Cipher

API Reference

Issue	**01**
Date	**2019-05-15**

HiSilicon (Shanghai) Technologies Co., Ltd.

Address:	New R&D Center, 49 Wuhe Road, Bantian, Longgang District, Shenzhen 518129 P. R. China
Website:	http://www.hisilicon.com/en/
Email:	support@hisilicon.com

About This Document

Purpose

As a security algorithm module of the HiSilicon digital media processing platform, the cipher module provides the advanced encryption standard (AES), data encryption standard (DES), triple data encryption standard (3DES) symmetric encryption and decryption algorithms, hash and hashed message authentication code (HMAC) digest algorithms, random number generation, and Rivest-Shamir-Adleman (RSA) asymmetric algorithm for encryption.

For some chips, the AES algorithm supports CCM and GCM, and the digest algorithms support SHA224, 384, and 512, mainly used for encryption and decryption of audio and video streams as well as data validity verification.

NOTE

- Unless otherwise stated, Hi3559C V100 and Hi3559A V100 contents are consistent.
- Unless otherwise stated, Hi3516D V300, Hi3559 V200, Hi3556 V200, Hi3516A V300, and Hi3516C V500 contents are consistent.
- Unless otherwise stated, Hi3516E V200, Hi3516E V300, Hi3516D V200, and Hi3518E V300 contents are consistent.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3559A	V100ES
Hi3559A	V100
Hi3559C	V100
Hi3519A	V100
Hi3516C	V500
Hi3516D	V300
Hi3559	V200
Hi3556	V200
Hi3516A	V300
Hi3516E	V200

Product Name	Version
Hi3516E	V300
Hi3518E	V300
Hi3516D	V200

Intended Audience

This document is intended for:

- Technical support engineers
- Software development engineers

Change History

Changes between document issues are cumulative. The latest document issue contains all changes made in previous issues.

Issue 01 (2019-05-15)

This issue is the first official release, which incorporates the following changes:

Section 1.1 is modified.

Issue 00B09 (2019-03-30)

This issue is the ninth draft release, which incorporates the following changes:

The content related to the Hi3516D V200 is added.

Section 5.1 is modified.

Issue 00B08 (2019-03-01)

This issue is the eighth draft release, which incorporates the following changes:

The content related to the Hi3516A V300 is added.

Section 1.1 is modified.

In chapter 4, Table 4-1 is modified.

Issue 00B07 (2018-11-23)

This issue is the seventh draft release, which incorporates the following changes:

The contents related to the Hi3516E V200/Hi3516E V300/Hi3518E V300 are added.

Issue 00B06 (2018-10-30)

This issue is the sixth draft release

In chapter 2, HI_UNF_CIPHER_CreateHandle, HI_UNF_CIPHER_RsaPrivateDecrypt, and HI_UNF_CIPHER_RsaPrivateEncrypt are modified.

In chapter 3, the **Member** field of HI_UNF_CIPHER_CTRL_S is modified.

Issue 00B05 (2018-09-29)

This issue is the fifth draft release

Issue 00B04 (2018-09-04)

This issue is the fourth draft release, which incorporates the following changes:

The contents related to the Hi3516C V500/Hi3516D V300 are added.

Issue 00B03 (2018-04-28)

This issue is the third draft release, which incorporates the following changes:

The contents related to the Hi3519A V100 are added.

In chapter 4, Table 4-1 is modified.

Issue 00B02 (2018-01-15)

This issue is the second draft release, which incorporates the following changes:

The descriptions in the Hi3559A V100 and Hi3559C V100 are added.

Issue 00B01 (2017-05-27)

This issue is the first draft release.

Contents

About This Document ii

1 Introduction 1

1.1 Overview 1

1.2 Procedure 3

1.2.1 Encrypting/Decrypting a Single Data Packet 3

1.2.2 Encrypting/Decrypting Multiple Data Packets 6

1.2.3 Calculating the Hash Value 7

1.2.4 Calculating the HMAC Value 7

1.2.5 Random Number Generation 8

1.2.6 Encrypting/Decrypting Data by Using RSA 8

1.2.7 RSA Signing and Signature Verification Process 9

1.2.8 Encrypting/Decrypting Data by Using CCM/GCM 10

2 API Reference 12

3 Data Structures 35

4 Error Codes 59

5 Proc Debugging Information 62

5.1 Cipher Status 62

Figures

Figure 1-1 Scenario 1 4
Figure 1-2 Scenario 2 5

Tables

Table 4-1 Error codes for the cipher module 59

Table 4-2 Parameter description 63

1 Introduction

1.1 Overview

As a security algorithm module of the HiSilicon digital media processing platform, the cipher module provides the AES, DES, and 3DES symmetric encryption and decryption algorithms, RSA asymmetric encryption and decryption algorithm, and random number generation. The module also supports hash and HMAC digest algorithms for encryption and decryption of audio and video streams as well as user validity verification. The functions are divided as follows.

Symmetric Encryption and Decryption Algorithm

- AES: supports the ECB, CBC, CFB, OFB, CTR, CCM, GCM, and other operating modes. **Note that the CCM and GCM modes are not supported by Hi3516C V500/Hi3516D V300/Hi3516A V300/Hi3516E V200/Hi3516E V300/Hi3518E V300/Hi3516D V200.** In these modes, a tag value needs to be obtained after encryption and decryption. Other modes are supported by the chip.
- DES/3DES: supports the ECB, CBC, CFB, and OFB modes. The CFB and OFB modes support 1-/8-/64- bit width. **Note that the DES/3DES algorithm is not supported by Hi3516E V200/Hi3516EV300/Hi3518E V300/Hi3516D V200.**

Except CTR, CCM and GCM in the preceding algorithms, the data length of other algorithms and modes must be aligned by block size. N and A of CCM or GCM must depend on software according to the standard to encapsulate each field into block-size-aligned data blocks. In each working mode, the encryption and decryption algorithms can implement multiple blocks or a single block for each time. You can apply for a maximum of 7 channels.

A maximum of two channels can be applied for Hi3516E V200, Hi3516E V300, Hi3516D V200, and Hi3518E V300.

The DES/3DES algorithm is insecure, and even the 3DES ($K1 \neq K2 \neq K3$) algorithm is not as secure as the AES algorithm (128 bits or above). Therefore, the more secure AES algorithm is recommended.

Asymmetric Encryption and Decryption Algorithm

RSA: supports 1024-/2048-/3072-/4096- bit width keys. **Note that the 3072-bit width is not supported by Hi3516C V500/Hi3516D V300/Hi3516A V300/Hi3516E V200/Hi3516E V300/Hi3518E V300.** Other bit widths are supported by the chip.

The 1024-bit or below RSA key is known as an insecure algorithm in the industry and should be forbidden.

Random Number Generation

RNG: supports DRGB to obtain random numbers at a higher rate.

Digest Algorithm

HASH: supports SHA1, SHA224, SHA256, SHA384, SHA512 and SM3, supports HMAC1, HMAC224, HMAC256, HMAC384 and HMAC512, and supports multiple channels of software. You can apply for a maximum of 8 channels.

Hi3516E V200, Hi3516E V300, Hi3516D V200, and Hi3518E V300 do not support SHA384, SHA512, HMAC384, and HMAC512.

The SHA1 algorithm is insecure and cannot be used scenarios where digital signatures are generated. The SHA2 (256 bits or above) algorithm is recommended.

The algorithms and operation modes involved in the preceding functional modules comply with the following standards:

- The implementation of the AES algorithm complies with the Federal Information Processing Standard (FIPS) 197. The supported operation modes comply with the following standards:
  - The ECB, CBC, 1/8/128-CFB, 128-OFB, CTR modes comply with the National Institute of Standards and Technology (NIST) special 800-38a standard.
  - The CCM mode complies with the NIST special 800-38c standard.
  - The GCM mode complies with the NIST special 800-38d standard.
- The implementation of the DES or 3DES algorithm complies with the FIPS-46-3 standard. The operation modes comply with the following standards:
  - Supports the ECB, CBC, 1/8/64-CFB, and 1/8/64-OFB modes of operation and complies with the FIPS 81 standard.
- The RSA supports encryption using public keys and decryption using private keys, encryption using private keys and decryption using public keys, signature, verification and other functions. The data filling methods in different modes comply with the public-key cryptography standard PKCS#1.
  - The RSA encryption and decryption modes include the following: NO_PADDING, BLOCK_YTPE_0, BLOCK_YTPE_1, BLOCK_YTPE_2, RSAES_OAEP_SHA1, RSAES_OAEP_SHA224, RSAES_OAEP_SHA256, RSAES_OAEP_SHA384, RSAES_OAEP_SHA512, RSAES_PKCS1_V1_5 and others.
  - The RSA signature and verification modes include the following: RSASSA_PKCS1_V15_SHA1, RSASSA_PKCS1_V15_SHA224, RSASSA_PKCS1_V15_SHA256, RSASSA_PKCS1_V15_SHA384, RSASSA_PKCS1_V15_SHA512, RSASSA_PKCS1_PSS_SHA1, RSASSA_PKCS1_PSS_SHA224, RSASSA_PKCS1_PSS_SHA256, RSASSA_PKCS1_PSS_SHA384, RSASSA_PKCS1_PSS_SHA512 and others.

NOTE

- Symmetric encryption and decryption algorithms mainly include the following modes of operation: Electronic CodeBook Mode (ECB), Cipher-Block Chaining Mode (CBC), Cipher Feedback Mode (CFB), Output Feedback Mode (OFB), Counter Mode (CTR), Counter with CBC-MAC (CCM), and Galois/Counter Mode (GCM). CCM and GCM generate a check-value of a cipher-based message authentication code (CMAC) during encryption and decryption. The decryption is correct only when the encryption CMAC to and decryption CMAC are the same, which is usually used in the fields where both encryption and authentication are needed. For details about the algorithms, see the related documents.
- In block cryptography, message blocks for encryption and decryption can be divided into several blocks: In ECB mode, each block is independently encrypted/decrypted and the blocks are independent of each other. In non-ECB mode, blocks are dependent on each other, and the initialization vector (IV) is used in the first block to ensure the uniqueness of each message.

1.2 Procedure

1.2.1 Encrypting/Decrypting a Single Data Packet

Scenario

A single data packet is encrypted or decrypted. When a stream data segment in a physical memory needs to be encrypted or decrypted, you can obtain the physical address of the memory and then call the cipher module to perform encryption or decryption at the user layer.

Working Processes

To encrypt or decrypt data by using the DES, 3DES, or AES algorithm, perform the following steps:

Step 1 Initialize a cipher device by calling HI_UNF_CIPHER_Init.

Step 2 Create a cipher channel and obtain the cipher handle by calling HI_UNF_CIPHER_CreateHandle.

Step 3 Configure the cipher control information (including the key, initialization vector, encryption algorithm, and working mode) by calling HI_UNF_CIPHER_ConfigHandle or HI_UNF_CIPHER_ConfigHandleEx.

Step 4 Encrypt or decrypt data by calling one of the following APIs:

- Call HI_UNF_CIPHER_Encrypt to encrypt a single packet.
- Call HI_UNF_CIPHER_Decrypt to decrypt a single packet.

Step 5 If the counter with CBC-MAC (CCM) mode or Galois/Counter Mode (GCM) is used, call HI_UNF_CIPHER_GetTag to obtain the tag value. Otherwise, go to step 6.

Step 6 Destroy the cipher handle by calling HI_UNF_CIPHER_DestroyHandle.

Step 7 Stop the cipher device by calling HI_UNF_CIPHER_Deinit.

----End

Notes

Note the following when using the cipher module:

This interface supports the AES, DES/3DES, GCM, CMM symmetric encryption and decryption algorithms.

The algorithms support the ECB, CBC, CFB, OFB and CTR operations modes.

- A cipher handle must be obtained before encryption or decryption. Release the handle if it is not used for a long time. You are advised to obtain a handle for encryption and a handle for decryption. Each handle is used only for encryption or decryption.
- Only data in a consecutive physical memory space can be encrypted or decrypted. (You can obtain a physical memory by calling the HiSilicon interface HI_MMZ_New, and then map the physical memory to a virtual address by calling HI_MMZ_Map.)
- The cipher module transmits data in direct memory access (DMA) mode. Therefore, when data is encrypted or decrypted through the HI_UNF_CIPHER_Encrypt or HI_UNF_CIPHER_Decrypt interface, the input address parameter is the physical address of the data.
- The source addresses and destination addresses for encryption and decryption can be the same. That is, data can be encrypted and decrypted at the same address (the same buffer is used for storing both the ciphertext and plaintext).
- During symmetric data encryption or decryption, the length of each data packet must be less than 1 MB. If the data length is greater than or equal to 1 MB, the data must be split into multiple data packets before encryption or decryption.
- During cipher encryption/decryption in non-ECB mode, the IV must be used.
- The IV needs to be configured in the following two scenarios (taking data block decryption as an example):

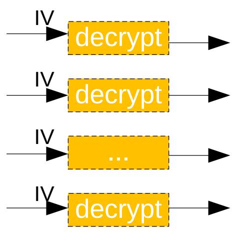
  Scenario 1:

  The IV needs to be updated each time the cipher module is called. In this case, set **stChangeFlags.bit1IV** to **2** and properly configure the IV value.

  For details about the API calling sequence, see the following code:

```
HI_UNF_CIPHER_ConfigHandle()   //should set stChangeFlags.bit1IV = 2 and
update u32IV
HI_UNF_CIPHER_Decrypt()
HI_UNF_CIPHER_ConfigHandle()   //should set stChangeFlags.bit1IV = 2 and
update u32IV
HI_UNF_CIPHER_Decrypt()
….
HI_UNF_CIPHER_ConfigHandle()   //should set stChangeFlags.bit1IV = 2 and
update u32IV
HI_UNF_CIPHER_Decrypt()
```

Figure 1-1 Scenario 1



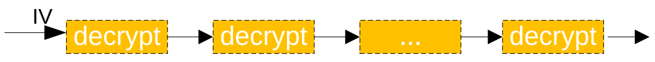
Scenario 2:

The IV needs to be updated only when the cipher module is called for the first time. In this case, set **stChangeFlags.bit1IV** to **1** and properly configure the IV value.

For details about the API calling sequence, see the following code:

```
HI_UNF_CIPHER_ConfigHandle()   //should set stChangeFlags.bit1IV = 1 and
update u32IV
HI_UNF_CIPHER_Decrypt()HI_UNF_CIPHER_Decrypt()….
HI_UNF_CIPHER_Decrypt()
```

Figure 1-2 Scenario 2



The IV value must be configured based on the actual scenario.

- The IVs used for single-packet encryption and decryption can be inherited. After a cipher channel is created and its attributes are configured (assuming that the IVs are required for the configured working mode), the IVs are used in turn when the single-packet encryption or decryption API is called.

  For example, you need to encrypt data 0 and data 1 in sequence and the IVs are a, b, c, and d. After data 0 is encrypted, the last block of data 0 uses b. When data 1 is encrypted, the first block of data 1 is encrypted by using c. If the APIs are called continuously, the IVs are used in the sequence of d, a, b, c, d, ….

  Ensure that the IVs are used in the same sequence during encryption and decryption. If the CIPHER control information is reconfigured, the first IV is used.

- If the member **bKeyByCA** of the data structure HI_UNF_CIPHER_CTRL_S is set to **HI_FALSE**, the normal mode is used, indicating that the key needs to be manually configured for data encryption and decryption. For example:

  ```
  memcpy(CipherCtrl.u32Key, u8KeyBuf, 32);
  ```

  For details, see the samples of the cipher module.

- If the member **bKeyByCA** is set to **HI_TRUE**, the embedded Key in the chip is used for data encryption and decryption.
- AES-CCM and AES-GCM can be configured through only HI_UNF_CIPHER_ConfigHandleEx. CCM and GCM need to obtain a tag value after computation. The decryption is successful only when the decryption tag value and the encryption tag value are the same.

Sample

For details about the sample, see **sample_cipher.c** in the SDK.

1.2.2 Encrypting/Decrypting Multiple Data Packets

Scenario

Multiple data packets are encrypted or decrypted. When multiple stream data segments in a physical memory need to be encrypted or decrypted, you can obtain the physical address of the memory and then call the cipher module to perform encryption or decryption at the user layer.

Working Processes

To encrypt or decrypt data by using the symmetric DES, 3DES, or AES algorithm, perform the following steps:

Step 1 Initialize a cipher device by calling HI_UNF_CIPHER_Init.

Step 2 Create a cipher channel and obtain the cipher handle by calling HI_UNF_CIPHER_CreateHandle.

Step 3 Configure the cipher control information (including the key, initialization vector, encryption algorithm, and working mode) by calling HI_UNF_CIPHER_ConfigHandle or HI_UNF_CIPHER_ConfigHandleEx.

Step 4 Encrypt or decrypt data by calling one of the following APIs:

- Call HI_UNF_CIPHER_EncryptMulti to encrypt multiple packets.
- Call HI_UNF_CIPHER_DecryptMulti to decrypt multiple packets.

Step 5 Destroy the cipher handle by calling HI_UNF_CIPHER_DestroyHandle.

Step 6 Stop the cipher device by calling HI_UNF_CIPHER_Deinit.

----End

Notes

- During multi-packet encryption or decryption, at most 128 packets can be encrypted or decrypted simultaneously.
- When multiple packets are encrypted or decrypted, each packet is calculated by using the vector configured by HI_UNF_CIPHER_ConfigHandle or HI_UNF_CIPHER_ConfigHandleEx. The IV scope is configurable. The vector calculation result of the previous packet can be used as the IV of the next packet. Or each packet is calculated independently (the result of the previous function invocation does not affect that of the next function invocation).
- For other notes, refer to chapter 1.2.1 "Encrypting/Decrypting a Single Data Packet."

Sample

For details about the sample, see **sample_multicipher.c** in the SDK.

1.2.3 Calculating the Hash Value

Scenario

The SHA1, SHA224, SHA256, SHA384 or SHA512 algorithm can be used to calculate the hash value of data.

Working Processes

Step 1 Initialize a cipher device by calling HI_UNF_CIPHER_Init.

Step 2 Create a hash channel, obtain the hash handle and select the hash algorithm by calling HI_UNF_CIPHER_HashInit.

Step 3 Input data and calculate the hash value by each data block in sequence by calling HI_UNF_CIPHER_HashUpdate.

Step 4 Repeat step 3 until the digest calculation is complete.

Step 5 Finish the input after digest calculation is complete and obtain the calculation result by calling HI_UNF_CIPHER_HashFinal.

Step 6 Stop the cipher device by calling HI_UNF_CIPHER_Deinit.

----End

Notes

This working process supports multiple channels for software. Multiple hash calculation tasks can be implemented at the same time, that is, when a hash calculation task is started in Step 2 and has not yet been completed (that is, before Step 5 is executed), a new channel can be applied to start another hash calculation task until there is no channel available.

At most eight hash software channels are supported. The eight channels can be enabled at the same time. However, only one channel can implement the calculation at the same time.

Sample

For details about the sample, see **sample_hash.c** in the SDK.

1.2.4 Calculating the HMAC Value

Scenario

The SHA1, SHA224, SHA256, SHA384 or SHA512 can be used to calculate the HMAC value of data.

Working Processes

To calculate the HMAC value, perform the following steps:

Step 1 Initialize a cipher device by calling HI_UNF_CIPHER_Init.

Step 2 Select the hash algorithm, configure the key used for HMAC calculation, and initialize the hash module by calling HI_UNF_CIPHER_HashInit.

Step 3 Input data by calling HI_UNF_CIPHER_HashUpdate. The data can be input by block.

Step 4 Finish the input and output the HMAC value by calling HI_UNF_CIPHER_HashFinal.

Step 5 Deinitialize the cipher device by calling HI_UNF_CIPHER_Deinit.

----End

Notes

This working process supports multiple channels for software. Multiple HMAC calculation tasks can be implemented at the same time, that is, when an HMAC calculation task is started in Step 2 and has not yet been completed (that is, before Step 5 is executed), a new channel can be applied to start another HMAC calculation task until there is no channel available.

HMAC and hash calculation tasks share eight software channels. Eight channels can be enabled at the same time. However, only one channel can implement the calculation at the same time.

Sample

For details about the sample, see **sample_hash.c** in the SDK.

1.2.5 Random Number Generation

Scenario

Obtain the true random numbers generated by hardware.

Working Processes

To generate the random number, perform the following steps:

Step 1 Initialize a cipher device by calling HI_UNF_CIPHER_Init.

Step 2 Obtain a 32-bit random number by calling HI_UNF_CIPHER_GetRandomNumber.

Step 3 Stop the cipher device by calling HI_UNF_CIPHER_Deinit.

----End

Notes

None

Sample

For details about the sample, see **sample_rng.c** in the SDK.

1.2.6 Encrypting/Decrypting Data by Using RSA

Scenario

Encrypt or decrypt data by using the RSA asymmetric algorithm. When the public key is used to encrypt data, the private key must be used to decrypt the data, and vice versa.

For details about the algorithm, see rfc3447. RSA Cryptography Specifications.

Working Processes

To encrypt or decrypt data by using the RSA asymmetric algorithm, perform the following steps:

Step 1 Initialize a cipher device by calling HI_UNF_CIPHER_Init.

Step 2 Encrypt or decrypt data, verify the signature, or generate the key pair by calling one of the following APIs based on the used key:

- Call HI_UNF_CIPHER_RsaPublicEncrypt to encrypt a public key.
- Call HI_UNF_CIPHER_RsaPrivateDecrypt to decrypt a private key.
- Call HI_UNF_CIPHER_RsaPrivateEncrypt to encrypt a private key.
- Call HI_UNF_CIPHER_RsaPublicDecrypt to decrypt a public key.
- Call HI_UNF_CIPHER_RsaSign to sign a private key.
- Call HI_UNF_CIPHER_RsaVerify to verify a public key.

Step 3 Stop the cipher device by calling HI_UNF_CIPHER_Deinit.

----End

Notes

The bit width of the RSA key can be 1024, 2048, 3072 or 4096 bits. According to the rule of the RSA algorithm, the size of the plaintext and ciphertext must be smaller than that of the public key *N*. Therefore, the length of the data to be encrypted/decrypted must be less than or equal to the length of the key. Typically, 0 is added to the upper bits of the data to be encrypted/decrypted, so that its length becomes equal to that of the public key *N* but its value is smaller than *N*. The PKCS#1 standard defines the data stuffing methods, which are Block Type 0, Block Type 1, Block Type 2, RSAES-OAEP, and RSAES-PKCS1-v1_5.

Sample

For details about the sample, see **sample_rsa_enc.c** in the SDK.

1.2.7 RSA Signing and Signature Verification Process

Scenario

To perform RSA signing and signature verification for data, use the private key for data signing and the public key for signature verification.

For details about the algorithm, see rfc3447. RSA Cryptography Specifications.

Working Processes

To sign the data using the RSA asymmetric algorithm or verify the signature, perform the following steps:

Step 1 Initialize a cipher device by calling HI_UNF_CIPHER_Init.

Step 2 Sign the data or verify the signature by calling the following interfaces:

- Call HI_UNF_CIPHER_RsaSign to sign a private key.
- Call HI_UNF_CIPHER_RsaVerify to verify a public key.

Step 3 Stop the cipher device by calling HI_UNF_CIPHER_Deinit.

----End

Notes

The bit width of the RSA key can be 1024, 2048, 3072 or 4096 bits. According to the rule of the RSA algorithm, the size of the plaintext and cipher text must be smaller than that of the public key. Therefore, the length of the data to be encrypted/decrypted must be less than or equal to the length of the key. Typically, calculate the HASH value of the data to be encrypted/decrypted, stuff the HASH value to a data equal to the public key N in length but smaller than *N* in value, and then encrypt the data. The PKCS#1 standard defines two data stuffing methods, which are RSASSA-PSS and RSAES-PKCS1-v1_5.

Sample

For details about the sample, see **sample_rsa_sign.c** in the SDK.

1.2.8 Encrypting/Decrypting Data by Using CCM/GCM

Scenario

Encrypt or decrypt data by using CCM. For details about the algorithm, see SP800-38C_updated-July20_2007_CCM. The CCM Mode for Authentication and Confidentiality.

Encrypt or decrypt data by using GCM. For details about the algorithm, see SP-800-38D-GCM. Galois/Counter Mode (GCM) and GMAC.

Working Processes

To encrypt or decrypt data by using the CCM/GCM symmetric algorithm, perform the following steps:

Step 1 Initialize a cipher device by calling HI_UNF_CIPHER_Init.

Step 2 Obtain the cipher handle by calling HI_UNF_CIPHER_CreateHandle.

Step 3 Configure cipher parameters by calling HI_UNF_CIPHER_ConfigHandleEx.

Step 4 Encrypt or decrypt data by calling the following interfaces:

- Call HI_UNF_CIPHER_Encrypt to encrypt data.
- Call HI_UNF_CIPHER_Decrypt to decrypt data.

Step 5 Obtain the TAG data of CCM/GCM by calling HI_UNF_CIPHER_GetTag.

Step 6 Release the cipher handle by calling HI_UNF_CIPHER_DestroyHandle.

Step 7 Deinitialize the cipher device by calling HI_UNF_CIPHER_Deinit.

----End

Notes

- The length of the CCM/GCM private key can be 128, 192 or 256 bits. The encryption result is correct only when the tag value generated by CCM/GCM decryption is the same as that by CCM/GCM encryption.
- The AES-CCM mode consists of AES CTR and AES CBC modes, which ensures data confidentiality and integrity.
  - According to the CCM algorithm principle, the IV length **u32IVLen** can be {7, 8, 9, 10, 11, 12, 13} bytes, IV stores nonce (N) of the algorithm standard, and the length of encrypted data is represented by **n** bytes. They must meet the following requirements: **u32IVLen** + **n** = 15. Therefore, when **u32IVLen** is 13, **n** is 2. The maximum length of encrypted data is 65,536 bytes, and the rule applies.
  - The values of the vector N and associated data A for CCM encryption must be the same as those for CCM decryption.
- The AES-GCM mode consists of AES CTR and GHASH modes, which ensures data confidentiality and integrity.
  - According to the GCM algorithm principle, the length **u32IVLen** of the GCM IV can be [1, 16].
  - The value of the associated data A for GCM encryption must be the same as that for GCM decryption.

Sample

For details about the sample, see **sample_cipher.c** in the SDK.

2 API Reference

The cipher module provides the following APIs:

- HI_UNF_CIPHER_Init: Initializes a cipher module.
- HI_UNF_CIPHER_Deinit: Deinitializes a cipher module.
- HI_UNF_CIPHER_Open: Opens a cipher module.
- HI_UNF_CIPHER_Close: Close a cipher module.
- HI_UNF_CIPHER_CreateHandle: Creates a cipher handle in a channel.
- HI_UNF_CIPHER_DestroyHandle: Destroys an existing cipher handle.
- HI_UNF_CIPHER_ConfigHandle: Configures the cipher control information.
- HI_UNF_CIPHER_ConfigHandleEx: Configures the cipher control information (extended).
- HI_UNF_CIPHER_GetHandleConfig: Obtains the configuration information of a cipher channel.
- HI_UNF_CIPHER_Encrypt: Encrypts a single data packet.
- HI_UNF_CIPHER_Decrypt: Decrypts a single data packet.
- HI_UNF_CIPHER_EncryptVir: Encrypts data.
- HI_UNF_CIPHER_DecryptVir: Decrypts data.
- HI_UNF_CIPHER_EncryptMulti: Encrypts multiple data packets.
- HI_UNF_CIPHER_DecryptMulti: Decrypts multiple data packets.
- HI_UNF_CIPHER_HashInit: Initializes the hash and HMAC calculation.
- HI_UNF_CIPHER_HashUpdate: Inputs data for hash and HMAC calculation.
- HI_UNF_CIPHER_HashFinal: Outputs the hash and HMAC calculation result.
- HI_UNF_CIPHER_GetRandomNumber: Obtains random numbers.
- HI_UNF_CIPHER_GetTag: Obtains the tag value.
- HI_UNF_CIPHER_RsaPublicEncrypt: Encrypts a plaintext by using the public key.
- HI_UNF_CIPHER_RsaPrivateDecrypt: Decrypts a ciphertext by using the private key.
- HI_UNF_CIPHER_RsaPrivateEncrypt: Encrypts a plaintext by using the private key.
- HI_UNF_CIPHER_RsaPublicDecrypt: Decrypts a ciphertext by using the public key.
- HI_UNF_CIPHER_RsaSign: Signs user data by using the private key.
- HI_UNF_CIPHER_RsaVerify: Verifies the validity and integrity of user data by using the public key.

- HI_UNF_CIPHER_KladEncryptKey: Encrypts the transparent key by using the Klad.

HI_UNF_CIPHER_Init

[Description]

Initializes a cipher module.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_Init(HI_VOID);
```

[Parameter]

None

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

See **sample_cipher.c**

HI_UNF_CIPHER_Deinit

[Description]

Deinitializes a cipher module.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_Deinit(HI_VOID);
```

[Parameter]

None

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

See **sample_cipher.c**

HI_UNF_CIPHER_Open

[Description]

Opens a cipher module.

[Syntax]

```
#define HI_UNF_CIPHER_Open(HI_VOID) HI_UNF_CIPHER_Init(HI_VOID);
```

[Parameter]

None

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

See **sample_cipher.c**

HI_UNF_CIPHER_Close

[Description]

Closes a cipher module.

[Syntax]

```
#define HI_UNF_CIPHER_Close(HI_VOID) HI_UNF_CIPHER_Deinit(HI_VOID);
```

[Parameter]

None

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

See **sample_cipher.c**

HI_UNF_CIPHER_CreateHandle

[Description]

Creates a cipher handle in a channel.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_ CreateHandle(HI_HANDLE* phCipher, const
HI_UNF_CIPHER_ATTS_S *pstCipherAttr);
```

[Parameter]

Parameter	Description	Input/Output
phCipher	Pointer to the cipher handle	Output
pstCipherAtt	Pointer to the cipher attribute	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

- The **phCipher** and **pstCipherAttr** parameters cannot be null.
- The handle **phCipher** is the input for data encryption and decryption.
- A maximum of seven cipher channels are supported.
- After a cipher channel is used, it must be destroyed.

[Example]

See **sample_cipher.c**.

HI_UNF_CIPHER_DestroyHandle

[Description]

Destroys an existing cipher handle.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_DestroyHandle(HI_HANDLE hCipher);
```

[Parameter]

Parameter	Description	Input/Output
hCipher	Cipher handle	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

After a created channel is used, it must be destroyed

[Example]

See **sample_cipher.c**

HI_UNF_CIPHER_ConfigHandle

[Description]

Configures the cipher control information. For details, see the data structure HI_UNF_CIPHER_CTRL_S.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_ConfigHandle(HI_HANDLE hCipher, HI_UNF_CIPHER_CTRL_S*
pstCtrl);
```

[Parameter]

Parameter	Description	Input/Output
hCipher	Cipher handle	Input
pstCtrl	Pointer to the control information	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

The pointer to the control information cannot be null.

[Example]

See **sample_cipher.c**

HI_UNF_CIPHER_ConfigHandleEx

[Description]

Configures the cipher control information (extended). For details, see the data structure HI_UNF_CIPHER_CTRL_EX_S.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_ConfigHandleEx(HI_HANDLE hCipher,
HI_UNF_CIPHER_CTRL_EX_S* pstExCtrl);
```

[Parameter]

Parameter	Description	Input/Output
hCipher	Cipher handle	Input
pstExCtrl	Pointer to the control extended information	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

The pointer to the cipher control extended information cannot be null.

[Example]

See **sample_cipher.c**.

HI_UNF_CIPHER_GetHandleConfig

[Description]

Obtains the configuration information of a cipher channel.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_GetHandleConfig(HI_HANDLE hCipher,
HI_UNF_CIPHER_CTRL_S* pstCtrl);
```

[Parameter]

Parameter	Description	Input/Output
hCipher	Cipher handle	Input
pstCtrl	Configuration information of a cipher channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

None

HI_UNF_CIPHER_Encrypt

[Description]

Encrypts a single data packet.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_Encrypt(HI_HANDLE hCipher, HI_U32 u32SrcPhyAddr, HI_U32
u32DestPhyAddr, HI_U32 u32ByteLength);
```

[Parameter]

Parameter	Description	Input/Output
hCipher	Cipher handle	Input
u32SrcPhyAddr	Physical address of the source data (data to be encrypted)	Input
u32DestPhyAddr	Physical address of the encrypted data	Input
u32ByteLength	Data length, in byte	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

- A cipher handle must be created before you call this API.
- This API can be called repeatedly.
- The data length must be greater than or equal to 16 bytes.

[Example]

See **sample_cipher.c**

HI_UNF_CIPHER_Decrypt

[Description]

Decrypts a single data packet.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_Decrypt(HI_HANDLE hCipher, HI_U32 u32SrcPhyAddr, HI_U32
u32DestPhyAddr, HI_U32 u32ByteLength);
```

[Parameter]

Parameter	Description	Input/Output
hCipher	Cipher handle	Input
u32SrcPhyAddr	Physical address of the source data (data to be decrypted)	Input
u32DestPhyAddr	Physical address of the decrypted data	Input
u32ByteLength	Data length, in byte	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

- A cipher handle must be created before you call this API.
- This API can be called repeatedly.
- The data length must be greater than or equal to 16 bytes.

[Example]

See **sample_cipher.c**

HI_UNF_CIPHER_EncryptVir

[Description]

Encrypts data.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_EncryptVir(HI_HANDLE hCipher, const HI_U8 *pu8SrcData,
HI_U8 *pu8DestData, HI_U32 u32ByteLength);
```

[Parameter]

Parameter	Description	Input/Output
hCipher	Cipher handle	Input
*pu8SrcData	Virtual address of the source data (data to be encrypted)	Input
*pu8DestData	Virtual address of the encryption result	Output
u32ByteLength	Data length, in bytes.	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files:**hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

- A cipher handle must be created before you call this API.
- This API can be called repeatedly.
- The data length in CTR, CCM, or GCM mode can be arbitrary. For other modes, the data length must be aligned by block.

[Example]

See **sample_cipher.c**.

HI_UNF_CIPHER_DecryptVir

[Description]

Decrypts data.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_DecryptVir(HI_HANDLE hCipher, const HI_U8 *pu8SrcData,
HI_U8 *pu8DestData, HI_U32 u32ByteLength);
```

[Parameter]

Parameter	Description	Input/Output
hCipher	Cipher handle	Input
*pu8SrcData	Virtual address of the source data (data to be decrypted)	Input
*pu8DestData	Virtual address of the decryption result	Output
u32ByteLength	Data length, in bytes.	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files:**hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

- A cipher handle must be created before you call this API.
- This API can be called repeatedly.
- The data length in CTR, CCM, or GCM mode can be arbitrary. For other modes, the data length must be aligned by block.

[Example]

See **sample_cipher.c**.

HI_UNF_CIPHER_EncryptMulti

[Description]

Encrypts multiple data packets.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_EncryptMulti(HI_HANDLE hCipher, None
 *pstDataPkg, HI_U32 u32DataPkgNum);
```

[Parameter]

Parameter	Description	Input/Output
hCipher	Cipher handle	Input
*pstDataPkg	Data packet to be encrypted	Input

Parameter	Description	Input/Output
u32DataPkgNum	Number of data packets to be encrypted	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

- A cipher handle must be created before you call this API.
- This API can be called repeatedly.
- A maximum of 128 data packets can be encrypted at a time.
- When multiple data packets are encrypted/decrypted, each data packet is calculated by using the vector configured by HI_UNF_CIPHER_ConfigHandle. Each data packet is calculated independently, that is, the calculation result of the previous data packet is not used in the calculation of the next data packet, and the calculation result of the previous function invocation does not affect that of the next function invocation.

[Example]

See **sample_multiciphe.c**.

HI_UNF_CIPHER_DecryptMulti

[Description]

Decrypts multiple data packets.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_DecryptMulti(HI_HANDLE hCipher, None
 *pstDataPkg, HI_U32 u32DataPkgNum);
```

[Parameter]

Parameter	Description	Input/Output
hCipher	Cipher handle	Input
*pstDataPkg	Data packet to be decrypted	Input
u32DataPkgNum	Number of data packets to be decrypted	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

- A cipher handle must be created before you call this API.
- This API can be called repeatedly.
- A maximum of 128 data packets can be decrypted at a time.
- When multiple data packets are encrypted/decrypted, each data packet is calculated by using the vector configured by HI_UNF_CIPHER_ConfigHandle. Each data packet is calculated independently, that is, the calculation result of the previous data packet is not used in the calculation of the next data packet, and the calculation result of the previous function invocation does not affect that of the next function invocation.

[Example]

See **sample_multiciphe.c**.

HI_UNF_CIPHER_HashInit

[Description]

Initializes the hash module.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_HashInit(HI_UNF_CIPHER_HASH_ATTS_S *pstHashAttr,
HI_HANDLE *pHashHandle);
```

[Parameter]

Parameter	Description	Input/Output
pstHashAttr	Parameter used for calculating the hash value	Input
pHashHandle	Output hash handle	Output

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

A code indicating failure is returned if the hash module is being used by other programs.

[Example]

None

HI_UNF_CIPHER_HashUpdate

[Description]

Calculates the hash value.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_HashUpdate(HI_HANDLE hHashHandle, HI_U8 *pu8InputData,
HI_U32 u32InputDataLen);
```

[Parameter]

Parameter	Description	Input/Output
hHashHandl	Hash handle	Input
pu8InputData	Input data buffer	Input
u32InputDataLen	Input data length (unit: byte)	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

- The input data block must be 64-byte-aligned. However, there is no such limitation on the last block.
- A hash handle must be created before you call this API.
- This API can be called repeatedly. Multiple data blocks are calculated at a time.

[Example]

See **sample_hash.c**.

HI_UNF_CIPHER_HashFinal

[Description]

Obtains the hash value.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_HashFinal(HI_HANDLE hHashHandle, HI_U8
*pu8OutputHash);
```

[Parameter]

Parameter	Description	Input/Output
hHashHandl	Hash handle	Input
pu8OutputHash	Output hash value	Output

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

See **sample_hash.c**.

HI_UNF_CIPHER_GetRandomNumber

[Description]

Generates a random number.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_GetRandomNumber(HI_U32 *pu32RandomNumber);
```

[Parameter]

Parameter	Description	Input/Output
pu32RandomNumber	Output random number	Output

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

See **sample_rng.c**.

HI_UNF_CIPHER_GetTag

[Description]

Obtains the tag value after encryption/decryption is performed in CCM mode or GCM.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_GetTag(HI_HANDLE hCipher, HI_U8 *pstTag);
```

[Parameter]

Parameter	Description	Input/Output
hCipher	Cipher handle	Input
pstTag	Tag value	Output

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

This API is valid only in CCM mode or GCM.

[Example]

See **sample_cipher.c**.

HI_UNF_CIPHER_RsaPublicEncrypt

[Description]

Encrypts a piece of plaintext by using the RSA public key.

[Syntax]

```
HI_UNF_CIPHER_RsaPublicEncrypt(HI_UNF_CIPHER_RSA_PUB_ENC_S*pstRsaEnc,
HI_U8 *pu8Input, HI_U32 u32InLen, HI_U8 *pu8Output, HI_U32 *pu32OutLen);
```

[Parameter]

Parameter	Description	Input/Output
pstRsaEnc	Public key encryption attribute	Input
pu8Input	Data to be encrypted	Input
u32InLen	Length of the data to be encrypted (unit: byte)	Input
pu8Output	Encryption result data	Output
pu32OutLen	Length of the encryption result data (unit: byte)	Output

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

See **sample_rsa_enc.c**.

HI_UNF_CIPHER_RsaPrivateDecrypt

[Description]

Decrypts a piece of ciphertext by using the RSA private key.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_RsaPrivateDecrypt(HI_UNF_CIPHER_RSA_PRI_ENC_S
*pstRsaDec, HI_U8 *pu8Input, HI_U32 u32InLen, HI_U8 *pu8Output, HI_U32
*pu32OutLen);
```

[Parameter]

Parameter	Description	Input/Output
pstRsaDec	Private key decryption attribute	Input
pu8Input	Data to be decrypted	Input
u32InLen	Length of the data to be decrypted (unit: byte)	Input
pu8Output	Decryption result data	Output
pu32OutLen	Length of the decryption result data (unit: byte)	Output

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

See **sample_rsa_enc.c**.

HI_UNF_CIPHER_RsaPrivateEncrypt

[Description]

Encrypts a piece of plaintext by using the RSA private key.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_RsaPrivateEncrypt(HI_UNF_CIPHER_RSA_PRI_ENC_S
*pstRsaEnc, HI_U8 *pu8Input, HI_U32 u32InLen, HI_U8 *pu8Output, HI_U32
*pu32OutLen);
```

[Parameter]

Parameter	Description	Input/Output
pstRsaEnc	Private key encryption attribute	Input
pu8Input	Data to be encrypted	Input
u32InLen	Length of the data to be encrypted (unit: byte)	Input
pu8Output	Encryption result data	Output
pu32OutLen	Length of the encryption result data (unit: byte)	Output

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

See **sample_rsa_enc.c**.

HI_UNF_CIPHER_RsaPublicDecrypt

[Description]

Decrypts a piece of ciphertext by using the RSA public key.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_RsaPrivateDecrypt (HI_UNF_CIPHER_RSA_PUB_ENC_S
*pstRsaDec, HI_U8 *pu8Input, HI_U32 u32InLen, HI_U8 *pu8Output, HI_U32
*pu32OutLen);
```

[Parameter]

Parameter	Description	Input/Output
pstRsaDec	Public key decryption attribute	Input
pu8Input	Data to be decrypted	Input
u32InLen	Length of the data to be decrypted (unit: byte)	Input
pu8Output	Decryption result data	Output
pu32OutLen	Length of the decryption result data (unit: byte)	Output

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

See **sample_rsa_enc.c**.

HI_UNF_CIPHER_RsaSign

[Description]

Signs a piece of text by using the RSA private key.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_RsaSign(HI_UNF_CIPHER_RSA_SIGN_S *pstRsaSign, HI_U8
*pu8InData, HI_U32 u32InDataLen, HI_U8 *pu8HashData, HI_U8 *pu8OutSign,
HI_U32 *pu32OutSignLen);
```

[Parameter]

Parameter	Description	Input/Output
pstRsaSign	Signature attribute	Input
pu8InData	Data to be signed. If **pu8HashData** is not null, **pu8HashData** is used for the signature and **pu8InData** is ignored.	Input

Parameter	Description	Input/Output
u32InDataLen	Length of the data to be signed (unit: byte)	Input
pu8HashData	Hash digest of the text to be signed. If **pu8HashData** is null, the hash digest of the data to be signed (**pu8InData**) is automatically calculated and used for the signature.	Input
pu8OutSign	Signature result data	Output
pu32OutSignLen	Length of the signature result data (unit: byte)	Output

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

See **sample_rsa_sign.c**.

HI_UNF_CIPHER_RsaVerify

[Description]

Verifies a piece of text by using the RSA public key.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_RsaVerify(HI_UNF_CIPHER_RSA_VERIFY_S*pstRsaVerify,
HI_U8 *pu8InData, HI_U32 u32InDataLen, HI_U8 *pu8HashData, HI_U8 *pu8InSign,
HI_U32 u32InSignLen);
```

[Parameter]

Parameter	Description	Input/Output
pstRsaVerify	Signature verification attribute	Input
pu8InData	Data to be verified. If **pu8HashData** is not null, **pu8HashData** is used for the verification and **pu8InData** is ignored.	Input

Parameter	Description	Input/Output
u32InDataLen	Length of the data to be verified (unit: byte)	Input
pu8HashData	Hash digest of the text to be verified. If **pu8HashData** is null, the hash digest of the data to be verified (**pu8InData**) is automatically calculated and used for the verification.	Input
pu8InSign	Signature data to be verified	Input
u32InSignLen	Length of the signature data to be verified (unit: byte)	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library file: **libhi_cipher.a**

[Note]

None

[Example]

See **sample_rsa_sign.c**.

HI_UNF_CIPHER_KladEncryptKey

[Description]

Encrypts the transparent key by using the Klad.

[Syntax]

```
HI_S32 HI_UNF_CIPHER_KladEncryptKey(HI_UNF_CIPHER_CA_TYPE_E enRootKey,
HI_UNF_CIPHER_KLAD_TARGET_E enTarget, HI_U8 *pu8CleanKey,
HI_U8* pu8EcnryptKey, HI_U32 u32KeyLen);
```

[Parameter]

Parameter	Description	Input/Output
enRootKey	Klad root key select. Only the eFUSE key can be selected.	Input
enTarget	Module using the key	Input

Parameter	Description	Input/Output
pu8CleanKey	Transparent key	Input
pu8EcnryptKey	Encrypted key	Output
u32KeyLen	Length of the key. The value must be an integral multiple of 16.	Input

[Return Value]

Return Value	Description
0	Success
Other values	For details, see chapter 4 "Error Codes."

[Requirement]

- Header files: **hi_error_mpi.h**, **hi_type.h**, and **hi_unf_cipher.h**
- Library files: **libhi_cipher.a**

[Note]

None

[See Also]

See **sample_rsa_enc.c** under the **cipher sample** directory.

3 Data Structures

The cipher data types are as follows:

- HI_HANDLE: Defines the handle type of the cipher module.
- HI_UNF_CIPHER_WORK_MODE_E: Defines the working mode of the cipher module.
- HI_UNF_CIPHER_ALG_E: Defines the encryption and decryption algorithms of the cipher module.
- HI_UNF_CIPHER_KEY_LENGTH_E: Defines the key length of the cipher module.
- HI_UNF_CIPHER_BIT_WIDTH_E: Defines the encryption bit width of the cipher module.
- HI_UNF_CIPHER_CTRL_CHANGE_FLAG_S: Defines the information structure of the cipher in CCM mode.
- HI_UNF_CIPHER_CA_TYPE_E: Defines the source of the cipher key.
- HI_UNF_CIPHER_KLAD_TARGET_E: Defines the target to which the Klad key is sent.
- HI_UNF_CIPHER_TYPE_E: Defines the cipher type.
- HI_UNF_CIPHER_ATTS_S: Defines the cipher type structure.
- HI_UNF_CIPHER_CTRL_S: Defines the structure of the cipher control information.
- HI_UNF_CIPHER_CTRL_AES_S: Defines the AES cipher control information structure extension.
- HI_UNF_CIPHER_CTRL_AES_CCM_GCM_S: Defines the AES-CCM and AES-GCM cipher control information structure.
- HI_UNF_CIPHER_CTRL_DES_S: Defines the DES cipher control information structure extension.
- HI_UNF_CIPHER_CTRL_3DES_S: Defines the 3DES cipher control information structure.
- HI_UNF_CIPHER_CTRL_EX_S: Defines the cipher control information extended structure as the algorithm.
- HI_UNF_CIPHER_DATA_S: Defines the data encrypted/decrypted by the cipher module.
- HI_UNF_CIPHER_HASH_TYPE_E: Defines the type of the cipher hash algorithm.
- HI_UNF_CIPHER_HASH_ATTS_S: Defines the initialization input of the cipher hash algorithm.
- HI_UNF_CIPHER_RSA_ENC_SCHEME_E: Defines the padding schemes for RSA data encryption.

- HI_UNF_CIPHER_RSA_SIGN_SCHEME_E: Defines the RSA data signature policies.
- HI_UNF_CIPHER_RSA_PUB_KEY_S: Defines the RSA public key structure.
- HI_UNF_CIPHER_RSA_PRI_KEY_S: Defines the RSA private key structure.
- HI_UNF_CIPHER_RSA_PUB_ENC_S: Defines the parameter structure of the RSA public key encryption and decryption algorithms.
- HI_UNF_CIPHER_RSA_PRI_ENC_S: Defines the parameter structure of the RSA private key decryption algorithm.
- HI_UNF_CIPHER_RSA_SIGN_S: Defines the parameter input structure of the RSA signature algorithm.
- HI_UNF_CIPHER_RSA_VERIFY_S: Defines the parameter input structure of the RSA signature verification algorithm.
- CIPHER_IV_CHANGE_ONE_PKG: Updates the IV of only one data packet when IVs are set for data packets in the cipher module.
- CIPHER_IV_CHANGE_ALL_PKG: Updates the IVs of all data packets when IVs are set for data packets in the cipher module.

HI_HANDLE

[Description]

Defines the handle type of the cipher module.

[Syntax]

```
typedef HI_U32 HI_HANDLE;
```

[Member]

None

[Note]

None

[See Also]

None

HI_UNF_CIPHER_WORK_MODE_E

[Description]

Defines the working mode of the cipher module.

[Syntax]

```
typedef enum hiHI_UNF_CIPHER_WORK_MODE_E
{
    HI_UNF_CIPHER_WORK_MODE_ECB,
    HI_UNF_CIPHER_WORK_MODE_CBC,
    HI_UNF_CIPHER_WORK_MODE_CFB,
    HI_UNF_CIPHER_WORK_MODE_OFB,
    HI_UNF_CIPHER_WORK_MODE_CTR,
    HI_UNF_CIPHER_WORK_MODE_CCM,
    HI_UNF_CIPHER_WORK_MODE_GCM,
    HI_UNF_CIPHER_WORK_MODE_CBC_CTS,
    HI_UNF_CIPHER_WORK_MODE_BUTT    = 0xffffffff
}HI_UNF_CIPHER_WORK_MODE_E;
```

[Member]

Member	Description
HI_UNF_CIPHER_WORK_MODE_ECB	Electronic code book (ECB) mode
HI_UNF_CIPHER_WORK_MODE_CBC	Cipher block chaining (CBC) mode
HI_UNF_CIPHER_WORK_MODE_CFB	Cipher feedback (CFB) mode
HI_UNF_CIPHER_WORK_MODE_OFB	Output feedback (OFB) mode
HI_UNF_CIPHER_WORK_MODE_CTR	Counter (CTR) mode
HI_UNF_CIPHER_WORK_MODE_CCM	CCM (Counter with Cipher Block Chaining-Message Authentication) mode
HI_UNF_CIPHER_WORK_MODE_GCM	GCM (Galois/Counter Mode) mode
HI_UNF_CIPHER_WORK_MODE_CBC_CTS	CBC CTS (Community Tissue Services) mode
HI_UNF_CIPHER_WORK_MODE_BUTT	Invalid mode

[Note]

None

[See Also]

None

HI_UNF_CIPHER_ALG_E

[Description]

Defines the encryption and decryption algorithms of the cipher module.

[Syntax]

```
typedef enum hiHI_UNF_CIPHER_ALG_E
{
    HI_UNF_CIPHER_ALG_DES  = 0x0,
    HI_UNF_CIPHER_ALG_3DES = 0x1,
    HI_UNF_CIPHER_ALG_AES  = 0x2,
    HI_UNF_CIPHER_ALG_SM1          = 0x3,
    HI_UNF_CIPHER_ALG_SM4          = 0x4,
    HI_UNF_CIPHER_ALG_DMA          = 0x5,
    HI_UNF_CIPHER_ALG_BUTT          = 0x6,
    HI_UNF_CIPHER_ALG_INVALID      = 0xffffffff,
}HI_UNF_CIPHER_ALG_E;
```

[Member]

Member	Description
HI_UNF_CIPHER_ALG_DES	DES algorithm
HI_UNF_CIPHER_ALG_3DES	3DES algorithm
HI_UNF_CIPHER_ALG_AES	AES algorithm
HI_UNF_CIPHER_ALG_SM1	SM1 algorithm
HI_UNF_CIPHER_ALG_SM4	SM4 algorithm
HI_UNF_CIPHER_ALG_DMA	Direct DMA copying without encryption or decryption calculation
HI_UNF_CIPHER_ALG_BUTT	Invalid algorithm
HI_UNF_CIPHER_ALG_INVALID	Invalid value

[Note]

None

[See Also]

None

HI_UNF_CIPHER_KEY_LENGTH_E

[Description]

Defines the key length of the cipher module.

[Syntax]

```
typedef enum hiHI_UNF_CIPHER_KEY_LENGTH_E
{
    HI_UNF_CIPHER_KEY_AES_128BIT = 0x0,
    HI_UNF_CIPHER_KEY_AES_192BIT = 0x1,
    HI_UNF_CIPHER_KEY_AES_256BIT = 0x2,
    HI_UNF_CIPHER_KEY_DES_3KEY = 0x2,
    HI_UNF_CIPHER_KEY_DES_2KEY = 0x3,
    HI_UNF_CIPHER_KEY_DEFAULT = 0x0,
    HI_UNF_CIPHER_KEY_INVALID = 0xffffffff,
}HI_UNF_CIPHER_KEY_LENGTH_E;
```

[Member]

Member	Description
HI_UNF_CIPHER_KEY_AES_128BIT	128-bit key for the AES algorithm
HI_UNF_CIPHER_KEY_AES_192BIT	192-bit key for the AES algorithm
HI_UNF_CIPHER_KEY_AES_256BIT	256-bit key for the AES algorithm
HI_UNF_CIPHER_KEY_DES_3KEY	Three keys for the DES algorithm
HI_UNF_CIPHER_KEY_DES_2KEY	Two keys for the DES algorithm
HI_UNF_CIPHER_KEY_DEFAULT	Default key length DES: 8 bytes SM1: 48 bytes, SM4: 16 bytes
HI_UNF_CIPHER_KEY_INVALID	Invalid value

[Note]

- The key length for the AES algorithm is 128 bits, 192 bits, or 256 bits.
- The number of keys for the 3DES algorithm is 2 or 3. Each key is 64 bits and is used for DES encryption.
- This data type is invalid for the DES algorithm.

[See Also]

None

HI_UNF_CIPHER_BIT_WIDTH_E

[Description]

Defines the encryption bit width of the cipher module.

[Syntax]

```
typedef enum hiHI_UNF_CIPHER_BIT_WIDTH_E
{
   HI_UNF_CIPHER_BIT_WIDTH_64BIT  = 0x0,
   HI_UNF_CIPHER_BIT_WIDTH_8BIT   = 0x1,
   HI_UNF_CIPHER_BIT_WIDTH_1BIT   = 0x2,
   HI_UNF_CIPHER_BIT_WIDTH_128BIT = 0x3,
   HI_UNF_CIPHER_BIT_WIDTH_INVALID = 0xffffffff
}HI_UNF_CIPHER_BIT_WIDTH_E;
```

[Member]

Member	Description
HI_UNF_CIPHER_BIT_WIDTH_64BIT	Bit width of 64 bits

Member	Description
HI_UNF_CIPHER_BIT_WIDTH_8BIT	Bit width of 8 bits
HI_UNF_CIPHER_BIT_WIDTH_1BIT	Bit width of 1 bit
HI_UNF_CIPHER_BIT_WIDTH_128BIT	Bit width of 128 bits
HI_UNF_CIPHER_BIT_WIDTH_INVALID	Invalid value

[Note]

None

[See Also]

None

HI_UNF_CIPHER_CTRL_CHANGE_FLAG_S

[Description]

Defines the information structure of the cipher in CCM mode.

[Syntax]

```
typedef struct hiUNF_CIPHER_CTRL_CHANGE_FLAG_S
{
   HI_U32   bit1IV:        2;
   HI_U32   bitsResv:      30;
} HI_UNF_CIPHER_CTRL_CHANGE_FLAG_S;
```

[Member]

Member	Description
bit1IV	Vector change **0** indicates no change. **1** indicates changes in the first packet. **2** indicates changes in each packet.
bitsResv	Reserved

[Note]

None

[See Also]

None

HI_UNF_CIPHER_CA_TYPE_E

[Description]

Defines the source of the cipher key.

[Syntax]

```
typedef enum hiHI_UNF_CIPHER_CA_TYPE_E
{
    HI_UNF_CIPHER_KEY_SRC_USER = 0x0,
    HI_UNF_CIPHER_KEY_SRC_KLAD_1,
    HI_UNF_CIPHER_KEY_SRC_KLAD_2,
    HI_UNF_CIPHER_KEY_SRC_KLAD_3,
    HI_UNF_CIPHER_KEY_SRC_BUTT,
    HI_UNF_CIPHER_KEY_SRC_INVALID = 0xffffffff,
} HI_UNF_CIPHER_CA_TYPE_E;
```

[Member]

Member	Description
HI_UNF_CIPHER_KEY_SRC_USER	Key configured by the user
HI_UNF_CIPHER_KEY_SRC_KLAD_1	Group 1 key of eFUSE
HI_UNF_CIPHER_KEY_SRC_KLAD_2	Group 2 key of eFUSE
HI_UNF_CIPHER_KEY_SRC_KLAD_3	Group 3 key of eFUSE
HI_UNF_CIPHER_KEY_SRC_BUTT	Invalid type
HI_UNF_CIPHER_KEY_SRC_INVALID	Invalid value

[Note]

None

[See Also]

None

HI_UNF_CIPHER_KLAD_TARGET_E

[Description]

Defines the target to which the Klad key is sent.

[Syntax]

```
typedef struct
{
    HI_UNF_CIPHER_KLAD_TARGET_AES,
    HI_UNF_CIPHER_KLAD_TARGET_RSA,
    HI_UNF_CIPHER_KLAD_TARGET_BUTT,
 } HI_UNF_CIPHER_KLAD_TARGET_E;
```

[Member]

Member	Description
HI_UNF_CIPHER_KLAD_TARGET_AES	The Klad key is sent to the AES.
HI_UNF_CIPHER_KLAD_TARGET_RSA	The Klad key is sent to the RSA.
HI_UNF_CIPHER_KLAD_TARGET_BUTT	This parameter is invalid.

[Note]

None

[See Also]

None

HI_UNF_CIPHER_TYPE_E

[Description]

Defines the cipher type.

[Syntax]

```
typedef enum
{
    HI_UNF_CIPHER_TYPE_NORMAL  = 0x0,
    HI_UNF_CIPHER_TYPE_COPY_AVOID,
    HI_UNF_CIPHER_TYPE_BUTT,
    HI_UNF_CIPHER_TYPE_INVALID = 0xffffffff,
}HI_UNF_CIPHER_TYPE_E;
```

[Member]

Member	Description
HI_UNF_CIPHER_TYPE_NORMAL	Direct memory access (DMA) mode of channel 1–7
HI_UNF_CIPHER_TYPE_COPY_AVOID	CPU copying mode of channel 0
HI_UNF_CIPHER_TYPE_BUTT	Invalid type
HI_UNF_CIPHER_TYPE_INVALID	Invalid value

[Note]

None

[See Also]

None

HI_UNF_CIPHER_ATTS_S

[Description]

Defines the cipher type structure.

[Syntax]

```
typedef struct
{
    HI_UNF_CIPHER_TYPE_E enCipherType;
}HI_UNF_CIPHER_ATTS_S;
```

[Member]

Member	Description
enCipherType	Variable of the cipher type structure

[Note]

None

[See Also]

None

HI_UNF_CIPHER_CTRL_S

[Description]

Defines the structure of the control information.

[Syntax]

```
typedef struct hiHI_UNF_CIPHER_CTRL_S
{
    HI_U32                  u32Key[8];
    HI_U32                  u32IV[4];
    HI_BOOL                 bKeyByCA;
    HI_UNF_CIPHER_CA_TYPE_E enCaType;
    HI_UNF_CIPHER_ALG_E       enAlg;
    HI_UNF_CIPHER_BIT_WIDTH_E  enBitWidth;
    HI_UNF_CIPHER_WORK_MODE_E  enWorkMode;
    HI_UNF_CIPHER_KEY_LENGTH_E enKeyLen;
    HI_UNF_CIPHER_CTRL_CHANGE_FLAG_S stChangeFlags;
} HI_UNF_CIPHER_CTRL_S;
```

[Member]

Member	Description
u32Key[8]	Key

Member	Description
u32IV[4]	Initialization vector (IV)
bKeyByCA	Whether to use the CA key for encryption and decryption
enCaType	CA type
enAlg	Encryption algorithm
enBitWidth	Bit width for encryption or decryption
enWorkMode	Working mode
enKeyLen	Key length
stChangeFlags	Flag of IV changes, indicating whether the IV needs to be changed

[Note]

The initialization vector is not required in ECB mode.

[See Also]

None

HI_UNF_CIPHER_CTRL_AES_S

[Description]

Defines the AES cipher control information structure extension.

[Syntax]

```
typedef struct hiHI_UNF_CIPHER_CTRL_AES_S
{
    HI_U32 u32EvenKey[8];
    HI_U32 u32OddKey[8];
    HI_U32 u32IV[4];
    HI_UNF_CIPHER_BIT_WIDTH_E enBitWidth;
    HI_UNF_CIPHER_KEY_LENGTH_E enKeyLen;
    HI_UNF_CIPHER_CTRL_CHANGE_FLAG_S stChangeFlags;
} HI_UNF_CIPHER_CTRL_AES_S;
```

[Member]

Member	Description
u32EvenKey	Even key (by default)
u32OddKey	Odd key
u32IV	IV
enBitWidth	Encryption bit width

Member	Description
enKeyLen	Length of the encryption key
stChangeFlags	Flag of IV changes

[Note]

AES supports the ECB, CBC, CFB, OFB and CTR modes of operation. The CFB mode supports 1, 8, and 128 bit width, and the OFB mode supports only 128 bit width.

[See Also]

None

HI_UNF_CIPHER_CTRL_AES_CCM_GCM_S

[Description]

Defines the AES-CCM and AES-GCM cipher control information structure.

[Syntax]

```
typedef struct hiHI_UNF_CIPHER_CTRL_AES_CCM_GCM_S
{
   HI_U32 u32Key[8];
   HI_U32 u32IV[4];
   HI_UNF_CIPHER_KEY_LENGTH_E enKeyLen;
   HI_U32 u32IVLen;
   HI_U32 u32TagLen;
   HI_U32 u32ALen;
   HI_U32 u32APhyAddr;
} HI_UNF_CIPHER_CTRL_AES_CCM_GCM_S;
```

[Member]

Member	Description
u32Key	Even key (by default)
u32IV	IV
enKeyLen	Length of the encryption key
IVLen	IV length
u32TagLen	Flag of tag length
u32ALen	Length of associated data A
u32APhyAddr	Physical address of associated data A

[Notes]

- CCM: The IV length **u32IVLen** can be {7, 8, 9, 10, 11, 12, 13} bytes, IV stores nonce (N) of the algorithm standard, and the length of encrypted data is represented by **n** bytes. They must meet the following requirements: **u32IVLen** + **n** = 15. Therefore, when **u32IVLen** is 13, **n** is 2. The maximum length of encrypted data is 65,536 bytes, and the rule applies. The length of a tag can be {4, 6, 8, 10, 12, 14, 16} bytes. The values of the vector N and associated data A for CCM encryption must be the same as those for CCM decryption.
- GCM: The IV length **u32IVLen** can be 1–16 bytes. The length of a tag **u32TagLen** can be {12, 13, 14, 15, 16} bytes and sometimes can be {4, 8} bytes. The value of associated data A for GCM encryption must be the same as that for GCM decryption.

[See Also]

None

HI_UNF_CIPHER_CTRL_DES_S

[Description]

Defines the DES cipher control information structure extension.

[Syntax]

```
typedef struct hiHI_UNF_CIPHER_CTRL_DES_S
{
    HI_U32 u32Key[2];
    HI_U32 u32IV[2];
    HI_UNF_CIPHER_BIT_WIDTH_E enBitWidth;
    HI_UNF_CIPHER_CTRL_CHANGE_FLAG_S stChangeFlags;
} HI_UNF_CIPHER_CTRL_DES_S;
```

[Member]

Member	Description
u32Key	Key
u32IV	IV
enBitWidth	Encryption bit width
stChangeFlags	Flag of IV change

[Notes]

This algorithm is not secure and therefore is not recommended for encryption or decryption.

[See Also]

None

HI_UNF_CIPHER_CTRL_3DES_S

[Description]

Defines the 3DES cipher control information structure.

[Syntax]

```
typedef struct hiHI_UNF_CIPHER_CTRL_3DES_S
{
    HI_U32 u32Key[6];
    HI_U32 u32IV[2];
    HI_UNF_CIPHER_BIT_WIDTH_E enBitWidth;
    HI_UNF_CIPHER_KEY_LENGTH_E enKeyLen;
    HI_UNF_CIPHER_CTRL_CHANGE_FLAG_S stChangeFlags;
} HI_UNF_CIPHER_CTRL_3DES_S;
```

[Member]

Member	Description
u32Key	Key
u32IV	IV
enBitWidth	Encryption bit width
enKeyLen	Length of the encryption key
stChangeFlags	Flag of IV changes

[Note]

- 3DES encryption: Use K1, K2, and K3 for encryption, decryption, and encryption in sequence. When the key for the first encryption and the key for decryption are the same but not equal to the key for the second decryption, that is, (K1 = K3)!= K2, two keys are used and you need to set only K1 and K2.
- DES/3DES supports the ECB, CBC, CFB, and OFB modes of operation. The CFB and OFB modes support 1, 8, and 64 bit width.

[See Also]

None

HI_UNF_CIPHER_CTRL_EX_S

[Description]

Defines the cipher control information extended structure as the algorithm. This parameter can be used for different algorithm encryption or decryption. Newly added algorithms such as SM1, SM4, CCM, and GCM are not applicable to parameter setting through HI_UNF_CIPHER_CTRL_S.

[Syntax]

```
typedef struct hiHI_UNF_CIPHER_CTRL_EX_S
{
    HI_UNF_CIPHER_ALG_E enAlg;
    HI_UNF_CIPHER_WORK_MODE_E enWorkMode;
    HI_BOOL bKeyByCA;
    HI_VOID *pParam;
} HI_UNF_CIPHER_CTRL_EX_S;
```

[Member]

Member	Description
enAlg	Encryption/decryption algorithm
enWorkMode	Working Mode
bKeyByCA	Whether to use the hardware key
pParam	Cipher control information structure pointing to various algorithms

[Note]

- The **HI_VOID *pParam** parameter, as the input parameter for the HI_UNF_CIPHER_ConfigHandleEx interface, corresponds to the following parameters based on different algorithm types:
  - For AES, the pointer points to HI_UNF_CIPHER_CTRL_AES_S.
  - For AES_CCM or AES_GCM, the pointer points to HI_UNF_CIPHER_CTRL_AES_CCM_GCM_S.
  - For DES, the pointer points to HI_UNF_CIPHER_CTRL_DES_S.
  - For 3DES, the pointer points to HI_UNF_CIPHER_CTRL_3DES_S.

[See Also]

None

HI_UNF_CIPHER_DATA_S

[Description]

Defines the data encrypted/decrypted by the cipher module.

[Syntax]

```
typedef struct hiHI_UNF_CIPHER_DATA_S
{
    HI_SIZE_T szSrcPhyAddr;
    HI_SIZE_T szDestPhyAddr;
    HI_U32  u32ByteLength;
    HI_BOOL bOddKey;
} HI_UNF_CIPHER_DATA_S;
```

[Member]

Member	Description
u32SrcPhyAddr	Physical address of the source data
u32DestPhyAddr	Physical address of the target data
u32ByteLength	Length of the data to be encrypted/decrypted
bOddKey	Whether to use the odd key (the even key is used by default)

[Note]

None

[See Also]

None

HI_UNF_CIPHER_HASH_TYPE_E

[Description]

Defines the type of the cipher hash algorithm.

[Syntax]

```
typedef enum hiHI_UNF_CIPHER_HASH_TYPE_E
{
    HI_UNF_CIPHER_HASH_TYPE_SHA1,
    HI_UNF_CIPHER_HASH_TYPE_SHA224,
    HI_UNF_CIPHER_HASH_TYPE_SHA256,
    HI_UNF_CIPHER_HASH_TYPE_SHA384,
    HI_UNF_CIPHER_HASH_TYPE_SHA512,
    HI_UNF_CIPHER_HASH_TYPE_HMAC_SHA1,
    HI_UNF_CIPHER_HASH_TYPE_HMAC_SHA224,
    HI_UNF_CIPHER_HASH_TYPE_HMAC_SHA256,
    HI_UNF_CIPHER_HASH_TYPE_HMAC_SHA384,
    HI_UNF_CIPHER_HASH_TYPE_HMAC_SHA512,
    HI_UNF_CIPHER_HASH_TYPE_SM3,
    HI_UNF_CIPHER_HASH_TYPE_BUTT,
    HI_UNF_CIPHER_HASH_TYPE_INVALID  = 0xffffffff,
}HI_UNF_CIPHER_HASH_TYPE_E;
```

[Member]

Member	Description
HI_UNF_CIPHER_HASH_TYPE_SHA1	SHA1 hash algorithm
HI_UNF_CIPHER_HASH_TYPE_SHA224	SHA224 hash algorithm

Member	Description
HI_UNF_CIPHER_HASH_TYPE_SHA256	SHA256 hash algorithm
HI_UNF_CIPHER_HASH_TYPE_SHA384	SHA384 hash algorithm
HI_UNF_CIPHER_HASH_TYPE_SHA512	SHA512 hash algorithm
HI_UNF_CIPHER_HASH_TYPE_HMAC_SHA1	HMAC_SHA1 hash algorithm
HI_UNF_CIPHER_HASH_TYPE_HMAC_SHA224	HMAC_SHA224 hash algorithm
HI_UNF_CIPHER_HASH_TYPE_HMAC_SHA256	HMAC_SHA256 hash algorithm
HI_UNF_CIPHER_HASH_TYPE_HMAC_SHA384	HMAC_SHA384 hash algorithm
HI_UNF_CIPHER_HASH_TYPE_HMAC_SHA512	HMAC_SHA512 hash algorithm
HI_UNF_CIPHER_HASH_TYPE_SM3	SM3 hash algorithm
HI_UNF_CIPHER_HASH_TYPE_BUTT	Invalid algorithm
HI_UNF_CIPHER_HASH_TYPE_INVALID	Invalid value

[Note]

None

[See Also]

None

HI_UNF_CIPHER_HASH_ATTS_S

[Description]

Defines the initialization input of the cipher hash algorithm.

[Syntax]

```
typedef struct
{
    HI_U8 *pu8HMACKey;
    HI_U32 u32HMACKeyLen;
    HI_UNF_CIPHER_HASH_TYPE_E eShaType;
}HI_UNF_CIPHER_HASH_ATTS_S;
```

[Member]

Member	Description
pu8HMACKey	HMAC key
u32HMACKeyLen	HMAC key length
eShaType	Hash algorithm type

[Note]

None

[See Also]

None

HI_UNF_CIPHER_RSA_ENC_SCHEME_E

[Description]

Defines the padding schemes for RSA data encryption.

[Syntax]

```
typedef enum hiHI_UNF_CIPHER_RSA_ENC_SCHEME_E
{
    HI_UNF_CIPHER_RSA_ENC_SCHEME_NO_PADDING,
    HI_UNF_CIPHER_RSA_ENC_SCHEME_BLOCK_TYPE_0,
    HI_UNF_CIPHER_RSA_ENC_SCHEME_BLOCK_TYPE_1,
    HI_UNF_CIPHER_RSA_ENC_SCHEME_BLOCK_TYPE_2,
    HI_UNF_CIPHER_RSA_ENC_SCHEME_RSAES_OAEP_SHA1,
    HI_UNF_CIPHER_RSA_ENC_SCHEME_RSAES_OAEP_SHA224,
    HI_UNF_CIPHER_RSA_ENC_SCHEME_RSAES_OAEP_SHA256,
    HI_UNF_CIPHER_RSA_ENC_SCHEME_RSAES_OAEP_SHA384,
    HI_UNF_CIPHER_RSA_ENC_SCHEME_RSAES_OAEP_SHA512,
    HI_UNF_CIPHER_RSA_ENC_SCHEME_RSAES_PKCS1_V1_5,
    HI_UNF_CIPHER_RSA_ENC_SCHEME_BUTT,
    HI_UNF_CIPHER_RSA_ENC_SCHEME_INVALID  = 0xffffffff,
}HI_UNF_CIPHER_RSA_ENC_SCHEME_E;
```

[Member]

Member	Description
HI_UNF_CIPHER_RSA_ENC_SCHEME_ NO_PADDING	No padding
HI_UNF_CIPHER_RSA_ENC_SCHEME_ BLOCK_TYPE_0,	Block type 0 padding scheme of PKCS #1
HI_UNF_CIPHER_RSA_ENC_SCHEME_ BLOCK_TYPE_1	Block type 1 padding scheme of PKCS #1
HI_UNF_CIPHER_RSA_ENC_SCHEME_ BLOCK_TYPE_2	Block type 2 padding scheme of PKCS #1
HI_UNF_CIPHER_RSA_ENC_SCHEME_ RSAES_OAEP_SHA1	RSAES-OAEP-SHA1 padding scheme of PKCS #1

Member	Description
HI_UNF_CIPHER_RSA_ENC_SCHEME_ RSAES_OAEP_SHA224	RSAES-OAEP-SHA224 padding scheme of PKCS #1
HI_UNF_CIPHER_RSA_ENC_SCHEME_ RSAES_OAEP_SHA256	RSAES-OAEP-SHA256 padding scheme of PKCS #1
HI_UNF_CIPHER_RSA_ENC_SCHEME_ RSAES_OAEP_SHA384	RSAES-OAEP-SHA384 padding scheme of PKCS #1
HI_UNF_CIPHER_RSA_ENC_SCHEME_ RSAES_OAEP_SHA512	RSAES-OAEP-SHA512 padding scheme of PKCS #1
HI_UNF_CIPHER_RSA_ENC_SCHEME_ RSAES_PKCS1_V1_5	PKCS1_V1_5 padding scheme of PKCS #1
HI_UNF_CIPHER_RSA_SCHEME_BUTT	Null value
HI_UNF_CIPHER_RSA_ENC_SCHEME_INVALID	Invalid value

[Note]

None

[See Also]

None

HI_UNF_CIPHER_RSA_SIGN_SCHEME_E

[Description]

Defines the RSA data signature policies.

[Syntax]

```
typedef enum hiHI_UNF_CIPHER_RSA_SIGN_SCHEME_E
{
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_V15_SHA1 = 0x100,
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_V15_SHA224,
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_V15_SHA256,
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_V15_SHA384,
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_V15_SHA512,
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_PSS_SHA1,
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_PSS_SHA224,
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_PSS_SHA256,
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_PSS_SHA384,
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_PSS_SHA512,
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_BUTT,
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_INVALID  = 0xffffffff,
}HI_UNF_CIPHER_RSA_SIGN_SCHEME_E;
```

[Member]

Member	Description
HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_V15_SHA1	PKCS #1 RSASSA_PKCS1_V15_SHA1 signature algorithm
HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_V15_SHA224	PKCS #1 RSASSA_PKCS1_V15_SHA224 signature algorithm
HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_V15_SHA256	PKCS #1 RSASSA_PKCS1_V15_SHA256 signature algorithm
HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_V15_SHA384	PKCS #1 RSASSA_PKCS1_V15_SHA384 signature algorithm
HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_V15_SHA512	PKCS #1 RSASSA_PKCS1_V15_SHA512 signature algorithm
HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_PSS_SHA1	PKCS #1 RSASSA_PKCS1_PSS_SHA1 signature algorithm
HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_PSS_SHA224	PKCS #1 RSASSA_PKCS1_PSS_SHA224 signature algorithm
HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_PSS_SHA256	PKCS #1 RSASSA_PKCS1_PSS_SHA256 signature algorithm
HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_PSS_SHA384	PKCS #1 RSASSA_PKCS1_PSS_SHA384 signature algorithm
HI_UNF_CIPHER_RSA_SIGN_SCHEME_RSASSA_PKCS1_PSS_SHA512	PKCS #1 RSASSA_PKCS1_PSS_SHA512 signature algorithm
HI_UNF_CIPHER_RSA_SIGN_SCHEME_BUTT	Invalid algorithm
HI_UNF_CIPHER_RSA_SIGN_SCHEME_INVALID	Invalid value

[Note]

None

[See Also]

None

HI_UNF_CIPHER_RSA_PUB_KEY_S

[Description]

Defines the RSA public key structure.

[Syntax]

```
typedef struct
{
    HI_U8  *pu8N;
     HI_U8  *pu8E;
    HI_U16 u16NLen;
    HI_U16 u16ELen;
}HI_UNF_CIPHER_RSA_PUB_KEY_S;
```

[Member]

Member	Description
pu8N	Pointer to the RSA public key *N*
pu8E	Pointer to the RSA public key *E*
u16NLen	Length of the RSA public key *N*
u16ELen	Length of the RSA public key *E*

[Note]

None

[See Also]

None

HI_UNF_CIPHER_RSA_PRI_KEY_S

[Description]

Defines the RSA private key structure.

[Syntax]

```
typedef struct
{
    HI_U8 *pu8N;
    HI_U8 *pu8E;
    HI_U8 *pu8D;
    HI_U8 *pu8P;
    HI_U8 *pu8Q;
    HI_U8 *pu8DP;
    HI_U8 *pu8DQ;
    HI_U8 *pu8QP;
    HI_U16 u16NLen;
    HI_U16 u16ELen;
    HI_U16 u16DLen;
    HI_U16 u16PLen;
    HI_U16 u16QLen;
    HI_U16 u16DPLen;
    HI_U16 u16DQLen;
    HI_U16 u16QPLen;
}HI_UNF_CIPHER_RSA_PRI_KEY_S;
```

[Member]

Member	Description
pu8N	Pointer to the RSA public key *N*
pu8E	Pointer to the RSA public key *E*
pu8D	Pointer to the RSA public key *D*
pu8P	Pointer to the RSA public key *P*
pu8Q	Pointer to the RSA public key *Q*
pu8DP	Pointer to the RSA public key *DP*
pu8DQ	Pointer to the RSA public key *DQ*
pu8QP	Pointer to the RSA public key *QP*
u16NLen	Length of the RSA public key *N*
u16ELen	Length of the RSA public key *E*
u16DLen	Length of the RSA public key *D*
u16PLen	Length of the RSA public key *P*
u16QLen	Length of the RSA public key *Q*
u16DPLen	Length of the RSA public key *DP*
u16DQLen	Length of the RSA public key *DQ*
u16QPLen	Length of the RSA public key *QP*

[Note]

None

[See Also]

None

HI_UNF_CIPHER_RSA_PUB_ENC_S

[Description]

Defines the parameter structure of the RSA public key encryption and decryption algorithms.

[Syntax]

```
typedef struct
{
    HI_UNF_CIPHER_RSA_ENC_SCHEME_E enScheme;
    HI_UNF_CIPHER_RSA_PUB_KEY_S stPubKey;
    HI_UNF_CIPHER_CA_TYPE_E enCaType;
}HI_UNF_CIPHER_RSA_PUB_ENC_S;
```

[Member]

Member	Description
enScheme	RSA data encryption and decryption algorithm policy
stPubKey	RSA public key structure
enCaType	RSA private key source select

[Note]

The **enCaType** parameter can be **CPU Key** or **Klad Key**.

[See Also]

None

HI_UNF_CIPHER_RSA_PRI_ENC_S

[Description]

Defines the parameter structure of the RSA private key decryption algorithm.

[Syntax]

```
typedef struct
{
    HI_UNF_CIPHER_RSA_ENC_SCHEME_E enScheme;
    HI_UNF_CIPHER_RSA_PRI_KEY_S stPriKey;
    HI_UNF_CIPHER_CA_TYPE_E enCaType;
}HI_UNF_CIPHER_RSA_PRI_ENC_S;
```

[Member]

Member	Description
enScheme	RSA data encryption and decryption algorithm policy
stPriKey	RSA private key structure

Member	Description
enCaType	RSA private key source select

[Note]

The **enCaType** parameter can be **CPU Key** or **Klad Key**.

[See Also]

None

HI_UNF_CIPHER_RSA_SIGN_S

[Description]

Defines the parameter input structure of the RSA signature algorithm.

[Syntax]

```
typedef struct
{
  HI_UNF_CIPHER_RSA_SIGN_SCHEME_E enScheme;
  HI_UNF_CIPHER_RSA_PRI_KEY_S stPriKey;
  HI_UNF_CIPHER_CA_TYPE_E enCaType;
} HI_UNF_CIPHER_RSA_SIGN_S;
```

[Member]

Member	Description
enScheme	RSA signature algorithm policy
stPriKey	RSA private key structure
enCaType	RSA private key source select

[Note]

The **enCaType** parameter can be **CPU Key** or **Klad Key**.

[See Also]

None

HI_UNF_CIPHER_RSA_VERIFY_S

[Description]

Defines the parameter input structure of the RSA signature verification algorithm.

[Syntax]

```
typedef struct
{
    HI_UNF_CIPHER_RSA_SIGN_SCHEME_E enScheme;
    HI_UNF_CIPHER_RSA_PUB_KEY_S stPubKey;
}HI_UNF_CIPHER_RSA_VERIFY_S;
```

[Member]

Member	Description
enScheme	RSA data encryption and decryption algorithm policy
stPubKey	RSA public key structure

[Note]

None

[See Also]

None

CIPHER_IV_CHANGE_ONE_PKG

[Description]

Updates the IV of only one data packet when IVs are set for data packets in the cipher module.

[Syntax]

```
#define CIPHER_IV_CHANGE_ONE_PKG (1)
```

[Note]

None

[See Also]

None

CIPHER_IV_CHANGE_ALL_PKG

[Description]

Updates the IVs of all data packets when IVs are set for data packets in the cipher module.

[Syntax]

```
#define CIPHER_IV_CHANGE_ALL_PKG (2)
```

[Note]

The macro is used only for multiple-packet encryption.

[See Also]

None

4 Error Codes

Table 4-1 describes the error codes for the cipher module.

Table 4-1 Error codes for the cipher module

Error Code	Macro Definition	Description
0x804D0001	HI_ERR_CIPHER_NOT_INIT	The cipher device is not initialized.
0x804D0002	HI_ERR_CIPHER_INVALID_HANDLE	The handle ID is invalid.
0x804D0003	HI_ERR_CIPHER_INVALID_POINT	The pointer is null.
0x804D0004	HI_ERR_CIPHER_INVALID_PARA	The parameter is invalid.
0x804D0005	HI_ERR_CIPHER_FAILED_INIT	The cipher module fails to be initialized.
0x804D0006	HI_ERR_CIPHER_FAILED_GETHANDLE	The handle fails to be obtained.
0x804D0007	HI_ERR_CIPHER_FAILED_RELEASEHANDLE	The handle fails to be released.
0x804D0008	HI_ERR_CIPHER_FAILED_CONFIGAES	The AES configuration is invalid.
0x804D0009	HI_ERR_CIPHER_FAILED_CONFIGDES	The DES configuration is invalid.
0x804D000A	HI_ERR_CIPHER_FAILED_ENCRYPT	Encryption fails.
0x804D000B	HI_ERR_CIPHER_FAILED_DECRYPT	Decryption fails.
0x804D000C	HI_ERR_CIPHER_BUSY	The cipher module is busy.
0x804D000D	HI_ERR_CIPHER_NO_AVAILABLE_RNG	There is no available random number.

Error Code	Macro Definition	Description
0x804D000E	HI_ERR_CIPHER_FAILED_MEM	The memory fails to be allocated.
0x804D000F	HI_ERR_CIPHER_UNAVAILABLE	The cipher module is unavailable.
0x804D0010	HI_ERR_CIPHER_OVERFLOW	Data overflow occurs.
0x804D0011	HI_ERR_CIPHER_HARD_STATUS	The hardware status is incorrect.
0x804D0012	HI_ERR_CIPHER_TIMEOUT	The cipher module timed out.
0x804D0013	HI_ERR_CIPHER_UNSUPPORTED	The configuration is not supported.
0x804D0014	HI_ERR_CIPHER_REGISTER_IRQ	The interrupt ID is invalid.
0x804D0015	HI_ERR_CIPHER_ILLEGAL_UUID	The UUID is invalid.
0x804D0016	HI_ERR_CIPHER_ILLEGAL_KEY	The key is invalid.
0x804D0017	HI_ERR_CIPHER_INVALID_ADDR	The address is invalid.
0x804D0018	HI_ERR_CIPHER_INVALID_LENGTH	The length is invalid.
0x804D0019	HI_ERR_CIPHER_ILLEGAL_DATA	The data is invalid.
0x804D001A	HI_ERR_CIPHER_RSA_SIGN	RSA signing fails.
0x804D001B	HI_ERR_CIPHER_RSA_VERIFY	RSA verification fails.
0x804D001E	HI_ERR_CIPHER_RSA_CRYPT_FAILED	RSA encryption and decryption fail.
–1	HI_FAILURE	The operation fails.
0x004D0001	HI_LOG_ERR_MEM	The memory operation fails.
0x004D0002	HI_LOG_ERR_SEM	The semaphore operation fails.
0x004D0003	HI_LOG_ERR_FILE	The file operation fails.
0x004D0004	HI_LOG_ERR_LOCK	The lock operation fails.
0x004D0005	HI_LOG_ERR_PARAM	The parameter is invalid.
0x004D0006	HI_LOG_ERR_TIMER	A timer error occurs.
0x004D0007	HI_LOG_ERR_THREAD	The thread fails.
0x004D0008	HI_LOG_ERR_TIMEOUT	A timeout occurs.

Error Code	Macro Definition	Description
0x004D0009	HI_LOG_ERR_DEVICE	The device operation fails.
0x004D0010	HI_LOG_ERR_STATUS	A status error occurs.
0x004D0011	HI_LOG_ERR_IOCTRL	The I/O operation fails.
0x004D0012	HI_LOG_ERR_INUSE	The resource is being used.
0x004D0013	HI_LOG_ERR_EXIST	The exit fails.
0x004D0014	HI_LOG_ERR_NOEXIST	The resource does not exit.
0x004D0015	HI_LOG_ERR_UNSUPPORTED	The parameter is not supported.
0x004D0016	HI_LOG_ERR_UNAVAILABLE	The resource is unavailable.
0x004D0017	HI_LOG_ERR_UNINITED	The resource is not initialized.
0x004D0018	HI_LOG_ERR_DATABASE	A database error occurs.
0x004D0019	HI_LOG_ERR_OVERFLOW	A resource overflow occurs.
0x004D0020	HI_LOG_ERR_EXTERNAL	An external error occurs.
0x004D0021	HI_LOG_ERR_UNKNOWNED	The location is incorrect.
0x004D0022	HI_LOG_ERR_FLASH	The flash memory operation fails.
0x004D0023	HI_LOG_ERR_ILLEGAL_IMAGE	The image is invalid.
0x004D0023	HI_LOG_ERR_ILLEGAL_UUID	The UUID is invalid.
0x004D0023	HI_LOG_ERR_NOPERMISSION	This operation is not allowed.

5 Proc Debugging Information

5.1 Cipher Status

[Debugging Information]

Debugging information is as follows:

```
Chnid   Status   Decrypt   Alg   Mode   KeyLen
0       close      1      DES      ECB     008
1       close      1      DES      ECB     008
2       close      0      DES      ECB     008
3       close      0      DES      ECB     008
4       close      0      DES      ECB     008
5       close      0      DES      ECB     008
6       close      0      DES      ECB     008
7       close      0      DES      ECB     008
Phy-Addr in/out      KeyFrom  INT-RAW in/out  INT-EN in/out INT_OCNTCFG
00000420/00000080    HW          0/0            0/0            00000000
00000000/00000000    SW          0/0            1/0            00000001
00000000/00000000    SW          0/0            1/0            00000001
00000000/00000000    SW          0/0            1/0            00000001
00000000/00000000    SW          0/0            1/0            00000001
00000000/00000000    SW          0/0            1/0            00000001
00000000/00000000    SW          0/0            1/0            00000001
00000000/00000000    SW          0/0            1/0            00000001
```

[Analysis]

Debugging information records the configuration information of each channel in the current cipher.

[Parameter]

Table 5-1 Parameter description

Parameter		Description
Cipher attributes	Chnid	Channel ID
	Status	Enable/Disable
	Decrypt	Encryption/Decryption
	Alg	Algorithm, such as AES, DES, or 3DES
	Mode	Mode, such as ECB, CBC, CFB, or CTR
	KeyLen	Key length, which can be 128, 192, or 256
	Phy-Addr	Input/Output physical address
	KeyFrom	Key source, which is the CPU or eFUSE
	INT-RAW	Whether a raw interrupt is generated
	INT-EN	Whether an interrupt is enabled
	INT_OCNTCFG	Whether an interrupt is generated