4.2 Advanced encryption accelerator

1. Experiment purpose

In this lesson, we mainly learn function of Advanced encryption accelerator of K210.

2.Experiment preparation

2.1 components

Advanced encryption accelerator AES in K210 chip

2.2 Component characteristics

K210 has built-in AES (Advanced Encryption Accelerator), which can greatly improve AES operation speed compared to software. The AES accelerator supports multiple encryption/decryption modes (ECB, CBC, GCM), and multiple lengths of KEY (128, 192, 256) operations.

AES accelerator is a module used for encryption and decryption. The specific performance is as follows:

- Support ECB, CBC, GCM three encryption methods
- Support 128-bit, 192-bit, 256-bit KEY length
- KEY can be configured by software and protected by hardware circuit
- Support DMA transfer

2.3 SDK API function

The header file is aes.h

We will provide users with the following interfaces:

- **aes_ecb128_hard_encrypt:** AES-ECB-128 encryption operation. Both input and output data are transmitted by CPU. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

- **aes_ecb128_hard_decrypt**: AES-ECB-128 decryption operation. Both input and output data are transmitted by CPU. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

- **aes_ecb192_hard_encrypt**: AES-ECB-192 encryption operation. Both input and output data are transmitted by CPU. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

- **aes_ecb192_hard_decrypt**: AES-ECB-192 decryption operation. Both input and output data are transmitted by CPU. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

- **aes_ecb256_hard_encrypt**: AES-ECB-256 encryption operation. Both input and output data are transmitted by CPU. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

• **aes_ecb256_hard_decrypt**: AES-ECB-256 decryption operation. Both input and output data are transmitted by CPU. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

• **aes_cbc128_hard_encrypt**: AES-CBC-128 encryption operation. Both input and output data are transmitted by CPU. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

• **aes_cbc128_hard_decrypt**: AES-CBC-128 decryption operation. Both input and output data are transmitted by CPU. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

• **aes_cbc192_hard_encrypt**: AES-CBC-192 encryption operation. Both input and output data are transmitted by CPU. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

• **aes_cbc192_hard_decrypt**: AES-CBC-192 decryption operation. Both input and output data are transmitted by CPU. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

• **aes_cbc256_hard_encrypt**: AES-CBC-256 encryption operation. Both input and output data are transmitted by CPU. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

• **aes_cbc256_hard_decrypt**: AES-CBC-256 decryption operation. Both input and output data are transmitted by CPU. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

• **aes_gcm128_hard_encrypt**: AES-GCM-128 encryption operation. Both input and output data are transmitted by CPU.

• **aes_gcm128_hard_decrypt**: AES-GCM-128 decryption operation. Both input and output data are transmitted by CPU.

• **aes_gcm192_hard_encrypt**: AES-GCM-192 encryption operation. Both input and output data are transmitted by CPU.

• **aes_gcm192_hard_decrypt**: AES-GCM-192 decryption operation. Both input and output data are transmitted by CPU.

• **aes_gcm256_hard_encrypt**: AES-GCM-256 encryption operation. Both input and output data are transmitted by CPU.

• **aes_gcm256_hard_decrypt**: AES-GCM-256 decryption operation. Both input and output data are transmitted by CPU.

• **aes_ecb128_hard_encrypt_dma**: AES-ECB-128 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

• **aes_ecb128_hard_decrypt_dma**: AES-ECB-128 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

• **aes_ecb192_hard_encrypt_dma**: AES-ECB-192 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

• **aes_ecb192_hard_decrypt_dma**: AES-ECB-192 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

• **aes_ecb256_hard_encrypt_dma**: AES-ECB-256 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

• **aes_ecb256_hard_decrypt_dma**: AES-ECB-256 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

• **aes_cbc128_hard_encrypt_dma**: AES-CBC-128 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

• **aes_cbc128_hard_decrypt_dma**: AES-CBC-128 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

• **aes_cbc192_hard_encrypt_dma**: AES-CBC-192 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

• **aes_cbc192_hard_decrypt_dma**: AES-CBC-192 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

• **aes_cbc256_hard_encrypt_dma**: AES-CBC-256 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

• **aes_cbc256_hard_decrypt_dma**: AES-CBC-256 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

• **aes_gcm128_hard_encrypt_dma**: AES-GCM-128 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma.

• **aes_gcm128_hard_decrypt_dma**: AES-GCM-128 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma.

• **aes_gcm192_hard_encrypt_dma**: AES-GCM-192 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma.

• **aes_gcm192_hard_decrypt_dma**: AES-GCM-192 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma.

• **aes_gcm256_hard_encrypt_dma**: AES-GCM-256 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma.

• **aes_gcm256_hard_decrypt_dma**: AES-GCM-256 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma.

• **aes_init**: AES Initialization of the hardware module.

• **aes_process:** AES hardware module performs encryption and decryption operations.

• **gcm_get_tag**: Get the tag after the AES-GCM calculation is completed.

3. Experiment principle

3.1 Through a for loop, the time corresponding to the three modes of AES (ECB/CBC/GCM) will be printed out.

```
for (cipher = AES_ECB; cipher < AES_CIPHER_MAX; cipher++)
{
    printf("[%s] test all byte ... \n", cipher_name[cipher]);
    if (AES_CHECK_FAIL == aes_check_all_byte(cipher))
    {
        printf("aes %s check_all_byte fail\n", cipher_name[cipher]);
        return -1;
    }

    printf("[%s] test all key ... \n", cipher_name[cipher]);
    if (AES_CHECK_FAIL == aes_check_all_key(cipher))
    {
        printf("aes %s check_all_key fail\n", cipher_name[cipher]);
        return -1;
    }

    printf("[%s] test all iv ... \n", cipher_name[cipher]);
    if (AES_CHECK_FAIL == aes_check_all_iv(cipher))
    {
        printf("aes %s check_all_iv fail\n", cipher_name[cipher]);
        return -1;
    }

    printf("[%s] [%ld bytes] cpu time = %ld us, dma time = %ld us, soft time = %ld us\n", cipher_name[cipher],
           AES_TEST_DATA_LEN,
           cycle[cipher][AES_HARD][AES_CPU]/(sysctl_clock_get_freq(SYSCTL_CLOCK_CPU)/1000000),
           cycle[cipher][AES_HARD][AES_DMA]/(sysctl_clock_get_freq(SYSCTL_CLOCK_CPU)/1000000),
           cycle[cipher][AES_SOFT][AES_CPU]/(sysctl_clock_get_freq(SYSCTL_CLOCK_CPU)/1000000));
}
```

3.2 AES detects all bytes.

```
check_result_t aes_check_all_byte(aes_cipher_mode_t cipher)
{
    uint32_t check_tag = 0;
    uint32_t index = 0;
    size_t data_len = 0;
    memset(aes_hard_in_data, 0, AES_TEST_PADDING_LEN);
    if (cipher == AES_GCM)
        iv_len = iv_gcm_len;
    for (index = 0; index < (AES_TEST_DATA_LEN < 256 ? AES_TEST_DATA_LEN : 256); index++)
    {
        aes_hard_in_data[index] = index;
        data_len++;

        AES_DBG("[%s] test num: %ld \n", cipher_name[cipher], data_len);
        if (aes_check(aes_key, key_len, aes_iv, iv_len, aes_aad, aad_len, cipher, aes_hard_in_data, data_len)
            == AES_CHECK_FAIL)
            check_tag = 1;
    }

    memset(aes_hard_in_data, 0, AES_TEST_PADDING_LEN);
    get_time_flag = 1;
    data_len = AES_TEST_DATA_LEN;
    AES_DBG("[%s] test num: %ld \n", cipher_name[cipher], data_len);
    for (index = 0; index < data_len; index++)
        aes_hard_in_data[index] = index % 256;
    if (aes_check(aes_key, key_len, aes_iv, iv_len, aes_aad, aad_len, cipher, aes_hard_in_data, data_len)
        == AES_CHECK_FAIL)
        check_tag = 1;
    get_time_flag = 0;
    if(check_tag)
        return AES_CHECK_FAIL;
    else
        return AES_CHECK_PASS;
}
```

3.3 AES detects all keys.

```
check_result_t aes_check_all_key(aes_cipher_mode_t cipher)
{
    size_t data_len = 0;
    uint32_t index = 0;
    uint32_t i = 0;
    uint32_t check_tag = 0;

    memset(aes_hard_in_data, 0, AES_TEST_PADDING_LEN);
    if (cipher == AES_GCM)
        iv_len = iv_gcm_len;
    data_len = AES_TEST_DATA_LEN;
    for (index = 0; index < data_len; index++)
        aes_hard_in_data[index] = index;
    for (i = 0; i < (256 / key_len); i++)
    {
        for (index = i * key_len; index < (i * key_len) + key_len; index++)
            aes_key[index - (i * key_len)] = index;
        if (aes_check(aes_key, key_len, aes_iv, iv_len, aes_aad, aad_len, cipher, aes_hard_in_data, data_len)
            == AES_CHECK_FAIL)
            check_tag = 1;
    }
    if(check_tag)
        return AES_CHECK_FAIL;
    else
        return AES_CHECK_PASS;
}
```

3.4 AES detects all IVs.

```
check_result_t aes_check_all_iv(aes_cipher_mode_t cipher)
{
    size_t data_len = 0;
    uint32_t index = 0;
    uint32_t i = 0;
    uint8_t check_tag = 0;

    memset(aes_hard_in_data, 0, AES_TEST_PADDING_LEN);
    if (cipher == AES_GCM)
        iv_len = iv_gcm_len;
    data_len = AES_TEST_DATA_LEN;
    for (index = 0; index < data_len; index++)
        aes_hard_in_data[index] = index;
    for (i = 0; i < (256 / iv_len); i++)
    {
        for (index = i * iv_len; index < (i * iv_len) + iv_len; index++)
            aes_iv[index - (i * iv_len)] = index;
        if (aes_check(aes_key, key_len, aes_iv, iv_len, aes_aad, aad_len, cipher, aes_hard_in_data, data_len)
            == AES_CHECK_FAIL)
            check_tag = 1;
    }
    if(check_tag)
        return AES_CHECK_FAIL;
    else
        return AES_CHECK_PASS;
}
```

3.5 Compile and debug, burn and run

Copy the aes256 to the src directory in the SDK.

Then, enter the build directory and run the following command to compile.

cmake .. -DPROJ=aes256 -G "MinGW Makefiles"

make

```
[100%] Linking C executable aes256
Generating .bin file ...
[100%] Built target aes256
PS C:\K210\SDK\kendryte-standalone-sdk-develop\build>
```

After the compilation is complete, the **aes256.bin** file will be generated in the build folder.

We need to use the type-C data cable to connect the computer and the K210 development board.

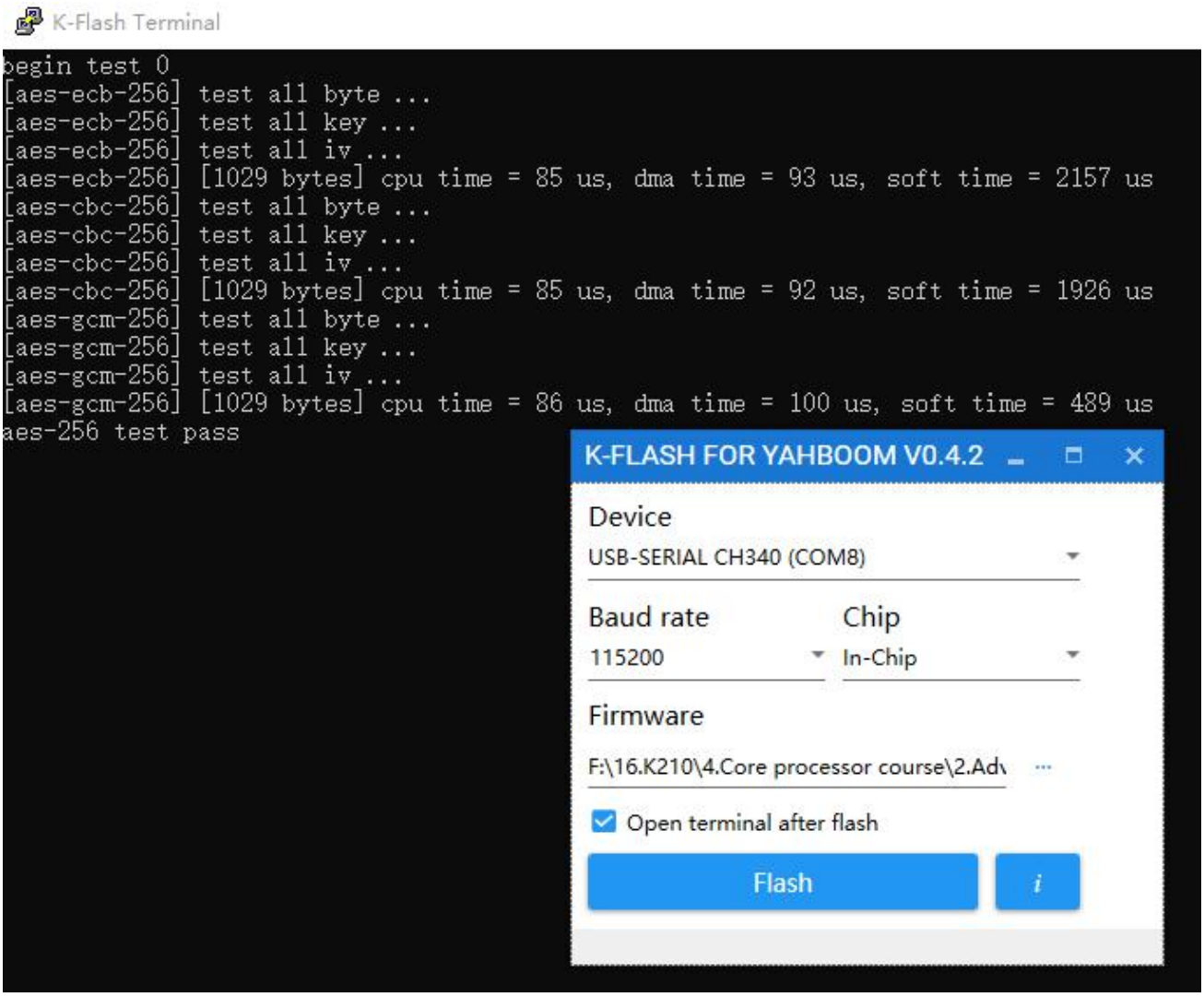
Open kflash, select the corresponding device, and then burn the **aes256.bin** file to the K210 development board.

4. Experimental phenomenon

After the firmware is write, a terminal interface will pop up.It will be printed out hash value.

It will print out the data encryption processing time of AES ECB mode, CBC mode and GCM mode, including the comparison time of software and hardware.

Tip: The CPU and DMA use hardware AES.



6. Experiment summary

6.1 AES encryption algorithm is divided into multiple modes, and the encryption method of each mode is different.

6.2 AES hardware accelerator saves time than using software encryption alone.

Appendix -- API

Header file is aes.h

aes_ecb128_hard_encrypt

Description:

AES-ECB-128 encryption operation. Both input and output data are transmitted by CPU. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

Function prototype:

void aes_ecb128_hard_encrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
input_key	AES-ECB-128 Encrypted key	Input
input_data	AES-ECB-128 Waiting for encrypted visible data	Input
input_len	AES-ECB-128 Length of waiting for encrypted visible data	Input
output_data	AES-ECB-128 Result of AES-ECB-128 encryption operation is stored in this buffer. The size of this buffer needs to be aligned with 16 bytes.	Output

Return value: no

aes_ecb128_hard_decrypt

Description:

AES-ECB-128 decryption operation. Both input and output data are transmitted by CPU. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

Function prototype:

void aes_ecb128_hard_decrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
input_key	AES-ECB-128 Decrypted key	Input
input_data	AES-ECB-128 Waiting for decrypted invisible data	Input
input_len	AES-ECB-128 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-ECB-128 decryption operation is stored in this buffer. The size of this buffer needs to be aligned with 16 bytes.	Output

Return value: no

aes_ecb192_hard_encrypt

Description:

AES-ECB-192 encryption operation. Both input and output data are transmitted by CPU. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

Function prototype:

void aes_ecb192_hard_encrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
input_key	AES-ECB-192 Encrypted key	Input
input_data	AES-ECB-192 Waiting for encrypted visible data	Input
input_len	AES-ECB-192 Length of waiting for encrypted visible data	Input
output_data	The result of AES-ECB-192 encryption operation is stored in this buffer. The size of this buffer needs to be aligned with 16 bytes.	Output

Return value: no

aes_ecb192_hard_decrypt

Description:

AES-ECB-192 decryption operation. Both input and output data are transmitted by CPU. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

Function prototype:

void aes_ecb192_hard_decrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
input_key	AES-ECB-192 Decrypted key	Input
input_data	AES-ECB-192 Waiting for decrypted invisible data	Input
input_len	AES-ECB-192 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-ECB-1 decryption operation is stored in this buffer. The size of this buffer needs to be aligned with 16 bytes.	Output

Return value: no

aes_ecb256_hard_encrypt

Description:

AES-ECB-256 encryption operation. Both input and output data are transmitted by CPU. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

Function prototype:

void aes_ecb256_hard_encrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
input_key	AES-ECB-256 Encrypted key	Input
input_data	AES-ECB-256 Waiting for encrypted visible data	Input
input_len	AES-ECB-256 Length of waiting for encrypted visible data	Input
output_data	The result of AES-ECB-256 encryption operation is stored in this buffer. The size of this buffer needs to be aligned with 16 bytes.	Output

Return value: no

aes_ecb256_hard_decrypt

Description:

AES-ECB-256 decryption operation. Both input and output data are transmitted by CPU. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

Function prototype:

void aes_ecb256_hard_decrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
input_key	AES-ECB-256 Decrypted key	Input
input_data	AES-ECB-256 Waiting for decrypted invisible data	Input
input_len	AES-ECB-256 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-ECB-256 decryption operation is stored in this buffer. The size of this buffer needs to be aligned with 16 bytes.	Output

Return value: no

aes_cbc128_hard_encrypt

Description:

AES-CBC-128 encryption operation. Both input and output data are transmitted by CPU. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

Function prototype:

void aes_cbc128_hard_encrypt(cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
context	AES-CBC-128 Encryption calculation structure, including encryption key and offset vector	Input
input_data	AES-CBC-128 Waiting for encrypted visible data	Input
input_len	AES-CBC-128 Length of waiting for encrypted visible data	Input
output_data	The result of AES-CBC-128 encryption operation is stored in this buffer. The size of this buffer needs to be aligned with 16 bytes.	Output

Return value: no

aes_cbc128_hard_decrypt

Description:

AES-CBC-128 decryption operation. Both input and output data are transmitted by CPU. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

Function prototype:

void aes_cbc128_hard_decrypt(cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
context	AES-CBC-128 The structure of the decryption calculation, including the decryption key and offset vector	Input
input_data	AES-CBC-128 Waiting for decrypted invisible data	Input
input_len	AES-CBC-128 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-ECB-128 decryption operation is stored in this buffer. The size of this buffer needs to be aligned with 16 bytes.	Output

Return value: no

aes_cbc192_hard_encrypt

Description:

AES-CBC-192 encryption operation. Both input and output data are transmitted by CPU. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

Function prototype:

void aes_cbc192_hard_encrypt(cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
context	AES-CBC-192 Encryption calculation structure, including encryption key and offset vector	Input
input_data	AES-CBC-192 Waiting for encrypted visible data	Input
input_len	AES-CBC-192 Length of waiting for encrypted visible data	Input
output_data	The result of AES-ECB-192 decryption operation is stored in this buffer. The size of this buffer needs to be aligned with 16 bytes.	Output

Return value: no

aes_cbc192_hard_decrypt

Description:

AES-CBC-192 decryption operation. Both input and output data are transmitted by CPU. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

Function prototype:

void aes_cbc192_hard_decrypt(cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
context	AES-CBC-192 The structure of the decryption calculation, including the decryption key and offset vector	Input
input_data	AES-CBC-192 Waiting for decrypted invisible data	Input
input_len	AES-CBC-192 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-ECB-192 decryption operation is stored in this buffer. The size of this buffer needs to be aligned with 16 bytes.	Output

Return value: no

aes_cbc256_hard_encrypt

Description:

AES-CBC-256 encryption operation. Both input and output data are transmitted by CPU. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

Function prototype:

void aes_cbc256_hard_encrypt(cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
context	AES-CBC-256 Encryption calculation structure, including encryption key and offset vector	Input
input_data	AES-CBC-256 Waiting for encrypted visible data	Input
input_len	AES-CBC-256 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-ECB-256 decryption operation is stored in this buffer. The size of this buffer needs to be aligned with 16 bytes.	Output

Return value: no

aes_cbc256_hard_decrypt

Description:

AES-CBC-256 decryption operation. Both input and output data are transmitted by CPU. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

Function prototype:

void aes_cbc256_hard_decrypt(uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
context	AES-CBC-256 The structure of the decryption calculation, including the decryption key and offset vector	Input
input_data	AES-CBC-256 Waiting for decrypted invisible data	Input
input_len	AES-CBC-256 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-ECB-256 decryption operation is stored in this buffer. The size of this buffer needs to be aligned with 16 bytes.	Output

Return value: no

aes_gcm128_hard_encrypt

Description:

AES-GCM-128 encryption operation. Both input and output data are transmitted by CPU.

Function prototype:

void aes_gcm128_hard_encrypt(gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data, uint8_t *gcm_tag)

Parameter:

Parameter name	Description	Input/Output
context	AES-GCM-128 Encryption calculation structure, including encryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-128 Waiting for encrypted visible data	Input
input_len	AES-GCM-128 Length of waiting for encrypted visible data	Input

Parameter name	Description	Input/Output
output_data	The result of AES-ECB-128 encryption operation is stored in this buffer.	Output
gcm_tag	The tag after AES-GCM-128 encryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_gcm128_hard_decrypt

Description:

AES-GCM-128 decryption operation. Both input and output data are transmitted by CPU.

Function prototype:

void aes_gcm128_hard_decrypt(gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data, uint8_t *gcm_tag)

Parameter:

Parameter name	Description	Input/Output
context	AES-GCM-128 The structure of the decryption calculation, including the decryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-128 Waiting for decrypted invisible data	Input
input_len	AES-GCM-128 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-ECB-128 decryption operation is stored in this buffer.	Input
gcm_tag	The tag after AES-GCM-128 decryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_gcm192_hard_encrypt

Description:

AES-GCM-192 encryption operation. Both input and output data are transmitted by CPU.

Function prototype:

void aes_gcm192_hard_encrypt(gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data, uint8_t *gcm_tag)

Parameter:

Parameter name	Description	Input/Output
context	AES-GCM-192 Encryption calculation structure, including encryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-192 Waiting for encrypted visible data	Input
input_len	AES-GCM-192 Length of waiting for encrypted visible data	Input
output_data	The result of AES-ECB-192 encryption operation is stored in this buffer.	Output

Parameter name	Description	Input/Output
gcm_tag	The result of AES-GCM-192 encryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_gcm192_hard_decrypt

Description:

AES-GCM-192 decryption operation. Both input and output data are transmitted by CPU.

Function prototype:

void aes_gcm192_hard_decrypt(gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data, uint8_t *gcm_tag)

Parameter:

Parameter name	Description	Input/Output
context	AES-GCM-192 The structure of the decryption calculation, including the decryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-192 Waiting for decrypted invisible data	Input
input_len	AES-GCM-192 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-GCM-192 decryption operation is stored in this buffer.	Output
gcm_tag	The tag after AES-GCM-192 decryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_gcm256_hard_encrypt

Description:

AES-GCM-256 encryption operation. Both input and output data are transmitted by CPU.

Function prototype:

void aes_gcm256_hard_encrypt(gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data, uint8_t *gcm_tag)

Parameter:

Parameter name	Description	Input/Output
context	AES-GCM-256 Encryption calculation structure, including encryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-256 Waiting for encrypted visible data	Input
input_len	AES-GCM-256 Length of waiting for encrypted visible data	Input
output_data	The result of AES-GCM-256 encryption operation is stored in this buffer.	Output
gcm_tag	The result of AES-GCM-256 encryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_gcm256_hard_decrypt

Description:

AES-GCM-256 decryption operation. Both input and output data are transmitted by CPU.

Function prototype:

void aes_gcm256_hard_decrypt(gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data, uint8_t *gcm_tag)

Parameter:

Parameter name	Description	Input/Output
context	AES-GCM-256 The structure of the decryption calculation, including the decryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-256 Waiting for decrypted invisible data	Input
input_len	AES-GCM-256 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-GCM-256 decryption operation is stored in this buffer.	Output
gcm_tag	The tag after AES-GCM-256 decryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_ecb128_hard_encrypt_dma

Description:

AES-ECB-128 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

Function prototype:

void aes_ecb128_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
input_key	AES-ECB-128 encryption key	Input
input_data	AES-ECB-128 Waiting for encrypted visible data	Input
input_len	AES-ECB-128 Length of waiting for encrypted visible data	Input
output_data	The result of AES-ECB-128 encryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_ecb128_hard_decrypt_dma

Description:

AES-ECB-128 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

Function prototype:

void aes_ecb128_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
input_key	AES-ECB-128 decryption key	Input
input_data	AES-ECB-128 Waiting for decrypted invisible data	Input
input_len	AES-ECB-128 Length of waiting for decrypted invisible data	Input
output_data	The tag after AES-ECB-128 decryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_ecb192_hard_encrypt_dma

Description:

AES-ECB-192 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

Function prototype:

void aes_ecb192_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
input_key	AES-ECB-192 encryption key	Input
input_data	AES-ECB-192 Waiting for encrypted visible data	Input
input_len	AES-ECB-192 Length of waiting for encrypted visible data	Input
output_data	The result of AES-ECB-192 encryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_ecb192_hard_decrypt_dma

Description:

AES-ECB-192 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

Function prototype:

void aes_ecb192_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
input_key	AES-ECB-192 decryption key	Input
input_data	AES-ECB-192 Waiting for decrypted invisible data	Input
input_len	AES-ECB-192 Length of waiting for decrypted invisible data	Input
output_data	The tag after AES-ECB-192 decryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_ecb256_hard_encrypt_dma

Description:

AES-ECB-256 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

Function prototype:

void aes_ecb256_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
input_key	AES-ECB-256 encryption key	Input
input_data	AES-ECB-256 waiting for encrypted visible data	Input
input_len	AES-ECB-256 length of waiting for encrypted visible data	Input
output_data	The result of AES-ECB-256 encryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 byte	Output

Return value: no

aes_ecb256_hard_decrypt_dma

Description:

AES-ECB-256 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma. ECB is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled. ECB mode does not use vectors.

Function prototype:

void aes_ecb256_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
input_key	AES-ECB-256 decryption key	Input
input_data	AES-ECB-256 Waiting for decrypted invisible data	Input
input_len	AES-ECB-256 Length of waiting for decrypted invisible data	Input
output_data	The tag after AES-ECB-256 decryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_cbc128_hard_encrypt_dma

Description:

AES-CBC-128 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

Function prototype:

void aes_cbc128_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num, cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-CBC-128 encryption calculation structure, including encryption key and offset vector	Input
input_data	AES-CBC-128 waiting for encrypted visible data	Input
input_len	AES-CBC-128 length of waiting for encrypted visible data	Input
output_data	The result of AES-ECB-128 encryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 byte	Output

Return value: no

aes_cbc128_hard_decrypt_dma

Description:

AES-CBC-128 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

Function prototype:

void aes_cbc128_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-CBC-128 decryption calculation structure, including decryption key and offset vector	Input
input_data	AES-CBC-128 Waiting for decrypted invisible data	Input
input_len	AES-CBC-128 Length of waiting for decrypted invisible data	Input
output_data	The tag after AES-CBC-128 decryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_cbc192_hard_encrypt_dma

Description:

AES-CBC-192 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

Function prototype:

void aes_cbc192_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num, cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-CBC-192 encryption calculation structure, including encryption key and offset vector	Input
input_data	AES-CBC-192 waiting for encrypted visible data	Input
input_len	AES-CBC-192 length of waiting for encrypted visible data	Input
output_data	The result of AES-CBC-192 encryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 byte	Output

Return value: no

aes_cbc192_hard_decrypt_dma

Description:

AES-CBC-192 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

Function prototype:

void aes_cbc192_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-CBC-192 decryption calculation structure, including decryption key and offset vector	Input
input_data	AES-CBC-192 Waiting for decrypted invisible data	Input
input_len	AES-CBC-192 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-CBC-192 decryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_cbc256_hard_encrypt_dma

Description:

AES-CBC-256 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

Function prototype:

void aes_cbc256_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num, cbc_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-CBC-256 encryption calculation structure, including encryption key and offset vector	Input
input_data	AES-CBC-256 waiting for encrypted visible data	Input
input_len	AES-CBC-256 length of waiting for encrypted visible data	Input
output_data	The result of AES-CBC-256 encryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 byte	Output

Return value: no

aes_cbc256_hard_decrypt_dma

Description:

AES-CBC-256 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma. CBC is encrypted according to a fixed-size 16bytes block. If the block size is insufficient, it will be filled.

Function prototype:

void aes_cbc256_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, uint8_t *input_key, uint8_t *input_data, size_t input_len, uint8_t *output_data)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-CBC-256 decryption calculation structure, including decryption key and offset vector	Input
input_data	AES-CBC-256 Waiting for decrypted invisible data	Input
input_len	AES-CBC-256 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-CBC-256 decryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_gcm128_hard_encrypt_dma

Description:

AES-GCM-128 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma.

Function prototype:

void aes_gcm128_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num, gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data, uint8_t *gcm_tag)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-GCM-128 encryption calculation structure, including encryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-128 waiting for encrypted visible data	Input
input_len	AES-GCM-128 length of waiting for encrypted visible data	Input
output_data	The result of AES-GCM-128 encryption operation is stored in this buffer. Since the minimum granularity of DMA handling data is 4 bytes, Therefore, it is necessary to ensure that the buffer size is at least an integer multiple of 4bytes.	Output
gcm_tag	The tag after AES-GCM-128 encryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_gcm128_hard_decrypt_dma

Description:

AES-GCM-128 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma.

Function prototype:

void aes_gcm128_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data, uint8_t *gcm_tag)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-GCM-128 decryption calculation structure, including decryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-128 Waiting for decrypted invisible data	Input
input_len	AES-GCM-128 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-GCM-128 decryption operation is stored in this buffer. Since the minimum granularity of DMA handling data is 4 bytes, Therefore, it is necessary to ensure that the buffer size is at least an integer multiple of 4bytes.	Input
gcm_tag	The tag after AES-GCM-128 decryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_gcm192_hard_encrypt_dma

Description:

AES-GCM-192 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma.

Function prototype:

void aes_gcm192_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num, gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data, uint8_t *gcm_tag)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-GCM-192 encryption calculation structure, including encryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-192 waiting for encrypted visible data	Input
input_len	AES-GCM-192 length of waiting for encrypted visible data	Input
output_data	The result of AES-GCM-192 encryption operation is stored in this buffer. Since the minimum granularity of DMA handling data is 4 bytes,	Output

Parameter name	Description	Input/Output
	Therefore, it is necessary to ensure that the buffer size is at least an integer multiple of 4bytes.	
gcm_tag	The tag after AES-GCM-192 encryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_gcm192_hard_decrypt_dma

Description:

AES-GCM-192 decryption operation. Input data is transmitted by CPU, output data is transmitted by dma.

Function prototype:

void aes_gcm192_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data, uint8_t *gcm_tag)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-GCM-192 decryption calculation structure, including decryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-192 Waiting for decrypted invisible data	Input
input_len	AES-GCM-192 Length of waiting for decrypted invisible data	Input
output_data	The result of AES-GCM-19 decryption operation is stored in this buffer. Since the minimum granularity of DMA handling data is 4 bytes, Therefore, it is necessary to ensure that the buffer size is at least an integer multiple of 4bytes.	Output
gcm_tag	The tag after AES-GCM-128 decryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_gcm256_hard_encrypt_dma

Description:

AES-GCM-256 encryption operation. Input data is transmitted by CPU, output data is transmitted by dma.

Function prototype:

void aes_gcm256_hard_encrypt_dma(dmac_channel_number_t dma_receive_channel_num, gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data, uint8_t *gcm_tag)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-GCM-256 encryption calculation structure, including encryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-256 waiting for encrypted visible data	Input
input_len	AES-GCM-256 length of waiting for encrypted visible data	Input
output_data	The result of AES-GCM-256 encryption operation is stored in this buffer. Since the minimum granularity of DMA handling data is 4 bytes, Therefore, it is necessary to ensure that the buffer size is at least an integer multiple of 4bytes.	Output
gcm_tag	The tag after AES-GCM-256 encryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes	Output

Return value: no

aes_gcm256_hard_decrypt_dma

Description:

AES-GCM-256 decryption operation. Input data is transmitted using cpu, and output data is transmitted using dma.

Function prototype:

void aes_gcm256_hard_decrypt_dma(dmac_channel_number_t dma_receive_channel_num, gcm_context_t *context, uint8_t *input_data, size_t input_len, uint8_t *output_data, uint8_t *gcm_tag)

Parameter:

Parameter name	Description	Input/Output
dma_receive_channel_num	DMA channel number of AES output data	Input
context	AES-GCM-256 decryption calculation structure, including decryption key/offset vector/aad/aad length	Input
input_data	AES-GCM-256 Waiting for decrypted invisible data	Input
input_len	AES-GCM-256 Length of waiting for decrypted invisible data	Input

Parameter name	Description	Input/Output
output_data	The result of AES-GCM-256 decryption operation is stored in this buffer. Since the minimum granularity of DMA handling data is 4 bytes, Therefore, it is necessary to ensure that the buffer size is at least an integer multiple of 4bytes.	Output
gcm_tag	The tag after AES-GCM-256 decryption operation is stored in this buffer. The buffer size needs to be guaranteed to be 16 bytes.	Output

Return value: no

aes_init

Description:

Initialization of AES hardware module.

Function prototype:

void aes_init(uint8_t *input_key, size_t input_key_len, uint8_t *iv,size_t iv_len, uint8_t *gcm_aad,aes_cipher_mode_t cipher_mode, aes_encrypt_sel_t encrypt_sel, size_t gcm_aad_len, size_t input_data_len)

Parameter:

Parameter name	Description	Input/Output
input_key	The key to be encrypted/decrypted	Input
input_key_len	The length of the key to be encrypted/decrypted	Input
iv	Iv data used for AES encryption and decryption	Input
iv_len	The length of iv data used for AES encryption and decryption is fixed at 16 bytes for CBC and 12 bytes for GCM.	Output
gcm_aad	Aad data used for AES-GCM encryption and decryption	Output
cipher_mode	The type of encryption and decryption performed by the AES hardware module, supporting AES_CBC/AES_ECB/AES_GCM	Input
encrypt_sel	AES hardware module execution mode: encryption or decryption	Input
gcm_aad_len	The length of aad data used for AES-GCM encryption and decryption	Input
input_data_len	Length of data to be encrypted/decrypted	Input

Return value: no

aes_process

Description:
AES hardware module performs encryption and decryption operations.
Function prototype:
void aes_process(uint8_t *input_data, uint8_t *output_data, size_t input_data_len, aes_cipher_mode_t cipher_mode)
Parameter:

Parameter name	Description	Input/Output
input_data	This buffer stores the data to be encrypted/decrypted	Input
output_data	This buffer stores the output result of encryption/decryption	Output
input_data_len	The length of the data to be encrypted/decrypted	Input
cipher_mode	The type of encryption and decryption performed by the AES hardware module, supporting AES_CBC/AES_ECB/AES_GCM	Input

Return value: no

gcm_get_tag
Description:
Get the tag after the AES-GCM calculation is completed.\
Function prototype:
void gcm_get_tag(uint8_t *gcm_tag)
Parameter:

Parameter name	Description	Input/Output
gcm_tag	This buffer stores the AES-GCM encrypted/decrypted tag, which is fixed at 16 bytes	Output

Return value: no

Eg:

```
cbc_context_t cbc_context;
cbc_context.input_key = cbc_key;
cbc_context.iv = cbc_iv;
aes_cbc128_hard_encrypt(&cbc_context, aes_input_data, 16L, aes_output_data);
memcpy(aes_input_data, aes_output_data, 16L);
aes_cbc128_hard_decrypt(&cbc_context, aes_input_data, 16L, aes_output_data);
```

Data type
The related data types and data structure are defined as follows:

- **aes_cipher_mode_t:** AES encryption/decryption method.

aes_cipher_mode_t
Description:
AES encryption/decryption method.
Define

```
typedef enum _aes_cipher_mode
{
    AES_ECB = 0,
    AES_CBC = 1,
    AES_GCM = 2,
    AES_CIPHER_MAX
} aes_cipher_mode_t;
```

- **gcm_context_t**:The structure used by the parameters of AES-GCM encryption/decryption

gcm_context_t

Description:

The structure used for the AES-GCM parameters, including the key, offset vector, aad data, and aad data length.

Define

```
typedef struct _gcm_context
{
    uint8_t *input_key;
    uint8_t *iv;
    uint8_t *gcm_aad;
    size_t gcm_aad_len;
} gcm_context_t;
```

cbc_context_t：The structure used by the parameters of AES-CBC encryption/decryption

cbc_context_t

Description:

The structure used for AES-CBC parameters, including key and offset vector.

Define

```
typedef struct _cbc_context
{
    uint8_t *input_key;
    uint8_t *iv;
} cbc_context_t;
```

member

Member name	Description
AES_ECB	ECB encryption/decryption
AES_CBC	CBC encryption/decryption
AES_GCM	GCM encryption/decryption