKeyId(keyId).withDatakeyLength(AesKeyBitLength));

    // 4. Create a DEK.
    final CreateDatakeyResponse createDatakeyResponse =
kmsClient.createDatakey(createDatakeyRequest);

    // 5. Receive the created DEK information.
    // It is recommended that the ciphertext key and KeyId be stored locally so that the
plaintext key can be easily obtained for data decryption.
    // The plaintext key should be used immediately after being created. Before using it,
convert the hexadecimal plaintext key to a byte array.
    final String cipherText = createDatakeyResponse.getCipherText();
    final byte[] plainKey = hexToBytes(createDatakeyResponse.getPlainText());

    // 6. Prepare the file to be encrypted.
    // inFile: file to be encrypted
    // outEncryptFile: file generated after encryption

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

    // 7. If the AES algorithm is used for encryption, you can create an initial vector.
    final byte[] iv = new byte[GcmIvLength];
    final SecureRandom secureRandom = new SecureRandom();
    secureRandom.nextBytes(iv);

    // 8. Encrypt the file and store the encrypted file.
    doFileFinal(Cipher.ENCRYPT_MODE, inFile, outEncryptFile, plainKey, iv);
}

/**
 * Encrypting and decrypting a file
 *
 * @param cipherMode: Encryption mode. It can be Cipher.ENCRYPT_MODE or
```

```
Cipher.DECRYPT_MODE.  
 * @param inFile: file to be encrypted or decrypted  
 * @param outFile: file generated after encryption and decryption  
 * @param keyPlain: plaintext key  
 * @param iv: initial vector  
 */  
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());  
    }  
}
```

Decrypting a Local File

1. Prepare basic authentication information.
 - **ACCESS_KEY**: access key of the Huawei ID
 - **SECRET_ACCESS_KEY**: secret access key of the Huawei ID
 - **PROJECT_ID**: project ID of a HUAWEI CLOUD site. For details, see [Project](#).
 - **KMS_ENDPOINT**: endpoint for accessing KMS. For details, see [Endpoints](#).
 - There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt the AK/SK in the configuration file or environment variables for storage.
 - In this example, the AK/SK stored in the environment variables are used for identity authentication. Configure the environment variables **HUAWEICLOUD_SDK_AK** and **HUAWEICLOUD_SDK_SK** in the local environment first.

2. Decrypt a local file.
Example code is as follows.

- CMK is the ID of the key created on the HUAWEI CLOUD management console.
- The data file generated after encryption is **SecondEncryptFile.jpg**.
- The data file generated after encryption and decryption is **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;

/**
 * Use a DEK to encrypt and decrypt files.
 * To enable the assert syntax, add -ea to enable VM_OPTIONS.
 */
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>";

    //Version of the KMS interface. Currently, the value is fixed to v1.0.
    private static final String KMS_INTERFACE_VERSION = "v1.0";

    /**
     * AES algorithm flags:
     * - AES_KEY_BIT_LENGTH: bit length of the AES256 key
     * - AES_KEY_BYTE_LENGTH: byte length of the AES256 key
     * - AES_ALG: AES256 algorithm. In this example, the Group mode is GCM and the padding
     mode is PKCS5Padding.
     * - AES_FLAG: AES algorithm flag
     * - GCM_TAG_LENGTH: GCM tag length
     * - GCM_IV_LENGTH: length of the GCM initial vector
     */
    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) {
        // ID of the CMK you created on the HUAWEI CLOUD management console
        final String keyId = args[0];
        // Returned ciphertext DEK after DEK creation
        final String cipherText = args[1];

        decryptFile(keyId, cipherText);
    }

    /**
     * Using a DEK to encrypt and decrypt a file
     *
     * @param keyId: user CMK ID
     * @param cipherText: ciphertext data key
     */
    static void decryptFile(String keyId, String cipherText) {
        // 1. Prepare the authentication information for accessing HUAWEI CLOUD.
        final BasicCredentials auth = new
        BasicCredentials().withAk(ACCESS_KEY).withSk(SECRET_ACCESS_KEY)
        .withProjectId(PROJECT_ID);

        // 2. Initialize the SDK and transfer the authentication information and the address for the
        KMS to access the client.
        final KmsClient kmsClient =
        KmsClient.newBuilder().withCredential(auth).withEndpoint(KMS_ENDPOINT).build();
    }
}
```

```
// 3. Prepare the file to be encrypted.
// inFile: file to be encrypted
// outEncryptFile: file generated after encryption
// outDecryptFile: file generated after encryption and decryption
final File inFile = new File("FirstPlainFile.jpg");
final File outEncryptFile = new File("SecondEncryptFile.jpg");
final File outDecryptFile = new File("ThirdDecryptFile.jpg");

// 4. Use the same initial vector for AES encryption and decryption.
final byte[] iv = new byte[GCM_IV_LENGTH];

// 5. Assemble the request message for decrypting the DEK. cipherText is the ciphertext
DEK returned after DEK creation.
final DecryptDatakeyRequest decryptDatakeyRequest = new DecryptDatakeyRequest()
    .withVersionId(KMS_INTERFACE_VERSION).withBody(new
DecryptDatakeyRequestBody()
    .withKeyId(keyId).withCipherText(cipherText).withDatakeyCipherLength(AES_KEY
_BYTE_LENGTH));

// 6. Decrypt the DEK and convert the returned hexadecimal plaintext key into a byte array.
final byte[] decryptDataKey =
hexToBytes(kmsClient.decryptDatakey(decryptDatakeyRequest).getDataKey());

// 7. Decrypt the file and store the decrypted file.
// iv at the end of the statement is the initial vector created in the encryption example.
doFileFinal(Cipher.DECRYPT_MODE, outEncryptFile, outDecryptFile, decryptDataKey, iv);

// 8. Compare the original file with the decrypted file.
assert getFileSha256Sum(inFile).equals(getFileSha256Sum(outDecryptFile));

}

/**
 * Encrypting and decrypting a file
 *
 * @param cipherMode: Encryption mode. It can be Cipher.ENCRYPT_MODE or
Cipher.DECRYPT_MODE.
 * @param inFile: file to be encrypted or decrypted
 * @param outFile: file generated after encryption and decryption
 * @param keyPlain: plaintext key
 * @param iv: initial vector
 */
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());
    }
}

/**
 * Converting a hexadecimal string to a byte array
 *
 * @param hexString: a hexadecimal string

```

```
* @return: byte array
*/
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;
}

/**
 * Calculate the SHA256 digest of the file.
 */
* @param file
* @return SHA256 digest
*/
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 Using KMS to Encrypt and Decrypt Data for Cloud Services

1.2.1 Overview

KMS is a secure, reliable, and easy-to-use cloud service that helps users create, manage, and protect keys in a centralized manner.

After your cloud services are integrated with KMS, to encrypt data on cloud, you simply need to select a CMK managed by KMS for encryption.

You can select a Default Master Key (DMK) automatically created by a cloud service through KMS, or a key you created or imported to KMS. For details, see [Differences Between a CMK and a Default Master Key](#).

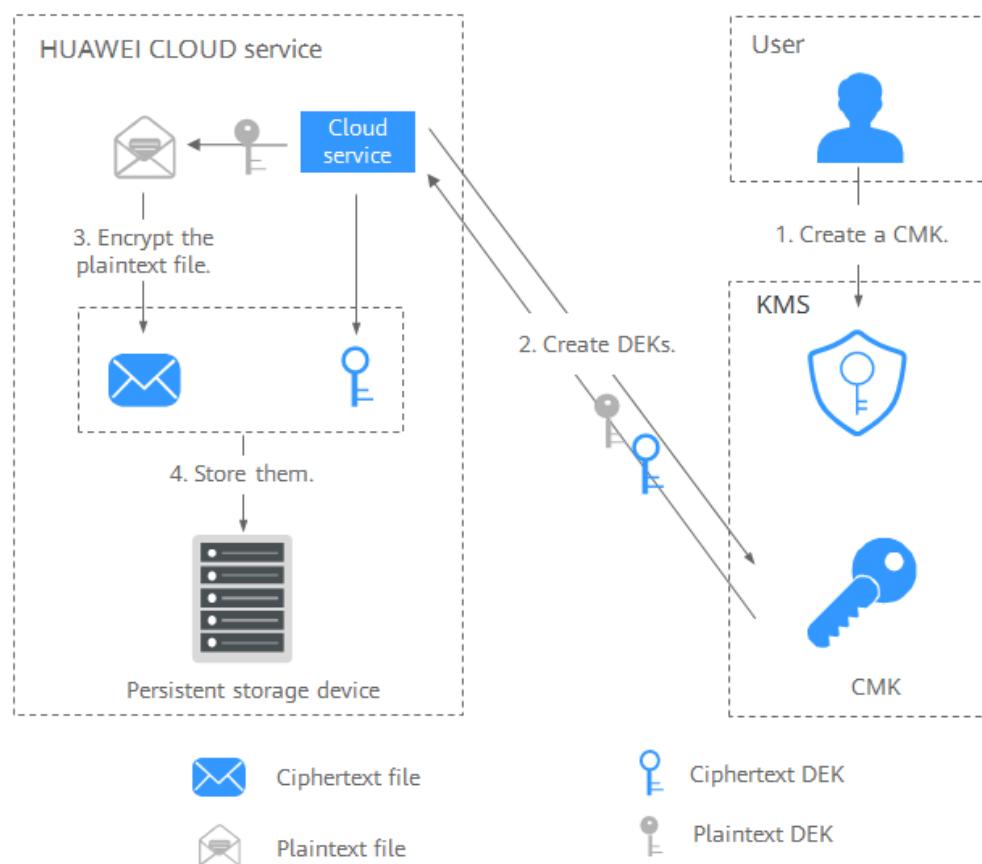
Table 1-1 Cloud services that use KMS encryption

Category	Service	Encryption Mode
Computing	Elastic Cloud Server (ECS)	You can encrypt an image or EVS disk in ECS. <ul style="list-style-type: none">• When creating an ECS, if you select an encrypted image, the system disk of the created ECS automatically has encryption enabled, with its encryption mode same as the image encryption mode.• When creating an ECS, you can encrypt added data disks.
	Image Management Service (IMS)	Encrypting Data in IMS
Storage	Object Storage Service (OBS)	Encrypting Data in OBS
	Elastic Volume Service (EVS)	Encrypting Data in EVS
	Volume Backup Service (VBS)	VBS generally creates online backups for a single EVS disk (system or data disk) of the server. If it is encrypted, its backup data will be stored in encrypted mode.
	Cloud Server Backup Service (CSBS)	CSBS mainly creates consistency backups online for all EVS disks of the server. CSBS backups will also be displayed on the VBS page. If it is encrypted, its backup data will be stored in encrypted mode.
Database	RDS for MySQL	Encrypting an RDS DB Instance
	RDS for PostgreSQL	
	RDS for SQL Server	
	Document Database Service (DDS)	Encrypting a DDS DB Instance

Encryption Process

HUAWEI CLOUD services use the envelope encryption technology and call KMS APIs to encrypt service resources. Your CMKs are under your own management. With your grant, HUAWEI CLOUD services use a specific CMK of yours to encrypt data.

Figure 1-6 How Huawei Cloud uses KMS for encryption



The encryption process is as follows:

1. Create a CMK on KMS.
2. A HUAWEI CLOUD service calls the **create-datakey** API of the KMS to create a DEK. A plaintext DEK and a ciphertext DEK are generated.

NOTE

Ciphertext DEKs are generated when you use a CMK to encrypt the plaintext DEKs.

3. The HUAWEI CLOUD service uses the plaintext DEK to encrypt a plaintext file, generating a ciphertext file.
4. The HUAWEI CLOUD service saves the ciphertext DEK and the ciphertext file together in a permanent storage device or a storage service.

NOTE

When users download the data from the HUAWEI CLOUD service, the service uses the CMK specified by KMS to decrypt the ciphertext DEK, uses the decrypted DEK to decrypt data, and then provides the decrypted data for users to download.

1.2.2 Encrypting Data in ECS

Overview

KMS supports one-click encryption for ECS. The images and data disks of ECS can be encrypted.

- When creating an ECS, if you select an encrypted image, the system disk of the created ECS automatically has encryption enabled, with its encryption mode same as the image encryption mode.
- When creating an ECS, you can encrypt added data disks.

For details about how to encrypt an image, see [Encrypting Data in IMS](#).

For details about how to encrypt a data disk, see [Encrypting Data in EVS](#).

1.2.3 Encrypting Data in OBS

Overview

After server-side encryption is enabled, data of an object uploaded to Object Storage Service (OBS) is encrypted on the server before being stored. When the object is downloaded, data is decrypted on the server first.

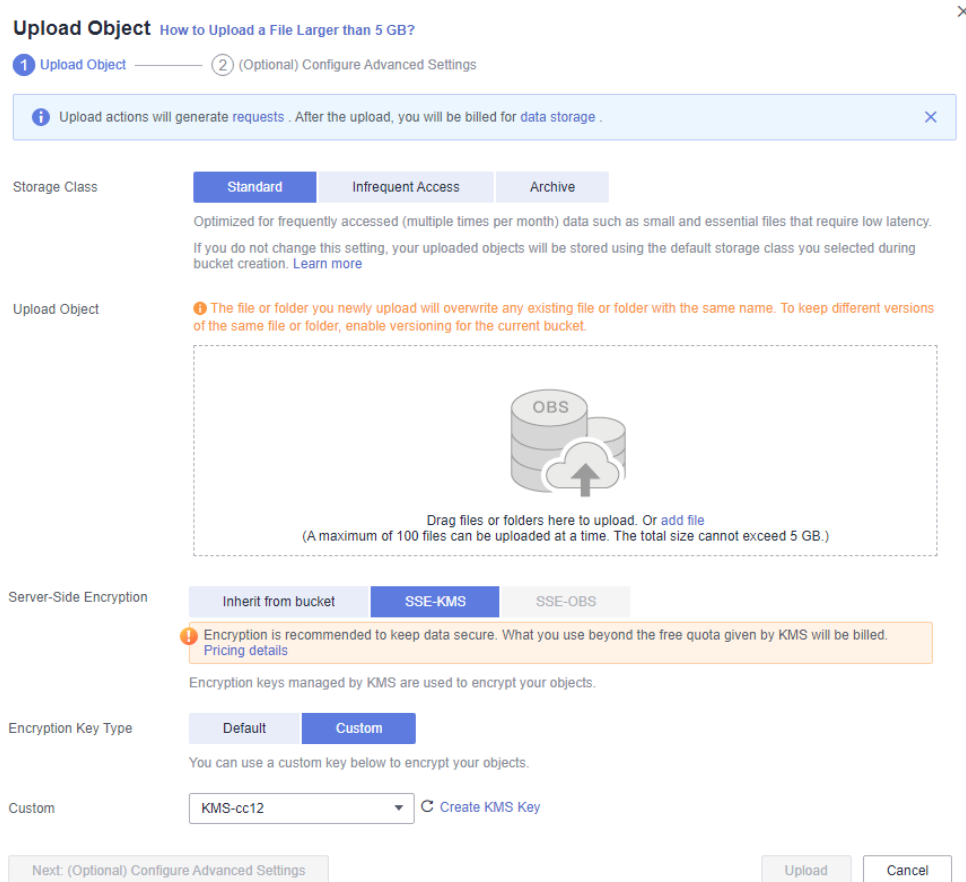
KMS uses a third-party hardware security module (HSM) to protect keys, enabling you to create and manage encryption keys easily. Keys are not displayed in plaintext outside HSMs, which prevents key disclosure. With KMS, all operations on keys are controlled and logged, and usage records of all keys can be provided to meet regulatory compliance requirements.

Server-side encryption with KMS-managed keys (SSE-KMS) can be implemented for the objects to be uploaded. You need to create a key using KMS or use the default key provided by KMS. Then you can use the key to encrypt the object on the server when uploading the object to OBS.

Uploading Files in Server-side Encryption Mode (on the Console)

- Step 1** In the bucket list on the OBS console, click a bucket to go to the **Overview** page.
- Step 2** In the navigation tree on the left, choose **Objects**.
- Step 3** Click **Upload Object**. The **Upload Object** dialog box is displayed.
- Step 4** Click **Add File**, select the file to be uploaded, and click **Open**.
- Step 5** Set **Server-Side Encryption** to **SSE-KMS**, select the default key or a custom key, and click **Upload**.

Figure 1-7 Encrypting an object to be uploaded



Key name: Name of the custom key. The key is created in DEW and is used for encrypted protection for data. OBS provides a default key **obs/default**. You can use the default key or create a key in DEW.

Step 6 After uploading the object, click it to view its encryption status.

NOTE

- The object encryption status cannot be changed.
- A key in use cannot be deleted. Otherwise, the object encrypted with this key cannot be downloaded.

----End

Uploading Files in Server-side Encryption Mode (Through an API)

You can call the required API of OBS to upload a file in SSE-KMS mode. For details, see *Object Storage Service API Reference*.

1.2.4 Encrypting Data in EVS

Overview

In case your services require encryption for the data stored on disks in Elastic Volume Service (EVS), EVS provides you with the encryption function. You can

encrypt newly created EVS disks. Keys used by encrypted EVS disks are provided by KMS of DEW, secure and convenient. Therefore, you do not need to establish and maintain the key management infrastructure.

Disk encryption is used for data disks only. System disk encryption relies on the image. For details, see [Encrypting Data in IMS](#).

Who Can Use the Disk Encryption Function?

- Security administrators (users having Security Administrator rights) can grant the KMS access rights to EVS for using disk encryption.
- When a common user who does not have the Security Administrator rights needs to use the disk encryption feature, the condition varies depending on whether the user is the first one ever in the current region or project to use this feature.
 - If the user is the first, the user must contact a user having the Security Administrator rights to grant the KMS access rights to EVS. Then, the user can use the disk encryption feature.
 - If the user is not the first, the user can use the disk encryption function directly.

From the perspective of a tenant, as long as the KMS access rights have been granted to EVS in a region, all users in the same region can directly use the disk encryption feature.

If there are multiple projects in the current region, the KMS access rights need to be granted to each project in this region.

Keys Used for EVS Disk Encryption

The keys provided by KMS for disk encryption include a Default Master Key and Customer Master Keys (CMKs).

- **Default Master Key:** A key that is automatically created by EVS through KMS and named **evs/default**.
The Default Master Key cannot be disabled and does not support scheduled deletion.
- **CMKs:** Keys created by users. You can use existing CMKs or create one. For details, see [Creating a CMK](#).

If disks are encrypted using a CMK, which is then disabled or scheduled for deletion, the disks can no longer be read from or written to, and data on these disks may never be restored. See [Table 1-2](#) for more information.

Table 1-2 Impact on encrypted disks after a CMK becomes unavailable

CMK Status	Impact on Encrypted Disks	Restoration Method
Disabled	<ul style="list-style-type: none"> If an encrypted disk is then attached to an ECS, the disk can still be used, but normal read/write operations are not guaranteed permanently. If an encrypted disk is then detached, re-attaching the disk will fail. 	Enable the CMK. For details, see Enabling One or More CMKs .
Pending deletion		Cancel the scheduled deletion for the CMK. For details, see Canceling the Scheduled Deletion of One or More CMKs .
Deleted		Data on the disks can never be restored.

NOTICE

You will be charged for the CMKs you use. If basic keys are used, ensure that your account balance is sufficient. If professional keys are used, renew your order timely. Otherwise, your services may be interrupted and your data may never be restored as the encrypted disks become unreadable and unwritable.

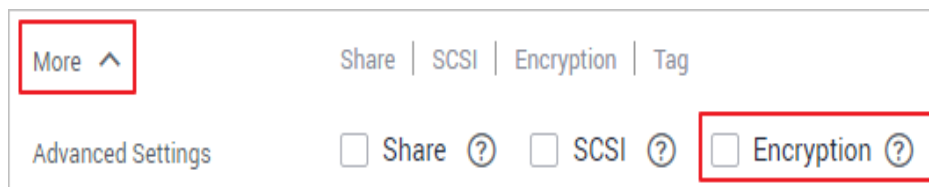
Using KMS to Encrypt a Disk (on the Console)

Step 1 On the EVS management console, click **Buy Disk**.

Step 2 Select the **Encryption** check box.

1. Click **More**. The **Encryption** check box is displayed.

Figure 1-8 More



2. Create an agency.
Select **Encrypt**. If EVS is not authorized to access KMS, the **Create Agency** dialog box is displayed. In this case, click **Yes** to authorize it. After the authorization, EVS can obtain KMS keys to encrypt and decrypt disks.

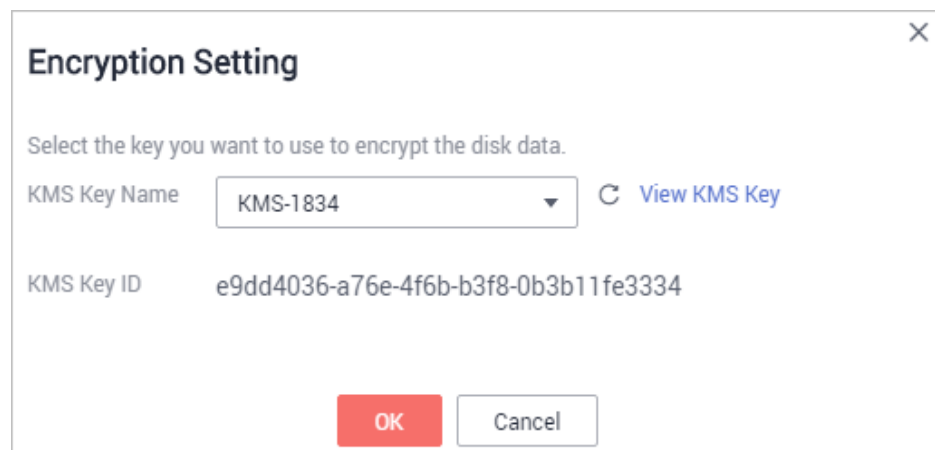
 **NOTE**

Before you use the disk encryption function, KMS access rights need to be granted to EVS. If you have the right for granting, grant the KMS access rights to EVS directly. If you do not have the right, contact a user with the Security Administrator rights to grant the KMS access rights to EVS, then repeat the preceding operations.

3. Set encryption parameters.

Select **Encrypt**. If the authorization succeeded, the **Encrypt Setting** dialog box is displayed.

Figure 1-9 Encryption settings



Select either of the following types of keys from the **KMS Key Name** drop-down list:

- Default Master Key. After the KMS access rights have been granted to EVS, the system automatically creates a Default Master Key named **evs/default**.
- An existing or new CMK. For details about how to create one, see [Creating a CMK](#).

Step 3 Configure other parameters for the disk. For details about the parameters, see [Purchase an EVS Disk](#).

----End

Using KMS to Encrypt a Disk (Through an API)

You can call the required API of EVS to purchase an encrypted EVS disk. For details, see *Elastic Volume Service API Reference*.

1.2.5 Encrypting Data in IMS

You can create an encrypted image in Image Management Service (IMS) to securely store data.

Restrictions

- DEW must be enabled.

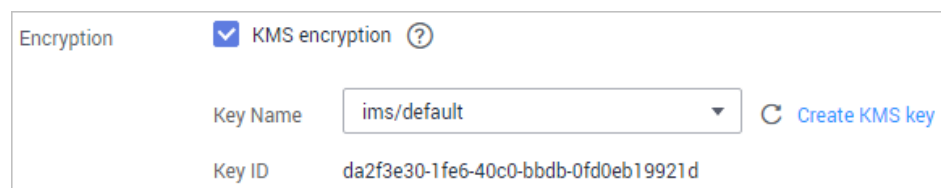
- An encrypted image cannot be shared with other users.
- An encrypted image cannot be published in the Marketplace.
- If an ECS has an encrypted system disk, the private image created using the ECS is also encrypted.
- The key used for encrypting an image cannot be changed.
- If the key used for encrypting an image is disabled or deleted, the image is unavailable.
- The system disk of an ECS created using an encrypted image is also encrypted, and its key is the same as the image key.

Using KMS to Encrypt a Private Image (on the Console)

You can create an encrypted image using an encrypted ECS or an external image file.

- Create an encrypted image using an encrypted ECS.
When you use an ECS to create a private image, if the system disk of the ECS is encrypted, the private image created using the ECS is also encrypted. The key used for encrypting the image is the one used for creating the system disk.
- Create an encrypted image using an external image file.
When you use an external image file that has been uploaded to an OBS bucket to create a private image, you can select KMS encryption when registering the image to encrypt the image.
When uploading an image file, you can select **KMS encryption** and use a key provided by KMS to encrypt the uploaded file, as shown in [Figure 1-10](#).
 - a. On the IMS management console, click **Create Private Image**.
 - b. Set **Type** to **System disk image**.
 - c. Set **Source** to **Image File**.
 - d. Select **KMS encryption**.

Figure 1-10 Encrypting data in IMS



The screenshot shows the 'Encryption' section of the IMS console. It features a checked checkbox for 'KMS encryption' with a help icon. Below this, there is a 'Key Name' dropdown menu currently set to 'ims/default', a 'Key ID' field displaying 'da2f3e30-1fe6-40c0-bbdb-0fd0eb19921d', and a 'Create KMS key' button.

Select either of the following types of keys from the **Key Name** drop-down list:

- Default Master Key **ims/default** created by KMS
 - An existing or new CMK. For details about how to create one, see [Creating a CMK](#).
- e. Configure other parameters. For details about the parameters, see [Registering an Image](#).

Using KMS to Encrypt a Private Image (Through an API)

You can call the required API of IMS to encrypt the image file. For details, see *Image Management Service API Reference*.

1.2.6 Encrypting an RDS DB Instance

Overview

Relational Database Service (RDS) supports MySQL and PostgreSQL engines.

After encryption is enabled, disk data will be encrypted and stored on the server when you create a DB instance or expand disk capacity. When you download encrypted objects, the encrypted data will be decrypted on the server and displayed in plaintext.

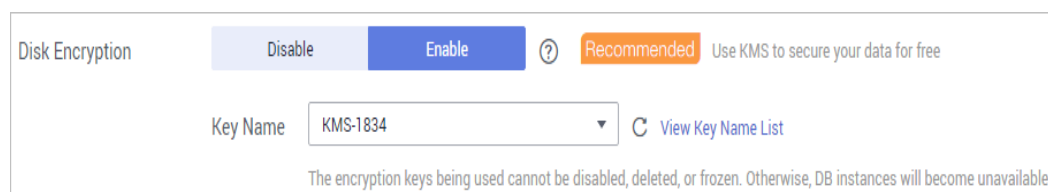
Restrictions

- The KMS Administrator right must be granted to the user in the region of RDS by using Identity and Access Management (IAM). For details about how to assign permissions to user groups, see "How Do I Manage User Groups and Grant Permissions to Them?" in *Identity and Access Management User Guide*.
- To use a user-defined key to encrypt objects to be uploaded, create a key using DEW. For details, see [Creating a CMK](#).
- Once the disk encryption function is enabled, you cannot disable it or change the key after a DB instance is created. The backup data stored in OBS will not be encrypted.
- After an RDS DB instance is created, do not disable or delete the key that is being used. Otherwise, RDS will be unavailable and data cannot be restored.
- If you scale up a DB instance with disks encrypted, the expanded storage space will be encrypted using the original encryption key.

Using KMS to Encrypt a DB Instance (on the Console)

When a user purchases a database instance from Relational Database Service (RDS), the user can select **Disk encryption** and use the key provided by KMS to encrypt the disk of the database instance. For more information, see [Buy a MySQL DB Instance](#) and [Buy a PostgreSQL DB Instance](#).

Figure 1-11 Encrypting data in RDS



Using KMS to Encrypt a DB Instance (Through an API)

You can also call the required API of RDS to purchase encrypted DB instances. For details, see *Relational Database Service API Reference*.

1.2.7 Encrypting a DDS DB Instance

Overview

After encryption is enabled, disk data will be encrypted and stored on the server when you create a DB instance or expand disk capacity. When you download encrypted objects, the encrypted data will be decrypted on the server and displayed in plaintext.

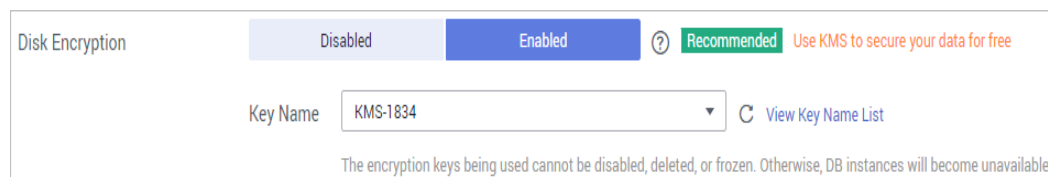
Restrictions

- The KMS Administrator right must be added in the region of RDS using IAM. For details about how to assign permissions to user groups, see "How Do I Manage User Groups and Grant Permissions to Them?" in *Identity and Access Management User Guide*.
- To use a user-defined key to encrypt objects to be uploaded, create a key using DEW. For details, see [Creating a CMK](#).
- Once the disk encryption function is enabled, you cannot disable it or change the key after a DB instance is created. The backup data stored in OBS will not be encrypted.
- After a Document Database Service (DDS) DB instance is created, do not disable or delete the key that is being used. Otherwise, DDS will be unavailable and data cannot be restored.
- If you scale up a DB instance with disks encrypted, the expanded storage space will be encrypted using the original encryption key.

Using KMS to Encrypt a DB Instance (on the Console)

When you purchase a DB instance in DDS, you can set **Disk Encryption** to **Enable** and use the key provided by KMS to encrypt the disk of the DB instance. For more information, see [Buying a Cluster Instance](#).

Figure 1-12 Encrypting data in DDS



Using KMS to Encrypt a DB Instance (Through an API)

You can also call the required API of DDS to purchase encrypted DB instances. For details, see *Document Database Service API Reference*.

1.3 Using the Encryption SDK to Encrypt and Decrypt Local Files

You can use certain algorithms to encrypt your files, protecting them from being breached or tampered with.

Encryption SDK is a client password library that can encrypt and decrypt data and file streams. You can easily encrypt and decrypt massive amounts of data simply by calling APIs. It allows you to focus on developing the core functions of your applications without being distracted by the data encryption and decryption processes.

Scenario

If large files and images are sent to KMS through HTTPS for encryption, a large number of network resources will be consumed and the encryption will be slow. This section describes how to quickly encrypt a large amount of data.

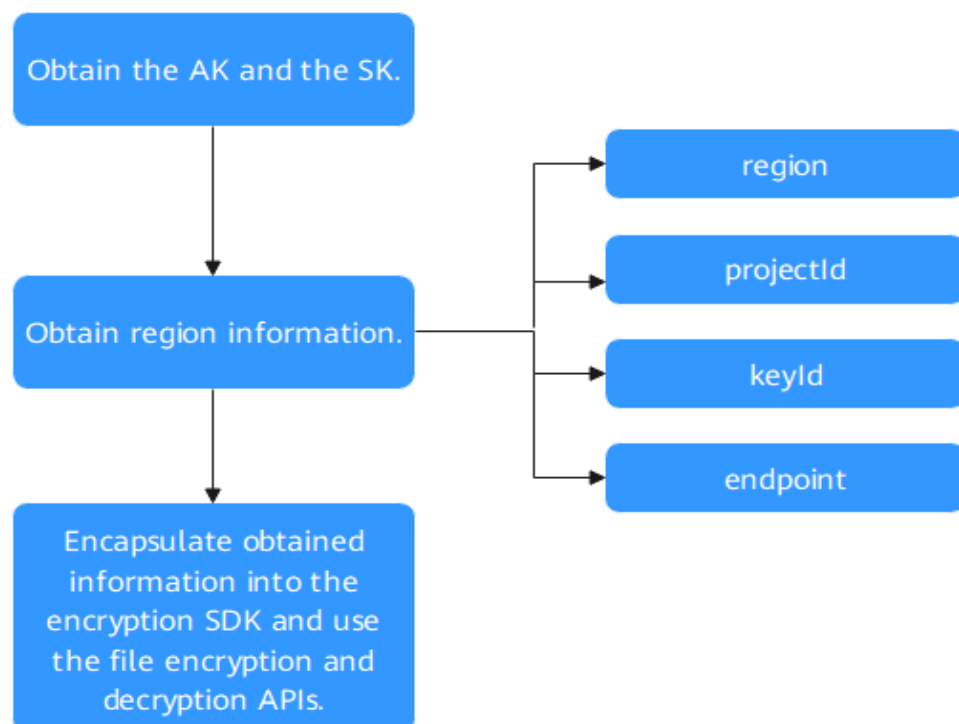
Solution

Encryption SDK performs envelope encryption on file streams segment by segment.

Data is encrypted within the SDK by using the DEK generated by KMS. Segmented encryption of files in the memory ensures the security and correctness of file encryption, because it does not require file transfer over the network.

The SDK loads a file to memory and processes it segment by segment. The next segment will not be read before the encryption or decryption of the current segment completes.

Process



Procedure

Step 1 Obtain the AK and the SK.

- **ACCESS_KEY**: Access key of the Huawei ID. For details, see [How Do I Obtain an Access Key \(AK/SK\)?](#)
- **SECRET_ACCESS_KEY**: Secret access key of the Huawei ID. For details, see [How Do I Obtain an Access Key \(AK/SK\)?](#)
- There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt the AK/SK in the configuration file or environment variables for storage.
- In this example, the AK/SK stored in the environment variables are used for identity authentication. Configure the environment variables **HUAWEICLOUD_SDK_AK** and **HUAWEICLOUD_SDK_SK** in the local environment first.

Step 2 Obtain region information.

1. [Log in to the management console.](#)
2. Hover over the username in the upper right corner and choose **My Credentials** from the drop-down list.
3. Obtain the **Project ID** and **Project Name**.

Figure 1-13 Obtaining the project ID and project name

Project ID	Project Name	Region
0d82edc16	10ea895171161f	cn-north-7
9de3819d1	52bd215af5db1	cn-north-1
a52e7697d	7e62d48243717bc	cn-north-4
e40df114bc	831a098bc103b	cn-north-2
6d537a841	953558194a69978	cn-north-9

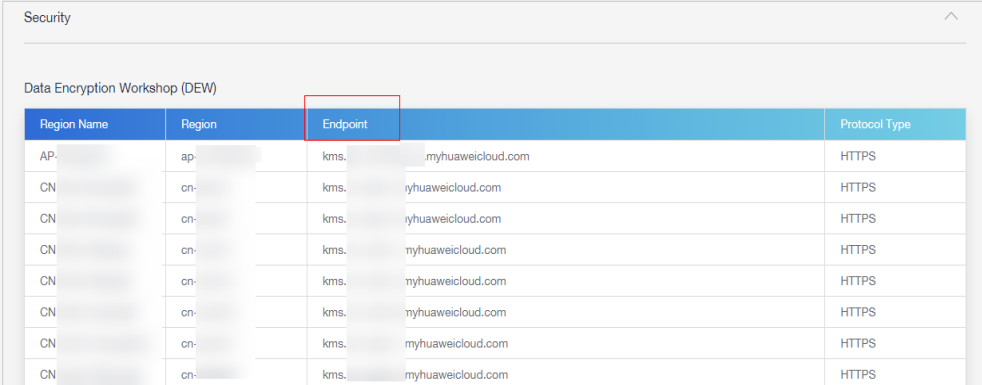
4. Click . Choose **Security & Compliance > Data Encryption Workshop**.
5. Obtain the ID of the CMK (**KEYID**) to be used in the current region.

Figure 1-14 Obtaining the CMK ID

AliasID	Status	Key Algorithm and Usage	Origin	Enterprise Project	Operation
KMS-e80l 8daa2980-719b-409c-a8c5-aa2f493a692	Enabled	AES_256 ENCRYPT_DECRYPT	Key Management Service	default	Disable Delete Add to Project
KMS-41e bc77738-161c-4844-81ce-actf5c8dc6d	Enabled	RSA_4096 ENCRYPT_DECRYPT	Key Management Service	DEW	Disable Delete Add to Project

6. Obtain the endpoint (**ENDPOINT**) required by the current region.
An endpoint is the **request address** for calling an API. Endpoints vary depending on services and regions. For the endpoints of all services, see [Regions and Endpoints](#).

Figure 1-15 Obtaining an endpoint



Region Name	Region	Endpoint	Protocol Type
AP-...	ap-...	kms. myhuaweicloud.com	HTTPS
CN	cn-	kms. yhuaweicloud.com	HTTPS
CN	cn-	kms. yhuaweicloud.com	HTTPS
CN	cn-	kms. nyhuaweicloud.com	HTTPS
CN	cn-	kms. nyhuaweicloud.com	HTTPS
CN	cn-	kms. nyhuaweicloud.com	HTTPS
CN	cn-	kms. nyhuaweicloud.com	HTTPS
CN	cn-	kms. myhuaweicloud.com	HTTPS
CN	cn-	kms. myhuaweicloud.com	HTTPS

Step 3 Encrypt and decrypt a file.

```
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 {  
        // Source file path  
        String encryptFileInPutPath = args[0];  
        // Path of the encrypted ciphertext file  
        String encryptFileOutPutPath = args[1];  
        // Path of the decrypted file  
        String decryptFileOutPutPath = args[2];  
        // Encryption context  
        Map<String, String> encryptContextMap = new HashMap<>();  
        encryptContextMap.put("encryption", "context");  
        encryptContextMap.put("simple", "test");  
        encryptContextMap.put("caching", "encrypt");  
        // Construct the encryption configuration  
        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);  
        // Set the key ring.  
        huaweiCrypto.withKeyring(new  
            KmsKeyringFactory().getKeyring(KeyringTypeEnum.KMS_MULTI_REGION.getType()));  
        // Encrypt the file.  
        encryptFile(encryptContextMap, huaweiCrypto, encryptFileInPutPath, encryptFileOutPutPath);  
        // Decrypt the file.  
        decryptFile(huaweiCrypto, encryptFileOutPutPath, decryptFileOutPutPath);  
    }  
  
    private static void encryptFile(Map<String, String> encryptContextMap, HuaweiCrypto huaweiCrypto,  
        String encryptFileInPutPath, String encryptFileOutPutPath) throws IOException {  
        // fileInputStream: input stream corresponding to the encrypted file  
        FileInputStream fileInputStream = new FileInputStream(encryptFileInPutPath);  
        // fileOutputStream: output stream corresponding to the source file  
        FileOutputStream fileOutputStream = new FileOutputStream(encryptFileOutPutPath);  
        // Encryption  
        huaweiCrypto.encrypt(fileInputStream, fileOutputStream, encryptContextMap);  
        fileInputStream.close();  
        fileOutputStream.close();  
    }  
}
```

```
private static void decryptFile(HuaweiCrypto huaweiCrypto, String decryptFileInPutPath, String
decryptFileOutPutPath) throws IOException {
    // in: input stream corresponding to the source file
    FileInputStream fileInputStream = new FileInputStream(decryptFileInPutPath);
    // out: output stream corresponding to the encrypted file
    FileOutputStream fileOutputStream = new FileOutputStream(decryptFileOutPutPath);
    // Decryption
    huaweiCrypto.decrypt(fileInputStream, fileOutputStream);
    fileInputStream.close();
    fileOutputStream.close();
}
}
```

 NOTE

For more information, see [Details](#).

----End

1.4 Encrypting and Decrypting Data Through Cross-region DR

Scenario

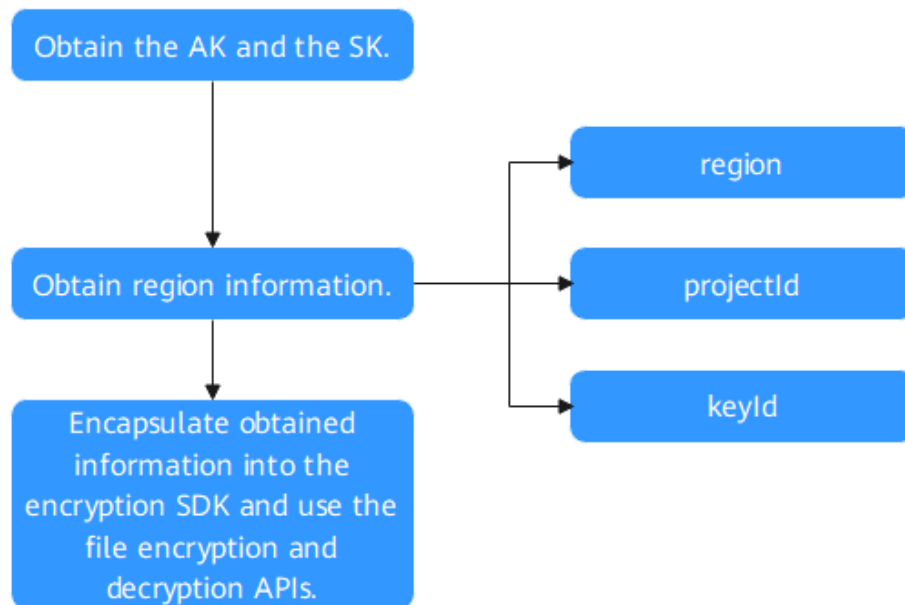
If a fault occurs during encryption or decryption in a region, you can use KMS to implement cross-region DR encryption and decryption, ensuring service continuity.

Solution

If KMS is faulty in one or multiple regions, encryption and decryption can be completed as long as a key in the key ring is available.

A cross-region key ring can use the CMKs of multiple regions to encrypt a piece of data and generate unique data ciphertext. To decrypt the data, you simply need to use a key ring that contains one or more available CMKs that were used for encrypting the data.

Process



Procedure

Step 1 Obtain the AK and the SK.

- **ACCESS_KEY**: Access key of the Huawei ID. For details, see [How Do I Obtain an Access Key \(AK/SK\)?](#)
- **SECRET_ACCESS_KEY**: Secret access key of the Huawei ID. For details, see [How Do I Obtain an Access Key \(AK/SK\)?](#)
- There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt the AK/SK in the configuration file or environment variables for storage.
- In this example, the AK/SK stored in the environment variables are used for identity authentication. Configure the environment variables **HUAWEICLOUD_SDK_AK** and **HUAWEICLOUD_SDK_SK** in the local environment first.

Step 2 Obtain region information.

1. [Log in to the management console.](#)
2. Hover over the username in the upper right corner and choose **My Credentials** from the drop-down list.
3. Obtain the **Project ID** and **Project Name**.

Figure 1-16 Obtaining the project ID and project name

Project ID	Project Name	Region
0d22edc16	10ea895171191f	cn-north-7
9def5916d	52bc02195af5061	cn-north-1
a52a7697d	9e62d482437170c	cn-north-4
e40af114bc	631a098bc193b	cn-north-2
6b537a841	953558184a69978	cn-north-9


4. Click . Choose **Security & Compliance > Data Encryption Workshop**.
5. Obtain the ID of the CMK (**KEYID**) to be used in the current region.

Figure 1-17 Obtaining the CMK ID

AliasID	Status	Key Algorithm and Usage	Origin	Enterprise Project	Operation
KMS-e80 86a2980-119b-409c-ab05-aa2f493a952	Enabled	AES_256 ENCRYPT_DECRYPT	Key Management Service	default	Disable Delete Add to Project
KMS-f41e bc77f73b-161c-4844-81ee-actf5cd6a2bd	Enabled	RSA_4096 ENCRYPT_DECRYPT	Key Management Service	DEV	Disable Delete Add to Project

Step 3 Use the key ring for encryption and decryption.

```
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>";

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

    // Data to be encrypted
    private static final String PLAIN_TEXT = "Hello World!";

    public static void main(String[] args) {
        // CMK list
        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));
        // Construct encryption-related information.
        HuaweiConfig multiConfig = HuaweiConfig.builder().buildSk(SECRET_ACCESS_KEY)
            .buildAk(ACCESS_KEY)
            .buildKmsConfig(kmsConfigList)
            .buildCryptoAlgorithm(CryptoAlgorithm.AES_256_GCM_NOPADDING)
            .build();
        // Select a key ring.
        KMSKeyring keyring = new
        KmsKeyringFactory().getKeyring(KeyringTypeEnum.KMS_MULTI_REGION.getType());
        HuaweiCrypto huaweiCrypto = new HuaweiCrypto(multiConfig).withKeyring(keyring);
        // Encryption context
        Map<String, String> encryptContextMap = new HashMap<>();
        encryptContextMap.put("key", "value");
        encryptContextMap.put("context", "encrypt");
        // Encryption
        CryptoResult<byte[]> encryptResult = huaweiCrypto.encrypt(new EncryptRequest(encryptContextMap,
        PLAIN_TEXT.getBytes(StandardCharsets.UTF_8)));
        // Decryption
        CryptoResult<byte[]> decryptResult = huaweiCrypto.decrypt(encryptResult.getResult());
        Assert.assertEquals(PLAIN_TEXT, new String(decryptResult.getResult()));
    }
}
```

----End

2 Secret Management

2.1 Using CSMS to Change Hard-coded Database Account Passwords

Generally, the secrets used for access are embedded in applications. To update a secret, you need to create a new secret and spend time updating your applications. If you have multiple applications using the same secret, you have to update all of them, or the applications you forgot to update will be unable to use the secret for login.

An easy-to-use, effective, and secure secret management tool will be helpful.

Cloud Secret Management Service (CSMS) has the following advantages:

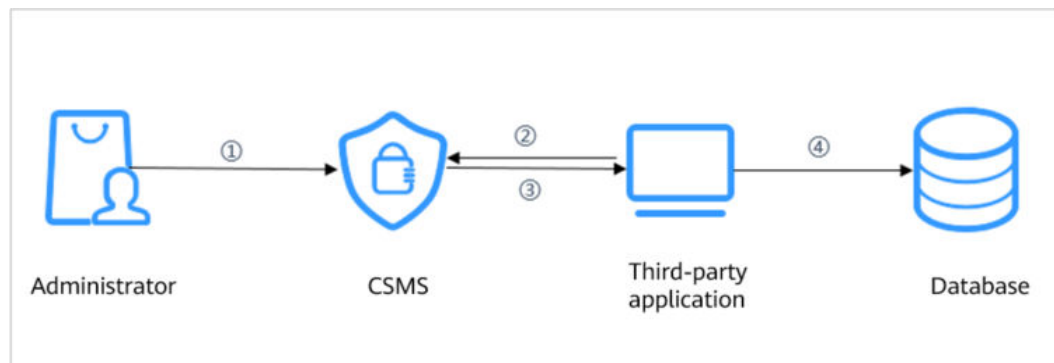
- You can host your secrets instead of using hardcoded secrets, improving the security of data and assets.
- Your services are not affected when you **manually rotate** secrets.
- Secure SDK access allows you to dynamically call your secrets.
- You can store many types of secrets. You can store service accounts, passwords, and database information, including but not limited to database names, IP addresses, and port numbers.

Logging In to a Database Using Secrets

You can create a secret and log in to your database by calling the secret via an API.

Ensure your account has the KMS Administrator or KMS CMKFullAccess permission. For details, see [DEW Permissions Management](#).

Figure 2-1 Secret-based login process



The process is as follows:

- Step 1** Create a secret on the [console](#) or via an [API](#) to store database information (such as the database address, port, and password).
- Step 2** Use an application to access the database. CSMS will query the secret created in **1**.
- Step 3** CSMS retrieves and decrypts the secret ciphertext and securely returns the information stored in the secret to the application through the secret management API.
- Step 4** The application obtains the decrypted plaintext secret and uses it to access the database.

----End

Secret Creation and Query APIs

You can call the following APIs to create secrets, save their content, and query secret information.

API	Description
Creating a Secret	This API is used to create a secret and store the secret value in the initial secret version.
Querying a Secret	This API is used to query a secret.

Creating and Querying Secrets via APIs

1. Prepare basic authentication information.
 - **ACCESS_KEY**: access key of the Huawei ID
 - **SECRET_ACCESS_KEY**: secret access key of the Huawei ID
 - **PROJECT_ID**: project ID of a Huawei Cloud site. For details, see [Project](#).
 - **CSMS_ENDPOINT**: endpoint for accessing CSMS. For details, see [Endpoints](#).

- There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt the AK/SK in the configuration file or environment variables for storage.
- In this example, the AK/SK stored in the environment variables are used for identity authentication. Configure the environment variables **HUAWEICLOUD_SDK_AK** and **HUAWEICLOUD_SDK_SK** in the local environment first.

2. Create and query secret information.

Secret name: **secretName**

Secret value: **secretString**

Secret version value: **LATEST_SECRET**

Secret version: **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 {
    /**
     * Basic authentication information:
     * - ACCESS_KEY: access key of the Huawei ID
     * - SECRET_ACCESS_KEY: secret access key of the Huawei ID
     * - PROJECT_ID: Huawei Cloud project ID. For details, see https://support.huaweicloud.com/intl/en-us/productdesc-iam/iam\_01\_0023.html
     * - CSMS_ENDPOINT: endpoint address for accessing CSMS. For details, see https://support.huaweicloud.com/intl/en-us/api-dew/dew\_02\_0052.html.
     * - There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt the AK/SK in the configuration file or environment variables for storage.
     * - In this example, the AK/SK stored in the environment variables are used for identity authentication. Configure the environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK in the local environment first.
     */
    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>";

    //Version ID used to query the latest secret version details
    private static final String LATEST_SECRET = "latest";

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

        //Create a secret.
        createSecret(secretName, secretString);

        //Query the content of the new secret based on the secret version latest or v1.
        ShowSecretVersionResponse latestVersion = showSecretVersion(secretName, LATEST_SECRET);
        ShowSecretVersionResponse firstVersion = showSecretVersion(secretName, "v1");

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

    /**
     * Create a secret.
     * @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);
}
/**
 * Query secret version details based on the secret version ID.
 * @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;
}
/**
 * Obtain the CSMS client.
 * @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();
}
}
```

Obtaining the Database Account and Password Through an Application

1. Obtain the dependency statement of the CSMS SDK.

Example:

```
<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. Establish a database connection and obtain the account and password.

Example:

```
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;

//Obtain the specified database account and password based on the secret information.
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 Using CSMS to Prevent AK and SK Leakage

CSMS is a secure, reliable, and easy-to-use credential hosting service. Users or applications can use CSMS to create, retrieve, update, and delete credentials in a unified manner throughout the credential lifecycle. CSMS can help you eliminate risks incurred by hardcoding, plaintext configuration, and permission abuse.

Scenario

Save the AK and SK to prevent leakage.

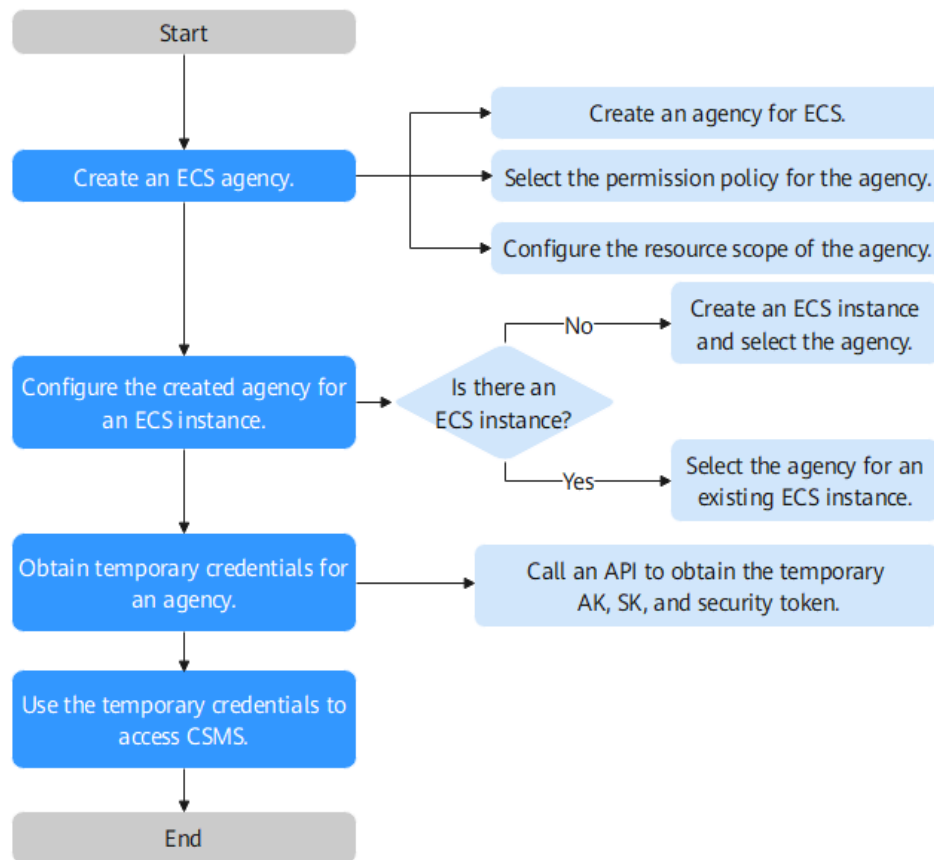
How It Works

You can use Identity and Access Management (IAM) to obtain temporary access keys for Elastic Cloud Server (ECS) to protect AKs and SKs.

Access credentials can be classified into permanent credentials and temporary credentials based on their validity periods. Permanent access credentials include usernames and passwords. Temporary access keys have a shorter validity period, are updated frequently, thus are more secure. You can assign an IAM agency to an ECS instance, so that applications in the ECS instance can use the temporary AK, SK, and security token to access CSMS. The temporary access keys are dynamically obtained every time they are required. They can also be cached in the memory and updated periodically.

Process Flow

Figure 2-2 ECS agency configuration process



Constraint

Only the administrator or an IAM user with the ECS permission can configure an agency for an ECS instance.

Procedure

Step 1 Create an ECS agency on IAM.


1. **Log in to the management console.**
2. Click  on the left of the page and choose **Management & Governance > Identity and Access Management**. The **Users** page is displayed.
3. In the navigation pane, choose **Agencies**.
4. Click **Create Agency** in the upper right corner.
5. Configure parameters in the **Create Agency** dialog box. For more information, see [Agency parameters](#).

Figure 2-3 Creating an agency

The screenshot shows a web form for creating an agency. It contains the following elements:

- Agency Name:** A text input field.
- Agency Type:** Two radio button options: "Account" (with subtext "Delegate another HUAWEI CLOUD account to perform operations on your resource") and "Cloud service" (with subtext "Delegate a cloud service to access your resources in other cloud services"). The "Cloud service" option is selected.
- Cloud Service:** A dropdown menu currently showing "Elastic Cloud Server (ECS) and Bare Metal Serv...".
- Validity Period:** A dropdown menu currently showing "Unlimited".
- Description:** A large text area with the placeholder "Enter a brief description." and a character count "0/255" at the bottom right.
- Buttons:** A red "Next" button and a white "Cancel" button.

Table 2-1 Agency parameters

Parameter Name	Description
Agency Name	Enter an agency name. Example: ECS_TO_CSMS
Agency Type	Select Cloud service .
Cloud Service	Select Elastic Cloud Server (ECS) and Bare Metal Server (BMS) .
Validity Period	Select a duration. The value can be Unlimited, 1 day, or Custom .
Description	(Optional) Enter agency description.

6. Click **Next** to go to the authorization page.
7. Click **Create Policy** in the upper right corner. If you already have a policy, skip this step.
 - a. Configure policy parameters. For more information, see [Policy parameters](#).

Figure 2-4 Creating a policy

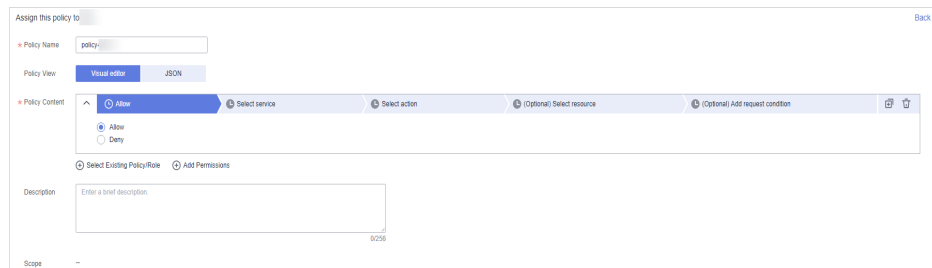


Table 2-2 Policy parameters

Parameter Name	Description
Policy Name	Enter a policy name.
Policy View	Select Visual editor .
Policy Content	<ul style="list-style-type: none"> ▪ Allow: Select Allow. ▪ Select service: Select Cloud Secret Management Service (CSMS). ▪ Select action: Select read and write permissions as required. ▪ (Optional) Select resource: Select the scope of resources. <ul style="list-style-type: none"> ○ Specific: Access specific credentials. <p>NOTE You can select Specify resource path, and then click Add Resource Path to specify an accessible credential.</p> ○ All: Access all credentials. ▪ (Optional) Add request condition: Click Add Request Condition, select a condition key and an operator, and enter values as required.
Description	(Optional) Enter policy description.

8. Select a policy for the agency. Click **Next**.
9. Select a scope and click **OK**.
 - **All resources:** IAM users will be able to use all resources, including those in enterprise projects, region-specific projects, and global services under your account based on assigned permissions.
 - **Enterprise projects:** The selected permissions will be applied to resources in the enterprise projects you select.
 - **Region-specific projects:** The selected permissions will be applied to resources in the region-specific projects you select.

Step 2 Assign an agency (for example, **ECS_TO_CSMS**) to an ECS instance.



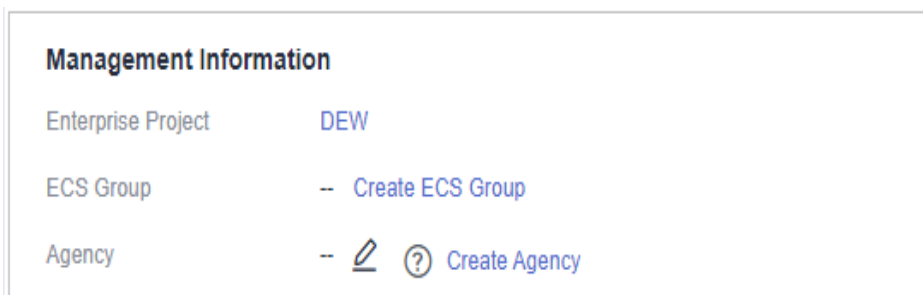
- To create an ECS instance, perform the operations described in [Creating an ECS](#). In **Step 3: Configure Advanced Settings**, select the new agency (for example, **ECS_TO_CSMS**).
- To use an existing ECS instance, perform the following steps:
 - a. Click  on the left of the page and choose **Management & Governance > Identity and Access Management**. Go to the ECS page.
 - b. Click the name of an ECS instance to go to the **Summary** page.
 - c. In the **Management Information** area, click  and select an agency (for example, **ECS_TO_CSMS**).

Figure 2-5 Selecting an agency



Step 3 In an application running on the ECS instance, call an API to obtain the temporary agency credentials, including the temporary AK, SK, and security token, to access CSMS.

1. Obtain the temporary AK and SK (in the **Security Key** directory). For details, see [Obtaining Metadata](#).

- URI
/openstack/latest/securitykey
- Method
GET request
- The following data is returned:

```
{
  "credential": {
    "access": "LDHZK30XXXXXXXXXXXXXV",
    "secret": "gyqcdzVXXXXXXXXXXXXXXXXXXXXXXXXXXXXMl6",
    "securitytoken": "El9FI2C65qXXXXXXXXXXXXXXXXXXXXnkaov",
    "expires_at": "2022-07-14T12:09:24.147000Z"
  }
}
```

 **NOTE**

- Extract the values of **access**, **secret**, and **securitytoken** to access CSMS.
 - ECS automatically rotates temporary credentials to ensure that they are secure and valid.
2. Use the temporary AK, SK, and security token to access CSMS.
 - An example of the credential list is as follows. For details, see [CSMS](#).


```
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) {
        String ak = "<YOUR AK>";
        String sk = "<YOUR SK>";
        String securitytoken = "<YOUR SecurityToken>";

        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());
        }
    }
}
```

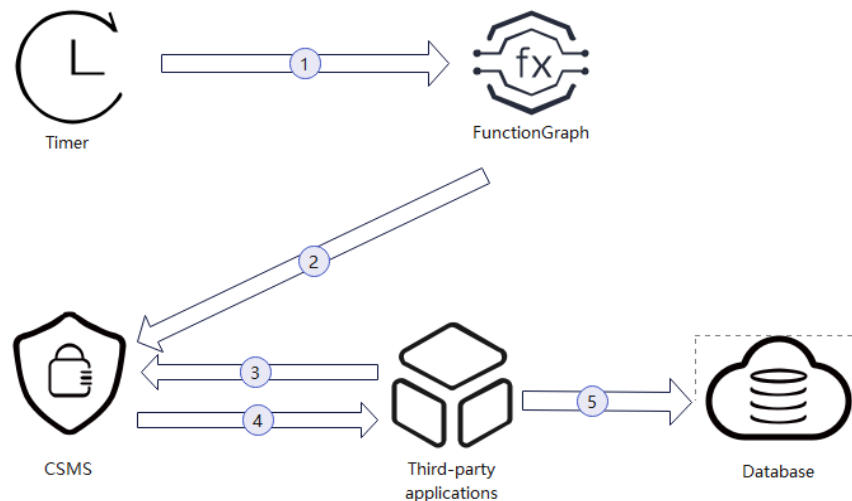
----End

2.3 Using CSMS to Automatically Rotate Security Passwords

This section describes how to use FunctionGraph and CSMS to generate and rotate strong secure passwords periodically, so that secure and compliant passwords can be generated, hosted, and automatically rotated.

Process

Figure 2-6 Password rotation



The process is as follows:


1. When a timer expires, a scheduled triggering event is published.
2. After receiving the event, FunctionGraph replaces the placeholder in the secret template with a new random password, and stores the password in the secret, which is considered as a new version of the secret.
3. Applications periodically call APIs or SDKs to obtain the latest secret version.
4. CSMS retrieves and decrypts the secret ciphertext and securely returns the information stored in the secret to the application through the secret management API.
5. After receiving the decrypted secret, applications use the new password for future access by using it to update the target object (such as the database or server).

Constraints

- CSMS is available in the region.
- FunctionGraph is available in the region.

Creating an Agency

Step 1 [Log in to the management console.](#)

Step 2 Click  on the left and choose **Management & Governance > Identity and Access Management** to access the **Users** page.

Step 3 In the navigation pane on the left, choose **Agencies**.

Step 4 Click **Create Agency** and configure the parameters as shown in **Figure 2-7**. **Table 2-3** describes the parameters.

Figure 2-7 Creating an agency

★ Agency Name

★ Agency Type Account
Delegate another Huawei Cloud account to perform operations on your resources.
 Cloud service
Delegate a cloud service to access your resources in other cloud services.

★ Cloud Service

★ Validity Period

Description
46/255

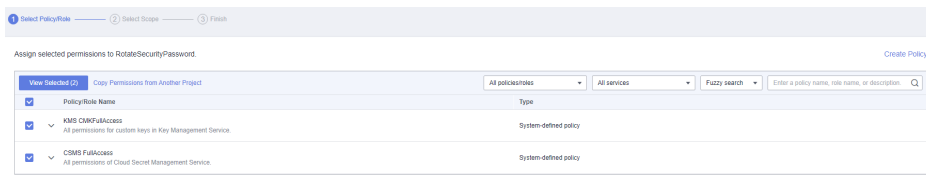
Table 2-3 Agency parameters

Parameter	Description
Agency Name	Set this parameter as required.
Agency Type	Select Cloud service .
Cloud Service	Choose FunctionGraph .
Validity Period	Set this parameter based on the application scenario of the function. If the function needs to be executed for a long time, choose Unlimited .
Description	Set this parameter as needed.

Step 5 Click **Next**.

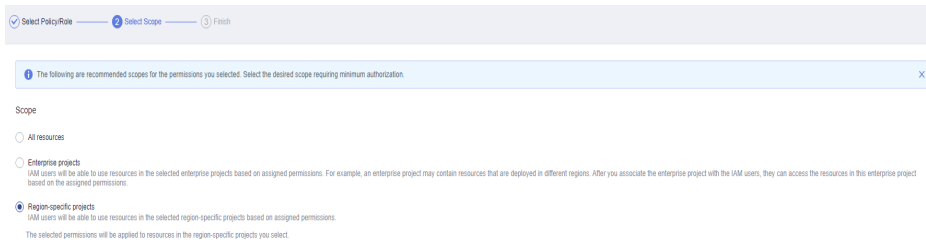
Step 6 Select **CSMS FullAccess** and **KMS CMKFullAccess**.

Figure 2-8 Selecting permissions



Step 7 Click **Next** and select a scope based on your service requirements.

Figure 2-9 Selecting a scope




Step 8 Click **OK**.

----End

Creating a Password Rotation Function

Step 1 **Log in to the management console.**

Step 2 Click  on the left and choose **Compute > FunctionGraph**.

Step 3 Click **Create Function** in the upper right corner and configure the parameters as shown in **Figure 2-10**. **Table 2-4** describes the parameters.

Figure 2-10 Creating a function

Basic Information

* Function Type

Event Function | HTTP Function

Processes event requests and can be triggered by APIG, OBS, and DIS events.

* Region

Region

Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region.

* Project

Project

* Function Name

rotate-security-password

Enter 1 to 60 characters, starting with a letter and ending with a letter or digit. Only letters, digits, hyphens (-), and underscores (_) are allowed.

Agency ⓘ

RotateSecurityPassword | Create Agency

Permission Policies

CSMS FullAccess | KMS CMKFullAccess | View Policies

These are the permission policies assigned to agency RotateSecurityPassword. To add or delete policies, go to the IAM console.

* Enterprise Project

default | View Enterprise Project

Enterprise Project Management Service (EPS) provides a unified method to manage cloud resources and personnel by enterprise project.

Runtime

Python 3.6

Select a language to compile the function. CodeArts IDE Online supports Node.js, Python, and PHP.

Table 2-4 Basic parameters

Parameter	Description
Region	Select the region where the function is deployed.
Project	Select the project where the function is deployed.
Function Name	Enter a custom function name.
Agency	Enter the name set in Creating an Agency .
Enterprise Project	If you have enabled enterprise projects, select one for you to add a function to it. If you have not enabled enterprise projects, this parameter will not be displayed. Skip this step. For details about how to enable an enterprise project, see Enabling Enterprise Center .
Runtime	Select a language for writing the function. Currently, Python is supported. NOTE Only Python 3.6, 3.9, and 3.10 are supported.

Step 4 Click **Create Function**.

Step 5 Choose the **Configuration** tab. In the navigation pane on the left, choose **Environment Variables**. Click **Add** and add environment variables in the variable configuration row. [Table 2-4](#) describes the parameters. Then, click **Save**.

Table 2-5 Environment variables

Parameter	Description	Example
region	Project name which is based on the region, for example, if the region is CN North-Beijing4, the project name should be cn-north-4 . To obtain your region, click your username in the upper right corner of the page, and choose My Credentials from the drop-down list.	cn-north-4
secret_name	Name of the secret to be rotated NOTE A secret must have been created. For details, see Creating a Secret .	rds-functionGraph-rotate
secret_content	Secret template which is specified using braces ({}), for example, the secret template is {"password":"password_placeholder"} . In this case, password_placeholder is the placeholder, which will be replaced by a secure password after the function is executed. The new content will be saved in the secret after the replacement. NOTE If multiple placeholders exist in a template, more than one password will be generated and replaced in sequence.	{"password":"password_placeholder"}
password_length	Password length. The value ranges from 8 to 128 . The default value is 16 .	16
password_format	Password format. The default value is 2 . Possible values are as follows: 1. Digits and letters 2. Digits, letters, and special characters (~!@#%^*_+=?) 3. Only digits 4. Only letters	2

Step 6 Choose the **Code** tab, add the following password rotation function to the editing window, and click **Deploy**.

```
# -*- 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'
```

----End

Debugging

Debug the created FunctionGraph. For details, see [Online Debugging](#).

Creating a Trigger

Create a trigger. For details, see [Using a Timer Trigger](#).

Viewing a Secret

Step 1 [Log in to the management console](#).

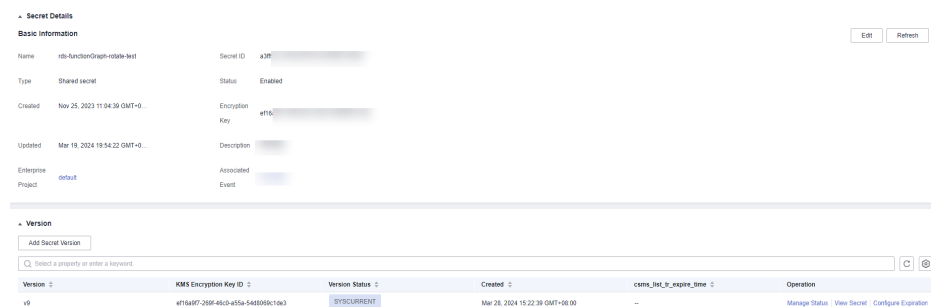
Step 2 Click . Choose **Security & Compliance > Data Encryption Workshop**.

Step 3 In the navigation pane, choose **Cloud Secret Management Service**.

Step 4 Search for the secret created in [Creating a Password Rotation Function](#).

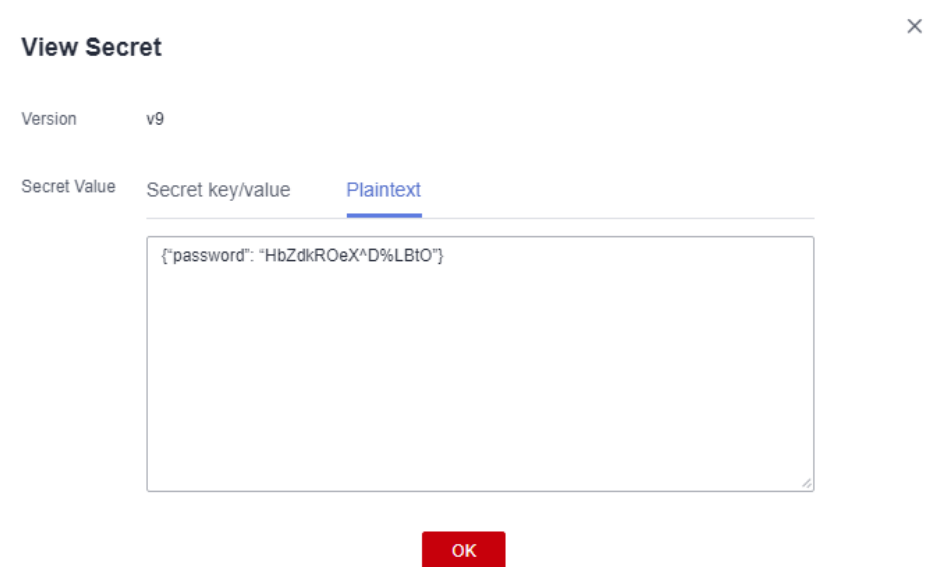
Step 5 Click the secret to view its details, including current and historical versions.

Figure 2-11 Viewing secret details



Step 6 In the **Version**, locate the secret, and click **View Secret** in the **Operation** column to view the password.

Figure 2-12 Viewing a secret value



----End

2.4 Rotating Secrets

2.4.1 Overview

If you have not updated secrets for a long time, important information (such as important passwords, tokens, certificates, SSH keys, and API keys) protected by these secrets are exposed to leakage risks. Periodically rotating secrets improves the security of protected plaintext information.

HUAWEI CLOUD provides two secret rotation policies:

- [Rotating a Secret for a User](#)
- [Rotating a Secret for a Two Users](#)

You can select a rotation policy as needed.

Procedure

1. The administrator adds a secret version and updates its content on the [console](#) or via an [API](#).
2. The application calls a CSMS API to obtain the latest secret version or the secret of a specified version status, and then rotate the secret.
3. Regularly repeat steps [1](#) and [2](#) to rotate secrets.

2.4.2 Rotating a Secret for a User

Overview

You can update the information of a user in a secret.

This is the most commonly used secret rotation policy.

Example:

- For database access, a database connection is not deleted during secret rotation. After the rotation, new connections use the new secrets.
- A user can create an account, for example, using an email address as the username. Generally, users can change passwords as needed, but cannot create other users or change usernames.
- Temporary users are created at the moment they are needed.
- Passwords are entered by users, not retrieved from CSMS by applications. These users do not need to change their usernames or passwords.

Constraints

Ensure your account has the KMS Administrator or KMS CMKFullAccess permission. For details, see [DEW Permissions Management](#).

Secret Rotation APIs

You can call the following APIs to rotate secrets locally.

API	Description
Creating a Secret Version	Create a secret version.
Querying the Secret Version and Value	Query the information about a specified secret version and the plaintext secret value in the version.

Example of Rotating the Secret of an Account

1. Create a secret on the HUAWEI CLOUD console. For details, see [Creating a Secret](#).
2. Prepare basic authentication information.
 - **ACCESS_KEY**: access key of the Huawei ID

- **SECRET_ACCESS_KEY**: secret access key of the Huawei ID
 - **PROJECT_ID**: project ID of a HUAWEI CLOUD site. For details, see [Project](#).
 - **CSMS_ENDPOINT**: endpoint for accessing CSMS. For details, see [Endpoints](#).
 - There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt the AK/SK in the configuration file or environment variables for storage.
 - In this example, the AK/SK stored in the environment variables are used for identity authentication. Configure the environment variables **HUAWEICLOUD_SDK_AK** and **HUAWEICLOUD_SDK_SK** in the local environment first.
3. Rotation the secret of a user.

In the example code,

- **secretName** indicates the name of the secret created on the Huawei Cloud console.
- **secretString** indicates the value saved in the secret created on the Huawei Cloud console.
- **versionId** indicates the secret ID automatically generated after a secret is created on the Huawei Cloud console.
- **LATEST_VERSION** indicates the secret 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 {
    /**
     * Basic authentication information:
     * - ACCESS_KEY: access key of the Huawei ID
     * - SECRET_ACCESS_KEY: secret access key of the Huawei ID
     * - PROJECT_ID: Huawei Cloud project ID. For details, see https://support.huaweicloud.com/intl/en-us/productdesc-iam/iam_01_0023.html
     * - CSMS_ENDPOINT: endpoint address for accessing CSMS. For details, see https://support.huaweicloud.com/intl/en-us/api-dew/dew_02_0052.html.
     * - There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt the AK/SK in the configuration file or environment variables for storage.
     * - In this example, the AK/SK stored in the environment variables are used for identity authentication. Configure the environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK in the local environment first.
     */
    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>";
    //Version ID used to query the latest secret version details.
    private static final String LATEST_VERSION = "latest";
    public static void main(String[] args) {
        String secretName = args[0];
        String secretString = args[1];
        singleAccountRotation(secretName, secretString);
    }

    /**
     * Example: secret rotation for a single account
     *
     * @param secretName Secret name
     * @param secretString Secret content
     */
}
```

```
*/
private static void singleAccountRotation(String secretName, String secretString) {
    //Host the new secret in CSMS.
    createNewSecretVersion(secretName, secretString);
    //Query the secret of the latest version based on the secret version latest.
    ShowSecretVersionResponse secretResponseByLatest =
showSecretVersionDetail(secretName, LATEST_VERSION);
    assert secretResponseByLatest.getVersion().getSecretString().equals(secretString);
}

/**
 * Example of creating a secret
 * If a secret version is added without version status, the program points the SYSCURRENT
version status to the version by default.
 *
 * @param secretName Secret name
 * @param secretString Secret content
 * @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;
}

/**
 * Query the secret of a specified version.
 *
 * @param secretName Secret_name
 * @param versionId Secret_version_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;
}

/**
 * Obtain the CSMS client.
 *
 * @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 Rotating a Secret for Two Users

Overview

You can update the information of two users in a secret.

For example, if your application requires high availability and you perform single-user rotation, you may fail to access the application when changing the user password and updating the secret content. In this case, you need to use the multi-user secret rotation policy.

You must have an account, for example, **Admin**, that has the permission to create and change the passwords of **user1** and **user2**. First, create a secret, create and save the account and password of **user1**, and record them as **user1/password1**. Then, create **user2** by adding secret version **v2**, and save the password of **user2** as **user2/password2**. If you change the password of **user1**, you need to add secret version **v3** and record it as **user1/password3**. When the secret version **v3** is being created, the application uses the existing secret version **v2** to obtain information. Once **v3** is ready, the temporary storage tag will be changed, and the application can use **v3** to obtain information.

Constraints

Ensure your account has the KMS Administrator or KMS CMKFullAccess permission. For details, see [DEW Permissions Management](#).

Secret Rotation APIs

You can call the following APIs to rotate secrets locally.

API	Description
Creating a Secret Version	Create a secret version.
Querying the Secret Version and Value	Query the information about a specified secret version and the plaintext secret value in the version.
Updating the Version Status of a Secret	Update the version status of a secret.
Querying the Status of a Secret Version	Query the status of a specified secret version.

Example of Rotating the Secret of Two Accounts

1. Create a secret on the HUAWEI CLOUD console as the administrator. For details, see [Creating a Secret](#).
2. Prepare basic authentication information.
 - **ACCESS_KEY**: access key of the Huawei ID
 - **SECRET_ACCESS_KEY**: secret access key of the Huawei ID

- **PROJECT_ID**: project ID of a HUAWEI CLOUD site. For details, see [Project](#).
 - **CSMS_ENDPOINT**: endpoint for accessing CSMS. For details, see [Endpoints](#).
 - There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt the AK/SK in the configuration file or environment variables for storage.
 - In this example, the AK/SK stored in the environment variables are used for identity authentication. Configure the environment variables **HUAWEICLOUD_SDK_AK** and **HUAWEICLOUD_SDK_SK** in the local environment first.
3. Rotate the secret for two users.

In the example code,

- **secretName** indicates the name of the secret created on the Huawei Cloud console.
- **secretString** indicates the value saved in the secret created on the Huawei Cloud console.
- **versionId** indicates the secret ID automatically generated after a secret is created on the Huawei Cloud console.
- **LATEST_VERSION** indicates the secret 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 {
    /**
     * Basic authentication information:
     * - ACCESS_KEY: access key of the Huawei ID
     * - SECRET_ACCESS_KEY: secret access key of the Huawei ID
     * - PROJECT_ID: Huawei Cloud project ID. For details, see https://support.huaweicloud.com/intl/en-us/productdesc-iam/iam\_01\_0023.html
     * - CSMS_ENDPOINT: endpoint address for accessing CSMS. For details, see https://support.huaweicloud.com/intl/en-us/api-dew/dew\_02\_0052.html.
     * - There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt the AK/SK in the configuration file or environment variables for storage;
     * - In this example, the AK/SK stored in the environment variables are used for identity authentication. Configure the environment variables HUAWEICLOUD_SDK_AK and HUAWEICLOUD_SDK_SK in the local environment first.
     */
    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>";

    //Version ID used to query the latest secret version details.
    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);
}

/**
 * Example: secret rotation for two accounts
 *
 * @param secretName
 * @param secretString
 */
private static void dualAccountRotation(String secretName, String secretString) {
    // Create a new secret version with a custom version status. Assume that the secret
    // content is the account password of service A.
    CreateSecretVersionResponse newSecretVersion =
    createNewSecretVersionWithStage(secretName,
        secretString, Collections.singletonList(HW_PENDING_STAGE));
    String versionId = newSecretVersion.getVersionMetadata().getId();

    // Before rotation, check whether the pending state is modified.
    ListSecretStageResponse listSecretStageResponse = showSecretStage(secretName,
    HW_PENDING_STAGE);
    assert listSecretStageResponse.getStage().getVersionId().equals(versionId);

    // After the account and password of service A are updated, the version status of the new
    // secret is updated to SYSCURRENT. The old secret is still stored in the service and cannot be
    // accessed by specifying latest.
    updateSecretStage(secretName, versionId, HW_CURRENT_STAGE);

    // Query the secret of the latest version by specifying latest to complete the dual-account
    // secret rotation.
    ShowSecretVersionResponse secretResponse = showVersionDetail(secretName,
    LATEST_VERSION);

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

/**
 * Example of creating a secret
 * If you add a secret version by customizing the version status, the program will not point
 * the SYSCURRENT version status to this version.
 *
 * @param secretName
 * @param newSecretVersionText
 * @param stageList
 * @return
 */
private static CreateSecretVersionResponse createNewSecretVersionWithStage(String
secretName,
                                String newSecretVersionText,
                                List<String> stageList) {
    CsmsClient csmsClient = getCsmsClient();

    //Host the new secret in CSMS.
    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;
}
```

```
}

/**
 * Point the input version state to the specified secret version.
 * @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);
}

/**
 * Query the secret of a specified version.
 * @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;
}

/**
 * Query the status of a specified version.
 * @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);
}

/**
 * Obtain the CSMS client.
 *
 * @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();  
}  
}
```

3 General

3.1 Retrying Failed DEW Requests by Using Exponential Backoff

Scenario

If you receive an error message when calling an API, you can use exponential backoff to retry the request.

How It Works

If consecutive errors (such as traffic limiting errors) are reported by the service side, continuous access will keep causing conflicts. Exponential backoff can help you avoid such errors.

Constraints

The current account has an enabled key.

Example

1. Prepare basic authentication information.
 - **ACCESS_KEY**: Access key of the Huawei ID. For details, see [How Do I Obtain an Access Key \(AK/SK\)?](#)
 - **SECRET_ACCESS_KEY**: Secret access key of the Huawei ID. For details, see [How Do I Obtain an Access Key \(AK/SK\)?](#)
 - **PROJECT_ID**: site project ID. For details, see [Obtaining a Project ID](#).
 - **KMS_ENDPOINT**: endpoint for accessing KMS. For details, see [Endpoints](#).
 - There will be security risks if the AK/SK used for authentication is directly written into code. Encrypt the AK/SK in the configuration file or environment variables for storage.
 - In this example, the AK/SK stored in the environment variables are used for identity authentication. Configure the environment variables

HUAWEICLOUD_SDK_AK and **HUAWEICLOUD_SDK_SK** in the local environment first.

2. Code for exponential backoff:

```
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);
                    }
                }
            }
        }
    }
}
```

```
}  
}
```

A Change History

Release On	Description
2023-03-03	This is the sixth official release. Added section "Encrypting Data in ECS".
2022-12-13	This is the fifth official release. Added "Retrying Failed DEW Requests by Using Exponential Backoff".
2022-11-11	This is the fourth official release. <ul style="list-style-type: none">• Added "Using CSMS to Prevent AK and SK Leakage".• Changed "Using CSMS for Applications to Log In to Databases" to "Using CSMS to Change Hard-coded Database Account Passwords".
2022-03-14	This is the third official release. Added sections "Using CSMS for Applications to Log In to Databases" and "Rotating Secrets".
2021-09-30	This is the second official issue. Added "Encrypting or Decrypting a Large Amount of Data".
2021-06-30	This is the first official release.