

Data Encryption Workshop

Best Practice

Issue 03
Date 2023-03-03



Copyright © Huawei Technologies Co., Ltd. 2023. 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.

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.....	12
1.2.3 Encrypting Data in OBS.....	12
1.2.4 Encrypting Data in EVS.....	13
1.2.5 Encrypting Data in IMS.....	16
1.2.6 Encrypting an RDS DB Instance.....	17
1.2.7 Encrypting a DDS DB Instance.....	18
2 General.....	20
2.1 Retrying Failed DEW Requests by Using Exponential Backoff.....	20
A Change History.....	22

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.

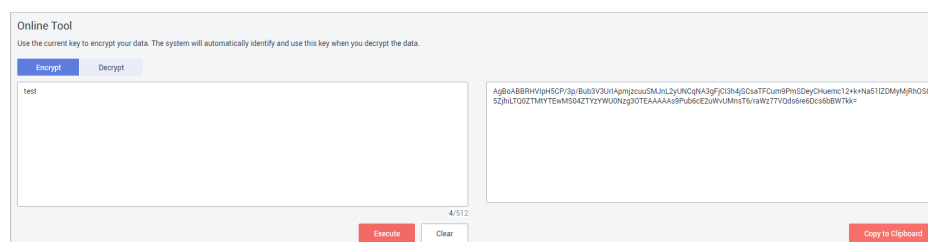
Encryption and Decryption Using Online Tools

- **Encrypting data**

Step 1 Click the alias of the desired CMK 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. For details, see [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**

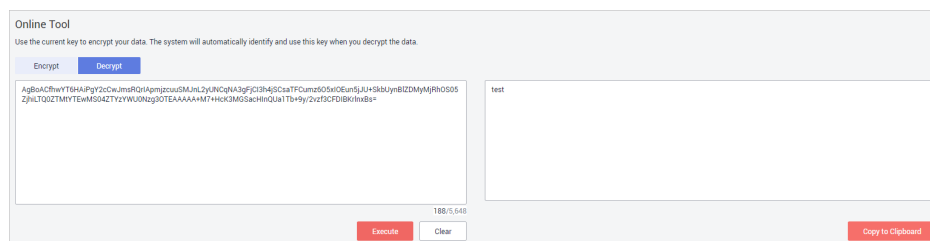
Step 4 You can click any CMK in **Enabled** status to go to the encryption and decryption page of the online tool.

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.
- However, if the CMK has been deleted, the decryption fails.

Figure 1-2 Decrypting data



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

 **NOTE**

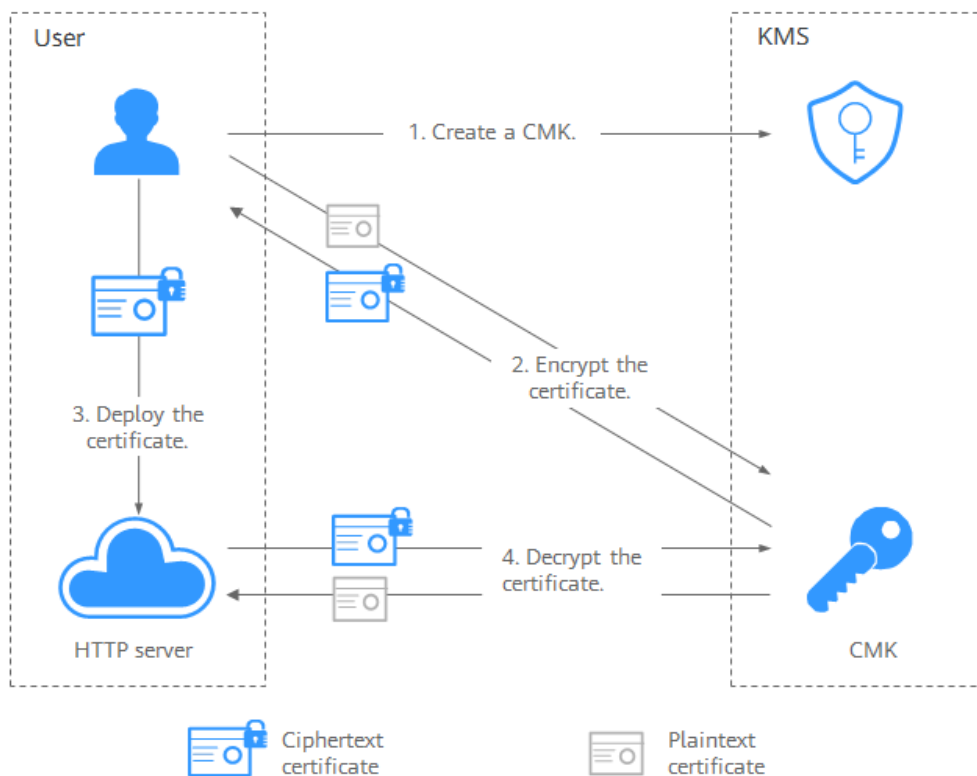
You can click **Copy to Clipboard** to copy the plaintext and save it in a local file.

----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.

1.1.2 Encrypting or Decrypting a Large Amount of Data

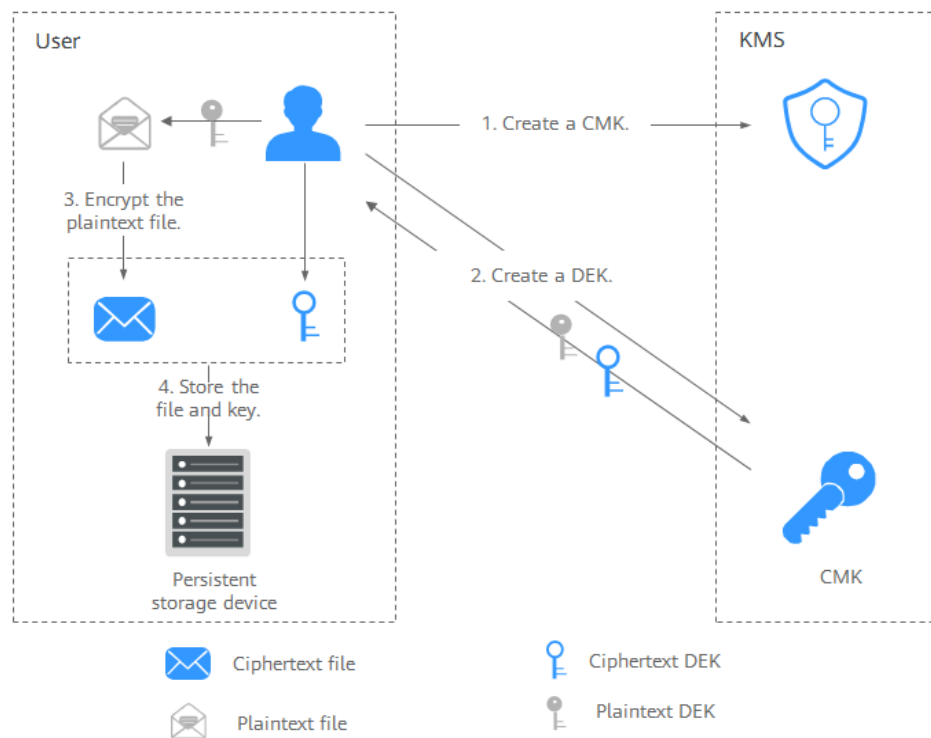
Scenario

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

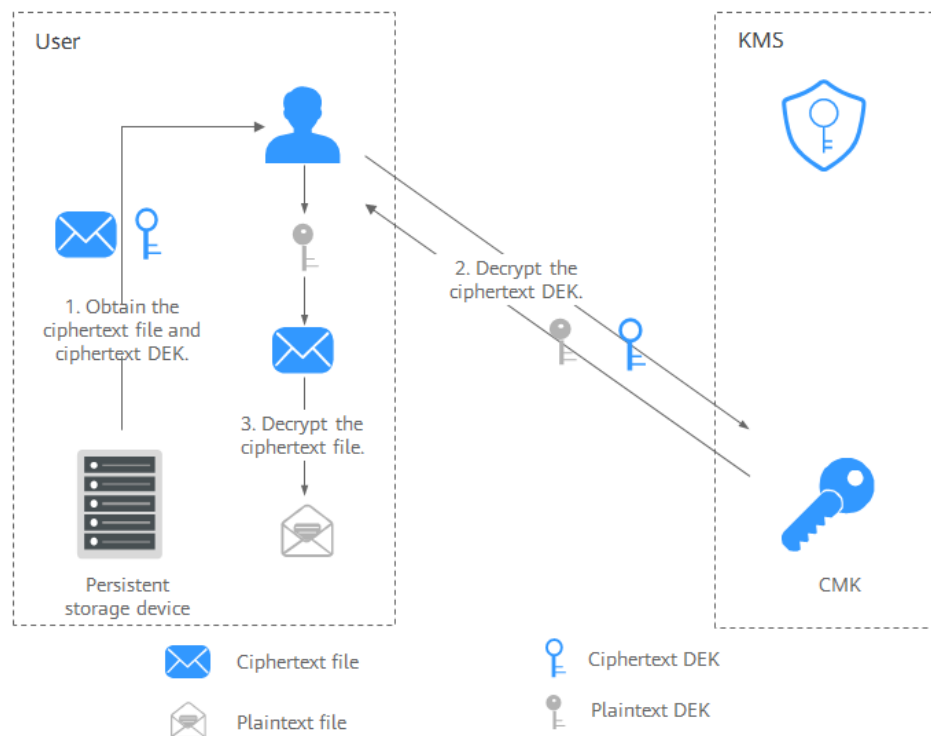
Figure 1-4 Encrypting a local file



The process is as follows:

- Create a CMK in KMS.
 - 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.
 - Use the plaintext DEK to encrypt a plaintext file, generating a ciphertext file.
 - 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:

- Read the ciphertext DEK and the ciphertext file from the permanent storage device or storage service.
- 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.
- 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
	This API is used to create a DEK.
	This API is used to decrypt a DEK with the specified CMK.

Encrypting a Local File

- Create a CMK on the management console. For details, see .
- Prepare basic authentication information.

- **ACCESS_KEY**: access key of the HUAWEI CLOUD account
 - **SECRET_ACCESS_KEY**: secret access key of the HUAWEI CLOUD account
 - **PROJECT_ID**: project ID of a HUAWEI CLOUD site. For details, see .
 - **KMS_ENDPOINT**: endpoint for accessing KMS. For details, see .
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 = "<AccessKey>";
    private static final String SECRET_ACCESS_KEY = "<SecretAccessKey>";
    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.ACCESS_KEY).withSk(SecretKey.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. Assemble the request message for creating a DEK.
        final CreateDatakeyRequest createDatakeyRequest = new
        CreateDatakeyRequest().withVersionId(KMS_INTERFACE_VERSION)
            .withBody(new
        CreateDatakeyRequestBody().withKeyId(keyId).withDatakeyLength(AES_KEY_BIT_LENGTH));

        // 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[GCM_IV_LENGTH];
        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 CLOUD account
 - **SECRET_ACCESS_KEY**: secret access key of the HUAWEI CLOUD account
 - **PROJECT_ID**: project ID of a HUAWEI CLOUD site. For details, see .
 - **KMS_ENDPOINT**: endpoint for accessing KMS. For details, see .
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 = "<AccessKey>";
    private static final String SECRET_ACCESS_KEY = "<SecretAccessKey>";
```

```
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 .

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.

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** Select the file to be uploaded and click **Open**.
- Step 5** Select **KMS encryption** and a key, as shown in [Figure 1-6](#). Then click **Upload**.

Figure 1-6 Encrypting an object to be uploaded

Upload Object [How to Upload a File Greater than 5 GB?](#) ✕

Storage Class: **Standard** | Infrequent Access | Archive

Optimized for frequently accessed (multiple times per month) data such as small and essential files that require low latency. The default storage class is the same as that of the bucket. You can change the storage class according to your actual needs. [Learn more](#)

Upload Object: **Note: If the bucket is not versioning-enabled, uploading a file/folder with the name that already exists in the bucket will replace the existing file/folder.**

1/100 Files Size 270.54 KB

Name	Size	Operation
[blurred]	270.54 KB	Remove

Encryption: Encrypts the file for secure storage. The encryption status of the encrypted file cannot be changed.

KMS encryption

Key name: Name of the primary 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 .

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 1-1](#) for more information.

Table 1-1 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. 	
Pending deletion		
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.

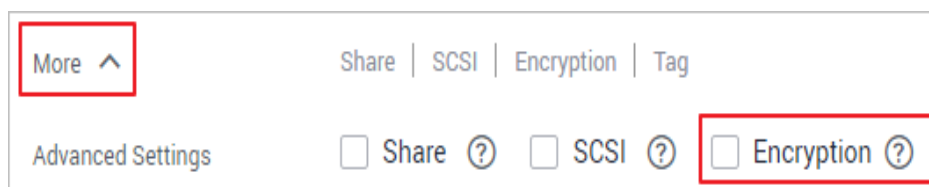
Using KMS to Encrypt a Disk (on the Console)

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 1-7 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.

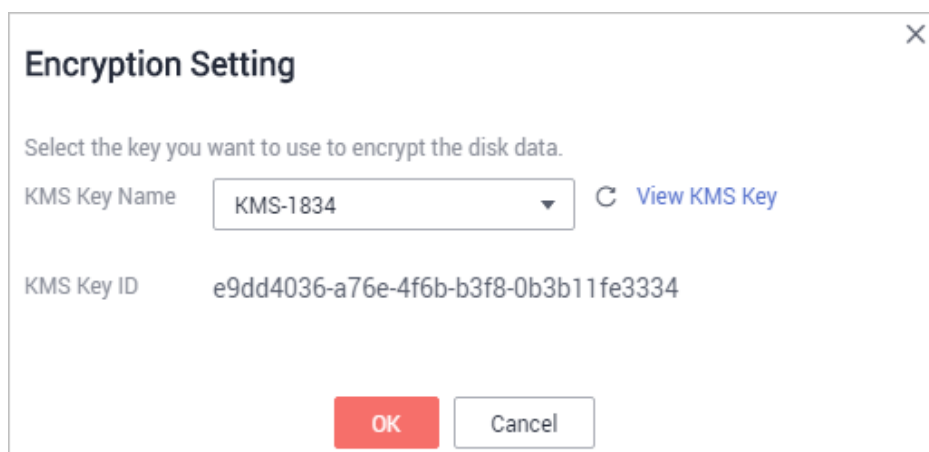
NOTE

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 1-8 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 .

Step 3 Configure other parameters for the disk. For details about the parameters, see .

----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.

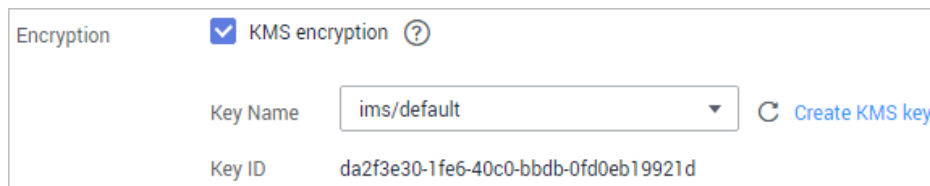
Using KMS to Encrypt a Private Image (on the Console)

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 1-9](#).

- 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 1-9 Encrypting data in IMS



Encryption	<input checked="" type="checkbox"/> KMS encryption ?
Key Name	ims/default ▼ Create KMS key
Key ID	da2f3e30-1fe6-40c0-bbdb-0fd0eb19921d

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 .
- e. Configure other parameters. For details about the parameters, see .

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

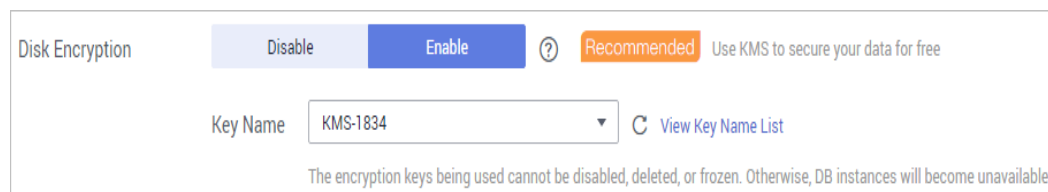
- The KMS Administrator right must be granted to the user in the region of RDS by using Identity and Access Management (IAM). For details about how to assign permissions to user groups, see "How Do I Manage User Groups and Grant Permissions to Them?" in *Identity and Access Management User Guide*.
- To use a user-defined key to encrypt objects to be uploaded, create a key using DEW. For details, see .
- 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 and .

Figure 1-10 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.

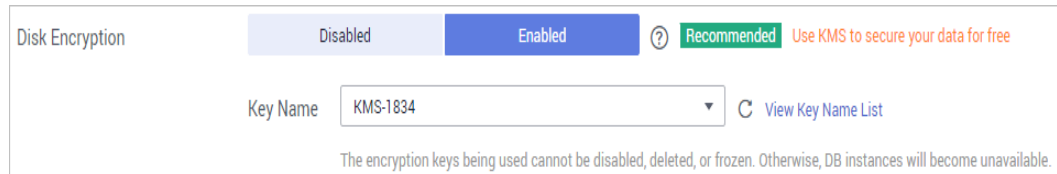
Restrictions

- 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 .
- 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 **Enabled** and use the key provided by KMS to encrypt the disk of the DB instance. For more information, see .

Figure 1-11 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*.

2 General

2.1 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 Cloud account. For details, see [How Do I Obtain an Access Key \(AK/SK\)?](#)
 - **SECRET_ACCESS_KEY**: Secret access key of the Huawei Cloud account. 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.
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 = "xxxx";

    private static final String SECRET_ACCESS_KEY = "xxxx";

    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 third official release. Added section "Encrypting Data in ECS".
2022-12-13	This is the second official issue. Added "Retrying Failed DEW Requests by Using Exponential Backoff".
2022-09-30	This is the first official release.