4.2 高级加密加速器

一、实验目的

本节课主要学习 K210 芯片中的高级加密加速器 AES 的功能。

二、实验准备

1.实验元件

K210 芯片中的高级加密加速器 AES

2.元件特性

K210 内置 AES(高级加密加速器),相对于软件可以极大的提高 AES 运算速度。AES 加速器支持多种加密/解密模式(ECB,CBC,GCM), 多种长度的 KEY(128,192,256)的运算。

AES 加速器是用来加密和解密的模块，具体性能如下：

- 支持 ECB，CBC，GCM 三种加密方式
- 支持 128 位，192 位，256 位三种长度的 KEY
- KEY 可以通过软件配置，受到硬件电路保护
- 支持 DMA 传输

4.SDK 中对应 API 功能

对应的头文件 aes.h

为用户提供以下接口：

- aes_ecb128_hard_encrypt:AES-ECB-128 加密运算。输入输出数据都使用 cpu 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

- aes_ecb128_hard_decrypt：AES-ECB-128 解密运算。输入输出数据都使用 cpu 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。
- aes_ecb192_hard_encrypt：AES-ECB-192 加密运算。输入输出数据都使用 cpu 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。
- aes_ecb192_hard_decrypt：AES-ECB-192 解密运算。输入输出数据都使用 cpu 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。
- aes_ecb256_hard_encrypt：AES-ECB-256 加密运算。输入输出数据都使用 cpu 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。
- aes_ecb256_hard_decrypt：AES-ECB-256 解密运算。输入输出数据都使用 cpu 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。
- aes_cbc128_hard_encrypt：AES-CBC-128 加密运算。输入输出数据都使用 cpu 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。
- aes_cbc128_hard_decrypt：AES-CBC-128 解密运算。输入输出数据都使用 cpu 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。
- aes_cbc192_hard_encrypt：AES-CBC-192 加密运算。输入输出数据都使用 cpu 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。
- aes_cbc192_hard_decrypt：AES-CBC-192 解密运算。输入输出数据都使用 cpu 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。
- aes_cbc256_hard_encrypt：AES-CBC-256 加密运算。输入输出数据都使用 cpu 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。
- aes_cbc256_hard_decrypt：AES-CBC-256 解密运算。输入输出数据都使用 cpu 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。
- aes_gcm128_hard_encrypt：AES-GCM-128 加密运算。输入输出数据都使用 cpu 传输。
- aes_gcm128_hard_decrypt：AES-GCM-128 解密运算。输入输出数据都使用 cpu 传输。
- aes_gcm192_hard_encrypt：AES-GCM-192 加密运算。输入输出数据都使用 cpu 传输。
- aes_gcm192_hard_decrypt：AES-GCM-192 解密运算。输入输出数据都使用 cpu 传输。
- aes_gcm256_hard_encrypt：AES-GCM-256 加密运算。输入输出数据都使用 cpu 传输。
- aes_gcm256_hard_decrypt：AES-GCM-256 解密运算。输入输出数据都使用 cpu 传输。
- aes_ecb128_hard_encrypt_dma：AES-ECB-128 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。
- aes_ecb128_hard_decrypt_dma：AES-ECB-128 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。
- aes_ecb192_hard_encrypt_dma：AES-ECB-192 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。
- aes_ecb192_hard_decrypt_dma：AES-ECB-192 解密运算。输入数据使用 cpu 传输，输出数

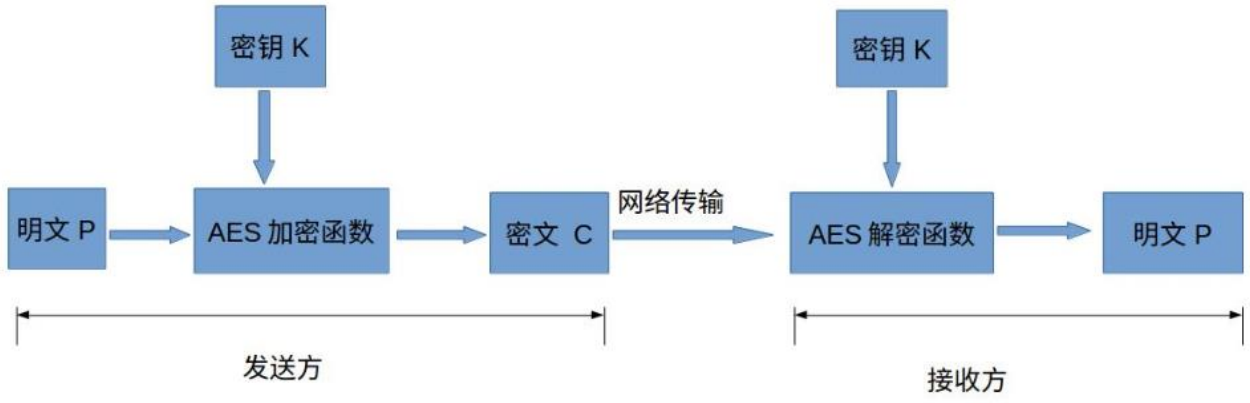
据都使用 dma 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

- aes_ecb256_hard_encrypt_dma：AES-ECB-256 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。
- aes_ecb256_hard_decrypt_dma：AES-ECB-256 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。
- aes_cbc128_hard_encrypt_dma：AES-CBC-128 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。
- aes_cbc128_hard_decrypt_dma：AES-CBC-128 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。
- aes_cbc192_hard_encrypt_dma：AES-CBC-192 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。
- aes_cbc192_hard_decrypt_dma：AES-CBC-192 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。
- aes_cbc256_hard_encrypt_dma：AES-CBC-256 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。
- aes_cbc256_hard_decrypt_dma：AES-CBC-256 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。
- aes_gcm128_hard_encrypt_dma：AES-GCM-128 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。
- aes_gcm128_hard_decrypt_dma：AES-GCM-128 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。
- aes_gcm192_hard_encrypt_dma：AES-GCM-192 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。
- aes_gcm192_hard_decrypt_dma：AES-GCM-192 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。
- aes_gcm256_hard_encrypt_dma：AES-GCM-256 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。
- aes_gcm256_hard_decrypt_dma：AES-GCM-256 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。
- aes_init：AES 硬件模块的初始化。
- aes_process：AES 硬件模块执行加密解密操作。
- gcm_get_tag：获取 AES-GCM 计算结束后的 tag。

三、实验原理

高级加密标准(AES,Advanced Encryption Standard)为最常见的对称加密算法。

对称加密算法也就是加密和解密用相同的密钥，具体的加密流程如下图



明文 P：没有经过加密的数据。

密钥 K：用来加密明文的密码，在对称加密算法中，加密与解密的密钥是相同的。

不可泄露。

密文 C：经过加密处理后的数据。

AES 解密函数：解密的具体实现方法，输入密文，还原出明文。

在 AES 标准规范中，分组长度只能是 128 位，也就是说，每个分组为 16 个字节（每个字节 8 位）。密钥的长度可以使用 128 位、192 位或 256 位。密钥的长度不同，推荐加密轮数也不同，如下表所示：

AES	密钥长度（32 位比特字）	分组长度(32 位比特字)	加密轮数
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

AES-GCM 模式，Galois/Counter Mode ：提供对消息的加密和完整性校验。

AES-ECB 模式，Electronic Codebook Book 电子密码本模式：将整个明文分成若干相同的组，然后对每一组进行加密。

AES-CBC 模式，Cipher Block Chaining 密码分组链接模式：将明文切分成若干个组，每个小组与初始组或者上一组的密文组进行异或运算，再与密钥进行加密。

四、实验过程

1. 通过一个 for 循环，把 AES 的三种模式（ECB/CBC/GCM）对应的时间打印出来。

```
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));
}
```

2. AES 检测所有 byte。

```
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. AES 检测所有 key。

```
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;
}
```

4. AES 检测所有 IV。

```
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;
}
```

5.编译调试，烧录运行

把本课程资料中的 aes256 复制到 SDK 中的 src 目录下，然后进入 build 目录，运行以下命令编译。

```
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>
```

编译完成后，在 build 文件夹下会生成 aes256.bin 文件。

使用 type-C 数据线连接电脑与 K210 开发板，打开 kflash，选择对应的设备，再将程序固件烧录到 K210 开发板上。

五、实验现象

烧录固件完成后，系统会自动弹出一个终端窗口，并且打印出 AES 的 ECB 模式、CBC 模式和 GCM 模式对数据加密处理的时间，包括软件和硬件的对比时间，CPU 和 DMA 使用的是硬件的 AES。

C:\Users\Administrator\AppData\Local\Temp\tmpF1A8.tmp

```
begin test 0
[aes-ecb-256] test all byte ...
[aes-ecb-256] test all key ...
[aes-ecb-256] test all iv ...
[aes-ecb-256] [1029 bytes] cpu time = 85 us, dma time = 94 us, soft time = 2151 us
[aes-cbc-256] test all byte ...
[aes-cbc-256] test all key ...
[aes-cbc-256] test all iv ...
[aes-cbc-256] [1029 bytes] cpu time = 85 us, dma time = 92 us, soft time = 1921 us
[aes-gcm-256] test all byte ...
[aes-gcm-256] test all key ...
[aes-gcm-256] test all iv ...
[aes-gcm-256] [1029 bytes] cpu time = 86 us, dma time = 100 us, soft time = 490 us
aes-256 test pass
```

六、实验总结

1. AES 加密算法分为多种模式，每种模式的加密方式不同。

2. AES 硬件加速器比单独使用软件加密更省时间。

附：API

对应的头文件 aes.h

aes_ecb128_hard_encrypt

描述

AES-ECB-128 加密运算。输入输出数据都使用 cpu 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

函数原型

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

参数

参数名称	描述	输入输出
input_key	AES-ECB-128 加密的密钥	输入
input_data	AES-ECB-128 待加密的明文数据	输入
input_len	AES-ECB-128 待加密明文数据的长度	输入
output_data	AES-ECB-128 加密运算后的结果存放在这个 buffer。这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_ecb128_hard_decrypt

描述

AES-ECB-128 解密运算。输入输出数据都使用 cpu 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

函数原型

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

参数

参数名称	描述	输入输出
input_key	AES-ECB-128 解密的密钥	输入
input_data	AES-ECB-128 待解密的密文数据	输入
input_len	AES-ECB-128 待解密密文数据的长度	输入
output_data	AES-ECB-128 解密运算后的结果存放在这个 buffer。这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_ecb192_hard_encrypt

描述

AES-ECB-192 加密运算。输入输出数据都使用 cpu 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

函数原型

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

参数

参数名称	描述	输入输出
input_key	AES-ECB-192 加密的密钥	输入
input_data	AES-ECB-192 待加密的明文数据	输入
input_len	AES-ECB-192 待加密明文数据的长度	输入
output_data	AES-ECB-192 加密运算后的结果存放在这个 buffer。这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_ecb192_hard_decrypt

描述

AES-ECB-192 解密运算。输入输出数据都使用 cpu 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

函数原型

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

参数

参数名称	描述	输入输出
input_key	AES-ECB-192 解密的密钥	输入
input_data	AES-ECB-192 待解密的密文数据	输入
input_len	AES-ECB-192 待解密密文数据的长度	输入
output_data	AES-ECB-192 解密运算后的结果存放在这个 buffer。这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_ecb256_hard_encrypt

描述

AES-ECB-256 加密运算。输入输出数据都使用 cpu 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

函数原型

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

参数

参数名称	描述	输入输出
input_key	AES-ECB-256 加密的密钥	输入
input_data	AES-ECB-256 待加密的明文数据	输入

参数名称	描述	输入输出
input_len	AES-ECB-256 待加密明文数据的长度	输入
output_data	AES-ECB-256 加密运算后的结果存放在这个 buffer。这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_ecb256_hard_decrypt

描述

AES-ECB-256 解密运算。输入输出数据都使用 cpu 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

函数原型

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

参数

参数名称	描述	输入输出
input_key	AES-ECB-256 解密的密钥	输入
input_data	AES-ECB-256 待解密的密文数据	输入
input_len	AES-ECB-256 待解密密文数据的长度	输入
output_data	AES-ECB-256 解密运算后的结果存放在这个 buffer。这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_cbc128_hard_encrypt

描述

AES-CBC-128 加密运算。输入输出数据都使用 cpu 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。

函数原型

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

参数

参数名称	描述	输入输出
context	AES-CBC-128 加密计算的结构体，包含加密密钥与偏移向量	输入
input_data	AES-CBC-128 待加密的明文数据	输入
input_len	AES-CBC-128 待加密明文数据的长度	输入
output_data	AES-CBC-128 加密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_cbc128_hard_decrypt

描述

AES-CBC-128 解密运算。输入输出数据都使用 cpu 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。

函数原型

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

参数

参数名称	描述	输入输出
context	AES-CBC-128 解密计算的结构体，包含解密密钥与偏移向量	输入
input_data	AES-CBC-128 待解密的密文数据	输入
input_len	AES-CBC-128 待解密密文数据的长度	输入
output_data	AES-CBC-128 解密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_cbc192_hard_encrypt

描述

AES-CBC-192 加密运算。输入输出数据都使用 cpu 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。

函数原型

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

参数

参数名称	描述	输入输出
context	AES-CBC-192 加密计算的结构体，包含加密密钥与偏移向量	输入
input_data	AES-CBC-192 待加密的明文数据	输入
input_len	AES-CBC-192 待加密明文数据的长度	输入
output_data	AES-CBC-192 加密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_cbc192_hard_decrypt

描述

AES-CBC-192 解密运算。输入输出数据都使用 cpu 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。

函数原型

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

参数

参数名称	描述	输入输出
context	AES-CBC-192 解密计算的结构体，包含解密密钥与偏移向量	输入
input_data	AES-CBC-192 待解密的密文数据	输入
input_len	AES-CBC-192 待解密密文数据的长度	输入
output_data	AES-CBC-192 解密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_cbc256_hard_encrypt

描述

AES-CBC-256 加密运算。输入输出数据都使用 cpu 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。

函数原型

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

参数

参数名称	描述	输入输出
context	AES-CBC-256 加密计算的结构体，包含加密密钥与偏移向量	输入
input_data	AES-CBC-256 待加密的明文数据	输入
input_len	AES-CBC-256 待加密明文数据的长度	输入
output_data	AES-CBC-256 加密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_cbc256_hard_decrypt

描述

AES-CBC-256 解密运算。输入输出数据都使用 cpu 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。

函数原型

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

参数

参数名称	描述	输入输出
context	AES-CBC-256 解密计算的结构体，包含解密密钥与偏移向量	输入
input_data	AES-CBC-256 待解密的密文数据	输入

参数名称	描述	输入输出
input_len	AES-CBC-256 待解密密文数据的长度	输入
output_data	AES-CBC-256 解密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_gcm128_hard_encrypt

描述

AES-GCM-128 加密运算。输入输出数据都使用 cpu 传输。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
context	AES-GCM-128 加密计算的结构体，包含加密密钥/偏移向量/aad/aad 长度	输入
input_data	AES-GCM-128 待加密的明文数据	输入
input_len	AES-GCM-128 待加密明文数据的长度	输入
output_data	AES-GCM-128 加密运算后的结果存放在这个 buffer	输出
gcm_tag	AES-GCM-128 加密运算后的 tag 存放在这个 buffer。 这个 buffer 大小需要保证为 16bytes	输出

返回值

无。

aes_gcm128_hard_decrypt

描述

AES-GCM-128 解密运算。输入输出数据都使用 cpu 传输。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
context	AES-GCM-128 解密计算的结构体，包含解密密钥/偏移向量/aad/aad 长度	输入
input_data	AES-GCM-128 待解密的密文数据	输入
input_len	AES-GCM-128 待解密密文数据的长度	输入
output_data	AES-GCM-128 解密运算后的结果存放在这个 buffer	输出
gcm_tag	AES-GCM-128 解密运算后的 tag 存放在这个 buffer。 这个 buffer 大小需要保证为 16bytes	输出

返回值

无。

aes_gcm192_hard_encrypt

描述

AES-GCM-192 加密运算。输入输出数据都使用 cpu 传输。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
context	AES-GCM-192 加密计算的结构体，包含加密密钥/偏移向量/aad/aad 长度	输入
input_data	AES-GCM-192 待加密的明文数据	输入
input_len	AES-GCM-192 待加密明文数据的长度	输入
output_data	AES-GCM-192 加密运算后的结果存放在这个 buffer	输出
gcm_tag	AES-GCM-192 加密运算后的 tag 存放在这个 buffer。 这个 buffer 大小需要保证为 16bytes	输出

返回值

无。

aes_gcm192_hard_decrypt

描述

AES-GCM-192 解密运算。输入输出数据都使用 cpu 传输。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
context	AES-GCM-192 解密计算的结构体，包含解密密钥/偏移向量/aad/aad 长度	输入
input_data	AES-GCM-192 待解密的密文数据	输入
input_len	AES-GCM-192 待解密密文数据的长度	输入
output_data	AES-GCM-192 解密运算后的结果存放在这个 buffer	输出
gcm_tag	AES-GCM-192 解密运算后的 tag 存放在这个 buffer。 这个 buffer 大小需要保证为 16bytes	输出

返回值

无。

aes_gcm256_hard_encrypt

描述

AES-GCM-256 加密运算。输入输出数据都使用 cpu 传输。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
context	AES-GCM-256 加密计算的结构体，包含加密密钥/偏移向量/aad/aad 长度	输入
input_data	AES-GCM-256 待加密的明文数据	输入
input_len	AES-GCM-256 待加密明文数据的长度	输入
output_data	AES-GCM-256 加密运算后的结果存放在这个 buffer	输出
gcm_tag	AES-GCM-256 加密运算后的 tag 存放在这个 buffer。 这个 buffer 大小需要保证为 16bytes	输出

返回值

无。

aes_gcm256_hard_decrypt

描述

AES-GCM-256 解密运算。输入输出数据都使用 cpu 传输。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
context	AES-GCM-256 解密计算的结构体，包含解密密钥/偏移向量/aad/aad 长度	输入
input_data	AES-GCM-256 待解密的密文数据	输入
input_len	AES-GCM-256 待解密密文数据的长度	输入
output_data	AES-GCM-256 解密运算后的结果存放在这个 buffer	输出
gcm_tag	AES-GCM-256 解密运算后的 tag 存放在这个 buffer。 这个 buffer 大小需要保证为 16bytes	输出

返回值

无。

aes_ecb128_hard_encrypt_dma

描述

AES-ECB-128 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入

参数名称	描述	输入输出
input_key	AES-ECB-128 加密的密钥	输入
input_data	AES-ECB-128 待加密的明文数据	输入
input_len	AES-ECB-128 待加密明文数据的长度	输入
output_data	AES-ECB-128 加密运算后的结果存放在这个 buffer。这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_ecb128_hard_decrypt_dma

描述

AES-ECB-128 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
input_key	AES-ECB-128 解密的密钥	输入
input_data	AES-ECB-128 待解密的密文数据	输入
input_len	AES-ECB-128 待解密密文数据的长度	输入
output_data	AES-ECB-128 解密运算后的结果存放在这个 buffer。这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_ecb192_hard_encrypt_dma

描述

AES-ECB-192 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
input_key	AES-ECB-192 加密的密钥	输入
input_data	AES-ECB-192 待加密的明文数据	输入
input_len	AES-ECB-192 待加密明文数据的长度	输入
output_data	AES-ECB-192 加密运算后的结果存放在这个 buffer。这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_ecb192_hard_decrypt_dma

描述

AES-ECB-192 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
input_key	AES-ECB-192 解密的密钥	输入
input_data	AES-ECB-192 待解密的密文数据	输入
input_len	AES-ECB-192 待解密密文数据的长度	输入

参数名称	描述	输入输出
output_data	AES-ECB-192 解密运算后的结果存放在这个 buffer。这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_ecb256_hard_encrypt_dma

描述

AES-ECB-256 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
input_key	AES-ECB-256 加密的密钥	输入
input_data	AES-ECB-256 待加密的明文数据	输入
input_len	AES-ECB-256 待加密明文数据的长度	输入
output_data	AES-ECB-256 加密运算后的结果存放在这个 buffer。这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_ecb256_hard_decrypt_dma

描述

AES-ECB-256 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。ECB 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。ECB 模式没有用到向量。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
input_key	AES-ECB-256 解密的密钥	输入
input_data	AES-ECB-256 待解密的密文数据	输入
input_len	AES-ECB-256 待解密密文数据的长度	输入
output_data	AES-ECB-256 解密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_cbc128_hard_encrypt_dma

描述

AES-CBC-128 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
context	AES-CBC-128 加密计算的结构体，包含加密密钥与偏移向量	输入
input_data	AES-CBC-128 待加密的明文数据	输入
input_len	AES-CBC-128 待加密明文数据的长度	输入
output_data	AES-CBC-128 加密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_cbc128_hard_decrypt_dma

描述

AES-CBC-128 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
context	AES-CBC-128 解密计算的结构体，包含解密密钥与偏移向量	输入
input_data	AES-CBC-128 待解密的密文数据	输入
input_len	AES-CBC-128 待解密密文数据的长度	输入
output_data	AES-CBC-128 解密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_cbc192_hard_encrypt_dma

描述

AES-CBC-192 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
context	AES-CBC-192 加密计算的结构体，包含加密密钥与偏移向量	输入
input_data	AES-CBC-192 待加密的明文数据	输入
input_len	AES-CBC-192 待加密明文数据的长度	输入
output_data	AES-CBC-192 加密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_cbc192_hard_decrypt_dma

描述

AES-CBC-192 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
context	AES-CBC-192 解密计算的结构体，包含解密密钥与偏移向量	输入
input_data	AES-CBC-192 待解密的密文数据	输入
input_len	AES-CBC-192 待解密密文数据的长度	输入
output_data	AES-CBC-192 解密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_cbc256_hard_encrypt_dma

描述

AES-CBC-256 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
context	AES-CBC-256 加密计算的结构体，包含加密密钥与偏移向量	输入
input_data	AES-CBC-256 待加密的明文数据	输入
input_len	AES-CBC-256 待加密明文数据的长度	输入
output_data	AES-CBC-256 加密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_cbc256_hard_decrypt_dma

描述

AES-CBC-256 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。CBC 加密将明文按照固定大小 16bytes 的块进行加密的，块大小不足则进行填充。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
context	AES-CBC-256 解密计算的结构体，包含解密密钥与偏移向量	输入
input_data	AES-CBC-256 待解密的密文数据	输入
input_len	AES-CBC-256 待解密密文数据的长度	输入

参数名称	描述	输入输出
output_data	AES-CBC-256 解密运算后的结果存放在这个 buffer。 这个 buffer 的大小需要保证 16bytes 对齐。	输出

返回值

无。

aes_gcm128_hard_encrypt_dma

描述

AES-GCM-128 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
context	AES-GCM-128 加密计算的结构体，包含加密密钥/偏移向量/aad/aad 长度	输入
input_data	AES-GCM-128 待加密的明文数据	输入
input_len	AES-GCM-128 待加密明文数据的长度。	输入
output_data	AES-GCM-128 加密运算后的结果存放在这个 buffer。。 由于 DMA 搬运数据的最小粒度为 4bytes， 所以需要保证这个 buffer 大小至少为 4bytes 的整数倍。	输出
gcm_tag	AES-GCM-128 加密运算后的 tag 存放在这个 buffer。 这个 buffer 大小需要保证为 16bytes	输出

返回值

无。

aes_gcm128_hard_decrypt_dma

描述

AES-GCM-128 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
context	AES-GCM-128 解密计算的结构体，包含解密密钥/偏移向量/aad/aad 长度	输入
input_data	AES-GCM-128 待解密的密文数据	输入
input_len	AES-GCM-128 待解密密文数据的长度。	输入
output_data	AES-GCM-128 解密运算后的结果存放在这个 buffer。 由于 DMA 搬运数据的最小粒度为 4bytes， 所以需要保证这个 buffer 大小至少为 4bytes 的整数倍。	输出
gcm_tag	AES-GCM-128 解密运算后的 tag 存放在这个 buffer。 这个 buffer 大小需要保证为 16bytes	输出

返回值

无。

aes_gcm192_hard_encrypt_dma

描述

AES-GCM-192 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
context	AES-GCM-192 加密计算的结构体，包含加密密钥/偏移向量	输入

参数名称	描述	输入输出
	/aad/aad 长度	
input_data	AES-GCM-192 待加密的明文数据	输入
input_len	AES-GCM-192 待加密明文数据的长度。	输入
output_data	AES-GCM-192 加密运算后的结果存放在这个 buffer。 由于 DMA 搬运数据的最小粒度为 4bytes， 所以需要保证这个 buffer 大小至少为 4bytes 的整数倍。	输出
gcm_tag	AES-GCM-192 加密运算后的 tag 存放在这个 buffer。 这个 buffer 大小需要保证为 16bytes	输出

返回值

无。

aes_gcm192_hard_decrypt_dma

描述

AES-GCM-192 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
context	AES-GCM-192 解密计算的结构体，包含解密密钥/偏移向量 /aad/aad 长度	输入
input_data	AES-GCM-192 待解密的密文数据	输入
input_len	AES-GCM-192 待解密密文数据的长度。	输入
output_data	AES-GCM-192 解密运算后的结果存放在这个 buffer。 由于 DMA 搬运数据的最小粒度为 4bytes， 所以需要保证这个 buffer 大小至少为 4bytes 的整数倍。	输出
gcm_tag	AES-GCM-192 解密运算后的 tag 存放在这个 buffer。 这个 buffer 大小需要保证为 16bytes	输出

返回值

无。

aes_gcm256_hard_encrypt_dma

描述

AES-GCM-256 加密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
context	AES-GCM-256 加密计算的结构体，包含加密密钥/偏移向量/aad/aad 长度	输入
input_data	AES-GCM-256 待加密的明文数据	输入
input_len	AES-GCM-256 待加密明文数据的长度。	输入
output_data	AES-GCM-256 加密运算后的结果存放在这个 buffer。 由于 DMA 搬运数据的最小粒度为 4bytes， 所以需要保证这个 buffer 大小至少为 4bytes 的整数倍。	输出
gcm_tag	AES-GCM-256 加密运算后的 tag 存放在这个 buffer。 这个 buffer 大小需要保证为 16bytes	输出

返回值

无。

aes_gcm256_hard_decrypt_dma

描述

AES-GCM-256 解密运算。输入数据使用 cpu 传输，输出数据都使用 dma 传输。

函数原型

```
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)
```

参数

参数名称	描述	输入输出
dma_receive_channel_num	AES 输出数据的 DMA 通道号	输入
context	AES-GCM-256 解密计算的结构体，包含解密密钥/偏移向量/aad/aad 长度	输入
input_data	AES-GCM-256 待解密的密文数据	输入
input_len	AES-GCM-256 待解密密文数据的长度。	输入
output_data	AES-GCM-256 解密运算后的结果存放在这个 buffer。 由于 DMA 搬运数据的最小粒度为 4bytes， 所以需要保证这个 buffer 大小至少为 4bytes 的整数倍。	输出
gcm_tag	AES-GCM-256 解密运算后的 tag 存放在这个 buffer。 这个 buffer 大小需要保证为 16bytes	输出

返回值

无。

aes_init

描述

AES 硬件模块的初始化

函数原型

```
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)
```

参数

参数名称	描述	输入输出
input_key	待加密/解密的密钥	输入
input_key_len	待加密/解密密钥的长度	输入
iv	AES 加密解密用到的 iv 数据	输入
iv_len	AES 加密解密用到的 iv 数据的长度，CBC 固定为 16bytes，GCM 固定为 12bytes	输出
gcm_aad	AES-GCM 加密解密用到的 aad 数据	输出
cipher_mode	AES 硬件模块执行的加密解密类型，支持 AES_CBC/AES_ECB/AES_GCM	输入
encrypt_sel	AES 硬件模块执行的模式：加密或解密	输入

参数名称	描述	输入输出
gcm_aad_len	AES-GCM 加密解密用到的 aad 数据的长度	输入
input_data_len	待加密/解密的数据长度	输入

返回值

无。

aes_process

描述

AES 硬件模块执行加密解密操作

函数原型

```
void aes_process(uint8_t *input_data, uint8_t *output_data, size_t
input_data_len, aes_cipher_mode_t cipher_mode)
```

参数

参数名称	描述	输入输出
input_data	这个 buffer 存放待加密/解密的数据	输入
output_data	这个 buffer 存放加密/解密的输出结果	输出
input_data_len	待加密/解密的数据的长度	输入
cipher_mode	AES 硬件模块执行的加密解密类型，支持 AES_CBC/AES_ECB/AES_GCM	输入

返回值

无。

gcm_get_tag

描述

获取 AES-GCM 计算结束后的 tag

函数原型

```
void gcm_get_tag(uint8_t *gcm_tag)
```

参数

参数名称	描述	输入输出
gcm_tag	这个 buffer 存放 AES-GCM 加密/解密后的 tag，固定为 16bytes 的大小	输出

返回值

无。

举例

```
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);
```

数据类型

相关数据类型、数据结构定义如下：

- `aes_cipher_mode_t`：AES 加密/解密的方式。

aes_cipher_mode_t

描述

AES 加密/解密的方式。

定义

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

- `gcm_context_t`：AES-GCM 加密/解密时参数用到的结构体

gcm_context_t

描述

AES-GCM 参数用到的结构体，包括密钥、偏移向量、aad 数据、aad 数据长度。

定义

```
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`：AES-CBC 加密/解密时参数用到的结构体

cbc_context_t

描述

AES-CBC 参数用到的结构体，包括密钥、偏移向量。

定义

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

成员

成员名称	描述
AES_ECB	ECB 加密/解密
AES_CBC	CBC 加密/解密
AES_GCM	GCM 加密/解密