4-6 学时实践要求（30 分）

1. 参考相关内容，在 Ubuntu或openEuler中（推荐 openEuler）中使用OpenSSL库编程实现调用SM2（加密解密，签名验签），SM3（摘要计算，HMAC 计算），SM4（加密解密）算法，使用Markdown记录详细记录实践过程，每完成一项git commit 一次。（5'）

- SM3 hash实现
  - 过程与验证

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# nano sm3_hash.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# gcc sm3_hash.c
-o sm3_hash -lssl -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# ./sm3_hash
"20221414xlm"
SM3 hash:
5fa790c6e893e4e724edea87d4c58c3426ba5487c5314a15d9d8c990364920e5
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# git commit -m
"shiyan1-2/Task01:finish SM3 MAC"
[master d854781] shiyan1-2/Task01:finish SM3 MAC
2 files changed, 51 insertions(+)
create mode 100755 shiyan1-2/task01/test_sm3/sm3_hash
create mode 100644 shiyan1-2/task01/test_sm3/sm3_hash.c
```

  - 具体代码

```c
#include <stdio.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <string.h>

void sm3_hash(const char* data) {
    EVP_MD_CTX* mdctx;
    const EVP_MD* md;
    unsigned char md_value[EVP_MAX_MD_SIZE];
    unsigned int md_len;

    // 使用SM3哈希算法
    OpenSSL_add_all_digests();
    md = EVP_get_digestbyname("sm3");
    if (md == NULL) {
        printf("Unknown message digest sm3\n");
        return;
    }

    // 创建并初始化散列上下文
    mdctx = EVP_MD_CTX_new();
    EVP_DigestInit_ex(mdctx, md, NULL);
```

```
    // 传递数据给散列函数
    EVP_DigestUpdate(mdctx, data, strlen(data));

    // 获取散列值
    EVP_DigestFinal_ex(mdctx, md_value, &md_len);

    // 打印散列值
    printf("SM3 hash: ");
    for (int i = 0; i < md_len; i++) {
        printf("%02x", md_value[i]);
    }
    printf("\n");

    // 清理
    EVP_MD_CTX_free(mdctx);
    EVP_cleanup();
}

int main(int argc, char* argv[]) {
    if (argc < 2) {
        printf("Usage: %s <string_to_hash>\n", argv[0]);
        return 1;
    }

    // 将命令行参数传递给sm3_hash函数
    sm3_hash(argv[1]);
    return 0;
}
```

- SM3 HAMC实现
  - 过程与验证

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# ls
sm3_hash  sm3_hash.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# nano sm3_hmac
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# gcc -o sm3_hmac
sm3_hmac.c -lcrypto
cc1: fatal error: sm3_hmac.c: No such file or directory
compilation terminated.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# mv sm3_hmac
sm3_hmac.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# gcc -o sm3_hmac
sm3_hmac.c -lcrypto
sm3_hmac.c: In function ‘main’:
sm3_hmac.c:21:5: warning: ‘HMAC_CTX_new’ is deprecated: Since OpenSSL
3.0 [-Wdeprecated-declarations]
21 |     ctx = HMAC_CTX_new();
    |     ^~~
In file included from sm3_hmac.c:3:
```

```
/usr/include/openssl/hmac.h:33:33: note: declared here
33 | OSSL_DEPRECATEDIN_3_0 HMAC_CTX *HMAC_CTX_new(void);
    |                                 ^~~~~~~~~~~~
sm3_hmac.c:27:5: warning: ‘HMAC_Init_ex’ is deprecated: Since OpenSSL
3.0 [-Wdeprecated-declarations]
27 |     if (HMAC_Init_ex(ctx, key, key_len, md, NULL) != 1) {
    |     ^~
In file included from sm3_hmac.c:3:
/usr/include/openssl/hmac.h:43:27: note: declared here
43 | OSSL_DEPRECATEDIN_3_0 int HMAC_Init_ex(HMAC_CTX *ctx, const void
*key, int len,
    |                           ^~~~~~~~~~~~
sm3_hmac.c:33:5: warning: ‘HMAC_Update’ is deprecated: Since OpenSSL
3.0 [-Wdeprecated-declarations]
33 |     if (HMAC_Update(ctx, data, data_len) != 1) {
    |     ^~
In file included from sm3_hmac.c:3:
/usr/include/openssl/hmac.h:45:27: note: declared here
45 | OSSL_DEPRECATEDIN_3_0 int HMAC_Update(HMAC_CTX *ctx, const
unsigned char *data,
    |                           ^~~~~~~~~~~
sm3_hmac.c:41:5: warning: ‘HMAC_Final’ is deprecated: Since OpenSSL 3.0
[-Wdeprecated-declarations]
41 |     if (HMAC_Final(ctx, hmac_value, &hmac_len) != 1) {
    |     ^~
In file included from sm3_hmac.c:3:
/usr/include/openssl/hmac.h:47:27: note: declared here
47 | OSSL_DEPRECATEDIN_3_0 int HMAC_Final(HMAC_CTX *ctx, unsigned char
*md,
    |                           ^~~~~~~~~~
sm3_hmac.c:47:5: warning: ‘HMAC_CTX_free’ is deprecated: Since OpenSSL
3.0 [-Wdeprecated-declarations]
47 |     HMAC_CTX_free(ctx);
    |     ^~~~~~~~~~~~~
In file included from sm3_hmac.c:3:
/usr/include/openssl/hmac.h:35:28: note: declared here
35 | OSSL_DEPRECATEDIN_3_0 void HMAC_CTX_free(HMAC_CTX *ctx);
    |                            ^~~~~~~~~~~~~
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# nano sm3_hmac
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# nano sm3_hmac.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# gcc -o sm3_hmac
sm3_hmac.c -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3#
./sm3_hmac_updated "mysecretkey" "my message"
-bash: ./sm3_hmac_updated: No such file or directory
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# ./sm3_hmac
"mysecretkey" "my message"
HMAC: c1d7bc47622831a4b0882391ebb444550e16b517800a03b3af2427fb5b001567
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# ./sm3_hmac
"newsecretkey" "my message"
HMAC: 7d9285231fe7a97fc7d03c1f72f31ec5e984ae2572bc1c4984818232c8a404ff
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm3# git commit -m
"finish sm3 HMAC"
```

```
[master b8cfa84] finish sm3 HMAC
2 files changed, 64 insertions(+)
create mode 100755 shiyan1-2/task01/test_sm3/sm3_hmac
create mode 100644 shiyan1-2/task01/test_sm3/sm3_hmac.c
```

- 具体代码

```
#include <stdio.h>
#include <openssl/evp.h>
#include <string.h>

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <key> <data>\n", argv[0]);
        return 1;
    }

    unsigned char *key = (unsigned char *)argv[1];
    unsigned char *data = (unsigned char *)argv[2];
    size_t key_len = strlen((char *)key);
    size_t data_len = strlen((char *)data);

    // 创建和初始化EVP_MAC_CTX和EVP_MAC变量
    EVP_MAC_CTX *ctx;
    EVP_MAC *mac;
    OSSL_PARAM params[2];
    unsigned char *out;
    size_t out_len;

    // 设置算法类型
    mac = EVP_MAC_fetch(NULL, "HMAC", NULL);
    ctx = EVP_MAC_CTX_new(mac);

    // 设置参数：摘要算法
    params[0] = OSSL_PARAM_construct_utf8_string("digest", "SM3", 0);
    params[1] = OSSL_PARAM_construct_end();

    if (!EVP_MAC_init(ctx, key, key_len, params)) {
        fprintf(stderr, "Failed to initialize HMAC\n");
        return 1;
    }

    if (!EVP_MAC_update(ctx, data, data_len)) {
        fprintf(stderr, "Failed to update HMAC\n");
        return 1;
    }

    // 计算HMAC长度
    EVP_MAC_final(ctx, NULL, &out_len, 0);
    out = malloc(out_len);

    if (!EVP_MAC_final(ctx, out, &out_len, out_len)) {
```

```
        fprintf(stderr, "Failed to finalize HMAC\n");
        return 1;
    }

    // 打印结果
    printf("HMAC: ");
    for (size_t i = 0; i < out_len; i++) {
        printf("%02x", out[i]);
    }
    printf("\n");

    // 清理资源
    EVP_MAC_CTX_free(ctx);
    EVP_MAC_free(mac);
    free(out);

    return 0;
}
```

- SM4 加解密
  - 过程与验证(用命令生成密钥和IV文件)

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01# mkdir test_sm4
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01# cd test_sm4
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# nano
sm4_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# nano
sm4_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# ls
sm4_decrypt.c  sm4_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# gcc
sm4_encrypt.c -o sm4_encrypt -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# gcc
sm4_decrypt.c -o sm4_decrypt -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# echo
"20221414xlm" > plain.txt
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# touch
encrypted_file
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# rm
encrypted_file
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# touch
encrypted_file.txt
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# touch
decrypted_file.txt
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# ls
decrypted_file.txt  encrypted_file.txt  plain.txt  sm4_decrypt
sm4_decrypt.c  sm4_encrypt  sm4_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# ./sm4_encrypt
"0123456789abcdef" "1234567890abcdef" plain.txt en
crypted_file.txt
```

```
Encryption complete.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# ./sm4_decrypt
"0123456789abcdef" "1234567890abcdef" encrypted_fi
le.txt decrypted_file.txt
Decryption complete.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# cat
encrypted_file.txt
AQ��z�b�[6̰root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4#
cat decrypted_file.txt
20221414xlm
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# git commit -m
"finish SM4 encrypt & decrypt"
[master 7e4df4d] finish SM4 encrypt & decrypt
7 files changed, 145 insertions(+)
create mode 100644 shiyan1-2/task01/test_sm4/decrypted_file.txt
create mode 100644 shiyan1-2/task01/test_sm4/encrypted_file.txt
create mode 100644 shiyan1-2/task01/test_sm4/plain.txt
create mode 100755 shiyan1-2/task01/test_sm4/sm4_decrypt
create mode 100644 shiyan1-2/task01/test_sm4/sm4_decrypt.c
create mode 100755 shiyan1-2/task01/test_sm4/sm4_encrypt
create mode 100644 shiyan1-2/task01/test_sm4/sm4_encrypt.c
```

- 具体代码
  - SM4 加密程序

```c
#include <openssl/evp.h>
#include <stdio.h>
#include <stdlib.h>

void handleErrors(void) {
    fprintf(stderr, "An error occurred.\n");
    exit(1);
}

int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "Usage: %s <key> <iv> <input file> <output
file>\n", argv[0]);
        return 1;
    }

    char *key = argv[1];
    char *iv = argv[2];
    char *input_file = argv[3];
    char *output_file = argv[4];

    FILE *f_input, *f_output;
    unsigned char buffer[1024];
    unsigned char ciphertext[1024 + EVP_MAX_BLOCK_LENGTH];
    int bytes_read, ciphertext_len;
```

```
    EVP_CIPHER_CTX *ctx;

    // Initialise the library
    if(!OPENSSL_init_crypto(OPENSSL_INIT_LOAD_CRYPTO_STRINGS,
NULL))
        handleErrors();

    // Create and initialise the context
    if(!(ctx = EVP_CIPHER_CTX_new()))
        handleErrors();

    // Initialise the encryption operation.
    if(1 != EVP_EncryptInit_ex(ctx, EVP_sm4_cbc(), NULL, (unsigned
char*)key, (unsigned char*)iv))
        handleErrors();

    // Open files
    if(!(f_input = fopen(input_file, "rb"))) {
        fprintf(stderr, "Could not open %s for reading\n",
input_file);
        return 1;
    }

    if(!(f_output = fopen(output_file, "wb"))) {
        fprintf(stderr, "Could not open %s for writing\n",
output_file);
        return 1;
    }

    // Provide the message to be encrypted
    while((bytes_read = fread(buffer, 1, 1024, f_input)) > 0) {
        if(1 != EVP_EncryptUpdate(ctx, ciphertext,
&ciphertext_len, buffer, bytes_read))
            handleErrors();
        fwrite(ciphertext, 1, ciphertext_len, f_output);
    }

    // Finalise the encryption
    if(1 != EVP_EncryptFinal_ex(ctx, ciphertext + ciphertext_len,
&ciphertext_len))
        handleErrors();
    fwrite(ciphertext, 1, ciphertext_len, f_output);

    // Clean up
    EVP_CIPHER_CTX_free(ctx);
    fclose(f_input);
    fclose(f_output);

    printf("Encryption complete.\n");

    return 0;
}
```

- SM4 解密程序

```
#include <openssl/evp.h>
#include <stdio.h>
#include <stdlib.h>

void handleErrors(void) {
    fprintf(stderr, "An error occurred.\n");
    exit(1);
}

int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "Usage: %s <key> <iv> <input file> <output
file>\n", argv[0]);
        return 1;
    }

    char *key = argv[1];
    char *iv = argv[2];
    char *input_file = argv[3];
    char *output_file = argv[4];

    FILE *f_input, *f_output;
    unsigned char buffer[1024];
    unsigned char plaintext[1024 + EVP_MAX_BLOCK_LENGTH];
    int bytes_read, plaintext_len;

    EVP_CIPHER_CTX *ctx;

    // Initialise the library
    if(!OPENSSL_init_crypto(OPENSSL_INIT_LOAD_CRYPTO_STRINGS,
NULL))
        handleErrors();

    // Create and initialise the context
    if(!(ctx = EVP_CIPHER_CTX_new()))
        handleErrors();

    // Initialise the decryption operation.
    if(1 != EVP_DecryptInit_ex(ctx, EVP_sm4_cbc(), NULL, (unsigned
char*)key, (unsigned char*)iv))
        handleErrors();

    // Open files
    if(!(f_input = fopen(input_file, "rb"))) {
        fprintf(stderr, "Could not open %s for reading\n",
input_file);
        return 1;
    }
```

```
    if(!(f_output = fopen(output_file, "wb"))) {
        fprintf(stderr, "Could not open %s for writing\n",
output_file);
        return 1;
    }

    // Provide the message to be decrypted
    while((bytes_read = fread(buffer, 1, 1024, f_input)) > 0) {
        if(1 != EVP_DecryptUpdate(ctx, plaintext, &plaintext_len,
buffer, bytes_read))
            handleErrors();
        fwrite(plaintext, 1, plaintext_len, f_output);
    }

    // Finalise the decryption
    if(1 != EVP_DecryptFinal_ex(ctx, plaintext + plaintext_len,
&plaintext_len))
        handleErrors();
    fwrite(plaintext, 1, plaintext_len, f_output);

    // Clean up
    EVP_CIPHER_CTX_free(ctx);
    fclose(f_input);
    fclose(f_output);

    printf("Decryption complete.\n");

    return 0;
}
```

- SM2 加解密、签名与验证
  - SM2加解密
    - 过程和验证

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ls
decrypted_file.txt  plain.txt    sm2_decrypt.c  sm2_encrypt.c
sm2_public.pem
encrypted_file.txt  sm2_decrypt  sm2_encrypt    sm2_private.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm
sm2_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc
sm2_encrypt.c -o sm2_encrypt -lcrypto
sm2_encrypt.c: In function ‘main’:
sm2_encrypt.c:82:9: warning: implicit declaration of function
‘EVP_PKEY_encrypt_final’; did you mean ‘EVP_PKEY_encrypt_init’? [-
Wimplicit-function-declaration]
82 |     if (EVP_PKEY_encrypt_final(ctx, ciphertext,
```

```
&ciphertext_len) <= 0)
    |         ^~~~~~~~~~~~~~~~~~~~~
    |         EVP_PKEY_encrypt_init
/usr/bin/ld: /tmp/ccq6e5QM.o: in function `main':
sm2_encrypt.c:(.text+0x3fa): undefined reference to
`EVP_PKEY_encrypt_final'
collect2: error: ld returned 1 exit status
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm
sm2_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc
sm2_encrypt.c -o sm2_encrypt -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_encrypt sm2_public.pem plain.txt encrypted_file.txt
Encryption complete.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm
sm2_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc
sm2_decrypt.c -o sm2_decrypt -lcrypto
sm2_decrypt.c: In function ‘main’:
sm2_decrypt.c:65:37: warning: passing argument 3 of
‘EVP_PKEY_decrypt’ from incompatible pointer type [-Wincompatible-
pointer-types]
65 |     if (EVP_PKEY_decrypt(ctx, NULL, &plaintext_len, buffer,
sizeof(buffer)) <= 0)
    |                                     ^~~~~~~~~~~~~~
    |                                     |
    |                                     int *
In file included from sm2_decrypt.c:1:
/usr/include/openssl/evp.h:1914:50: note: expected ‘size_t *’ {aka
‘long unsigned int *’} but argument is of type ‘int ’
1914 |                     unsigned char *out, size_t *outlen,
    |                                         ~~~~~~~~^~~~~
sm2_decrypt.c:75:46: warning: passing argument 3 of
‘EVP_PKEY_decrypt’ from incompatible pointer type [-Wincompatible-
pointer-types]
75 |         if (EVP_PKEY_decrypt(ctx, plaintext, &plaintext_len,
buffer, bytes_read) <= 0)
    |                                              ^~~~~~~~~~~~~~
    |                                              |
    |                                              int *
In file included from sm2_decrypt.c:1:
/usr/include/openssl/evp.h:1914:50: note: expected ‘size_t *’ {aka
‘long unsigned int *’} but argument is of type ‘int ’
1914 |                     unsigned char *out, size_t *outlen,
    |                                         ~~~~~~~~^~~~~
sm2_decrypt.c:81:9: warning: implicit declaration of function
‘EVP_PKEY_decrypt_final’; did you mean ‘EVP_PKEY_decrypt_init’? [-
Wimplicit-function-declaration]
81 |     if (EVP_PKEY_decrypt_final(ctx, plaintext,
&plaintext_len) <= 0)
```

```
    |         ^~~~~~~~~~~~~~~~~~~~~
    |         EVP_PKEY_decrypt_init
/usr/bin/ld: /tmp/ccxLtmHg.o: in function `main':
sm2_decrypt.c:(.text+0x3fd): undefined reference to
`EVP_PKEY_decrypt_final'
collect2: error: ld returned 1 exit status
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm
sm2_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc
sm2_decrypt.c -o sm2_decrypt -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ls
decrypted_file.txt  plain.txt    sm2_decrypt.c  sm2_encrypt.c
sm2_public.pem
encrypted_file.txt  sm2_decrypt  sm2_encrypt    sm2_private.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_decrypt privkey.pem encrypted_file.txt decrypted_file.txt
Unable to open private key file privkey.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_decrypt sm2_private.pem encrypted_file.txt decrypted_file.
txt
An error occurred.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm
sm2_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc
sm2_decrypt.c -o sm2_decrypt -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_decrypt sm2_private.pem encrypted_file.txt
decrypted_file.txt
OpenSSL error: error:0680007B:asn1 encoding routines::header too
long
OpenSSL error: error:06800066:asn1 encoding routines::bad object
header
OpenSSL error: error:0688010A:asn1 encoding routines::nested asn1
error
OpenSSL error: error:1A800068:SM2 routines::reason(104)
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ls
decrypted_file.txt  plain.txt    sm2_decrypt.c  sm2_encrypt.c
sm2_public.pem
encrypted_file.txt  sm2_decrypt  sm2_encrypt    sm2_private.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# mv
decrypted_file.txt decrypted_file.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# mv
encrypted_file.txt encrypted_file.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_encrypt sm2_public.pem plain.txt encrypted_file.bin
Encryption complete.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_decrypt sm2_private.pem encrypted_file.bin decrypted_file.
bin
OpenSSL error: error:0680007B:asn1 encoding routines::header too
```

```
long
OpenSSL error: error:06800066:asn1 encoding routines::bad object
header
OpenSSL error: error:0688010A:asn1 encoding routines::nested asn1
error
OpenSSL error: error:1A800068:SM2 routines::reason(104)
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm
encrypted_file.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm
decrypted_file.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# touch
encrypted_file.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# touch
decrypted_file.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_encrypt sm2_public.pem plain.txt encrypted_file.bin
Encryption complete.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_decrypt sm2_private.pem encrypted_file.bin
decrypted_file.bin
OpenSSL error: error:068000A8:asn1 encoding routines::wrong tag
OpenSSL error: error:0688010A:asn1 encoding routines::nested asn1
error
OpenSSL error: error:1A800068:SM2 routines::reason(104)
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# openssl ec
-in sm2_private.pem -text -noout
read EC key
Private-Key: (256 bit)
priv:
    37:54:56:d4:3d:a4:00:7f:dc:15:3d:0c:fc:e5:88:
    db:dc:d8:49:1f:56:76:19:d2:ca:16:3a:06:78:98:
    78:95
pub:
    04:c6:19:31:a2:7c:eb:5a:62:e6:9f:0c:f8:58:47:
    72:eb:9c:c7:0e:a0:0e:a4:f4:43:19:16:ad:77:46:
    37:8a:d2:43:01:a6:6a:0b:40:01:70:d1:4e:42:55:
    54:d8:19:a9:b3:e2:23:55:26:7a:2a:d3:98:7d:0f:
    94:85:c4:84:ec
ASN1 OID: SM2
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# openssl ec
-in sm2_public.pem -pubin -text -noout
read EC key
Public-Key: (256 bit)
pub:
    04:c6:19:31:a2:7c:eb:5a:62:e6:9f:0c:f8:58:47:
    72:eb:9c:c7:0e:a0:0e:a4:f4:43:19:16:ad:77:46:
    37:8a:d2:43:01:a6:6a:0b:40:01:70:d1:4e:42:55:
    54:d8:19:a9:b3:e2:23:55:26:7a:2a:d3:98:7d:0f:
    94:85:c4:84:ec
ASN1 OID: SM2
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# openssl
pkeyutl -decrypt -in encrypted_file.bin -out decrypted_file.bin -
inkey sm2_private.pem -pkeyopt ec_scheme:sm2
pkeyutl: Can't set parameter "ec_scheme:sm2":
```

```
803BC4A8AD7F0000:error:03000093:digital envelope
routines:default_fixup_args:command not
supported:../crypto/evp/ctrl_params_translate.c:580:[action:2,
state:4] name=ec_scheme, value=sm2
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# openssl
sm2 -decrypt -in encrypted_file.bin -out decrypted_file.bin -inkey
sm2_private.pem
Invalid command 'sm2'; type "help" for a list.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# openssl
pkeyutl -decrypt -in encrypted_file.bin -out decrypted_file.bin -
inkey sm2_private.pem -pkeyopt ec_scheme:sm2
pkeyutl: Can't set parameter "ec_scheme:sm2":
80FB331A8E7F0000:error:03000093:digital envelope
routines:default_fixup_args:command not
supported:../crypto/evp/ctrl_params_translate.c:580:[action:2,
state:4] name=ec_scheme, value=sm2
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# openssl
version
OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022)
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# openssl
pkeyutl -decrypt -in encrypted_file.bin -out decrypted_file.bin -
inkey sm2_private.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ls
decrypted_file.bin  plain.txt    sm2_decrypt.c  sm2_encrypt.c
sm2_public.pem
encrypted_file.bin  sm2_decrypt  sm2_encrypt    sm2_private.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# cat
decrypted_file.bin
20221414xlm
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_decrypt sm2_private.pem encrypted_file.bin
decrypted_file.bin
OpenSSL error: error:068000A8:asn1 encoding routines::wrong tag
OpenSSL error: error:0688010A:asn1 encoding routines::nested asn1
error
OpenSSL error: error:1A800068:SM2 routines::reason(104)
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc
sm2_decrypt.c -o sm2_decrypt -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_decrypt sm2_private.pem encrypted_file.bin
decrypted_file.bin
OpenSSL error: error:068000A8:asn1 encoding routines::wrong tag
OpenSSL error: error:0688010A:asn1 encoding routines::nested asn1
error
OpenSSL error: error:1A800068:SM2 routines::reason(104)
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ls
decrypted_file.bin  plain.txt    sm2_decrypt.c  sm2_encrypt.c
sm2_public.pem
encrypted_file.bin  sm2_decrypt  sm2_encrypt    sm2_private.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm
sm2_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
```

```
sm2_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc -o
sm2_decrypt sm2_decrypt.c -lcrypto -lssl
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_decrypt private_key.pem encrypted_file.bin
decrypted_file.bin
Error opening private key file: No such file or directory
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_decrypt sm2_private.pem encrypted_file.bin decrypted_file.
bin
Decryption successful.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# cat
decrypted_file.bin
20221414xlm
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# git commit
-m "finish sm2 encrypt & decrypt"
[master aae0ff0] finish sm2 encrypt & decrypt
9 files changed, 199 insertions(+)
create mode 100644 shiyan1-2/task01/test_sm2/decrypted_file.bin
create mode 100644 shiyan1-2/task01/test_sm2/encrypted_file.bin
create mode 100644 shiyan1-2/task01/test_sm2/plain.txt
create mode 100755 shiyan1-2/task01/test_sm2/sm2_decrypt
create mode 100644 shiyan1-2/task01/test_sm2/sm2_decrypt.c
create mode 100755 shiyan1-2/task01/test_sm2/sm2_encrypt
create mode 100644 shiyan1-2/task01/test_sm2/sm2_encrypt.c
create mode 100644 shiyan1-2/task01/test_sm2/sm2_private.pem
create mode 100644 shiyan1-2/task01/test_sm2/sm2_public.pem
```

- 代码
  - 加密代码

```c
#include <openssl/evp.h>
#include <openssl/pem.h>
#include <stdio.h>
#include <stdlib.h>

void handleErrors(void) {
    fprintf(stderr, "An error occurred.\n");
    exit(1);
}

int main(int argc, char *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <pubkey.pem> <input file>
<output file>\n", argv[0]);
        return 1;
    }

    char *pubkey_filename = argv[1];
    char *input_file = argv[2];
```

```
    char *output_file = argv[3];

    FILE *f_input, *f_output, *f_pubkey;
    unsigned char buffer[1024];
    unsigned char *ciphertext;
    size_t ciphertext_len;
    size_t bytes_read;

    EVP_PKEY *pubkey = NULL;
    EVP_PKEY_CTX *ctx = NULL;

    // Load public key
    if (!(f_pubkey = fopen(pubkey_filename, "r"))) {
        fprintf(stderr, "Unable to open public key file
%s\n", pubkey_filename);
        return 1;
    }

    if (!(pubkey = PEM_read_PUBKEY(f_pubkey, NULL, NULL,
NULL))) {
        fprintf(stderr, "Error loading public key\n");
        fclose(f_pubkey);
        return 1;
    }
    fclose(f_pubkey);

    // Initialise the library
    if
(!OPENSSL_init_crypto(OPENSSL_INIT_LOAD_CRYPTO_STRINGS,
NULL))
        handleErrors();

    // Create and initialise the context
    if (!(ctx = EVP_PKEY_CTX_new(pubkey, NULL)))
        handleErrors();

    if (EVP_PKEY_encrypt_init(ctx) <= 0)
        handleErrors();

    // Open files
    if (!(f_input = fopen(input_file, "rb"))) {
        fprintf(stderr, "Could not open %s for reading\n",
input_file);
        return 1;
    }

    if (!(f_output = fopen(output_file, "wb"))) {
        fprintf(stderr, "Could not open %s for writing\n",
output_file);
        return 1;
    }

    // Determine buffer size for output
    if (EVP_PKEY_encrypt(ctx, NULL, &ciphertext_len, buffer,
```

```
sizeof(buffer)) <= 0)
        handleErrors();
    ciphertext = malloc(ciphertext_len);
    if (!ciphertext) {
        fprintf(stderr, "Memory allocation failed\n");
        return 1;
    }

    // Encrypt the data
    while ((bytes_read = fread(buffer, 1, sizeof(buffer),
f_input)) > 0) {
        if (EVP_PKEY_encrypt(ctx, ciphertext,
&ciphertext_len, buffer, bytes_read) <= 0)
            handleErrors();
        fwrite(ciphertext, 1, ciphertext_len, f_output);
    }

    // Clean up
    EVP_PKEY_free(pubkey);
    EVP_PKEY_CTX_free(ctx);
    fclose(f_input);
    fclose(f_output);
    free(ciphertext);

    printf("Encryption complete.\n");

    return 0;
}
```

- 解密代码

```
#include <openssl/ec.h>
#include <openssl/objects.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <openssl/pem.h>
#include <stdio.h>
#include <stdlib.h>

void handle_errors() {
    ERR_print_errors_fp(stderr);
    abort();
}

int main(int argc, char **argv) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <private_key.pem>
<encrypted_file> <decrypted_file>\n", argv[0]);
        return EXIT_FAILURE;
    }

    const char *private_key_file = argv[1];
```

```
    const char *encrypted_file = argv[2];
    const char *decrypted_file = argv[3];

    // Load private key
    EVP_PKEY *private_key = NULL;
    FILE *fp;
    fp = fopen(private_key_file, "r");
    if (!fp) {
        perror("Error opening private key file");
        return EXIT_FAILURE;
    }
    private_key = PEM_read_PrivateKey(fp, NULL, NULL, NULL);
    fclose(fp);
    if (!private_key) {
        handle_errors();
        return EXIT_FAILURE;
    }

    // Setup decryption
    EVP_PKEY_CTX *ctx = EVP_PKEY_CTX_new(private_key, NULL);
    if (!ctx) {
        handle_errors();
        return EXIT_FAILURE;
    }
    if (EVP_PKEY_decrypt_init(ctx) <= 0) {
        handle_errors();
        return EXIT_FAILURE;
    }

    // Read encrypted data
    FILE *fin = fopen(encrypted_file, "rb");
    if (!fin) {
        perror("Error opening encrypted file");
        return EXIT_FAILURE;
    }
    // Determine the size of the encrypted data
    fseek(fin, 0, SEEK_END);
    long encrypted_data_size = ftell(fin);
    fseek(fin, 0, SEEK_SET);
    unsigned char *encrypted_data =
malloc(encrypted_data_size);
    fread(encrypted_data, 1, encrypted_data_size, fin);
    fclose(fin);

    // Decrypt data
    unsigned char *decrypted_data = NULL;
    size_t decrypted_data_size = 0;
    if (EVP_PKEY_decrypt(ctx, NULL, &decrypted_data_size,
encrypted_data, encrypted_data_size) <= 0) {
        handle_errors();
        return EXIT_FAILURE;
    }
    decrypted_data = malloc(decrypted_data_size);
    if (EVP_PKEY_decrypt(ctx, decrypted_data,
```

```
    &decrypted_data_size, encrypted_data, encrypted_data_size) <=
    0) {
            handle_errors();
            return EXIT_FAILURE;
        }

        // Write decrypted data to file
        FILE *fout = fopen(decrypted_file, "wb");
        if (!fout) {
            perror("Error opening decrypted file");
            return EXIT_FAILURE;
        }
        fwrite(decrypted_data, 1, decrypted_data_size, fout);
        fclose(fout);

        // Clean up
        EVP_PKEY_CTX_free(ctx);
        EVP_PKEY_free(private_key);
        free(encrypted_data);
        free(decrypted_data);

        printf("Decryption successful.\n");
        return EXIT_SUCCESS;
    }
```

- SM2签名和验证
  - 过程和验证

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_sign.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc -o
sm2_sign sm2_sign.c -lcrypto -lssl
sm2_sign.c: In function ‘main’:
sm2_sign.c:29:22: warning: implicit declaration of function
‘PEM_read_PrivateKey’; did you mean ‘i2d_PrivateKey’? [-Wimplicit-
function-declaration]
29 |     EVP_PKEY *pkey = PEM_read_PrivateKey(fp, NULL, NULL,
NULL);
    |                      ^~~~~~~~~~~~~~~~~~
    |                      i2d_PrivateKey
sm2_sign.c:29:22: warning: initialization of ‘EVP_PKEY *’ {aka
‘struct evp_pkey_st *’} from ‘int’ makes pointer from integer
without a cast [-Wint-conversion]
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm
sm2_sign.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_sign.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc -o
sm2_sign sm2_sign.c -lcrypto -lssl
sm2_sign.c: In function ‘main’:
sm2_sign.c:30:22: warning: implicit declaration of function
```

```
‘PEM_read_Private_key’; did you mean ‘PEM_read_PrivateKey’? [-
Wimplicit-function-declaration]
30 |     EVP_PKEY *pkey = PEM_read_Private_key(fp, NULL, NULL,
NULL);
   |                       ^~~~~~~~~~~~~~~~~~~
   |                       PEM_read_PrivateKey
sm2_sign.c:30:22: warning: initialization of ‘EVP_PKEY *’ {aka
‘struct evp_pkey_st *’} from ‘int’ makes pointer from integer
without a cast [-Wint-conversion]
/usr/bin/ld: /tmp/ccIzsK7G.o: in function `main':
sm2_sign.c:(.text+0xed): undefined reference to
`PEM_read_Private_key'
collect2: error: ld returned 1 exit status
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm
sm2_sign.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_sign.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc -o
sm2_sign sm2_sign.c -lcrypto -lssl
sm2_sign.c:5:10: fatal error: openssl/sm2.h: No such file or
directory
    5 | #include <openssl/sm2.h>
      |          ^~~~~~~~~~~~~~~
compilation terminated.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_sign.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc -o
sm2_sign sm2_sign.c -lcrypto -lssl
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_sign
sm2_private.pem plain.txt signature.bin
Usage: ./sm2_sign <private_key.pem> <input_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_sign
sm2_private.pem plain.txt
Signature:
3046022100cbfb7639a9f9135a99b1993eabbd19fe2cfe573f89177053e6bb3744
07272cb5022100aa171c3d7a88e775bed2ad536bbc6fb6b36898e00e1200074335
8dbeca03af08
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm
sm2_sign.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_sign.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc -o
sm2_sign sm2_sign.c -lcrypto -lssl
sm2_sign.c:5:10: fatal error: openssl/sm2.h: No such file or
directory
    5 | #include <openssl/sm2.h>
      |          ^~~~~~~~~~~~~~~
compilation terminated.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_sign.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc -o
sm2_sign sm2_sign.c -lcrypto -lssl
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_sign
sm2_private.pem plain.txt signature.bin
```

```
Signature successfully written to signature.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_verify.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc
sm2_verify.c -o sm2_verify -lssl -lcrypto
sm2_verify.c:5:10: fatal error: openssl/sm2.h: No such file or
directory
    5 | #include <openssl/sm2.h>
      |          ^~~~~~~~~~~~~~~
compilation terminated.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_verify.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc
sm2_verify.c -o sm2_verify -lssl -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_verify sm2_public.pem plain.txt signature.bin
Signature verification successful!
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# git commit
-m "finish sm2 sign & verify"
[master 8575bbf] finish sm2 sign & verify
5 files changed, 228 insertions(+)
create mode 100644 shiyan1-2/task01/test_sm2/signature.bin
create mode 100755 shiyan1-2/task01/test_sm2/sm2_sign
create mode 100644 shiyan1-2/task01/test_sm2/sm2_sign.c
create mode 100755 shiyan1-2/task01/test_sm2/sm2_verify
create mode 100644 shiyan1-2/task01/test_sm2/sm2_verify.c
```

- 代码（使用evp接口）
  - 签名代码

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/evp.h>
#include <openssl/pem.h>
#include <openssl/err.h>

void handle_errors() {
    ERR_print_errors_fp(stderr);
    abort();
}

void sign_file(const char *private_key_file, const char
*input_file, const char *output_file) {
    FILE *fp = fopen(input_file, "rb");
    if (!fp) {
        perror("Unable to open input file");
        exit(EXIT_FAILURE);
    }
```

```
    // 读取文件内容
    fseek(fp, 0, SEEK_END);
    long file_size = ftell(fp);
    fseek(fp, 0, SEEK_SET);
    unsigned char *data = malloc(file_size);
    if (!data) {
        perror("Unable to allocate memory");
        fclose(fp);
        exit(EXIT_FAILURE);
    }
    fread(data, 1, file_size, fp);
    fclose(fp);

    // 读取私钥
    FILE *key_fp = fopen(private_key_file, "r");
    if (!key_fp) {
        perror("Unable to open private key file");
        free(data);
        exit(EXIT_FAILURE);
    }

    // 从 PEM 文件读取 SM2 私钥
    EVP_PKEY *pkey = PEM_read_PrivateKey(key_fp, NULL, NULL,
NULL);
    fclose(key_fp);
    if (!pkey) {
        handle_errors();
    }

    // 创建签名上下文
    EVP_MD_CTX *ctx = EVP_MD_CTX_new();
    if (!ctx) {
        handle_errors();
    }

    // 初始化签名操作
    if (EVP_DigestSignInit(ctx, NULL, EVP_sm3(), NULL, pkey)
!= 1) {
        handle_errors();
    }

    // 提供要签名的数据
    if (EVP_DigestSignUpdate(ctx, data, file_size) != 1) {
        handle_errors();
    }

    // 获取签名所需的缓冲区大小
    size_t sig_len;
    if (EVP_DigestSignFinal(ctx, NULL, &sig_len) != 1) {
        handle_errors();
    }

    // 分配内存用于签名
    unsigned char *sig = malloc(sig_len);
```

```
    if (!sig) {
        perror("Unable to allocate memory for signature");
        EVP_MD_CTX_free(ctx);
        EVP_PKEY_free(pkey);
        free(data);
        exit(EXIT_FAILURE);
    }

    // 获取签名
    if (EVP_DigestSignFinal(ctx, sig, &sig_len) != 1) {
        handle_errors();
    }

    // 将签名写入输出文件
    FILE *out_fp = fopen(output_file, "wb");
    if (!out_fp) {
        perror("Unable to open output file");
        free(sig);
        EVP_MD_CTX_free(ctx);
        EVP_PKEY_free(pkey);
        free(data);
        exit(EXIT_FAILURE);
    }

    fwrite(sig, 1, sig_len, out_fp);
    fclose(out_fp);

    printf("Signature successfully written to %s\n",
output_file);

    // 清理资源
    free(sig);
    EVP_MD_CTX_free(ctx);
    EVP_PKEY_free(pkey);
    free(data);
}

int main(int argc, char **argv) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <private_key.pem>
<input_file> <output_file>\n", argv[0]);
        return EXIT_FAILURE;
    }

    sign_file(argv[1], argv[2], argv[3]);
    return EXIT_SUCCESS;
}
```

- 验证代码

```
#include <stdio.h>
#include <stdlib.h>
```

```c
#include <string.h>
#include <openssl/evp.h>
#include <openssl/pem.h>
#include <openssl/err.h>

void handle_errors() {
    ERR_print_errors_fp(stderr);
    abort();
}

int verify_signature(const char *public_key_file, const char
*input_file, const char *signature_file) {
    // 打开输入文件
    FILE *fp = fopen(input_file, "rb");
    if (!fp) {
        perror("Unable to open input file");
        return EXIT_FAILURE;
    }

    // 读取文件内容
    fseek(fp, 0, SEEK_END);
    long file_size = ftell(fp);
    fseek(fp, 0, SEEK_SET);
    unsigned char *data = malloc(file_size);
    if (!data) {
        perror("Unable to allocate memory");
        fclose(fp);
        return EXIT_FAILURE;
    }
    fread(data, 1, file_size, fp);
    fclose(fp);

    // 打开签名文件
    FILE *sig_fp = fopen(signature_file, "rb");
    if (!sig_fp) {
        perror("Unable to open signature file");
        free(data);
        return EXIT_FAILURE;
    }

    // 读取签名
    fseek(sig_fp, 0, SEEK_END);
    long sig_size = ftell(sig_fp);
    fseek(sig_fp, 0, SEEK_SET);
    unsigned char *signature = malloc(sig_size);
    if (!signature) {
        perror("Unable to allocate memory for signature");
        fclose(sig_fp);
        free(data);
        return EXIT_FAILURE;
    }
    fread(signature, 1, sig_size, sig_fp);
    fclose(sig_fp);
```

```
    // 读取公钥
    FILE *key_fp = fopen(public_key_file, "r");
    if (!key_fp) {
        perror("Unable to open public key file");
        free(signature);
        free(data);
        return EXIT_FAILURE;
    }

    // 从 PEM 文件读取 SM2 公钥
    EVP_PKEY *pkey = PEM_read_PUBKEY(key_fp, NULL, NULL,
NULL);
    fclose(key_fp);
    if (!pkey) {
        handle_errors();
    }

    // 创建验证上下文
    EVP_MD_CTX *ctx = EVP_MD_CTX_new();
    if (!ctx) {
        handle_errors();
    }

    // 初始化验证操作
    if (EVP_DigestVerifyInit(ctx, NULL, EVP_sm3(), NULL,
pkey) != 1) {
        handle_errors();
    }

    // 提供要验证的数据
    if (EVP_DigestVerifyUpdate(ctx, data, file_size) != 1) {
        handle_errors();
    }

    // 验证签名
    int ret = EVP_DigestVerifyFinal(ctx, signature,
sig_size);
    if (ret == 1) {
        printf("Signature verification successful!\n");
    } else {
        printf("Signature verification failed!\n");
    }

    // 清理资源
    EVP_MD_CTX_free(ctx);
    EVP_PKEY_free(pkey);
    free(signature);
    free(data);

    return ret == 1 ? EXIT_SUCCESS : EXIT_FAILURE;
}

int main(int argc, char **argv) {
    if (argc != 4) {
```

```
        fprintf(stderr, "Usage: %s <public_key.pem>
<input_file> <signature_file>\n", argv[0]);
        return EXIT_FAILURE;
    }

    return verify_signature(argv[1], argv[2], argv[3]);
}
```

- 用代码生成SM4密钥和IV文件，同时对原有SM4加解密代码进行修改：
    - 密钥生成和IV生成
        - 密钥生成
            - 过程

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# ls
decrypted_file.txt  encrypted_file.txt  plain.txt
sm4_decrypt  sm4_decrypt.c  sm4_encrypt  sm4_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# nano
sm4_key_gen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# gcc -
o sm4_key_gen sm4_key_gen.c -lssl -lcrypto
sm4_key_gen.c: In function ‘generate_sm4_key’:
sm4_key_gen.c:29:9: warning: implicit declaration of function
‘RAND_bytes’ [-Wimplicit-function-declaration]
29 |     if (RAND_bytes(key, key_length) != 1) {
   |         ^~~~~~~~~~
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# rm
sm4_key_gen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# nano
sm4_key_gen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# gcc -
o sm4_key_gen sm4_key_gen.c -lssl -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4#
./sm4_key_gen
Generated SM4 Key: 7198690feff5fdd1e80c16438a38c1c1
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# rm
sm4_key_gen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# nano
sm4_key_gen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# gcc -
o sm4_key_gen sm4_key_gen.c -lssl -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4#
./sm4_key_gen
Usage: ./sm4_key_gen <key_filename>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4#
./sm4_key_gen my_sm4_key.bin
Key written to file: my_sm4_key.bin
Generated SM4 Key: 7339a72b904a019b19050853a62ce6a2
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4#
hexdump -C my_sm4_key.bin
00000000  73 39 a7 2b 90 4a 01 9b  19 05 08 53 a6 2c e6 a2
```

```
|s9.+.J.....S.,..|
00000010
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# git
add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# git
commit -m "finish openssl sm4 key gen"
[master acdac49] finish openssl sm4 key gen
3 files changed, 83 insertions(+)
create mode 100644 shiyan1-2/task01/test_sm4/my_sm4_key.bin
create mode 100755 shiyan1-2/task01/test_sm4/sm4_key_gen
create mode 100644 shiyan1-2/task01/test_sm4/sm4_key_gen.c
```

- 代码

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <openssl/rand.h>

// 错误处理函数
void handleErrors(void) {
    ERR_print_errors_fp(stderr);
    abort();
}

// SM4密钥生成函数
void generate_sm4_key(unsigned char *key, int key_length) {
    // 创建EVP_CTX上下文
    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
    if (ctx == NULL) {
        fprintf(stderr, "Failed to create EVP_CIPHER_CTX\n");
        handleErrors();
    }

    // 使用EVP_EncryptInit_ex初始化上下文
    if (EVP_EncryptInit_ex(ctx, EVP_sm4_cbc(), NULL, NULL,
NULL) != 1) {
        fprintf(stderr, "Failed to initialize EVP cipher\n");
        handleErrors();
    }

    // 生成随机密钥
    if (RAND_bytes(key, key_length) != 1) {
        fprintf(stderr, "Failed to generate random key\n");
        handleErrors();
    }

    // 清理上下文
    EVP_CIPHER_CTX_free(ctx);
}
```

```
// 将密钥写入文件的函数
void write_key_to_file(const char *filename, unsigned char
*key, int key_length) {
    FILE *file = fopen(filename, "wb");
    if (file == NULL) {
        fprintf(stderr, "Failed to open file %s for
writing\n", filename);
        handleErrors();
    }

    // 写入密钥到文件
    if (fwrite(key, 1, key_length, file) != key_length) {
        fprintf(stderr, "Failed to write key to file\n");
        handleErrors();
    }

    fclose(file);
    printf("Key written to file: %s\n", filename);
}

int main(int argc, char *argv[]) {
    // 检查命令行参数
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <key_filename>\n",
argv[0]);
        return EXIT_FAILURE;
    }

    // 定义密钥长度（128位 = 16字节）
    int key_length = 16;
    unsigned char key[16];

    // 生成SM4密钥
    generate_sm4_key(key, key_length);

    // 将密钥写入指定文件
    write_key_to_file(argv[1], key, key_length);

    // 打印生成的密钥（以十六进制格式）
    printf("Generated SM4 Key: ");
    for (int i = 0; i < key_length; i++) {
        printf("%02x", key[i]);
    }
    printf("\n");

    return EXIT_SUCCESS;
}
```

- IV文件生成
  - 过程

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# nano
sm4_iv_gen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# gcc -
o sm4_iv_gen sm4_iv_gen.c -lssl -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4#
./sm4_iv_gen my_sm4_iv.bin
IV written to file: my_sm4_iv.bin
Generated SM4 IV: b673ebec4b7baa1a8e92bea503b15e3f
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4#
hexdump -C my_sm4_iv.bin
00000000  b6 73 eb ec 4b 7b aa 1a  8e 92 be a5 03 b1 5e 3f
|.s..K{........^?|
00000010
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# git
add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# git
commit -m "finish openssl iv gen"
[master 0b46498] finish openssl iv gen
3 files changed, 67 insertions(+)
create mode 100644 shiyan1-2/task01/test_sm4/my_sm4_iv.bin
create mode 100755 shiyan1-2/task01/test_sm4/sm4_iv_gen
create mode 100644 shiyan1-2/task01/test_sm4/sm4_iv_gen.c
```

- 代码

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <openssl/rand.h>

// 错误处理函数
void handleErrors(void) {
    ERR_print_errors_fp(stderr);
    abort();
}

// 生成SM4的初始化向量（IV）
void generate_sm4_iv(unsigned char *iv, int iv_length) {
    // 生成随机IV
    if (RAND_bytes(iv, iv_length) != 1) {
        fprintf(stderr, "Failed to generate random IV\n");
        handleErrors();
    }
}

// 将IV写入文件的函数
void write_iv_to_file(const char *filename, unsigned char
*iv, int iv_length) {
    FILE *file = fopen(filename, "wb");
```

```
    if (file == NULL) {
        fprintf(stderr, "Failed to open file %s for
writing\n", filename);
        handleErrors();
    }

    // 写入IV到文件
    if (fwrite(iv, 1, iv_length, file) != iv_length) {
        fprintf(stderr, "Failed to write IV to file\n");
        handleErrors();
    }

    fclose(file);
    printf("IV written to file: %s\n", filename);
}

int main(int argc, char *argv[]) {
    // 检查命令行参数
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <iv_filename>\n",
argv[0]);
        return EXIT_FAILURE;
    }

    // 定义IV长度（128位 = 16字节）
    int iv_length = 16;
    unsigned char iv[16];

    // 生成SM4的IV
    generate_sm4_iv(iv, iv_length);

    // 将IV写入指定文件
    write_iv_to_file(argv[1], iv, iv_length);

    // 打印生成的IV（以十六进制格式）
    printf("Generated SM4 IV: ");
    for (int i = 0; i < iv_length; i++) {
        printf("%02x", iv[i]);
    }
    printf("\n");

    return EXIT_SUCCESS;
}
```

- SM4加解密
  - 过程与验证

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# ls
decrypted_file.txt  my_sm4_iv.bin   plain.txt    sm4_decrypt.c
sm4_encrypt.c  sm4_iv_gen.c  sm4_key_gen.c
encrypted_file.txt  my_sm4_key.bin  sm4_decrypt  sm4_encrypt
```

```
sm4_iv_gen     sm4_key_gen
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# nano
sm4_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# rm
sm4_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# nano
sm4_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# rm
sm4_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# nano
sm4_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4#
./sm4_encrypt key.bin iv.bin input.txt output.enc
Could not open input.txt for reading
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# gcc -o
sm4_encrypt sm4_encrypt.c -lssl -lcrypto
sm4_encrypt.c: In function ‘handleErrors’:
sm4_encrypt.c:12:5: warning: implicit declaration of function
‘ERR_print_errors_fp’ [-Wimplicit-function-declaration]
12 |     ERR_print_errors_fp(stderr); // 输出详细的错误信息
   |     ^~~~~~~~~~~~~~~~~~
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# rm
sm4_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# nano
sm4_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# gcc -o
sm4_encrypt sm4_encrypt.c -lssl -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# ls
decrypted_file.txt  my_sm4_iv.bin   plain.txt    sm4_decrypt.c
sm4_encrypt.c  sm4_iv_gen.c  sm4_key_gen.c
encrypted_file.txt  my_sm4_key.bin  sm4_decrypt  sm4_encrypt
sm4_iv_gen     sm4_key_gen
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4#
./sm4_encrypt
Usage: ./sm4_encrypt <key_file> <iv_file> <input_file>
<output_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4#
./sm4_encrypt  my_sm4_key.bin my_sm4_iv.bin plain.txt
encrypted_file.txt
Encryption complete.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# rm
sm4_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# nano
sm4_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# gcc -o
sm4_decrypt sm4_decrypt.c -lssl -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4#
./sm4_decrypt
Usage: ./sm4_decrypt <key_file> <iv_file> <input_file>
<output_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4#
./sm4_decrypt my_sm4_key.bin my_sm4_iv.bin encrypted_file.txt
decrypted_file.txt
Decryption complete.
```

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# cat
plain.txt
20221414xlm
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# cat
decrypted_file.txt
20221414xlm
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm4# git commit
-m "update openssl sm4 encrypt & decrypt code"
[master 9c6e06f] update openssl sm4 encrypt & decrypt code
5 files changed, 117 insertions(+), 27 deletions(-)
```

- 代码
  - 加密代码

```c
#include <openssl/evp.h>
#include <openssl/err.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void handleErrors(void) {
    fprintf(stderr, "An error occurred.\n");
    ERR_print_errors_fp(stderr);
    exit(1);
}

int read_key_and_iv(const char *key_file, const char
*iv_file, unsigned char *key, unsigned char *iv) {
    FILE *kf = fopen(key_file, "rb");
    FILE *ivf = fopen(iv_file, "rb");
    if (!kf || !ivf) {
        fprintf(stderr, "Could not open key or IV file.\n");
        return -1;
    }
    size_t key_len = fread(key, 1, 16, kf);
    if (key_len != 16) {
        fprintf(stderr, "Key must be 16 bytes long.\n");
        fclose(kf);
        fclose(ivf);
        return -1;
    }
    size_t iv_len = fread(iv, 1, 16, ivf);
    if (iv_len != 16) {
        fprintf(stderr, "IV must be 16 bytes long.\n");
        fclose(kf);
        fclose(ivf);
        return -1;
    }
    fclose(kf);
    fclose(ivf);
```

```
    return 0;
}

int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "Usage: %s <key_file> <iv_file>
<input_file> <output_file>\n", argv[0]);
        return 1;
    }

    unsigned char key[16], iv[16];
    if (read_key_and_iv(argv[1], argv[2], key, iv) != 0) {
        return 1;
    }

    FILE *f_input = fopen(argv[3], "rb");
    FILE *f_output = fopen(argv[4], "wb");
    if (!f_input || !f_output) {
        fprintf(stderr, "Could not open input or output
file.\n");
        return 1;
    }

    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
    if (!ctx) handleErrors();

    if (1 != EVP_EncryptInit_ex(ctx, EVP_sm4_cbc(), NULL,
key, iv))
        handleErrors();

    unsigned char buffer[1024], ciphertext[1024 +
EVP_MAX_BLOCK_LENGTH];
    int bytes_read, ciphertext_len, final_len;

    while ((bytes_read = fread(buffer, 1, sizeof(buffer),
f_input)) > 0) {
        if (1 != EVP_EncryptUpdate(ctx, ciphertext,
&ciphertext_len, buffer, bytes_read))
            handleErrors();
        fwrite(ciphertext, 1, ciphertext_len, f_output);
    }

    if (1 != EVP_EncryptFinal_ex(ctx, ciphertext +
ciphertext_len, &final_len))
        handleErrors();
    fwrite(ciphertext, 1, ciphertext_len + final_len,
f_output);

    EVP_CIPHER_CTX_free(ctx);
    fclose(f_input);
    fclose(f_output);
    printf("Encryption complete.\n");
    return 0;
}
```

- 解密代码

```
#include <openssl/evp.h>
#include <openssl/err.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void handleErrors(void) {
    fprintf(stderr, "An error occurred.\n");
    ERR_print_errors_fp(stderr);
    exit(1);
}

int read_key_and_iv(const char *key_file, const char
*iv_file, unsigned char *key, unsigned char *iv) {
    FILE *kf = fopen(key_file, "rb");
    FILE *ivf = fopen(iv_file, "rb");
    if (!kf || !ivf) {
        fprintf(stderr, "Could not open key or IV file.\n");
        return -1;
    }
    size_t key_len = fread(key, 1, 16, kf);
    if (key_len != 16) {
        fprintf(stderr, "Key must be 16 bytes long.\n");
        fclose(kf);
        fclose(ivf);
        return -1;
    }
    size_t iv_len = fread(iv, 1, 16, ivf);
    if (iv_len != 16) {
        fprintf(stderr, "IV must be 16 bytes long.\n");
        fclose(kf);
        fclose(ivf);
        return -1;
    }
    fclose(kf);
    fclose(ivf);
    return 0;
}

int main(int argc, char *argv[]) {
    if (argc != 5) {
        fprintf(stderr, "Usage: %s <key_file> <iv_file>
<input_file> <output_file>\n", argv[0]);
        return 1;
    }

    unsigned char key[16], iv[16];
    if (read_key_and_iv(argv[1], argv[2], key, iv) != 0) {
        return 1;
```

```
    }

    FILE *f_input = fopen(argv[3], "rb");
    FILE *f_output = fopen(argv[4], "wb");
    if (!f_input || !f_output) {
        fprintf(stderr, "Could not open input or output
file.\n");
        return 1;
    }

    EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
    if (!ctx) handleErrors();

    if (1 != EVP_DecryptInit_ex(ctx, EVP_sm4_cbc(), NULL,
key, iv))
        handleErrors();

    unsigned char buffer[1024], plaintext[1024 +
EVP_MAX_BLOCK_LENGTH];
    int bytes_read, plaintext_len, final_len;

    while ((bytes_read = fread(buffer, 1, sizeof(buffer),
f_input)) > 0) {
        if (1 != EVP_DecryptUpdate(ctx, plaintext,
&plaintext_len, buffer, bytes_read))
            handleErrors();
        fwrite(plaintext, 1, plaintext_len, f_output);
    }

    if (1 != EVP_DecryptFinal_ex(ctx, plaintext +
plaintext_len, &final_len))
        handleErrors();
    fwrite(plaintext, 1, plaintext_len + final_len,
f_output);

    EVP_CIPHER_CTX_free(ctx);
    fclose(f_input);
    fclose(f_output);
    printf("Decryption complete.\n");
    return 0;
}
```

- 用代码生成SM2的密钥并验证生成的密钥是否正确
  - 过程与验证

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ls
decrypted_file.bin  plain.txt      sm2_decrypt    sm2_encrypt
sm2_private.pem  sm2_sign    sm2_verify
encrypted_file.bin  signature.bin  sm2_decrypt.c  sm2_encrypt.c
sm2_public.pem   sm2_sign.c  sm2_verify.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_sign
```

```
Usage: ./sm2_sign <private_key.pem> <input_file> <output_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_verify
Usage: ./sm2_verify <public_key.pem> <input_file> <signature_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_sign
sm2_private.pem plain.txt signature.bin
Signature successfully written to signature.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_verify
sm2_public.pem plain.txt signature.bin
Signature verification successful!
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc -o
sm2_keygen sm2_keygen.c -lcrypto
sm2_keygen.c: In function ‘main’:
sm2_keygen.c:17:5: warning: implicit declaration of function
‘OPENSSL_init_ssl’; did you mean ‘OPENSSL_init’? [-Wimplicit-function-
declaration]
17 |     OPENSSL_init_ssl(0, NULL);
   |     ^~~~~~~~~~~~~~~~
   |     OPENSSL_init
sm2_keygen.c:46:9: warning: implicit declaration of function
‘PEM_write_PUBKEY’ [-Wimplicit-function-declaration]
46 |     if (PEM_write_PUBKEY(pub_key_file, pkey) != 1) {
   |         ^~~~~~~~~~~~~~~~
/usr/bin/ld: /tmp/ccC7svUU.o: in function `main':
sm2_keygen.c:(.text+0x57): undefined reference to `OPENSSL_init_ssl'
collect2: error: ld returned 1 exit status
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc -o
sm2_keygen sm2_keygen.c -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc -o
sm2_keygen sm2_keygen.c -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# rm sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# nano
sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# gcc -o
sm2_keygen sm2_keygen.c -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ls
decrypted_file.bin  signature.bin  sm2_encrypt    sm2_keygen.c
sm2_sign    sm2_verify.c
encrypted_file.bin  sm2_decrypt    sm2_encrypt.c  sm2_private.pem
sm2_sign.c
plain.txt           sm2_decrypt.c  sm2_keygen     sm2_public.pem
sm2_verify
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_keygen
sm2_private_key.pem sm2_public_key.pem
```

```
SM2 private key has been written to sm2_private_key.pem
SM2 public key has been written to sm2_public_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ls
decrypted_file.bin  signature.bin  sm2_encrypt    sm2_keygen.c
sm2_public.pem      sm2_sign.c
encrypted_file.bin  sm2_decrypt    sm2_encrypt.c  sm2_private.pem
sm2_public_key.pem  sm2_verify
plain.txt           sm2_decrypt.c  sm2_keygen     sm2_private_key.pem
sm2_sign            sm2_verify.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_encrypt
Usage: ./sm2_encrypt <pubkey.pem> <input file> <output file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_decrypt
Usage: ./sm2_decrypt <private_key.pem> <encrypted_file>
<decrypted_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_sign
sm2_private_key.pem plain.txt signature.bin
Signature successfully written to signature.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_verify
sm2_public_key.pem plain.txt signature.bin
Signature verification successful!
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_encrypt
sm2_public_key.pem plain.txt encrypted_file.bin
Encryption complete.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_decrypt
sm2_private_key.pem encrypted_file.bin decrypted_file.bin
Decryption successful.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# cat
decrypted_file.bin
test10/16
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# cat plain.txt
test10/16
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# git commit -m
"finish openssl sm2 key gen"
[master 281faf7] finish openssl sm2 key gen
7 files changed, 93 insertions(+), 1 deletion(-)
create mode 100755 shiyan1-2/task01/test_sm2/sm2_keygen
create mode 100644 shiyan1-2/task01/test_sm2/sm2_keygen.c
create mode 100644 shiyan1-2/task01/test_sm2/sm2_private_key.pem
create mode 100644 shiyan1-2/task01/test_sm2/sm2_public_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# git branch
* master
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# git push -u
origin master
Username for 'https://gitee.com': xu-luming
Password for 'https://xu-luming@gitee.com':
Enumerating objects: 17, done.
Counting objects: 100% (17/17), done.
Delta compression using up to 8 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (11/11), 5.43 KiB | 5.43 MiB/s, done.
Total 11 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Powered by GITEE.COM [1.1.5]
remote: Set trace flag 674073b4
```

```
To https://gitee.com/xu-luming/information-security-system-design-
experiment1-fundamentals-of-embedded-development.git
9c6e06f..281faf7  master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

- 代码

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <openssl/pem.h>
#include <openssl/provider.h>

void handleErrors() {
    ERR_print_errors_fp(stderr);
    abort();
}

int main(int argc, char *argv[]) {
    EVP_PKEY *pkey = NULL;
    EVP_PKEY_CTX *ctx = NULL;

    // 检查命令行参数
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <private_key_file.pem>
<public_key_file.pem>\n", argv[0]);
        return EXIT_FAILURE;
    }

    // 初始化 OpenSSL
    if (OPENSSL_init_crypto(0, NULL) == 0) {
        fprintf(stderr, "Failed to initialize OpenSSL\n");
        return EXIT_FAILURE;
    }
    ERR_load_crypto_strings();

    // 创建上下文
    ctx = EVP_PKEY_CTX_new_id(EVP_PKEY_SM2, NULL);
    if (!ctx) {
        fprintf(stderr, "Failed to create EVP_PKEY_CTX\n");
        handleErrors();
    }

    // 初始化密钥生成
    if (EVP_PKEY_keygen_init(ctx) <= 0) {
        fprintf(stderr, "Failed to initialize key generation\n");
        handleErrors();
    }

    // 生成密钥
```

```
    if (EVP_PKEY_keygen(ctx, &pkey) <= 0) {
        fprintf(stderr, "Failed to generate SM2 key\n");
        handleErrors();
    }

    // 输出生成的私钥到指定文件
    FILE *priv_key_file = fopen(argv[1], "w");
    if (!priv_key_file) {
        fprintf(stderr, "Failed to open file for writing private
key\n");
        handleErrors();
    }

    if (PEM_write_PrivateKey(priv_key_file, pkey, NULL, NULL, 0, NULL,
NULL) != 1) {
        fprintf(stderr, "Failed to write private key to file\n");
        handleErrors();
    }
    fclose(priv_key_file);
    printf("SM2 private key has been written to %s\n", argv[1]);

    // 输出生成的公钥到指定文件
    FILE *pub_key_file = fopen(argv[2], "w");
    if (!pub_key_file) {
        fprintf(stderr, "Failed to open file for writing public
key\n");
        handleErrors();
    }

    if (PEM_write_PUBKEY(pub_key_file, pkey) != 1) {
        fprintf(stderr, "Failed to write public key to file\n");
        handleErrors();
    }
    fclose(pub_key_file);
    printf("SM2 public key has been written to %s\n", argv[2]);

    // 释放资源
    EVP_PKEY_free(pkey);
    EVP_PKEY_CTX_free(ctx);
    ERR_free_strings();
    return EXIT_SUCCESS;
}
```

2. 参考相关内容，在 Ubuntu或openEuler中（推荐 openEuler）中使用GmSSL库编程实现调用SM2（加密解密，签名验签），SM3（摘要计算，HMAC 计算），SM4（加密解密）算法，使用Markdown记录详细记录实践过程，每完成一项git commit 一次。 (5')

- sm2加解密、签名与验证
    - sm2密钥对生成（密钥格式：PEM）
        - 过程

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01# ls
test_sm2  test_sm3  test_sm4
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2# mkdir task02
root@Youer:~/shiyan/shiyan01/shiyan1-2# cd task02
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# mkdir test_sm2
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# cd test_sm2
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# mkdir src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# cd src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# nano
sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# cd .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# mkdir bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# gcc -o
bin/sm2_keygen src/sm2_keygen.c -lcrypto -lssl
src/sm2_keygen.c: In function ‘generate_sm2_keypair’:
src/sm2_keygen.c:22:5: warning: ‘EC_KEY_new’ is deprecated: Since
OpenSSL 3.0 [-Wdeprecated-declarations]
22 |     key = EC_KEY_new();
   |     ^~~
In file included from src/sm2_keygen.c:1:
/usr/include/openssl/ec.h:968:31: note: declared here
968 | OSSL_DEPRECATEDIN_3_0 EC_KEY *EC_KEY_new(void);
   |                               ^~~~~~~~~~
src/sm2_keygen.c:30:5: warning: ‘EC_KEY_set_group’ is deprecated:
Since OpenSSL 3.0 [-Wdeprecated-declarations]
30 |     if (!EC_KEY_set_group(key, group)) {
   |     ^~
In file included from src/sm2_keygen.c:1:
/usr/include/openssl/ec.h:1042:27: note: declared here
1042 | OSSL_DEPRECATEDIN_3_0 int EC_KEY_set_group(EC_KEY *key,
const EC_GROUP *group);
   |                           ^~~~~~~~~~~~~~~~
src/sm2_keygen.c:37:5: warning: ‘EC_KEY_generate_key’ is
deprecated: Since OpenSSL 3.0 [-Wdeprecated-declarations]
37 |     if (!EC_KEY_generate_key(key)) {
   |     ^~
In file included from src/sm2_keygen.c:1:
/usr/include/openssl/ec.h:1101:27: note: declared here
1101 | OSSL_DEPRECATEDIN_3_0 int EC_KEY_generate_key(EC_KEY *key);
   |                           ^~~~~~~~~~~~~~~~~~~
src/sm2_keygen.c:59:10: warning: implicit declaration of function
‘PEM_write_ECPrivateKey’; did you mean ‘i2d_ECPrivateKey’? [-
Wimplicit-function-declaration]
59 |     if (!PEM_write_ECPrivateKey(priv_file, key, NULL, NULL,
0, NULL, NULL)) {
   |          ^~~~~~~~~~~~~~~~~~~~~
   |          i2d_ECPrivateKey
src/sm2_keygen.c:67:10: warning: implicit declaration of function
‘PEM_write_ECPublicKey’ [-Wimplicit-function-declaration]
67 |     if (!PEM_write_ECPublicKey(pub_file, key)) {
```

```
    |          ^~~~~~~~~~~~~~~~~~~~~
src/sm2_keygen.c:82:5: warning: ‘EC_KEY_free’ is deprecated: Since
OpenSSL 3.0 [-Wdeprecated-declarations]
82 |     if (key) EC_KEY_free(key);
    |     ^~
In file included from src/sm2_keygen.c:1:
/usr/include/openssl/ec.h:1003:28: note: declared here
1003 | OSSL_DEPRECATEDIN_3_0 void EC_KEY_free(EC_KEY *key);
    |                            ^~~~~~~~~~
/usr/bin/ld: /tmp/ccfzDR1u.o: in function `generate_sm2_keypair':
sm2_keygen.c:(.text+0x28b): undefined reference to
`PEM_write_ECPublicKey'
collect2: error: ld returned 1 exit status
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# rm
src/sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# cd src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# nano
sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# gcc -o
../bin/sm2_keygen sm2_keygen.c -lcrypto -lssl
sm2_keygen.c: In function ‘generate_sm2_keypair’:
sm2_keygen.c:23:5: warning: ‘EC_KEY_new_method’ is deprecated:
Since OpenSSL 3.0 [-Wdeprecated-declarations]
23 |     key = EC_KEY_new_method(NULL); // 使用新的API
    |     ^~~
In file included from sm2_keygen.c:1:
/usr/include/openssl/ec.h:1284:31: note: declared here
1284 | OSSL_DEPRECATEDIN_3_0 EC_KEY *EC_KEY_new_method(ENGINE
*engine);
    |                               ^~~~~~~~~~~~~~~~~
sm2_keygen.c:31:5: warning: ‘EC_KEY_set_group’ is deprecated:
Since OpenSSL 3.0 [-Wdeprecated-declarations]
31 |     if (!EC_KEY_set_group(key, group)) {
    |     ^~
In file included from sm2_keygen.c:1:
/usr/include/openssl/ec.h:1042:27: note: declared here
1042 | OSSL_DEPRECATEDIN_3_0 int EC_KEY_set_group(EC_KEY *key,
const EC_GROUP *group);
    |                           ^~~~~~~~~~~~~~~
sm2_keygen.c:38:5: warning: ‘EC_KEY_generate_key’ is deprecated:
Since OpenSSL 3.0 [-Wdeprecated-declarations]
38 |     if (!EC_KEY_generate_key(key)) {
    |     ^~
In file included from sm2_keygen.c:1:
/usr/include/openssl/ec.h:1101:27: note: declared here
1101 | OSSL_DEPRECATEDIN_3_0 int EC_KEY_generate_key(EC_KEY *key);
    |                           ^~~~~~~~~~~~~~~~~~
sm2_keygen.c:60:5: warning: ‘PEM_write_ECPrivateKey’ is
deprecated: Since OpenSSL 3.0 [-Wdeprecated-declarations]
60 |     if (!PEM_write_ECPrivateKey(priv_file, key, NULL, NULL,
0, NULL, NULL)) {
    |     ^~
In file included from sm2_keygen.c:4:
/usr/include/openssl/pem.h:462:1: note: declared here
```

```
462 | DECLARE_PEM_rw_cb_attr(OSSL_DEPRECATEDIN_3_0, ECPrivateKey,
EC_KEY)
    | ^~~~~~~~~~~~~~~~~~~~~~
sm2_keygen.c:63:7: error: ‘el’ undeclared (first use in this
function)
63 |     } el
    |       ^~
sm2_keygen.c:63:7: note: each undeclared identifier is reported
only once for each function it appears in
sm2_keygen.c:63:9: error: expected ‘;’ at end of input
63 |     } el
    |         ^
    |         ;
sm2_keygen.c:63:5: error: expected declaration or statement at end
of input
63 |     } el
    |     ^
sm2_keygen.c:56:9: error: label ‘cleanup’ used but not defined
56 |         goto cleanup;
    |         ^~~~
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# rm
sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# nano
sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# gcc -o
../bin/sm2_keygen sm2_keygen.c -lcrypto -lssl
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2#
./bin/sm2_keygen private_key.pem public_key.pem
Private key written to private_key.pem
Public key written to public_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# ls
bin  private_key.pem  public_key.pem  src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# git commit
-m "gmssl:create keys by code"
[master 0ae010f] gmssl:create keys by code
4 files changed, 98 insertions(+)
create mode 100755 shiyan1-2/task02/test_sm2/bin/sm2_keygen
create mode 100644 shiyan1-2/task02/test_sm2/private_key.pem
create mode 100644 shiyan1-2/task02/test_sm2/public_key.pem
create mode 100644 shiyan1-2/task02/test_sm2/src/sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# nano
src/sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# gcc -o
../bin/sm2_keygen src/sm2_keygen.c -lcrypto -lssl
src/sm2_keygen.c: In function ‘main’:
src/sm2_keygen.c:15:24: warning: implicit declaration of function
‘sm2_key_new’; did you mean ‘sm2_key_print’? [-Wimplicit-function-
declaration]
15 |     SM2_KEY *sm2_key = sm2_key_new();
    |                        ^~~~~~~~~~~
    |                        sm2_key_print
src/sm2_keygen.c:15:24: warning: initialization of ‘SM2_KEY *’
```

```
from ‘int’ makes pointer from integer without a cast [-Wint-
conversion]
src/sm2_keygen.c:21:9: error: too many arguments to function
‘sm2_key_generate’
21 |     if (sm2_key_generate(sm2_key, 256) != 1) {
   |         ^~~~~~~~~~~~~~~~
In file included from src/sm2_keygen.c:3:
/usr/local/include/gmssl/sm2.h:31:5: note: declared here
31 | int sm2_key_generate(SM2_KEY *key);
   |     ^~~~~~~~~~~~~~~~
src/sm2_keygen.c:23:9: warning: implicit declaration of function
‘sm2_key_free’; did you mean ‘sm2_key_print’? [-Wimplicit-
function-declaration]
23 |         sm2_key_free(sm2_key);
   |         ^~~~~~~~~~~~
   |         sm2_key_print
src/sm2_keygen.c:34:9: warning: implicit declaration of function
‘sm2_key_write_private_key’; did you mean
‘sm2_key_set_private_key’? [-Wimplicit-function-declaration]
34 |     if (sm2_key_write_private_key(sm2_key, fp_private) != 1)
{
   |         ^~~~~~~~~~~~~~~~~~~~~~~~
   |         sm2_key_set_private_key
src/sm2_keygen.c:49:9: warning: implicit declaration of function
‘sm2_key_write_public_key’; did you mean ‘sm2_key_set_public_key’?
[-Wimplicit-function-declaration]
49 |     if (sm2_key_write_public_key(sm2_key, fp_public) != 1) {
   |         ^~~~~~~~~~~~~~~~~~~~~~~
   |         sm2_key_set_public_key
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# rm
src/sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# nano
src/sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# ls
bin  private_key.pem  public_key.pem  sm2_keygen.c  src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# cp
./sm2_keygen.c ./src/
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# ls
bin  private_key.pem  public_key.pem  sm2_keygen.c  src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# rm
sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# gcc -o
../bin/sm2_keygen src/sm2_keygen.c -lcrypto -lssl
/usr/bin/ld: cannot open output file ../bin/sm2_keygen: No such
file or directory
collect2: error: ld returned 1 exit status
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# tree
.
├── bin
│   └── sm2_keygen
├── private_key.pem
├── public_key.pem
└── src
    └── sm2_keygen.c
```

```
2 directories, 4 files
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# gcc -o
../bin/sm2_keygen src/sm2_keygen.c -lcrypto -lssl
/usr/bin/ld: cannot open output file ../bin/sm2_keygen: No such
file or directory
collect2: error: ld returned 1 exit status
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# gcc -o
./bin/sm2_keygen src/sm2_keygen.c -lcrypto -lssl
/usr/bin/ld: /tmp/ccq6xvrX.o: in function `main':
sm2_keygen.c:(.text+0x95): undefined reference to
`sm2_key_generate'
/usr/bin/ld: sm2_keygen.c:(.text+0x123): undefined reference to
`sm2_private_key_info_to_pem'
/usr/bin/ld: sm2_keygen.c:(.text+0x1cc): undefined reference to
`sm2_public_key_info_to_pem'
collect2: error: ld returned 1 exit status
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# gcc -o
bin/sm2_keygen src/sm2_keygen.c -lcrypto -lssl
/usr/bin/ld: /tmp/ccR5ZVHB.o: in function `main':
sm2_keygen.c:(.text+0x95): undefined reference to
`sm2_key_generate'
/usr/bin/ld: sm2_keygen.c:(.text+0x123): undefined reference to
`sm2_private_key_info_to_pem'
/usr/bin/ld: sm2_keygen.c:(.text+0x1cc): undefined reference to
`sm2_public_key_info_to_pem'
collect2: error: ld returned 1 exit status
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# gcc -o
bin/sm2_keygen src/sm2_keygen.c -lcrypto -lssl -L/usr/local/lib -
lgmssl
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# tree
.
├── bin
│   └── sm2_keygen
├── private_key.pem
├── public_key.pem
└── src
    └── sm2_keygen.c

2 directories, 4 files
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2#
./bin/sm2_keygen private_key.pem public_key.pem
SM2 key pair generated successfully.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# ls
bin  private_key.pem  public_key.pem  src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# rm *.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2#
./bin/sm2_keygen private_key.pem public_key.pem
SM2 key pair generated successfully.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# ls
bin  private_key.pem  public_key.pem  src
```

- 代码

```c
#include <stdio.h>
#include <stdlib.h>
#include <gmssl/sm2.h>
#include <gmssl/error.h>
#include <gmssl/pem.h>

int main(int argc, char **argv) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <private_key_file>
<public_key_file>\n", argv[0]);
        return 1;
    }

    const char *private_key_file = argv[1];
    const char *public_key_file = argv[2];

    SM2_KEY sm2_key;
    FILE *fp;

    // 生成SM2密钥对
    if (sm2_key_generate(&sm2_key) != 1) {
        fprintf(stderr, "Failed to generate SM2 key pair.\n");
        return 1;
    }

    // 保存私钥
    if (!(fp = fopen(private_key_file, "wb"))) {
        perror("Failed to open private key file");
        return 1;
    }
    if (sm2_private_key_info_to_pem(&sm2_key, fp) != 1) {
        fprintf(stderr, "Failed to write private key to file.\n");
        fclose(fp);
        return 1;
    }
    fclose(fp);

    // 保存公钥
    if (!(fp = fopen(public_key_file, "wb"))) {
        perror("Failed to open public key file");
        return 1;
    }
    if (sm2_public_key_info_to_pem(&sm2_key, fp) != 1) {
        fprintf(stderr, "Failed to write public key to file.\n");
        fclose(fp);
        return 1;
    }
    fclose(fp);

    printf("SM2 key pair generated successfully.\n");
    return 0;
}
```

- sm2加解密
  - 过程与验证

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# rm
sm2_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# nano
sm2_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# gcc -o
../bin/sm2_encrypt sm2_encrypt.c -lgmssl -lssl -lcrypto
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# tree
.
├── bin
│   ├── sm2_decrypt
│   ├── sm2_encrypt
│   └── sm2_keygen
├── private_key.pem
├── public_key.pem
├── src
│   ├── sm2_decrypt.c
│   ├── sm2_encrypt.c
│   └── sm2_keygen.c
└── test
    ├── decrypt.bin
    ├── encrypt.bin
    └── plain.txt

3 directories, 11 files
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2#
./bin/sm2_encrypt public_key.pem ./test/plain.txt
./test/encrypt.bin
Encryption successful, output written to ./test/encrypt.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# cd src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# nano
sm2_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2#
./bin/sm2_decrypt private_key.pem ./test/encrypt.bin
./test/decrypt.bin
Invalid SM2 key length: must be 64 hex characters.
Invalid SM2 key.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# cd src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# ls
sm2_decrypt.c  sm2_encrypt.c  sm2_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# rm
sm2_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# nano
sm2_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# gcc -o
../bin/sm2_decrypt sm2_decrypt.c -lgmssl -lssl -lcryp
to
```

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2#
./bin/sm2_decrypt private_key.pem ./test/encrypt.bin
./test/decrypt.bin
Decryption successful, output written to ./test/decrypt.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# cat
./test/decrypt.bin
20221414xlm
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# git commit
-m "finish gmssl sm2 encrypt & decrypt"
[master 8ccde69] finish gmssl sm2 encrypt & decrypt
11 files changed, 231 insertions(+), 94 deletions(-)
create mode 100755 shiyan1-2/task02/test_sm2/bin/sm2_decrypt
create mode 100755 shiyan1-2/task02/test_sm2/bin/sm2_encrypt
rewrite shiyan1-2/task02/test_sm2/bin/sm2_keygen (60%)
create mode 100644 shiyan1-2/task02/test_sm2/src/sm2_decrypt.c
create mode 100644 shiyan1-2/task02/test_sm2/src/sm2_encrypt.c
rewrite shiyan1-2/task02/test_sm2/src/sm2_keygen.c (90%)
create mode 100644 shiyan1-2/task02/test_sm2/test/decrypt.bin
create mode 100644 shiyan1-2/task02/test_sm2/test/encrypt.bin
create mode 100644 shiyan1-2/task02/test_sm2/test/plain.txt
```

- 代码
  - 加密

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gmssl/sm2.h>
#include <gmssl/pem.h>

#define SM2_CIPHERTEXT_SIZE 1024  // 根据需要调整大小

void print_usage() {
    printf("Usage: sm2_encrypt <public_key.pem> <input_file>
<output_file>\n");
}

int main(int argc, char *argv[]) {
    if (argc != 4) {
        print_usage();
        return 1;
    }

    const char *public_key_file = argv[1];
    const char *input_file = argv[2];
    const char *output_file = argv[3];

    // Load public key
    SM2_KEY sm2_key;
```

```
    FILE *fp = fopen(public_key_file, "r");
    if (!fp) {
        perror("Failed to open public key file");
        return 1;
    }

    // Use the correct function to read the public key
    if (sm2_public_key_info_from_pem(&sm2_key, fp) != 1) {
        fprintf(stderr, "Failed to read public key from
PEM\n");
        fclose(fp);
        return 1;
    }
    fclose(fp);

    // Read input file
    FILE *in_fp = fopen(input_file, "rb");
    if (!in_fp) {
        perror("Failed to open input file");
        return 1;
    }

    fseek(in_fp, 0, SEEK_END);
    long input_len = ftell(in_fp);
    fseek(in_fp, 0, SEEK_SET);
    unsigned char *input_data = malloc(input_len);
    if (fread(input_data, 1, input_len, in_fp) != input_len)
{
        perror("Failed to read input file");
        free(input_data);
        fclose(in_fp);
        return 1;
    }
    fclose(in_fp);

    // Encrypt data
    unsigned char ciphertext[SM2_CIPHERTEXT_SIZE];
    size_t ciphertext_len = sizeof(ciphertext);

    if (sm2_encrypt(&sm2_key, input_data, input_len,
ciphertext, &ciphertext_len) != 1) {
        fprintf(stderr, "SM2 encryption failed\n");
        free(input_data);
        return 1;
    }

    free(input_data);

    // Write output file
    FILE *out_fp = fopen(output_file, "wb");
    if (!out_fp) {
        perror("Failed to open output file");
        return 1;
    }
```

```
    if (fwrite(ciphertext, 1, ciphertext_len, out_fp) !=
ciphertext_len) {
        perror("Failed to write output file");
        fclose(out_fp);
        return 1;
    }
    fclose(out_fp);

    printf("Encryption successful, output written to %s\n",
output_file);
    return 0;
}
```

- 解密

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gmssl/sm2.h>
#include <gmssl/pem.h>

#define SM2_CIPHERTEXT_SIZE 1024  // 根据需要调整大小

void print_usage() {
    printf("Usage: sm2_decrypt <private_key.pem> <input_file>
<output_file>\n");
}

int main(int argc, char *argv[]) {
    if (argc != 4) {
        print_usage();
        return 1;
    }

    const char *private_key_file = argv[1];
    const char *input_file = argv[2];
    const char *output_file = argv[3];

    // Load private key
    SM2_KEY sm2_key;
    FILE *fp = fopen(private_key_file, "r");
    if (!fp) {
        perror("Failed to open private key file");
        return 1;
    }

    // 使用 PEM 格式加载私钥
    if (sm2_private_key_info_from_pem(&sm2_key, fp) != 1) {
        fprintf(stderr, "Failed to read private key from
PEM\n");
        fclose(fp);
```

```
        return 1;
    }
    fclose(fp);

    // Read input file (ciphertext)
    FILE *in_fp = fopen(input_file, "rb");
    if (!in_fp) {
        perror("Failed to open input file");
        return 1;
    }

    fseek(in_fp, 0, SEEK_END);
    long input_len = ftell(in_fp);
    fseek(in_fp, 0, SEEK_SET);
    unsigned char *ciphertext = malloc(input_len);
    if (fread(ciphertext, 1, input_len, in_fp) != input_len)
{
        perror("Failed to read input file");
        free(ciphertext);
        fclose(in_fp);
        return 1;
    }
    fclose(in_fp);

    // 解密数据
    unsigned char decrypted[SM2_CIPHERTEXT_SIZE];  // 根据需要
调整大小
    size_t decrypted_len = sizeof(decrypted);

    if (sm2_decrypt(&sm2_key, ciphertext, input_len,
decrypted, &decrypted_len) != 1) {
        fprintf(stderr, "SM2 decryption failed\n");
        free(ciphertext);
        return 1;
    }

    free(ciphertext);

    // Write output file
    FILE *out_fp = fopen(output_file, "wb");
    if (!out_fp) {
        perror("Failed to open output file");
        return 1;
    }

    if (fwrite(decrypted, 1, decrypted_len, out_fp) !=
decrypted_len) {
        perror("Failed to write output file");
        fclose(out_fp);
        return 1;
    }
    fclose(out_fp);

    printf("Decryption successful, output written to %s\n",
```

```
output_file);
    return 0;
}
```

- sm2签名和验证
  - 过程与验证

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# rm
sm2_sign.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# nano
sm2_sign.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# gcc -o
../bin/sm2_sign sm2_sign.c -lgmssl -lssl -lcrypt
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src#
../bin/sm2_sign ../private_key.pem ../test/plain.txt
../test/signature.sig
/root/GmSSL/src/pem.c:100:pem_read():
/root/GmSSL/src/sm2_key.c:466:sm2_private_key_from_pem():
Error loading private key from PEM file: ../private_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src#
openssl pkey -in ../private_key.pem -out ../private_key_ec.pem -
outform PEM -traditional
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src#
../bin/sm2_sign ../private_key_ec.pem ../test/plain.txt ../t
est/signature.sig
Signature generated and saved to ../test/signature.sig
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# gmssl
sm2verify -pubkey ../public_key.pem -in ../test/plain.
txt -sig ../test/signature.sig
/root/GmSSL/src/sm2_sign.c:265:sm2_fast_verify():
/root/GmSSL/src/sm2_sign.c:671:sm2_verify_finish():
gmssl sm2verify: inner error
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# ls
bin  private_key.pem  private_key_ec.pem  public_key.pem  src
test
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# rm
private_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# mv
private_key_ec.pem private_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# cd src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src#
../bin/sm2_verify ../public_key.pem ../test/plain.txt
../test/signature.sig
Signature is valid.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# cd src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src#
../bin/sm2_sign ../private_key.pem ../test/plain.txt
../test/signature.sig
Signature generated and saved to ../test/signature.sig
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2/src#
```

```
../bin/sm2_verify ../public_key.pem ../test/plain.txt
../test/signature.sig
Signature is valid.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm2# git commit
-m "finish sm2 sign & verify by PKCS#8 key"
[master de0b95b] finish sm2 sign & verify by PKCS#8 key
10 files changed, 249 insertions(+), 205 deletions(-)
create mode 100755 shiyan1-2/task02/test_sm2/bin/sm2_sign
create mode 100755 shiyan1-2/task02/test_sm2/bin/sm2_verify
create mode 100644 shiyan1-2/task02/test_sm2/src/sm2_keygen_pem.c
rewrite shiyan1-2/task02/test_sm2/src/sm2_sign.c (73%)
rewrite shiyan1-2/task02/test_sm2/src/sm2_verify.c (85%)
create mode 100644 shiyan1-2/task02/test_sm2/test/signature.sig
```

- 代码
  - 签名

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gmssl/sm2.h>
#include <gmssl/pem.h>
#include <gmssl/error.h>
#include <gmssl/sm3.h>

int main(int argc, char **argv)
{
    if (argc != 4) {
        printf("Usage: %s <private_key.pem> <input.txt>
<signature.sig>\n", argv[0]);
        return 1;
    }

    const char *private_key_file = argv[1];
    const char *input_file = argv[2];
    const char *signature_file = argv[3];

    SM2_KEY sm2_key;
    FILE *key_fp = NULL;
    FILE *input_fp = NULL;
    FILE *sig_fp = NULL;
    unsigned char dgst[32];
    unsigned char sig[SM2_MAX_SIGNATURE_SIZE];
    size_t siglen;
    unsigned char buffer[1024];
    size_t len;

    // 加载私钥
    key_fp = fopen(private_key_file, "r");
    if (!key_fp) {
```

```
        perror("Failed to open private key file");
        return 1;
    }

    // 使用 PEM 格式加载私钥
    if (sm2_private_key_info_from_pem(&sm2_key, key_fp) != 1)
{
        fprintf(stderr, "Failed to read private key from
PEM\n");
        fclose(key_fp);
        return 1;
    }
    fclose(key_fp);

    // 读取输入文件并计算SM3哈希
    input_fp = fopen(input_file, "r");
    if (!input_fp) {
        fprintf(stderr, "Error opening input file: %s\n",
input_file);
        return 1;
    }

    SM3_CTX sm3_ctx;
    sm3_init(&sm3_ctx);

    while ((len = fread(buffer, 1, sizeof(buffer), input_fp))
> 0) {
        sm3_update(&sm3_ctx, buffer, len);
    }
    fclose(input_fp);

    sm3_finish(&sm3_ctx, dgst);

    // 生成签名
    siglen = sizeof(sig);  // 确保我们为签名的大小传递正确的变量
    if (sm2_sign(&sm2_key, dgst, sig, &siglen) != 1) {
        fprintf(stderr, "Error generating SM2 signature\n");
        return 1;
    }

    // 写入签名到文件
    sig_fp = fopen(signature_file, "wb");
    if (!sig_fp) {
        fprintf(stderr, "Error opening signature file: %s\n",
signature_file);
        return 1;
    }

    if (fwrite(sig, 1, siglen, sig_fp) != siglen) {
        fprintf(stderr, "Error writing signature to file:
%s\n", signature_file);
        fclose(sig_fp);
        return 1;
    }
```

```
    fclose(sig_fp);

    printf("Signature generated and saved to %s\n",
signature_file);
    return 0;
}
```

- 验证

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gmssl/sm2.h>
#include <gmssl/sm3.h>
#include <gmssl/pem.h>
#include <gmssl/error.h>

int main(int argc, char **argv) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <public key PEM file>
<input file> <signature file>\n", argv[0]);
        return 1;
    }

    const char *pubkey_filename = argv[1];
    const char *input_filename = argv[2];
    const char *signature_filename = argv[3];

    SM2_KEY key;
    FILE *fp;
    SM3_CTX sm3_ctx;
    uint8_t dgst[32]; // SM3摘要结果大小

    // 读取公钥
    if (!(fp = fopen(pubkey_filename, "r"))) {
        perror("Error opening public key file"); // 使用
perror 打印系统错误信息
        return 1;
    }
    if (!sm2_public_key_info_from_pem(&key, fp)) {
        fprintf(stderr, "Error reading public key from PEM
file %s\n", pubkey_filename);
        fclose(fp);
        return 1;
    }
    fclose(fp);

    // 读取输入文件并计算哈希
    if (!(fp = fopen(input_filename, "rb"))) {
        perror("Error opening input file"); // 使用 perror 打
印系统错误信息
        return 1;
```

```
    }

    sm3_init(&sm3_ctx);
    unsigned char buf[1024];
    size_t len;
    while ((len = fread(buf, 1, sizeof(buf), fp)) > 0) {
        sm3_update(&sm3_ctx, buf, len);
    }
    if (ferror(fp)) {
        fprintf(stderr, "Error reading input file %s\n",
input_filename);
        fclose(fp);
        return 1;
    }
    sm3_finish(&sm3_ctx, dgst);
    fclose(fp);

    // 读取签名
    unsigned char signature[256];
    size_t siglen;
    if (!(fp = fopen(signature_filename, "rb"))) {
        perror("Error opening signature file"); // 使用
perror 打印系统错误信息
        return 1;
    }
    siglen = fread(signature, 1, sizeof(signature), fp);
    if (ferror(fp)) {
        fprintf(stderr, "Error reading signature file %s\n",
signature_filename);
        fclose(fp);
        return 1;
    }
    fclose(fp);

    // 确保读取的签名长度有效
    if (siglen == 0) {
        fprintf(stderr, "Signature file %s is empty or could
not be read.\n", signature_filename);
        return 1;
    }

    // 验证签名
    int verify_result = sm2_verify(&key, dgst, signature,
siglen);
    if (verify_result == 1) {
        printf("Signature is valid.\n");
    } else {
        fprintf(stderr, "Signature is invalid. Please check
the public key, input file, and signature file.\n");
        return 1;
    }

    return 0;
}
```

- sm3摘要计算和HMAC
  - 过程与验证

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# mkdir test_sm3
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# cd test_sm3
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3# mkdir bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3# mkdir src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3# mkdir test
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3# cd src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# nano
sm3_digest.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# gcc -o
../bin/sm3_digest sm3_digest.c -lgmssl
sm3_digest.c: In function ‘main’:
sm3_digest.c:30:26: error: ‘SM3_DIGEST_LENGTH’ undeclared (first use in
this function); did you mean ‘SM3_DIGEST_CTX’?
30 |     unsigned char digest[SM3_DIGEST_LENGTH];
   |                          ^~~~~~~~~~~~~~~~~
   |                          SM3_DIGEST_CTX
sm3_digest.c:30:26: note: each undeclared identifier is reported only
once for each function it appears in
sm3_digest.c:48:5: warning: implicit declaration of function
‘sm3_final’; did you mean ‘sm3_finish’? [-Wimplicit-function-
declaration]
48 |     sm3_final(digest, &ctx);
   |     ^~~~~~~~~
   |     sm3_finish
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# ls
sm3_digest.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# rm
sm3_digest.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# nano
sm3_digest.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# gcc -o
../bin/sm3_digest sm3_digest.c -lgmssl
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3# cd test
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/test# echo
"20221414XLM" > input.txt
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/test# touch
output.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/test# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3# cd src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src#
../bin/sm3_digest ../test/input.txt ../test/output.bin
SM3 digest computed and written to ../test/output.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# cat
../test/output.bin
���*H
~ʕ�<w��Do�n%��{��I��jroot@Youer:~/shiyan/shiyan01/shiyan1-
```

```
2/task02/test_sm3/src#
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# rm
sm3_hmac.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# nano
sm3_hmac.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# gcc -o
../bin/sm3_hmac sm3_hmac.c -lgmssl
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src#
../bin/sm3_hmac ../test/hmac.bin ../test/input.txt ../test/output.txt
gmssl: illegal option 'dgst'
usage: gmssl command [options]
command -help

Commands:
help               Print this help message
version            Print version
rand               Generate random bytes
sm2keygen          Generate SM2 keypair
sm2sign            Generate SM2 signature
sm2verify          Verify SM2 signature
sm2encrypt         Encrypt with SM2 public key
sm2decrypt         Decrypt with SM2 private key
sm3                Generate SM3 hash
sm3hmac            Generate SM3 HMAC tag
sm3_pbkdf2         Hash password into key using PBKDF2 algoritm
sm3xmss_keygen     Generate SM3-XMSS keypair
sm4_ecb            Encrypt or decrypt with SM4 ECB
sm4_cbc            Encrypt or decrypt with SM4 CBC
sm4_ctr            Encrypt or decrypt with SM4 CTR
sm4_cfb            Encrypt or decrypt with SM4 CFB
sm4_ofb            Encrypt or decrypt with SM4 OFB
sm4_ccm            Encrypt or decrypt with SM4 CCM
sm4_gcm            Encrypt or decrypt with SM4 GCM
sm4_xts            Encrypt or decrypt with SM4 XTS
sm4_cbc_sm3_hmac   Encrypt or decrypt with SM4 CBC with SM3-HMAC
sm4_ctr_sm3_hmac   Encrypt or decrypt with SM4 CTR with SM3-HMAC
sm4_cbc_mac        Generate SM4 CBC-MAC
ghash              Generate GHASH
zuc                Encrypt or decrypt with ZUC
sm9setup           Generate SM9 master secret
sm9keygen          Generate SM9 private key
sm9sign            Generate SM9 signature
sm9verify          Verify SM9 signature
sm9encrypt         SM9 public key encryption
sm9decrypt         SM9 decryption
reqgen             Generate certificate signing request (CSR)
reqsign            Generate certificate from CSR
reqparse           Parse and print a CSR
crlget             Download the CRL of given certificate
crlgen             Sign a CRL with CA certificate and private key
crlverify          Verify a CRL with issuer's certificate
crlparse           Parse and print CRL
certgen            Generate a self-signed certificate
certparse          Parse and print certificates
```

```
certverify        Verify certificate chain
certrevoke        Revoke certificate and output RevokedCertificate
record
cmsparse          Parse CMS (cryptographic message syntax) file
cmsencrypt        Generate CMS EnvelopedData
cmsdecrypt        Decrypt CMS EnvelopedData
cmssign           Generate CMS SignedData
cmsverify         Verify CMS SignedData
sdfinfo           Print SDF device info
sdfdigest         Generate SM3 hash with SDF device
sdfexport         Export SM2 signing public key from SDF device
sdfsign           Generate SM2 signature with SDF internal private key
sdfencrypt        SM2/SM4-CBC hybrid encryption with SDF device
sdfdecrypt        SM2/SM4-CBC hybrid decryption with SDF device
sdftest           Test vendor's SDF library and device
tlcp_client       TLCP client
tlcp_server       TLCP server
tls12_client      TLS 1.2 client
tls12_server      TLS 1.2 server
tls13_client      TLS 1.3 client
tls13_server      TLS 1.3 server

run `gmssl <command> -help` to print help of the given command


Failed to read HMAC output: Success
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# gmssl
sm3hmac -help
usage: sm3hmac -key hex [-in file | -in_str str] [-bin|-hex] [-out
file]
Options

    -key hex               Hex string of the MAC key
    -in_str str            Input as text string
    -in file | stdin       Input file path
                          `-in_str` and `-in` should not be used together
                          If neither `-in` nor `-in_str` specified, read
from stdin
    -hex                   Output MAC-tag as hex string (by default)
    -bin                   Output MAC-tag as binary
                          `-hex` and `-bin` should not be used together
    -out file | stdout     Output file path. If not specified, output
to stdout

Examples

    KEY_HEX=`gmssl rand -outlen 16 -hex`
    gmssl sm3hmac -key $KEY_HEX -in_str abc

    gmssl sm3hmac -key $KEY_HEX -in_str abc -bin

    gmssl sm3hmac -key $KEY_HEX -in /path/to/file

When reading from stdin, make sure the trailing newline character is
```

```
removed

Linux/Mac:
    echo -n abc | gmssl sm3hmac -key $KEY_HEX

Windows:
    C:\> echo |set/p="abc" | gmssl sm3hmac -key
112233445566778811223344556677 88
```

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# rm
sm3_hmac.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# gcc -o
../bin/sm3_hmac sm3_hmac.c -lgmssl
cc1: fatal error: sm3_hmac.c: No such file or directory
compilation terminated.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# nano
sm3_hmac.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# gcc -o
../bin/sm3_hmac sm3_hmac.c -lgmssl
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src#
../bin/sm3_hmac  "key01" "20221414xlm"
/root/GmSSL/src/hex.c:108:hex2bin(): hex key01 len = 5
sm3hmac: invalid HEX digits
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src#
../bin/sm3_hmac  "1234567890abcdef1234567890abcdef" "20221414xlm"
HMAC-SM3 result:
ffdb72738ab2e33c7703a983cf79f11c29375893bee747f1056fa80396dee307
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src#
../bin/sm3_hmac  "0123456789abcdef1234567890abcdef" "2022141
4xlm"
HMAC-SM3 result:
f9d5646e7b3d76de7f4d7d524e81ca41d4181da01b1eb0d3d0c02ed7fd3e4db7
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3/src# git commit
-m "finish gmssl hash & hmac by code"
[master ea58d11] finish gmssl hash & hmac by code
2 files changed, 112 insertions(+)
create mode 100644 shiyan1-2/task02/test_sm3/src/sm3_digest.c
create mode 100644 shiyan1-2/task02/test_sm3/src/sm3_hmac.c
```

- 代码
  - 摘要计算

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gmssl/sm3.h>

#define SM3_DIGEST_LENGTH 32 // SM3 摘要长度为 32 字节
```

```
void print_usage() {
    fprintf(stderr, "Usage: sm3_digest <input_file>
<output_file>\n");
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        print_usage();
        return EXIT_FAILURE;
    }

    const char *input_file = argv[1];
    const char *output_file = argv[2];

    // 打开输入文件
    FILE *fp_input = fopen(input_file, "rb");
    if (!fp_input) {
        perror("Error opening input file");
        return EXIT_FAILURE;
    }

    // 读取输入文件内容
    unsigned char buffer[1024];
    size_t bytes_read;
    SM3_CTX ctx;
    unsigned char digest[SM3_DIGEST_LENGTH];

    // 初始化 SM3 上下文
    sm3_init(&ctx);

    // 逐块读取文件并更新摘要
    while ((bytes_read = fread(buffer, 1, sizeof(buffer),
fp_input)) > 0) {
        sm3_update(&ctx, buffer, bytes_read);
    }

    // 检查读取文件时是否发生错误
    if (ferror(fp_input)) {
        perror("Error reading input file");
        fclose(fp_input);
        return EXIT_FAILURE;
    }

    // 完成摘要计算
    sm3_finish(&ctx, digest); // 使用 sm3_finish 替代 sm3_final
    fclose(fp_input);

    // 打开输出文件
    FILE *fp_output = fopen(output_file, "wb");
    if (!fp_output) {
        perror("Error opening output file");
        return EXIT_FAILURE;
    }
```

```
    // 将摘要写入输出文件
    if (fwrite(digest, 1, sizeof(digest), fp_output) !=
sizeof(digest)) {
        perror("Error writing output file");
        fclose(fp_output);
        return EXIT_FAILURE;
    }

    fclose(fp_output);

    printf("SM3 digest computed and written to %s\n",
output_file);
    return EXIT_SUCCESS;
}
```

- HAMC实现(这里是通过用代码构建并执行命令的方式实现功能的，后面有用python的代码实现)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define HMAC_COMMAND_FORMAT "gmssl sm3hmac -key %s -in_str %s"

int main(int argc, char *argv[]) {
    if (argc != 3) {
        fprintf(stderr, "Usage: %s <key_hex> <input_string>\n",
argv[0]);
        return EXIT_FAILURE;
    }

    const char *key_hex = argv[1];    // 密钥的十六进制表示
    const char *input_string = argv[2]; // 输入字符串

    // 构建命令
    char command[512]; // 确保命令缓冲区足够大
    snprintf(command, sizeof(command), HMAC_COMMAND_FORMAT,
key_hex, input_string);

    // 执行命令并获取结果
    FILE *fp = popen(command, "r");
    if (fp == NULL) {
        perror("popen failed");
        return EXIT_FAILURE;
    }

    // 读取并打印标准输出
    char buffer[128]; // 用于读取输出
    printf("HMAC-SM3 result: ");
    while (fgets(buffer, sizeof(buffer), fp) != NULL) {
        printf("%s", buffer); // 打印输出，供调试使用
```

```
    }

    // 关闭命令输出流
    if (pclose(fp) == -1) {
        perror("pclose failed");
        return EXIT_FAILURE;
    }

    return EXIT_SUCCESS;
}
```

- sm4加解密
  - sm4密钥生成
    - 过程

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# mkdir test_sm4
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# cd test_sm4
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4# mkdir sin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4# rm -r sin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4# mkdir src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4# mkdir bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4# mkdir bin
mkdir: cannot create directory ‘bin’: File exists
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4# mkdir test
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4# cd src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# nano
sm4_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# gcc -o
../bin/sm4_keygen sm4_keygen.c -lgmssl
sm4_keygen.c: In function ‘generate_key’:
sm4_keygen.c:12:10: warning: implicit declaration of function
‘RAND_bytes’ [-Wimplicit-function-declaration]
12 |     if (!RAND_bytes(key, sizeof(key))) {
   |          ^~~~~~~~~~
/usr/bin/ld: /tmp/ccmkJaHx.o: in function `generate_key':
sm4_keygen.c:(.text+0x31): undefined reference to `RAND_bytes'
collect2: error: ld returned 1 exit status
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# rm
sm4_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# nano
sm4_keygen.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# gcc -o
../bin/sm4_keygen sm4_keygen.c -lgmssl
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src#
../bin/sm4_keygen
Usage: ../bin/sm4_keygen <output_filename>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src#
../bin/sm4_keygen ../test/sm4_key.pem
Key generated and saved to ../test/sm4_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# rm
```

```
../test/sm4_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src#
../bin/sm4_keygen ../test/sm4_key.bin
Key generated and saved to ../test/sm4_key.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# cat
../test/sm4_key.bin
���T��u�[�B�`�root@Youer:~/shiyan/shiyan01/shiyan1-
2/task02/test_sm4/src#
```

- 代码

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gmssl/sm4.h>
#include <gmssl/rand.h> // 引入 GmSSL 的随机数生成库

void generate_key(const char *filename) {
    // 定义 SM4 密钥
    unsigned char key[16];  // SM4 密钥长度为 16 字节
    FILE *file;

    // 生成随机密钥
    if (rand_bytes(key, sizeof(key)) != 1) { // 使用 GmSSL 提供的
rand_bytes 函数
        fprintf(stderr, "Error generating random bytes for the
key.\n");
        exit(EXIT_FAILURE);
    }

    // 打开文件以写入密钥
    file = fopen(filename, "wb");
    if (!file) {
        fprintf(stderr, "Error opening file %s for writing.\n",
filename);
        exit(EXIT_FAILURE);
    }

    // 写入密钥到文件
    size_t written = fwrite(key, sizeof(unsigned char),
sizeof(key), file);
    if (written != sizeof(key)) {
        fprintf(stderr, "Error writing key to file.\n");
        fclose(file);
        exit(EXIT_FAILURE);
    }

    fclose(file);
    printf("Key generated and saved to %s\n", filename);
}

int main(int argc, char *argv[]) {
```

```
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <output_filename>\n", argv[0]);
        return EXIT_FAILURE;
    }

    generate_key(argv[1]);
    return EXIT_SUCCESS;
}
```

- sm4加密和解密
  - 过程和验证

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# rm
sm4_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# nano
sm4_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# gcc
sm4_encrypt.c -o ../bin/sm4_encrypt -lgmssl
sm4_encrypt.c:7: warning: "SM4_BLOCK_SIZE" redefined
    7 | #define SM4_BLOCK_SIZE 16
      |
In file included from sm4_encrypt.c:5:
/usr/local/include/gmssl/sm4.h:24: note: this is the location of
the previous definition
24 | #define SM4_BLOCK_SIZE          (16)
      |
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# rm
sm4_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# nano
sm4_encrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# gcc
sm4_encrypt.c -o ../bin/sm4_encrypt -lgmssl
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src#
../bin/sm4_encrypt
Usage: ../bin/sm4_encrypt <key_file> <input_file> <output_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# ls
../test
sm4_key.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# echo
"20221414Xlm" > ../test/input.txt
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src#
../bin/sm4_encrypt ../test/sm4_key.bin ../test/input.txt
Usage: ../bin/sm4_encrypt <key_file> <input_file> <output_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# touch
output.bin ../test/
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src#
../bin/sm4_encrypt ../test/sm4_key.bin ../test/input.txt ../
test/output.txt
Encryption completed successfully.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# cat
input.txt
```

```
cat: input.txt: No such file or directory
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# cat
../test/input.txt
20221414Xlm
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# cat
../test/output.txt
4a1&N?m?root@Youer:~/shiyan/shiyan01/shiyan1-
2/task02/test_sm4/src# nano sm4_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# gcc
sm4_decrypt.c -o ../bin/sm4_decrypt -lgmssl
sm4_decrypt.c:7: warning: "SM4_BLOCK_SIZE" redefined
    7 | #define SM4_BLOCK_SIZE 16
      |
In file included from sm4_decrypt.c:5:
/usr/local/include/gmssl/sm4.h:24: note: this is the location of
the previous definition
24 | #define SM4_BLOCK_SIZE          (16)
      |
sm4_decrypt.c: In function ‘sm4_decrypt_file’:
sm4_decrypt.c:64:9: warning: implicit declaration of function
‘sm4_decrypt’; did you mean ‘sm4_encrypt’? [-Wimplicit-function-
declaration]
64 |         sm4_decrypt(&sm4_key, input, output);
      |         ^~~~~~~~~~
      |         sm4_encrypt
/usr/bin/ld: /tmp/ccd3Ysw9.o: in function `sm4_decrypt_file':
sm4_decrypt.c:(.text+0x1eb): undefined reference to `sm4_decrypt'
collect2: error: ld returned 1 exit status
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# cat
../test/output.txt
4a1&N?m?
        root@Youer:~/shiyan/shiyan01/shiyan1-
2/task02/test_sm4/src# rm sm4_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# nano
sm4_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# gcc
sm4_decrypt.c -o ../bin/sm4_decrypt -lgmssl
sm4_decrypt.c:7: warning: "SM4_BLOCK_SIZE" redefined
    7 | #define SM4_BLOCK_SIZE 16
      |
In file included from sm4_decrypt.c:5:
/usr/local/include/gmssl/sm4.h:24: note: this is the location of
the previous definition
24 | #define SM4_BLOCK_SIZE          (16)
      |
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# rm
sm4_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# nano
sm4_decrypt.c
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# gcc
sm4_decrypt.c -o ../bin/sm4_decrypt -lgmssl
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src#
../bin/sm4_decrypt
Usage: ../bin/sm4_decrypt <key_file> <input_file> <output_file>
```

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# ls
../test
input.txt  output.txt  sm4_key.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# mv
../test/output.txt en_output.txt ../test
mv: '../test/output.txt' and '../test/output.txt' are the same
file
mv: cannot stat 'en_output.txt': No such file or directory
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4# cd test
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/test# ls
input.txt  output.txt  sm4_key.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/test# mv
output.txt en_output.txt
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/test# touch
de_output.txt
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/test# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4# cd src
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src#
../bin/sm4_decrypt ../test/sm4_key.bin
../test/en_output.txt../test/de_output.txt
Usage: ../bin/sm4_decrypt <key_file> <input_file> <output_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src#
../bin/sm4_decrypt ../test/sm4_key.bin
../test/en_output.txt../test/de_output.txt
Usage: ../bin/sm4_decrypt <key_file> <input_file> <output_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src#
../bin/sm4_decrypt ../test/sm4_key.bin ../test/en_output.txt
../test/de_output.txt
Decryption completed successfully.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# cat
../test/de_output.txt
20221414Xlm

root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# git
add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm4/src# git
commit -m "finish gmssl sm4 ken_gen & decrypt & encrypt"
[master ba133d6] finish gmssl sm4 ken_gen & decrypt & encrypt
3 files changed, 202 insertions(+)
create mode 100644 shiyan1-2/task02/test_sm4/src/sm4_decrypt.c
create mode 100644 shiyan1-2/task02/test_sm4/src/sm4_encrypt.c
create mode 100644 shiyan1-2/task02/test_sm4/src/sm4_keygen.c
```

- 代码
  - 加密代码

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
```

```c
#include "gmssl/sm4.h"

// 使用 GmSSL 中定义的 SM4_BLOCK_SIZE
// #define SM4_BLOCK_SIZE 16 // 删除自定义的定义

// 函数声明
void handleErrors(const char *message);
void sm4_encrypt_file(const char *key_file, const char
*input_file, const char *output_file);

int main(int argc, char *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <key_file> <input_file>
<output_file>\n", argv[0]);
        return EXIT_FAILURE;
    }

    sm4_encrypt_file(argv[1], argv[2], argv[3]);

    return EXIT_SUCCESS;
}

void sm4_encrypt_file(const char *key_file, const char
*input_file, const char *output_file) {
    FILE *kf, *inf, *outf;
    SM4_KEY sm4_key;
    uint8_t key[SM4_KEY_SIZE];
    uint8_t input[SM4_BLOCK_SIZE];
    uint8_t output[SM4_BLOCK_SIZE];
    size_t bytes_read;

    // 打开密钥文件
    kf = fopen(key_file, "rb");
    if (!kf) {
        handleErrors("Failed to open key file");
    }

    // 读取密钥
    if (fread(key, 1, SM4_KEY_SIZE, kf) != SM4_KEY_SIZE) {
        fclose(kf);
        handleErrors("Failed to read key from file");
    }
    fclose(kf);

    // 设置 SM4 加密密钥
    sm4_set_encrypt_key(&sm4_key, key);

    // 打开输入文件
    inf = fopen(input_file, "rb");
    if (!inf) {
        handleErrors("Failed to open input file");
    }

    // 打开输出文件
```

```
    outf = fopen(output_file, "wb");
    if (!outf) {
        fclose(inf);
        handleErrors("Failed to open output file");
    }

    // 逐块读取输入文件并进行加密
    while ((bytes_read = fread(input, 1, SM4_BLOCK_SIZE,
inf)) > 0) {
        // 如果读取的块不足一个完整的块，进行填充
        if (bytes_read < SM4_BLOCK_SIZE) {
            memset(input + bytes_read, 0, SM4_BLOCK_SIZE -
bytes_read); // 填充零
        }
        sm4_encrypt(&sm4_key, input, output);
        fwrite(output, 1, SM4_BLOCK_SIZE, outf);
    }

    fclose(inf);
    fclose(outf);
    printf("Encryption completed successfully.\n");
}

void handleErrors(const char *message) {
    fprintf(stderr, "Error: %s\n", message);
    exit(EXIT_FAILURE);
}
```

- 解密代码

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "gmssl/sm4.h"

// 不再重新定义 SM4_BLOCK_SIZE
// #define SM4_BLOCK_SIZE 16

// 函数声明
void handleErrors(const char *message);
void sm4_decrypt_file(const char *key_file, const char
*input_file, const char *output_file);

int main(int argc, char *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <key_file> <input_file>
<output_file>\n", argv[0]);
        return EXIT_FAILURE;
    }

    sm4_decrypt_file(argv[1], argv[2], argv[3]);
```

```
    return EXIT_SUCCESS;
}

void sm4_decrypt_file(const char *key_file, const char
*input_file, const char *output_file) {
    FILE *kf, *inf, *outf;
    SM4_KEY sm4_key;
    uint8_t key[SM4_KEY_SIZE];
    uint8_t input[SM4_BLOCK_SIZE];
    uint8_t output[SM4_BLOCK_SIZE];
    size_t bytes_read;

    // 打开密钥文件
    kf = fopen(key_file, "rb");
    if (!kf) {
        handleErrors("Failed to open key file");
    }

    // 读取密钥
    if (fread(key, 1, SM4_KEY_SIZE, kf) != SM4_KEY_SIZE) {
        fclose(kf);
        handleErrors("Failed to read key from file");
    }
    fclose(kf);

    // 设置 SM4 解密密钥
    sm4_set_decrypt_key(&sm4_key, key);

    // 打开输入文件
    inf = fopen(input_file, "rb");
    if (!inf) {
        handleErrors("Failed to open input file");
    }

    // 打开输出文件
    outf = fopen(output_file, "wb");
    if (!outf) {
        fclose(inf);
        handleErrors("Failed to open output file");
    }

    // 逐块读取输入文件并进行解密
    while ((bytes_read = fread(input, 1, SM4_BLOCK_SIZE,
inf)) > 0) {
        // 调用 sm4_encrypt 进行解密
        sm4_encrypt(&sm4_key, input, output);
        fwrite(output, 1, SM4_BLOCK_SIZE, outf);
    }

    fclose(inf);
    fclose(outf);
    printf("Decryption completed successfully.\n");
}
```

```
void handleErrors(const char *message) {
    fprintf(stderr, "Error: %s\n", message);
    exit(EXIT_FAILURE);
}
```

- 通过python代码实现sm3的HMAC功能
  - 过程与验证

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3# python3
sm3_hmac_keygen.py ./test/sm3_hmac_key_01.bin
python3: can't open file '/root/shiyan/shiyan01/shiyan1-
2/task02/test_sm3/sm3_hmac_keygen.py': [Errno 2] No such file or
directory
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3# python3
./src/sm3_hmac_keygen.py ./test/sm3_hmac_key_01.bin
Generated HMAC key saved to ./test/sm3_hmac_key_01.bin. Key:
f8faf5b184b0438318d2ad5a0cd2ece72133abae7c5f5d1341624107b3e268c1
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3# python3
./src/sm3_hmac_keygen.py ./test/sm3_hmac_key_02.bin
Generated HMAC key saved to ./test/sm3_hmac_key_02.bin. Key:
02802bd5298c7d529eafdbe4e0320e8a67fd6ee5ff3174516c5001d1f3f9d37d
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3# python3
./src/hmac_sm3.py ./test/input.txt ./test/sm3_hmac_key_01.bin
HMAC-SM3计算结果:
54312aba6b190cd8a56f048ddca0164bb0d0692d5b6070ab2b19607f5d910e27
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02/test_sm3# python3
./src/hmac_sm3.py ./test/input.txt ./test/sm3_hmac_key_02.bin
HMAC-SM3计算结果:
0400741f7685030808e3e6e6ba227ad6a8a16c12255ef6e72acd6317ad0e6a1c
```

  - 代码
    - sm3的Hmac密钥生成代码

```
import sys
import secrets
import binascii
from gmssl import sm3, func

def generate_hmac_key(output_file):
    key = secrets.token_bytes(32)
    with open(output_file, 'wb') as f:
        f.write(key)
    return key

def main():
    if len(sys.argv)!= 2:
        print("Usage: python key_generator.py <output_file>")
```

```
        return
    output_file = sys.argv[1]
    key = generate_hmac_key(output_file)
    print(f"Generated HMAC key saved to {output_file}. Key:
{binascii.hexlify(key).decode()}")

if __name__ == "__main__":
    main()
```

- sm3的Hmac摘要计算代码

```
import sys
import secrets
import binascii
from gmssl import sm3, func

def generate_hmac_key(output_file):
    key = secrets.token_bytes(32)
    with open(output_file, 'wb') as f:
        f.write(key)
    return key

def main():
    if len(sys.argv)!= 2:
        print("Usage: python key_generator.py <output_file>")
        return
    output_file = sys.argv[1]
    key = generate_hmac_key(output_file)
    print(f"Generated HMAC key saved to {output_file}. Key:
{binascii.hexlify(key).decode()}")

if __name__ == "__main__":
    main()
```

3. 两人一组，在 Ubuntu或openEuler中（推荐 openEuler）中使用OpenSSL编程实现带签名的数字信封协议。使用OpenSSL库时，Alice发送，Bob接收。Ailice，Bob在实验中要替换为自己的8位学号+姓名。 使用Markdown记录详细记录实践过程，每完成一项git commit 一次。（5分）

- Alice,Bob生成自己的公私钥匙对，记作：（PKa，SKa），（PKb，SKb），Alice,Bob分别拥有：（PKa，SKa，PKb），（PKb，SKb，PKa），实验中把公钥文件拷贝给对方
- Alice发给Bob的明文plain.txt，内容为自己的姓名学号
- Alice：sm4 key使用gmssl rand 产生，16字节，记作k
- Alice：Sm4Enc(k,P) = C
- Alice：Sm2Enc(PKb,k) = KC
- Alice：Sm2Sign（SKa，C）= S1
- Alice： 数字信封 C||KC||S1 发给Bob
- Bob：Sm2Very（PKa，S1）
- Bob：Sm2Dec（SKb，KC）= k
- Bob：Sm4Dec（k，C）= P

(我们先做完gmssl，再做完openssl)

- 将可执行文件复制到task03文件夹

```
root@Youer:~/shiyan/shiyan01/shiyan1-2# ls
task01  task02
root@Youer:~/shiyan/shiyan01/shiyan1-2# mkdir task03
root@Youer:~/shiyan/shiyan01/shiyan1-2# cd task03
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# mkdir bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2# cp -r ./task01/bin ./task03
root@Youer:~/shiyan/shiyan01/shiyan1-2# tree
.
├── task01
│   ├── bin
│   │   ├── sm2_decrypt
│   │   ├── sm2_encrypt
│   │   ├── sm2_keygen
│   │   ├── sm2_sign
│   │   ├── sm2_verify
│   │   ├── sm3_hash
│   │   ├── sm3_hmac
│   │   ├── sm4_decrypt
│   │   ├── sm4_encrypt
│   │   ├── sm4_iv_gen
│   │   └── sm4_key_gen
│   ├── test_sm2
│   │   ├── decrypted_file.bin
│   │   ├── encrypted_file.bin
│   │   ├── plain.txt
│   │   ├── signature.bin
│   │   ├── sm2_decrypt
│   │   ├── sm2_decrypt.c
│   │   ├── sm2_encrypt
│   │   ├── sm2_encrypt.c
│   │   ├── sm2_keygen
│   │   ├── sm2_keygen.c
│   │   ├── sm2_private.pem
│   │   ├── sm2_private_key.pem
│   │   ├── sm2_public.pem
│   │   ├── sm2_public_key.pem
│   │   ├── sm2_sign
│   │   ├── sm2_sign.c
│   │   ├── sm2_verify
│   │   └── sm2_verify.c
│   ├── test_sm3
│   │   ├── sm3_hash
│   │   ├── sm3_hash.c
│   │   ├── sm3_hmac
│   │   └── sm3_hmac.c
│   └── test_sm4
│       ├── decrypted_file.txt
│       ├── encrypted_file.txt
```

```
│       ├── my_sm4_iv.bin
│       ├── my_sm4_key.bin
│       ├── plain.txt
│       ├── sm4_decrypt
│       ├── sm4_decrypt.c
│       ├── sm4_encrypt
│       ├── sm4_encrypt.c
│       ├── sm4_iv_gen
│       ├── sm4_iv_gen.c
│       ├── sm4_key_gen
│       └── sm4_key_gen.c
├── task02
│   ├── test_sm2
│   │   ├── bin
│   │   │   ├── sm2_decrypt
│   │   │   ├── sm2_encrypt
│   │   │   ├── sm2_keygen
│   │   │   ├── sm2_sign
│   │   │   └── sm2_verify
│   │   ├── private_key.pem
│   │   ├── public_key.pem
│   │   ├── src
│   │   │   ├── sm2_decrypt.c
│   │   │   ├── sm2_encrypt.c
│   │   │   ├── sm2_keygen.c
│   │   │   ├── sm2_sign.c
│   │   │   └── sm2_verify.c
│   │   └── test
│   │       ├── decrypt.bin
│   │       ├── encrypt.bin
│   │       ├── plain.txt
│   │       └── signature.sig
│   ├── test_sm3
│   │   ├── bin
│   │   │   ├── sm3_digest
│   │   │   └── sm3_hmac
│   │   ├── src
│   │   │   ├── sm3_digest.c
│   │   │   ├── sm3_hmac.c
│   │   │   ├── sm3_hmac.py
│   │   │   └── sm3_hmac_keygen.py
│   │   └── test
│   │       ├── hmac.bin
│   │       ├── input.txt
│   │       ├── output.bin
│   │       ├── sm3_hmac_key_01.bin
│   │       └── sm3_hmac_key_02.bin
│   └── test_sm4
│       ├── bin
│       │   ├── sm4_decrypt
│       │   ├── sm4_encrypt
│       │   └── sm4_keygen
│       ├── src
│       │   ├── sm4_decrypt.c
```

```
│       │   ├── sm4_encrypt.c
│       │   └── sm4_keygen.c
│       └── test
│           ├── de_output.txt
│           ├── en_output.txt
│           ├── input.txt
│           └── sm4_key.bin
└── task03
    └── bin
        ├── sm2_decrypt
        ├── sm2_encrypt
        ├── sm2_keygen
        ├── sm2_sign
        ├── sm2_verify
        ├── sm3_hash
        ├── sm3_hmac
        ├── sm4_decrypt
        ├── sm4_encrypt
        ├── sm4_iv_gen
        └── sm4_key_gen

20 directories, 94 files
root@Youer:~/shiyan/shiyan01/shiyan1-2# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2# git commit -m "Consolidate the
openssl executable files into task03"
[master 9737fa4] Consolidate the openssl executable files into task03
25 files changed, 3 insertions(+), 3 deletions(-)
create mode 100755 shiyan1-2/task01/bin/sm2_decrypt
create mode 100755 shiyan1-2/task01/bin/sm2_encrypt
create mode 100755 shiyan1-2/task01/bin/sm2_keygen
create mode 100755 shiyan1-2/task01/bin/sm2_sign
create mode 100755 shiyan1-2/task01/bin/sm2_verify
create mode 100755 shiyan1-2/task01/bin/sm3_hash
create mode 100755 shiyan1-2/task01/bin/sm3_hmac
create mode 100755 shiyan1-2/task01/bin/sm4_decrypt
create mode 100755 shiyan1-2/task01/bin/sm4_encrypt
create mode 100755 shiyan1-2/task01/bin/sm4_iv_gen
create mode 100755 shiyan1-2/task01/bin/sm4_key_gen
create mode 100755 shiyan1-2/task03/bin/sm2_decrypt
create mode 100755 shiyan1-2/task03/bin/sm2_encrypt
create mode 100755 shiyan1-2/task03/bin/sm2_keygen
create mode 100755 shiyan1-2/task03/bin/sm2_sign
create mode 100755 shiyan1-2/task03/bin/sm2_verify
create mode 100755 shiyan1-2/task03/bin/sm3_hash
create mode 100755 shiyan1-2/task03/bin/sm3_hmac
create mode 100755 shiyan1-2/task03/bin/sm4_decrypt
create mode 100755 shiyan1-2/task03/bin/sm4_encrypt
create mode 100755 shiyan1-2/task03/bin/sm4_iv_gen
create mode 100755 shiyan1-2/task03/bin/sm4_key_gen
root@Youer:~/shiyan/shiyan01/shiyan1-2#
```

- 交换公钥和数字信封

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# ls
bin  plain.txt  sm2_private_key.pem  sm2_public_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# cp ./sm2_public_key.pem
/mnt/d/xlm20/
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# cp
/mnt/d/xlm20/lyh_public_key.pem ./
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# ls
bin  lyh_public_key.pem  plain.txt  sm2_private_key.pem  sm2_public_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# ls
bin  lyh_public_key.pem  plain.txt  sm2_private_key.pem  sm2_public_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# cp /mnt/d/xlm20/*.bin ./
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# cp /mnt/d/xlm20/*.sig ./
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# ls
bin                encrypted_sm4_key.bin  lyh_public_key.pem
signed_encrypted_txt.sig  sm2_public_key.pem
encrypt_input.bin  iv.bin                 plain.txt
sm2_private_key.pem
```

- Sm2Very（PKa，S1）

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# ./bin/sm2_verify
Usage: ./bin/sm2_verify <public_key.pem> <input_file> <signature_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# ./bin/sm2_verify
lyh_public_key.pem encrypt_input.bin signed_encrypted_txt.sig
Signature verification successful!
```

- Sm2Dec（SKb，KC）= k

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# ./bin/sm2_decrypt
Usage: ./bin/sm2_decrypt <private_key.pem> <encrypted_file> <decrypted_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# touch lyh_sm4_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# ./bin/sm2_decrypt
sm2_private_key.pem encrypted_sm4_key.bin  lyh_sm4_key.pem
Decryption successful.
```

- Sm4Dec（k，C）= P

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# ./sm4_decrypt lyh_sm4_key.pem
iv.bin encrypt_input.bin lyh_plain.txt
Decryption complete.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task03# cat  lyh_plain.txt
20221425lyh&20221414xlm
```

4. 两人一组，在 Ubuntu或openEuler中（推荐 openEuler）中使用GmSSL编程实现带签名的数字信封协议。使用GmSSL库时，Bob发送，Alice接收。Ailice，Bob在实验中要替换为自己的8位学号+姓名。 使用

Markdown记录详细记录实践过程，每完成一项git commit 一次。（5分）

- 复制可执行文件

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# ls
test_sm2  test_sm3  test_sm4
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# mkdir bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# cp -r ./test_sm2/bin ./
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# cp -r ./test_sm3/bin ./
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# cp -r ./test_sm4/bin ./
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# tree
.
├── bin
│   ├── sm2_decrypt
│   ├── sm2_encrypt
│   ├── sm2_keygen
│   ├── sm2_sign
│   ├── sm2_verify
│   ├── sm3_digest
│   ├── sm3_hmac
│   ├── sm4_decrypt
│   ├── sm4_encrypt
│   └── sm4_keygen
├── test_sm2
│   ├── bin
│   │   ├── sm2_decrypt
│   │   ├── sm2_encrypt
│   │   ├── sm2_keygen
│   │   ├── sm2_sign
│   │   └── sm2_verify
│   ├── private_key.pem
│   ├── public_key.pem
│   ├── src
│   │   ├── sm2_decrypt.c
│   │   ├── sm2_encrypt.c
│   │   ├── sm2_keygen.c
│   │   ├── sm2_sign.c
│   │   └── sm2_verify.c
│   └── test
│       ├── decrypt.bin
│       ├── encrypt.bin
│       ├── plain.txt
│       └── signature.sig
├── test_sm3
│   ├── bin
│   │   ├── sm3_digest
│   │   └── sm3_hmac
│   ├── src
│   │   ├── sm3_digest.c
│   │   ├── sm3_hmac.c
│   │   ├── sm3_hmac.py
│   │   └── sm3_hmac_keygen.py
│   └── test
```

```
│       ├── hmac.bin
│       ├── input.txt
│       ├── output.bin
│       ├── sm3_hmac_key_01.bin
│       └── sm3_hmac_key_02.bin
└── test_sm4
    ├── bin
    │   ├── sm4_decrypt
    │   ├── sm4_encrypt
    │   └── sm4_keygen
    ├── src
    │   ├── sm4_decrypt.c
    │   ├── sm4_encrypt.c
    │   └── sm4_keygen.c
    └── test
        ├── de_output.txt
        ├── en_output.txt
        ├── input.txt
        └── sm4_key.bin

13 directories, 47 files
root@Youer:~/shiyan/shiyan01/shiyan1-2/task02# cd ..
root@Youer:~/shiyan/shiyan01/shiyan1-2# cp -r ./task02/bin ./task04/bin
root@Youer:~/shiyan/shiyan01/shiyan1-2# cd task04
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# tree
.
└── bin
    ├── sm2_decrypt
    ├── sm2_encrypt
    ├── sm2_keygen
    ├── sm2_sign
    ├── sm2_verify
    ├── sm3_digest
    ├── sm3_hmac
    ├── sm4_decrypt
    ├── sm4_encrypt
    └── sm4_keygen
```

- alice生成sm2密钥和新建明文plain.txt，明文内容是"20221414xlm&20221425lyh"

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# echo
"20221414xlm&20221425lyh" > plain.txt
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# ./bin/sm2_keygen
sm2_private_key.pem sm2_public_key.pem
SM2 key pair generated successfully.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# cat sm2_private_key.pem
-----BEGIN PRIVATE KEY-----
MIGTAgEAMBMGByqGSM49AgEGCCqBHM9VAYItBHkwdwIBAQQgDLA2XQ+AQGTGUKDU
5DJIi2z7QsL2qmLHymkzvSziXG6gCgYIKoEcz1UBgi2hRANCAASGOtexSfpnLskN
xQ4mGIXolf8UsAUgKmnkZ9kdwM/4NR+8LlLGtJjh7xvFiqmAYh3YDIvUuwUB5p4M
oDL5W2Mu
```

```
-----END PRIVATE KEY-----
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# cat sm2_public_key.pem
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoEcz1UBgi0DQgAEhjrXsUn6Zy7JDcUOJhiF6JX/FLAF
ICpp5GfZHcDP+DUfvC5SxrSY4e8bxYqpgGId2AyL1LsFAeaeDKAy+VtjLg==
-----END PUBLIC KEY-----
```

- sm4密钥生成K，获取bob公钥

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# ls
bin  bob_pubkey.pem  plain.txt  sm2_private_key.pem  sm2_public_key.pem
sm4_key.pem
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# git add .
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# git commit -m "get bob
pub_key"
[master 3832e91] get bob pub_key
5 files changed, 16 insertions(+)
create mode 100755 shiyan1-2/task04/bob_pubkey.pem
create mode 100644 shiyan1-2/task04/plain.txt
create mode 100644 shiyan1-2/task04/sm2_private_key.pem
create mode 100644 shiyan1-2/task04/sm2_public_key.pem
create mode 100644 shiyan1-2/task04/sm4_key.pem
```

- Alice：Sm4Enc(k,P) = C

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# ./bin/sm4_encrypt
Usage: ./bin/sm4_encrypt <key_file> <input_file> <output_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# touch encrypted_plain.bin
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# ./bin/sm4_encrypt sm4_key.pem
plain.txt encrypted_plain.bin
Encryption completed successfully.
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# cat encrypted_plain.bin
p�Ja�t�
    ��ʕ/?:/  ���@@o">@:@@root@Youer:~/shiyan/shiyan01/shiyan1-
2/task04#
```

- Alice：Sm2Enc(PKb,k) = KC

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# ./bin/sm2_encrypt
Usage: sm2_encrypt <public_key.pem> <input_file> <output_file>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# ./bin/sm2_encrypt
sm2_public_key.pem sm4_key.pem encrypted_sm4_key.bin
Encryption successful, output written to encrypted_sm4_key.bin
```

- Alice：Sm2Sign（SKa，C）= S1

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# ./bin/sm2_sign
Usage: ./bin/sm2_sign <private_key.pem> <input.txt> <signature.sig>
root@Youer:~/shiyan/shiyan01/shiyan1-2/task04# ./bin/sm2_sign
sm2_private_key.pem encrypted_plain.bin signed_encrypted_plain.sig
Signature generated and saved to signed_encrypted_plain.sig
```

5. 使用Rust完成带签名的数字信封协议（选做，10分）

6. 实验记录中提交 gitee 课程项目链接，提交本次实验相关 git log运行结果

```
root@Youer:~/shiyan/shiyan01/shiyan1-2# git log
commit b6dd24786d15c04fb0964a6cb141d2cec15c8273 (HEAD -> master,
origin/master)
commit 5e0f8a3ac2369e4ab61d42d543e46b1e77b52ca0 (HEAD -> master,
origin/master)
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sun Oct 20 19:11:44 2024 +0800

    fix openssl sm4 code

commit d1df4c8c043e777dbdd9f7fc46746f925bfc3257
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sun Oct 20 18:56:39 2024 +0800

    finish shiyan 1-2 task03

Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sun Oct 20 11:28:09 2024 +0800

    finish shiyan1-2 task04

commit 3377faf9179b1bab8a4c915dd31fc1d7708f5f7e
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sun Oct 20 11:03:22 2024 +0800

    Complete Alice's task in gmssl

commit 3832e9189738169233e52880b82ed92f66858c76
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sun Oct 20 10:47:33 2024 +0800

    get bob pub_key

commit f5e9c8bb6c5dba9d84a43c24290e8b7cbc6d9e9f
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sun Oct 20 10:31:42 2024 +0800

    Pause the openssl experiment and switch to the gmssl experiment.

commit a6090f3402200d1bc21815e2fdd8a89188ce8738
Author: 徐鹿鸣 <xlm20040219@qq.com>
```

```
Date:   Sun Oct 20 10:22:27 2024 +0800

    openssl create plain.txt and create sm2 key

commit 1d127b3b7cef96c1334d644f52db23a5c682e5af
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sun Oct 20 10:13:33 2024 +0800

    Consolidate the gmssl executable files into task04

commit 9737fa4ca8423468becab96d1693932a13eb9093
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sun Oct 20 10:07:54 2024 +0800

    Consolidate the openssl executable files into task03

commit 9637873880c8d37b6b954dd5c925892a35115b4b
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sat Oct 19 13:07:22 2024 +0800

    finish gmssl sm3_hmac func by python code

commit 281faf701eade856d935f0148486f018870e8a0d
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sat Oct 19 10:20:51 2024 +0800

    finish openssl sm2 key gen

commit 9c6e06f5a4a1b287c2da5b80af5f2e9f304b84e0
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Fri Oct 18 23:28:05 2024 +0800

    update openssl sm4 encrypt & decrypt code

commit 0b46498ee5867a88a5252970b75764d2d90b71dc
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Fri Oct 18 23:07:01 2024 +0800

    finish openssl iv gen

commit acdac499dbd3d7f29a49a870278fe583efef03f7
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Fri Oct 18 23:01:28 2024 +0800

    finish openssl sm4 key gen

commit 96de790b4217e431742606bfc89a6ce4a9f365fc
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Wed Oct 16 21:17:48 2024 +0800

    successful test openssl's sm2_sign & sm2_verify

commit 888e2ed3814ef278640174eee53a154f2432676f
Author: 徐鹿鸣 <xlm20040219@qq.com>
```

```
Date:   Wed Oct 16 19:24:13 2024 +0800

    Add all uncommitted files

commit ba133d69db495ea844d41fdac429371b905f4d7d
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Wed Oct 16 19:16:02 2024 +0800

    finish gmssl sm4 ken_gen & decrypt & encrypt

commit ea58d1106c2907500def86d4e8c771a4fab3f65c
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Wed Oct 16 18:38:42 2024 +0800

    finish gmssl hash & hmac by code

commit d8b17fd115784837e1183ded49775d8346f73328
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Wed Oct 16 13:38:34 2024 +0800

    delete useless file

commit de0b95b675d81ccf22ebb9cd7d1169aa76abf849
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Wed Oct 16 13:37:29 2024 +0800

    finish sm2 sign & verify by PKCS#8 key

commit 38420861accb7e9e6e8365c45614c295f2debae7
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 15 23:17:16 2024 +0800

    fail to finish gmssl sign & verify

commit 8ccde69556be28ab7c9d3814e96f9796e6d9895b
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 15 21:54:40 2024 +0800

    finish gmssl sm2 encrypt & decrypt

commit 0ae010fcaa83bc5dacf6a77ae8a7d040343ac7b5
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 15 19:40:38 2024 +0800

    gmssl:create keys by code

commit 8575bbfcdf37cc27962ec5bef109a8ff5fa8e48f
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 15 19:17:23 2024 +0800

    finish sm2 sign & verify

commit aae0ff05e7415659fb1ed5d62f10ec1d69111bd8
Author: 徐鹿鸣 <xlm20040219@qq.com>
```

```
Date:   Tue Oct 15 18:30:28 2024 +0800

    finish sm2 encrypt & decrypt

commit 7e4df4da0835fc24d6452bc78154ef3246e71003
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 15 17:03:07 2024 +0800

    finish SM4 encrypt & decrypt

commit b8cfa8417e1b28689c0ed7625006c5cf632fd46d
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 15 16:49:35 2024 +0800

    finish sm3 HMAC

commit d854781ccc2089ba45f1ba020b3a3a9548b5692e
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 15 16:43:31 2024 +0800

    shiyan1-2/Task01:finish SM3 MAC

commit 21252c42002a61e97702323b6a22392d90b7650c
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 15 16:16:30 2024 +0800

    task01 fail

commit d302e64543b673bbbb3fe6ead05808d7a9298d05
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 15 16:07:41 2024 +0800

    Implemented SM2 signing and verification

commit eea6c265819cebf9ac6b531096ec604c7baa9f7f
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 15 16:05:14 2024 +0800

    Implemented SM4 encryption and decryption

commit 7c6297bca248316a8c3f0cbebaa1e3a2e2de61c8
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 15 16:01:36 2024 +0800

    Implemented SM3 hash and HMAC computation

commit 50de39709984e74eecd522842ba8aeaf093b1183
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 15 15:54:04 2024 +0800

    Implemented SM2 encryption and decryption

commit 6dfbd075b6e3dca8b49005a705da424020577b44
Author: 徐鹿鸣 <xlm20040219@qq.com>
```

```
Date:   Tue Oct 15 15:51:10 2024 +0800

    Initial project setup with CMake and OpenSSL
```

7. 提交要求：

- 提交实践过程Markdown和转化的PDF文件
- 代码，文档托管到gitee或github等，推荐 gitclone
- 记录实验过程中遇到的问题，解决过程，反思等内容，用于后面实验报告
- 实验一项目gittee链接
- 实验过程文档GitHub链接

8.代码中遇到的问题记录与经验总结

- 第一步的sm2代码解密失败，提示ASN.1编码出错
    - 原代码

```
#include <openssl/evp.h>
#include <openssl/pem.h>
#include <stdio.h>
#include <stdlib.h>

void handleErrors(void) {
    fprintf(stderr, "An error occurred.\n");
    exit(1);
}

int main(int argc, char *argv[]) {
    if (argc != 4) {
        fprintf(stderr, "Usage: %s <privkey.pem> <input file> <output
file>\n", argv[0]);
        return 1;
    }

    char *privkey_filename = argv[1];
    char *input_file = argv[2];
    char *output_file = argv[3];

    FILE *f_input, *f_output, *f_privkey;
    unsigned char buffer[1024];
    unsigned char *plaintext;
    size_t plaintext_len;
    size_t bytes_read;  // 声明 bytes_read 变量

    EVP_PKEY *privkey = NULL;
    EVP_PKEY_CTX *ctx = NULL;
```

```
    // Load private key
    if (!(f_privkey = fopen(privkey_filename, "r"))) {
        fprintf(stderr, "Unable to open private key file %s\n",
privkey_filename);
        return 1;
    }

    if (!(privkey = PEM_read_PrivateKey(f_privkey, NULL, NULL, NULL)))
{
        fprintf(stderr, "Error loading private key\n");
        fclose(f_privkey);
        return 1;
    }
    fclose(f_privkey);

    // Initialise the library
    if (!OPENSSL_init_crypto(OPENSSL_INIT_LOAD_CRYPTO_STRINGS, NULL))
        handleErrors();

    // Create and initialise the context
    if (!(ctx = EVP_PKEY_CTX_new(privkey, NULL)))
        handleErrors();

    if (EVP_PKEY_decrypt_init(ctx) <= 0)
        handleErrors();

    // Open files
    if (!(f_input = fopen(input_file, "rb"))) {
        fprintf(stderr, "Could not open %s for reading\n", input_file);
        return 1;
    }

    if (!(f_output = fopen(output_file, "wb"))) {
        fprintf(stderr, "Could not open %s for writing\n",
output_file);
        return 1;
    }

    // Determine buffer size for output
    if (EVP_PKEY_decrypt(ctx, NULL, &plaintext_len, buffer,
sizeof(buffer)) <= 0)
        handleErrors();
    plaintext = malloc(plaintext_len);

    // Decrypt the data
    while ((bytes_read = fread(buffer, 1, sizeof(buffer), f_input)) >
0) {
        if (EVP_PKEY_decrypt(ctx, plaintext, &plaintext_len, buffer,
bytes_read) <= 0)
            handleErrors();
        fwrite(plaintext, 1, plaintext_len, f_output);
    }

    // Clean up
```

```
    EVP_PKEY_free(privkey);
    EVP_PKEY_CTX_free(ctx);
    fclose(f_input);
    fclose(f_output);
    free(plaintext);

    printf("Decryption complete.\n");

    return 0;
}
```

- 报错

```
root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_decrypt
sm2_private.pem encrypted_file.bin decrypted_file.bin
OpenSSL error: error:068000A8:asn1 encoding routines::wrong tag
OpenSSL error: error:0688010A:asn1 encoding routines::nested asn1 error
OpenSSL error: error:1A800068:SM2 routines::reason(104)
```

- 具体表现：
  - 无法通过代码进行解密，但可以通过openssl命令解密
  - 在询问中，AI的代码无法跳出这一bug，且总是重复一些已经排除的可能
- 最终解决：
  - 明确根本问题：需要一个调用openssl库且从命令行接受密钥和加解密文件的SM2解密函数
  - 最终舍弃了之前的讨论才得到了正确答案
- 解决问题的具体对话 ``` User: ./sm2_decrypt sm2_private.pem encrypted_file decrypted_file An error occurred. 这该怎么办

```
  Kimi: 出现 "An error occurred." 的错误消息通常意味着在程序执行过程中遇到了问题，但并没有提供具体的错误信息。为了进一步诊断问题，我们可以采取以下步骤：

  1. **检查输入文件**：确保 `encrypted_file` 是存在的，并且其内容是使用相应的公钥正确加密的。如果文件为空或损坏，解密操作可能会失败。

  2. **检查私钥**：确保 `sm2_private.pem` 文件包含正确的私钥，并且该私钥与用于加密的公钥是匹配的。如果私钥不正确或格式有误，解密也会失败。

  3. **查看 OpenSSL 错误队列**：OpenSSL 库维护了一个错误队列，当函数调用失败时，可以在程序中检查这些错误。修改 `handleErrors` 函数来打印出错误信息：

  ```c
  void handleErrors(void) {
      unsigned long err;
```
```

```
    while ((err = ERR_get_error()) != 0) {
        char buf[256];
        ERR_error_string_n(err, buf, sizeof(buf));
        fprintf(stderr, "OpenSSL error: %s\n", buf);
    }
    exit(1);
}
```

4. **确保正确的库和算法**：如果你正在使用 SM2 算法，确保你的 OpenSSL 库支持 SM2，并且你使用的是正确的初始化和加密函数。

5. **检查内存分配**：确保 `malloc` 为 `plaintext` 分配的内存是足够的。如果 `EVP_PKEY_decrypt` 需要更多的空间来存储解密后的数据，而分配的空间不足，这也可能导致错误。

6. **检查文件读写权限**：确保程序有权限读取输入文件和写入输出文件。

7. **调试程序**：如果可能的话，使用调试器来逐步执行程序，观察在哪一步出现了问题。

8. **查看程序输出**：运行程序时，注意查看是否有其他输出信息，这可能会提供问题的线索。

9. **检查代码逻辑**：重新检查代码逻辑，确保所有的步骤都是正确的，并且符合 OpenSSL 的使用要求。

如果在采取了上述步骤后仍然无法解决问题，你可能需要提供更多的上下文信息，例如私钥和加密文件的内容（如果可能的话），以便进一步分析。记住，不要在公共场合或不安全的环境中泄露你的私钥或敏感数据。

User: 请给我修改完成的代码

Kimi: 当然，以下是修改后的完整代码，包括了从 OpenSSL 错误队列中提取并打印错误信息的功能。这将帮助你诊断在运行程序时遇到的问题。

```c
#include <openssl/evp.h>
#include <openssl/pem.h>
#include <openssl/err.h>
#include <stdio.h>
#include <stdlib.h>

void handleErrors(void) {
    unsigned long err;
    while ((err = ERR_get_error()) != 0) {
        char buf[256];
        ERR_error_string_n(err, buf, sizeof(buf));
        fprintf(stderr, "OpenSSL error: %s\n", buf);
    }
    exit(1);
```

```
  }

  int main(int argc, char *argv[]) {
      if (argc != 4) {
          fprintf(stderr, "Usage: %s <privkey.pem> <input file> <output
file>\n", argv[0]);
          return 1;
      }

      char *privkey_filename = argv[1];
      char *input_file = argv[2];
      char *output_file = argv[3];

      FILE *f_input, *f_output, *f_privkey;
      unsigned char buffer[1024];
      unsigned char *plaintext;
      size_t bytes_read, plaintext_len;

      EVP_PKEY *privkey = NULL;
      EVP_PKEY_CTX *ctx = NULL;

      // Load private key
      if (!(f_privkey = fopen(privkey_filename, "r"))) {
          fprintf(stderr, "Unable to open private key file %s\n",
privkey_filename);
          return 1;
      }

      if (!(privkey = PEM_read_PrivateKey(f_privkey, NULL, NULL,
NULL))) {
          fprintf(stderr, "Error loading private key\n");
          fclose(f_privkey);
          handleErrors();
      }
      fclose(f_privkey);

      // Initialise the library
      if (!OPENSSL_init_crypto(OPENSSL_INIT_LOAD_CRYPTO_STRINGS, NULL))
          handleErrors();

      // Create and initialise the context
      if (!(ctx = EVP_PKEY_CTX_new(privkey, NULL)))
          handleErrors();

      if (EVP_PKEY_decrypt_init(ctx) <= 0)
          handleErrors();

      // Open files
      if (!(f_input = fopen(input_file, "rb"))) {
          fprintf(stderr, "Could not open %s for reading\n",
input_file);
```

```
            return 1;
        }

        if (!(f_output = fopen(output_file, "wb"))) {
            fprintf(stderr, "Could not open %s for writing\n",
output_file);
            return 1;
        }

        // Determine buffer size for output
        if (EVP_PKEY_decrypt(ctx, NULL, &plaintext_len, buffer,
sizeof(buffer)) <= 0)
            handleErrors();
        plaintext = malloc(plaintext_len);
        if (!plaintext) {
            fprintf(stderr, "Memory allocation failed\n");
            return 1;
        }

        // Decrypt the data
        while ((bytes_read = fread(buffer, 1, sizeof(buffer), f_input)) >
0) {
            if (EVP_PKEY_decrypt(ctx, plaintext, &plaintext_len, buffer,
bytes_read) <= 0)
                handleErrors();
            fwrite(plaintext, 1, plaintext_len, f_output);
        }

        // Clean up
        EVP_PKEY_free(privkey);
        EVP_PKEY_CTX_free(ctx);
        fclose(f_input);
        fclose(f_output);
        free(plaintext);

        printf("Decryption complete.\n");

        return 0;
    }
    ```

    ### 编译指令
    确保在编译时链接 OpenSSL 的加密库：
    ```bash
    gcc sm2_decrypt.c -o sm2_decrypt -lcrypto -lssl
    ```

    ### 使用说明
    运行程序时，确保提供正确的参数：私钥文件、加密的输入文件和要写入的输出文件。如果程序在执行过程中遇到错误，它将打印出 OpenSSL 错误信息，这将帮助你诊断问题。
```
```

如果有任何问题或需要进一步的帮助，请随时告诉我。

User: root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2# ./sm2_decrypt sm2_private.pem encrypted_file.txt decrypted_file.txt
OpenSSL error: error:0680007B:asn1 encoding routines::header too long
OpenSSL error: error:06800066:asn1 encoding routines::bad object header
OpenSSL error: error:0688010A:asn1 encoding routines::nested asn1 error
OpenSSL error: error:1A800068:SM2 routines::reason(104)

Kimi: 根据您提供的错误信息，这里有几个可能的原因和解决方案：

1. **ASN.1编码问题**：错误信息表明在处理ASN.1编码时出现了问题，这通常是由于输入文件的格式不正确。确保您的加密文件是使用正确的SM2公钥加密的，并且格式正确。如果您的加密文件是ASN.1编码的，确保它遵循正确的结构。

2. **密文格式问题**：SM2加密的密文通常包含三部分：C1、C2和C3。C1是椭圆曲线点，C2是密文数据，C3是SM3的摘要值。确保您的加密文件遵循正确的格式。在不同的实现中，C1C2C3和C1C3C2的顺序可能不同，这取决于使用的SM2标准版本。

3. **密钥不匹配**：确保您使用的私钥与加密时使用的公钥是匹配的。如果密钥不匹配，解密过程将失败。

4. **OpenSSL版本问题**：如果您使用的OpenSSL版本不支持SM2算法，或者存在某些Bug，可能会导致解密失败。确保您的OpenSSL版本是最新的，或者至少是支持SM2算法的版本。

5. **代码问题**：如果代码中有错误，比如错误的使用了`EVP_PKEY_decrypt_final`函数，也会导致解密失败。由于`EVP_PKEY_decrypt_final`并不是OpenSSL的标准API，所以在您的代码中应该移除对它的调用。

6. **内存分配问题**：确保为`plaintext`分配的内存足够大，能够容纳解密后的数据。

7. **文件读写问题**：确保程序有权限读取输入文件和写入输出文件，并且文件没有损坏。

8. **使用正确的解密函数**：在OpenSSL中，对于SM2算法，可能需要使用特定的解密函数，而不是通用的`EVP_PKEY_decrypt`。您可能需要查阅OpenSSL的文档，找到正确的函数来使用。

如果上述解决方案仍然无法解决问题，您可能需要提供更多的上下文信息，例如私钥和加密文件的内容（如果可能的话），以便进一步分析。记住，不要在公共场合或不安全的环境中泄露您的私钥或敏感数据。

User: 是不是不能把密文用txt文件保存

Kimi: 密文通常可以保存在任何类型的文件中，包括文本文件（.txt）。然而，密文

通常是由二进制数据组成的，直接以文本形式保存可能会导致编码问题，特别是如果密文包含无法打印的字符或二进制数据。

在使用文本文件保存密文时，需要确保在读取和写入时使用正确的模式。例如，在C语言中，应该使用“rb”（读取二进制）和“wb”（写入二进制）模式来处理文件，而不是“r”或“w”，这样可以确保二进制数据被正确处理。

如果密文是使用SM2算法加密的，并且遵循了正确的ASN.1编码格式，那么在保存和读取时需要特别注意编码和解码的问题。如果密文是以十六进制或Base64编码的字符串形式保存的，那么在解密之前需要先将其转换回原始的二进制形式。

这里是一个简单的示例，说明如何使用Base64编码和解码来处理二进制数据：

```c
#include <openssl/evp.h>
#include <openssl/bio.h>
#include <openssl/buffer.h>

// 将二进制数据编码为Base64字符串
char* base64_encode(const unsigned char* buffer, size_t length) {
    BIO *bio, *b64;
    BUF_MEM *bufferPtr;

    b64 = BIO_new(BIO_f_base64());
    bio = BIO_new(BIO_s_mem());
    bio = BIO_push(b64, bio);

    BIO_write(bio, buffer, length);
    BIO_flush(bio);
    BIO_get_mem_ptr(bio, &bufferPtr);

    char *base64text = (char*)malloc(bufferPtr->length);
    memcpy(base64text, bufferPtr->data, bufferPtr->length - 1);
    base64text[bufferPtr->length - 1] = 0;

    BIO_free_all(bio);
    return base64text;
}

// 将Base64字符串解码为二进制数据
unsigned char* base64_decode(const char* b64input, int length, size_t
*output_length) {
    BIO *bio, *b64;

    b64 = BIO_new(BIO_f_base64());
    bio = BIO_new_mem_buf(b64input, length);
    bio = BIO_push(b64, bio);

    BUF_MEM *bufferPtr;
    BIO_get_mem_ptr(bio, &bufferPtr);
```

```
      unsigned char *buffer = (unsigned char*)malloc(bufferPtr-
>length);
      BIO_read(bio, buffer, bufferPtr->length);

      *output_length = bufferPtr->length;
      BIO_free_all(bio);
      return buffer;
  }
  ```
```

使用这些函数，你可以在保存和读取密文时进行Base64编码和解码，以确保二进制数据的正确处理。记住，无论使用哪种编码方式，都需要在解密之前将编码的密文转换回原始的二进制形式。

User: 如何存储sm2加密的密文

Kimi: SM2加密的密文存储格式有特定的标准，根据搜索结果，主要有两种格式：

1. **GM/T0003.4 规定的标准密文格式**：这种格式通常表示为 `C1||C3||C2`（新标准）或 `C1||C2||C3`（旧标准）。其中：
  - `C1` 是随机生成的公钥，格式为 `04||X1||Y1`，其中 `04` 代表未压缩格式，`X1` 和 `Y1` 是椭圆曲线上的点坐标。
  - `C2` 是裸密文，即加密后的字符串。
  - `C3` 是摘要值，计算方式为 `Hash(X2||M||Y2)`，其中 `M` 是原文，`X2`、`Y2` 是用户公钥的坐标。

2. **GM/T0009

User: root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test_sm2#
./sm2_decrypt sm2_private.pem encrypted_file.bin decrypted_file.bin
OpenSSL error: error:068000A8:asn1 encoding routines::wrong tag
OpenSSL error: error:0688010A:asn1 encoding routines::nested asn1
error
OpenSSL error: error:1A800068:SM2 routines::reason(104)

Kimi: 根据您提供的错误信息和搜索结果，出现 "error:068000A8:asn1 encoding routines::wrong tag" 和 "error:1A800068:SM2 routines::reason(104)" 错误的原因可能是：

1. **密文格式不正确**：密文可能不是使用正确的SM2公钥加密的，或者密文的格式不符合SM2算法的要求。确保加密时使用的是正确的公钥，并且加密过程没有错误。

2. **ASN.1编码问题**：错误信息表明在处理ASN.1编码时出现了问题，这通常是由于输入文件的格式不正确。如果您的加密文件是ASN.1编码的，确保它遵循正确的结构。

3. **私钥格式问题**：如果私钥不是PKCS#1或PKCS#8标准格式，可能会导致解密时出现错误。您可能需要将私钥转换为正确的格式。

4. **OpenSSL版本问题**：如果您使用的OpenSSL版本不支持SM2算法，或者存在某些Bug，可能会导致解密失败。确保您的OpenSSL版本是最新的，或者至少是支持SM2算法的版本。
```

5. **密钥不匹配**：确保您使用的私钥与加密时使用的公钥是匹配的。如果密钥不匹配，解密过程将失败。

6. **文件读写问题**：确保程序有权限读取输入文件和写入输出文件，并且文件没有损坏。

为了解决这个问题，您可以尝试以下步骤：

- 确保您的加密文件（`encrypted_file.bin`）是使用正确的SM2公钥加密的，并且格式正确。
- 确保您的私钥（`sm2_private.pem`）是正确的，并且与用于加密的公钥匹配。
- 如果您的私钥是PKCS#8格式，您可能需要将其转换为PKCS#1格式，以确保与OpenSSL的兼容性。
- 确保您的OpenSSL版本支持SM2算法。如果不支持，您可能需要升级OpenSSL或使用其他支持SM2的库。
- 检查您的文件读写权限，确保程序可以正确读取和写入文件。

如果问题仍然存在，您可能需要提供更多的上下文信息，例如私钥和加密文件的内容（如果可能的话），以便进一步分析。记住，不要在公共场合或不安全的环境中泄露您的私钥或敏感数据。

User: 如何将sm2生成的密文转变ASN.1编码格式

Kimi: SM2加密的密文可以转换为ASN.1编码格式，这通常是为了确保密文的格式符合特定的标准或要求。ASN.1（Abstract Syntax Notation One）是一种标准的语法，用于数据结构的描述和编码。在SM2加密中，密文通常由C1、C2、C3三部分组成，其中C1是椭圆曲线点，C2是密文数据，C3是摘要值。

根据搜索结果，转换SM2密文为ASN.1编码格式通常涉及以下步骤：

1. **定义ASN.1结构**：首先需要定义一个ASN.1结构来表示SM2密文的C1、C2、C3三部分。这可能需要使用ASN.1的语法来描述这些部分的数据类型和顺序。

2. **编码密文**：使用ASN.1结构，将C1、C2、C3的数据编码为ASN.1格式。这可能涉及到将原始的二进制数据转换为ASN.1的DER（Distinguished Encoding Rules）或BER（Basic Encoding Rules）编码。

3. **处理长度和标签**：在ASN.1编码中，每个数据元素都有一个标签来标识其类型，以及一个长度字段来指定数据的大小。确保这些长度和标签被正确处理是转换过程中的关键部分。

4. **使用工具或库**：有一些工具和库可以帮助进行ASN.1编码和解码，例如OpenSSL、Bouncy Castle等。这些工具通常提供了命令行接口或编程接口，可以用来处理SM2密文的ASN.1编码。

例如，根据搜索结果中提供的信息，可以通过以下代码示例来实现SM2密文的ASN.1编码转换（摘自 ）：

```c
#include <openssl/ec.h>
```

```
#include <openssl/asn1.h>
#include <openssl/asn1t.h>

typedef struct SM2_Ciphertext_st SM2_Ciphertext;
DECLARE_ASN1_FUNCTIONS(SM2_Ciphertext)

struct SM2_Ciphertext_st {
    BIGNUM *C1x;
    BIGNUM *C1y;
    ASN1_OCTET_STRING *C3;
    ASN1_OCTET_STRING *C2;
};

ASN1_SEQUENCE(SM2_Ciphertext) = {
    ASN1_SIMPLE(SM2_Ciphertext, C1x, BIGNUM),
    ASN1_SIMPLE(SM2_Ciphertext, C1y, BIGNUM),
    ASN1_SIMPLE(SM2_Ciphertext, C3, ASN1_OCTET_STRING),
    ASN1_SIMPLE(SM2_Ciphertext, C2, ASN1_OCTET_STRING),
} ASN1_SEQUENCE_END(SM2_Ciphertext)

IMPLEMENT_ASN1_FUNCTIONS(SM2_Ciphertext)

// 然后使用这些函数来编码和解码SM2密文
```

请注意，这只是一个示例，实际的实现可能需要根据您的具体需求和OpenSSL的版本进行调整。如果您需要进一步的帮助，可以提供更多的上下文信息，例如密文的具体格式和您希望实现的具体功能。

User: 在openssl中，如何将sm2生成的密文转变ASN.1编码格式或者确保其生成的就是asn.1的格式

Kimi: 在 OpenSSL 中，SM2 加密的密文通常不直接生成 ASN.1 编码格式，但可以转换为 ASN.1 编码格式。ASN.1 (Abstract Syntax Notation One) 是一种标准的语法，用于数据结构的描述和编码。在 SM2 加密中，密文通常由 C1、C2、C3 三部分组成，其中 C1 是椭圆曲线点，C2 是密文数据，C3 是摘要值。

要将 SM2 生成的密文转换为 ASN.1 编码格式，可以按照以下步骤操作：

1. **定义 ASN.1 结构**：首先需要定义一个 ASN.1 结构来表示 SM2 密文的 C1、C2、C3 三部分。这可能需要使用 ASN.1 的语法来描述这些部分的数据类型和顺序。

2. **编码密文**：使用定义的 ASN.1 结构，将 C1、C2、C3 的数据编码为 ASN.1 格式。这可能涉及到将原始的二进制数据转换为 ASN.1 的 DER (Distinguished Encoding Rules) 或 BER (Basic Encoding Rules) 编码。

3. **处理长度和标签**：在 ASN.1 编码中，每个数据元素都有一个标签来标识其类型，以及一个长度字段来指定数据的大小。确保这些长度和标签被正确处理是转换过程中的关键部分。

根据搜索结果，可以通过以下代码示例来实现 SM2 密文的 ASN.1 编码转换 ：

```c
#include <string.h>
#include <openssl/ec.h>
#include <openssl/asn1.h>
#include <openssl/asn1t.h>

typedef struct SM2_Ciphertext_st SM2_Ciphertext;
DECLARE_ASN1_FUNCTIONS(SM2_Ciphertext)

struct SM2_Ciphertext_st {
    BIGNUM *C1x;
    BIGNUM *C1y;
    ASN1_OCTET_STRING *C3;
    ASN1_OCTET_STRING *C2;
};

ASN1_SEQUENCE(SM2_Ciphertext) = {
    ASN1_SIMPLE(SM2_Ciphertext, C1x, BIGNUM),
    ASN1_SIMPLE(SM2_Ciphertext, C1y, BIGNUM),
    ASN1_SIMPLE(SM2_Ciphertext, C3, ASN1_OCTET_STRING),
    ASN1_SIMPLE(SM2_Ciphertext, C2, ASN1_OCTET_STRING),
} ASN1_SEQUENCE_END(SM2_Ciphertext)

IMPLEMENT_ASN1_FUNCTIONS(SM2_Ciphertext)

// 然后使用这些函数来编码和解码 SM2 密文
```

在实际应用中，您可能需要根据具体的加密数据和需求来调整上述代码。这可能包括处理不同的密文长度、添加错误检查和处理以及其他必要的调整。如果您需要进一步的帮助，可以提供更多的上下文信息，例如密文的具体格式和您希望实现的具体功能。

User: 请帮我编写一个调用openssl库且从命令行接受密钥和加解密文件的SM2解密函数

Kimi: 下面是一个使用 OpenSSL 库的示例 C 程序，它从命令行接受私钥文件、加密文件（密文）和输出文件（明文）路径，并执行 SM2 解密操作。请注意，这个示例假设你已经有了一个正确格式的 SM2 密文文件，并且你的 OpenSSL 库已经支持 SM2。

```c
#include <openssl/ec.h>
#include <openssl/objects.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <openssl/pem.h>
#include <stdio.h>
#include <stdlib.h>

void handle_errors() {
    ERR_print_errors_fp(stderr);
```

```
        abort();
    }

    int main(int argc, char **argv) {
        if (argc != 4) {
            fprintf(stderr, "Usage: %s <private_key.pem> <encrypted_file>
<decrypted_file>\n", argv[0]);
            return EXIT_FAILURE;
        }

        const char *private_key_file = argv[1];
        const char *encrypted_file = argv[2];
        const char *decrypted_file = argv[3];

        // Load private key
        EVP_PKEY *private_key = NULL;
        FILE *fp;
        fp = fopen(private_key_file, "r");
        if (!fp) {
            perror("Error opening private key file");
            return EXIT_FAILURE;
        }
        private_key = PEM_read_PrivateKey(fp, NULL, NULL, NULL);
        fclose(fp);
        if (!private_key) {
            handle_errors();
            return EXIT_FAILURE;
        }

        // Setup decryption
        EVP_PKEY_CTX *ctx = EVP_PKEY_CTX_new(private_key, NULL);
        if (!ctx) {
            handle_errors();
            return EXIT_FAILURE;
        }
        if (EVP_PKEY_decrypt_init(ctx) <= 0) {
            handle_errors();
            return EXIT_FAILURE;
        }

        // Read encrypted data
        FILE *fin = fopen(encrypted_file, "rb");
        if (!fin) {
            perror("Error opening encrypted file");
            return EXIT_FAILURE;
        }
        // Determine the size of the encrypted data
        fseek(fin, 0, SEEK_END);
        long encrypted_data_size = ftell(fin);
        fseek(fin, 0, SEEK_SET);
        unsigned char *encrypted_data = malloc(encrypted_data_size);
```

```
        fread(encrypted_data, 1, encrypted_data_size, fin);
        fclose(fin);

        // Decrypt data
        unsigned char *decrypted_data = NULL;
        size_t decrypted_data_size = 0;
        if (EVP_PKEY_decrypt(ctx, NULL, &decrypted_data_size,
encrypted_data, encrypted_data_size) <= 0) {
            handle_errors();
            return EXIT_FAILURE;
        }
        decrypted_data = malloc(decrypted_data_size);
        if (EVP_PKEY_decrypt(ctx, decrypted_data, &decrypted_data_size,
encrypted_data, encrypted_data_size) <= 0) {
            handle_errors();
            return EXIT_FAILURE;
        }

        // Write decrypted data to file
        FILE *fout = fopen(decrypted_file, "wb");
        if (!fout) {
            perror("Error opening decrypted file");
            return EXIT_FAILURE;
        }
        fwrite(decrypted_data, 1, decrypted_data_size, fout);
        fclose(fout);

        // Clean up
        EVP_PKEY_CTX_free(ctx);
        EVP_PKEY_free(private_key);
        free(encrypted_data);
        free(decrypted_data);

        printf("Decryption successful.\n");
        return EXIT_SUCCESS;
    }
    ```

    编译这个程序时，确保链接了 OpenSSL 的加密库：

    ```bash
    gcc -o sm2_decrypt sm2_decrypt.c -lcrypto -lssl
    ```

    运行程序时，使用以下命令格式：

    ```bash
    ./sm2_decrypt private_key.pem encrypted_file encrypted_file_out
    ```

    其中 `private_key.pem` 是你的 SM2 私钥文件，`encrypted_file` 是包含 SM2
```
```

加密数据的文件，`encrypted_file_out` 是你希望写入解密数据的输出文件。

请注意，这个示例代码假设你的 OpenSSL 库已经支持 SM2 算法，并且你有一个正确格式的 SM2 密文文件。如果你的 OpenSSL 版本不支持 SM2，你可能需要使用其他库或更新你的 OpenSSL 版本。

```

- 这一问题的具体解释（AI）

```
User: 代码1:
        #include <openssl/evp.h>
        #include <openssl/pem.h>
        #include <stdio.h>
        #include <stdlib.h>

        void handleErrors(void) {
            fprintf(stderr, "An error occurred.\n");
            exit(1);
        }

        int main(int argc, char *argv[]) {
            if (argc != 4) {
                fprintf(stderr, "Usage: %s <privkey.pem> <input file>
<output file>\n", argv[0]);
                return 1;
            }

            char *privkey_filename = argv[1];
            char *input_file = argv[2];
            char *output_file = argv[3];

            FILE *f_input, *f_output, *f_privkey;
            unsigned char buffer[1024];
            unsigned char *plaintext;
            size_t plaintext_len;
            size_t bytes_read; // 声明 bytes_read 变量

            EVP_PKEY *privkey = NULL;
            EVP_PKEY_CTX *ctx = NULL;

            // Load private key
            if (!(f_privkey = fopen(privkey_filename, "r"))) {
                fprintf(stderr, "Unable to open private key file %s\n",
privkey_filename);
                return 1;
            }

            if (!(privkey = PEM_read_PrivateKey(f_privkey, NULL, NULL,
NULL))) {
```
```

```
                fprintf(stderr, "Error loading private key\n");
                fclose(f_privkey);
                return 1;
            }
            fclose(f_privkey);

            // Initialise the library
            if (!OPENSSL_init_crypto(OPENSSL_INIT_LOAD_CRYPTO_STRINGS,
NULL))
                handleErrors();

            // Create and initialise the context
            if (!(ctx = EVP_PKEY_CTX_new(privkey, NULL)))
                handleErrors();

            if (EVP_PKEY_decrypt_init(ctx) <= 0)
                handleErrors();

            // Open files
            if (!(f_input = fopen(input_file, "rb"))) {
                fprintf(stderr, "Could not open %s for reading\n",
input_file);
                return 1;
            }

            if (!(f_output = fopen(output_file, "wb"))) {
                fprintf(stderr, "Could not open %s for writing\n",
output_file);
                return 1;
            }

            // Determine buffer size for output
            if (EVP_PKEY_decrypt(ctx, NULL, &plaintext_len, buffer,
sizeof(buffer)) <= 0)
                handleErrors();
            plaintext = malloc(plaintext_len);

            // Decrypt the data
            while ((bytes_read = fread(buffer, 1, sizeof(buffer),
f_input)) > 0) {
                if (EVP_PKEY_decrypt(ctx, plaintext, &plaintext_len,
buffer, bytes_read) <= 0)
                    handleErrors();
                fwrite(plaintext, 1, plaintext_len, f_output);
            }

            // Clean up
            EVP_PKEY_free(privkey);
            EVP_PKEY_CTX_free(ctx);
            fclose(f_input);
            fclose(f_output);
            free(plaintext);

            printf("Decryption complete.\n");
```

```
            return 0;
        }
```

代码2：

```
                #include <openssl/ec.h>
                #include <openssl/objects.h>
                #include <openssl/evp.h>
                #include <openssl/err.h>
                #include <openssl/pem.h>
                #include <stdio.h>
                #include <stdlib.h>

                void handle_errors() {
                    ERR_print_errors_fp(stderr);
                    abort();
                }

                int main(int argc, char **argv) {
                    if (argc != 4) {
                        fprintf(stderr, "Usage: %s <private_key.pem>
<encrypted_file> <decrypted_file>\n", argv[0]);
                        return EXIT_FAILURE;
                    }

                    const char *private_key_file = argv[1];
                    const char *encrypted_file = argv[2];
                    const char *decrypted_file = argv[3];

                    // Load private key
                    EVP_PKEY *private_key = NULL;
                    FILE *fp;
                    fp = fopen(private_key_file, "r");
                    if (!fp) {
                        perror("Error opening private key file");
                        return EXIT_FAILURE;
                    }
                    private_key = PEM_read_PrivateKey(fp, NULL, NULL,
NULL);
                    fclose(fp);
                    if (!private_key) {
                        handle_errors();
                        return EXIT_FAILURE;
                    }

                    // Setup decryption
                    EVP_PKEY_CTX *ctx = EVP_PKEY_CTX_new(private_key,
NULL);
                    if (!ctx) {
                        handle_errors();
                        return EXIT_FAILURE;
                    }
                    if (EVP_PKEY_decrypt_init(ctx) <= 0) {
                        handle_errors();
                        return EXIT_FAILURE;
```

```
                    }

                    // Read encrypted data
                    FILE *fin = fopen(encrypted_file, "rb");
                    if (!fin) {
                        perror("Error opening encrypted file");
                        return EXIT_FAILURE;
                    }
                    // Determine the size of the encrypted data
                    fseek(fin, 0, SEEK_END);
                    long encrypted_data_size = ftell(fin);
                    fseek(fin, 0, SEEK_SET);
                    unsigned char *encrypted_data =
malloc(encrypted_data_size);
                    fread(encrypted_data, 1, encrypted_data_size, fin);
                    fclose(fin);

                    // Decrypt data
                    unsigned char *decrypted_data = NULL;
                    size_t decrypted_data_size = 0;
                    if (EVP_PKEY_decrypt(ctx, NULL,
&decrypted_data_size, encrypted_data, encrypted_data_size) <= 0) {
                        handle_errors();
                        return EXIT_FAILURE;
                    }
                    decrypted_data = malloc(decrypted_data_size);
                    if (EVP_PKEY_decrypt(ctx, decrypted_data,
&decrypted_data_size, encrypted_data, encrypted_data_size) <= 0) {
                        handle_errors();
                        return EXIT_FAILURE;
                    }

                    // Write decrypted data to file
                    FILE *fout = fopen(decrypted_file, "wb");
                    if (!fout) {
                        perror("Error opening decrypted file");
                        return EXIT_FAILURE;
                    }
                    fwrite(decrypted_data, 1, decrypted_data_size,
fout);
                    fclose(fout);

                    // Clean up
                    EVP_PKEY_CTX_free(ctx);
                    EVP_PKEY_free(private_key);
                    free(encrypted_data);
                    free(decrypted_data);

                    printf("Decryption successful.\n");
                    return EXIT_SUCCESS;
                }
```

请你分析为什么第一个代码会有文件格式（ASN.1）的报错而第二个代码是正确的

Kimi: 两个代码都旨在使用OpenSSL库进行解密操作，但是它们在处理输入数据和调用解

密函数的方式上有所不同。以下是可能导致第一个代码出现ASN.1格式错误的原因，以及第二个代码为何能够正确执行的分析：

1. **一次性读取与分块读取**：
- 第一个代码尝试分块读取加密数据，每次读取固定大小的缓冲区（`buffer`），然后对每个缓冲区进行解密。这种方法在理论上是可行的，但前提是加密数据的块边界能够正确对齐。如果加密数据的块边界没有正确对齐，可能会导致ASN.1格式错误，因为ASN.1编码的数据结构可能被错误地分割。
- 第二个代码一次性读取整个加密文件到内存中，然后一次性进行解密。这种方法更可靠，因为它确保了整个加密数据作为一个整体被处理，避免了块边界对齐的问题。

2. **输出缓冲区大小的确定**：
- 在第一个代码中，`plaintext_len`是在第一次调用`EVP_PKEY_decrypt`时确定的，但是这个调用并没有传入实际的加密数据（`buffer`），这可能导致`plaintext_len`的值不准确。此外，如果加密数据的块边界没有正确对齐，每次解密操作可能需要不同的输出缓冲区大小。
- 在第二个代码中，`decrypted_data_size`是在第一次调用`EVP_PKEY_decrypt`时确定的，这次调用传入了整个加密数据。这确保了输出缓冲区大小的准确性，因为整个加密数据作为一个整体被处理。

3. **错误处理**：
- 第一个代码在出现错误时调用`handleErrors`函数，该函数只是打印一条错误消息并退出程序，没有提供详细的错误信息。
- 第二个代码在出现错误时调用`handle_errors`函数，该函数使用`ERR_print_errors_fp`函数打印详细的错误信息，这有助于诊断问题。

4. **内存分配**：
- 在第一个代码中，`plaintext`的内存分配是基于第一次调用`EVP_PKEY_decrypt`时返回的`plaintext_len`，这可能不准确，如前所述。
- 在第二个代码中，`decrypted_data`的内存分配是基于第一次调用`EVP_PKEY_decrypt`时返回的`decrypted_data_size`，这更可靠，因为它是基于整个加密数据的。

综上所述，第一个代码可能出现ASN.1格式错误的原因可能是由于分块读取加密数据导致的块边界对齐问题，以及输出缓冲区大小的确定不准确。第二个代码通过一次性读取和处理整个加密数据，避免了这些问题，因此能够正确执行解密操作。

    - 反思：
        - 关键在于明确根本问题，之前尝试解决时猜测了很多解决思路，比如转换密文格式等，但这些只是猜测，最终浪费了自己的时间
            - 这就是"XY"问题的翻版
        - 同时不要就一个代码问AI太久，AI跳不出原有的框架，就会一直给出错误代码。及时开启新对话。
        - 以及加解密存储的文件还是保存为.bin比较好，不容易有格式错误
- 一些经验
    - openssl
        - openssl调用sm2等算法时一般要用evp接口，不能直接调用。同时，要明确告诉AI使用EVP接口，可以有效减少工作量。

        - openssl中不存在"openssl/sm2"等头文件
    - AI调用gmssl库时经常混用openssl的代码
        - 比较突出的是头文件"gmssl/err.h"，但其实gmssl的错误处理的头文件是"gmssl/error.h"
        - 同时还有对"gmssl/evp.h"等不存在的头文件的使用
        - 要明确提示AI哪些文件不可用，哪些文件的正确用法是什么
    - 同时要声明使用的库的版本，不然AI会调用一些旧的API
    - C语言实在写不出来就用python
- 私钥格式问题
    - 这次遇到了哪些格式
        - PEM格式(EC格式)
        - PKCS#8格式
    - 不同函数对私钥格式支持不一样，即sm2_private_key_from_pem（支持PKCS#8格式）和sm2_private_key_info_from_pem（支持EC格式）
    - AI的协助：
        - 我的提问

请参考下面的私钥加载过程修改之后的签名生成程序：
私钥加载过程：

```
// Load private key
SM2_KEY sm2_key;
FILE *fp = fopen(private_key_file, "r");
if (!fp) {
perror("Failed to open private key file");
return 1;
}

// 使用 PEM 格式加载私钥
if (sm2_private_key_info_from_pem(&sm2_key, fp) != 1) {
    fprintf(stderr, "Failed to read private key from PEM\n");
    fclose(fp);
    return 1;
}
fclose(fp);
```

签名程序：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gmssl/sm2.h>
#include <gmssl/pem.h>
#include <gmssl/error.h>
#include <gmssl/sm3.h>

int main(int argc, char **argv)
{
if (argc != 4) {
printf("Usage: %s <private_key.pem> <input.txt>
<signature.sig>\n", argv[0]);
return 1;
}
```

```
const char *private_key_file = argv[1];
const char *input_file = argv[2];
const char *signature_file = argv[3];

SM2_KEY sm2_key;
FILE *key_fp = NULL;
FILE *input_fp = NULL;
FILE *sig_fp = NULL;
unsigned char dgst[32];
unsigned char sig[SM2_MAX_SIGNATURE_SIZE];
size_t siglen;
unsigned char buffer[1024];
size_t len;

// 读取私钥
if (!(key_fp = fopen(private_key_file, "r"))) {
    fprintf(stderr, "Error opening private key file: %s\n",
private_key_file);
    return 1;
}

if (sm2_private_key_from_pem(&sm2_key, key_fp) != 1) {
    fprintf(stderr, "Error loading private key from PEM file:
%s\n", private_key_file);
    fclose(key_fp);
    return 1;
}
fclose(key_fp);

// 读取输入文件并计算SM3哈希
if (!(input_fp = fopen(input_file, "r"))) {
    fprintf(stderr, "Error opening input file: %s\n", input_file);
    return 1;
}

SM3_CTX sm3_ctx;
sm3_init(&sm3_ctx);

while ((len = fread(buffer, 1, sizeof(buffer), input_fp)) > 0) {
    sm3_update(&sm3_ctx, buffer, len);
}
fclose(input_fp);

sm3_finish(&sm3_ctx, dgst);

// 生成签名
siglen = sizeof(sig);  // 确保我们为签名的大小传递正确的变量
if (sm2_sign(&sm2_key, dgst, sig, &siglen) != 1) {
    fprintf(stderr, "Error generating SM2 signature\n");
    return 1;
}

// 写入签名到文件
if (!(sig_fp = fopen(signature_file, "wb"))) {
```

```
    fprintf(stderr, "Error opening signature file: %s\n",
signature_file);
    return 1;
}

if (fwrite(sig, 1, siglen, sig_fp) != siglen) {
    fprintf(stderr, "Error writing signature to file: %s\n",
signature_file);
    fclose(sig_fp);
    return 1;
}
fclose(sig_fp);

printf("Signature generated and saved to %s\n", signature_file);
return 0;
}
```

- AI的回答：

您提供的代码中有一个私钥加载过程和一个签名生成程序。为了将私钥加载过程合并到签名生成程序中，我们需要替换原有的私钥加载部分，以使用您提供的加载过程。这是一个经过修改的完整签名生成程序，使用了新的私钥加载逻辑。
修改后的签名生成程序

```
#include &lt;stdio.h&gt;
#include &lt;stdlib.h&gt;
#include &lt;string.h&gt;
#include &lt;gmssl/sm2.h&gt;
#include &lt;gmssl/pem.h&gt;
#include &lt;gmssl/error.h&gt;
#include &lt;gmssl/sm3.h&gt;

int main(int argc, char **argv)
{
    if (argc != 4) {
        printf("Usage: %s &lt;private_key.pem&gt;
&lt;input.txt&gt; &lt;signature.sig&gt;\n", argv[0]);
        return 1;
    }

    const char *private_key_file = argv[1];
    const char *input_file = argv[2];
    const char *signature_file = argv[3];

    SM2_KEY sm2_key;
    FILE *key_fp = NULL;
    FILE *input_fp = NULL;
    FILE *sig_fp = NULL;
    unsigned char dgst[32];
    unsigned char sig[SM2_MAX_SIGNATURE_SIZE];
    size_t siglen;
    unsigned char buffer[1024];
    size_t len;
```

```
    // 加载私钥
    key_fp = fopen(private_key_file, "r");
    if (!key_fp) {
        perror("Failed to open private key file");
        return 1;
    }

    // 使用 PEM 格式加载私钥
    if (sm2_private_key_info_from_pem(&amp;sm2_key, key_fp) != 1)
{
        fprintf(stderr, "Failed to read private key from PEM\n");
        fclose(key_fp);
        return 1;
    }
    fclose(key_fp);

    // 读取输入文件并计算SM3哈希
    input_fp = fopen(input_file, "r");
    if (!input_fp) {
        fprintf(stderr, "Error opening input file: %s\n",
input_file);
        return 1;
    }

    SM3_CTX sm3_ctx;
    sm3_init(&amp;sm3_ctx);

    while ((len = fread(buffer, 1, sizeof(buffer), input_fp)) &gt;
0) {
        sm3_update(&amp;sm3_ctx, buffer, len);
    }
    fclose(input_fp);

    sm3_finish(&amp;sm3_ctx, dgst);

    // 生成签名
    siglen = sizeof(sig);  // 确保我们为签名的大小传递正确的变量
    if (sm2_sign(&amp;sm2_key, dgst, sig, &amp;siglen) != 1) {
        fprintf(stderr, "Error generating SM2 signature\n");
        return 1;
    }

    // 写入签名到文件
    sig_fp = fopen(signature_file, "wb");
    if (!sig_fp) {
        fprintf(stderr, "Error opening signature file: %s\n",
signature_file);
        return 1;
    }

    if (fwrite(sig, 1, siglen, sig_fp) != siglen) {
        fprintf(stderr, "Error writing signature to file: %s\n",
signature_file);
```

```
        fclose(sig_fp);
        return 1;
    }
    fclose(sig_fp);

    printf("Signature generated and saved to %s\n",
signature_file);
    return 0;
}
```

修改的要点

1.私钥加载逻辑：

2.使用提供的私钥加载过程替换了原有的 sm2_private_key_from_pem 函数调用。
3.采用了 sm2_private_key_info_from_pem 来从 PEM 文件中加载私钥。

4.错误处理：

5.在每个打开文件和加载私钥的步骤中增加了适当的错误处理，确保在出现错误时能够输出相应的信息并退出程序。

6.变量和文件指针管理：

7.确保在所有文件操作后关闭文件指针，防止资源泄漏。

注意事项

8.请确保您拥有所需的库和头文件（如 gmssl）以支持 SM2 签名和 SM3 哈希计算。
9.在实际使用中，请根据您的需求调整输入输出文件的路径和名称。

- 原理解答(AI)
- 反思：
  - 提问时要学会借鉴成功经验。
  - 比如在这里，我之前的sm2加解密就可以使用PKCS#8格式的私钥，那么就可以参考他们的代码去实现或者给AI这方面的提示