[toc]

《密码系统设计》实验

实验二 密码算法实现

1-3 学时实践要求（30 分）

1. 在 Ubuntu或openEuler中（推荐 openEuler）中调试运行商用密码检测中心 https://www.scctc.org.cn/xzzx/sfydm/ydmxz/提供的源代码，至少运行SM2，SM3，SM4代码。使用GmSSL命令验证你代码的正确性。使用Markdown记录详细记录实践过程，每完成一项功能或者一个函数git commit 一次。（14分）

- SM3测试
  - 代码修改（测试使用的main函数）

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

const char* testCases[] = {
    "xlm",
    "20221414xlm20221414xlm",
};

int main() {

    // 调用自检函数并输出结果
    int selfTestResult = SM3_SelfTest();
    if (selfTestResult == 0) {
        printf("SM3 SelfTest passed.\n");
    }
    else {
        printf("SM3 SelfTest failed.\n");
    }

    // 计算并输出每个测试用例的哈希值
    for (int i = 0; i < sizeof(testCases) / sizeof(testCases[0]); i++)
{
        unsigned char MsgHash[32] = { 0 };
        const char* testCase = testCases[i];
        int len = strlen(testCase);

        // 计算哈希值
        SM3_256((unsigned char*)testCase, len, MsgHash);

        // 打印计算出的哈希值
        printf("Computed Hash for '%s': ", testCase);
        for (int j = 0; j < 32; j++) {
            printf("%02x", MsgHash[j]);
        }
```

```
        printf("\n");
    }

    return 0;
}
```

- 验证过程

```
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm3# nano SM3.c
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm3# gcc -o SM3 SM3.c -lm
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm3# ./SM3
SM3 SelfTest passed.
Computed Hash for 'xlm':
99f620e94508ee9445bf0722bac8d9d9942cd1a9821f99b2e9e416960e926596
Computed Hash for '20221414xlm20221414xlm':
39d0ccbf88ba55a3de4ad2b370288eee8d37f900ae14afff3ba95095cbf1b352
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm3# echo -n "xlm" | gmssl sm3
99f620e94508ee9445bf0722bac8d9d9942cd1a9821f99b2e9e416960e926596
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm3# echo -n
"20221414xlm20221414xlm" | gmssl sm3
39d0ccbf88ba55a3de4ad2b370288eee8d37f900ae14afff3ba95095cbf1b352
```

- SM4测试
  - 代码
    - main函数(特点：从命令行接受加密数据，按PKCS#7填充数据)

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

#define KEY_SIZE 16  // 定义密钥长度为16字节
#define BLOCK_SIZE 16  // SM4 block size is 16 bytes

void pkcs7_padding(unsigned char* input, int input_length,
unsigned char* output, int* output_length) {
    int padding_length = BLOCK_SIZE - (input_length % BLOCK_SIZE);
    *output_length = input_length + padding_length;

    // Copy input to output and apply PKCS#7 padding
    memcpy(output, input, input_length);
    memset(output + input_length, padding_length, padding_length);
}

void pkcs7_unpadding(unsigned char* input, int input_length,
unsigned char* output, int* output_length) {
    // Get padding value
    int padding_value = input[input_length - 1];
    *output_length = input_length - padding_value;
```

```
    memcpy(output, input, *output_length);
}

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

    // Self-check
    if (SM4_SelfCheck() == 0) {
        printf("自检成功! \n");
    }
    else {
        printf("自检失败! \n");
        return 1; // Self-check failed, exit program
    }

    // Generate random key
    unsigned char key[KEY_SIZE];
    srand((unsigned int)time(NULL)); // 使用当前时间作为随机数种子

    for (int i = 0; i < KEY_SIZE; i++) {
        key[i] = rand() % 256; // 生成0-255之间的随机数
    }

    // Output generated key
    printf("生成的随机密钥: ");
    for (int i = 0; i < KEY_SIZE; i++) {
        printf("%02x", key[i]);  // Hexadecimal output without
space
    }
    printf("\n");

    // Prepare plaintext
    const char* plaintext_input = argv[1];
    unsigned char plaintext[BLOCK_SIZE] = { 0 }; // Buffer for
plaintext
    unsigned char padded_plaintext[BLOCK_SIZE] = { 0 }; // Buffer
for padded plaintext
    unsigned char ciphertext[BLOCK_SIZE] = { 0 }; // Buffer for
ciphertext
    unsigned char decrypted[BLOCK_SIZE] = { 0 }; // Buffer for
decrypted text
    int padded_length = 0;

    // Apply PKCS#7 padding to the plaintext
    pkcs7_padding((unsigned char*)plaintext_input,
strlen(plaintext_input), padded_plaintext, &padded_length);

    // Encrypt plaintext
    SM4_Encrypt(key, padded_plaintext, ciphertext);
    printf("明文 (%s) 的密文: ", plaintext_input);
    for (int i = 0; i < BLOCK_SIZE; i++) {
```

```
        printf("%02x", ciphertext[i]);  // Hexadecimal output
without space
    }
    printf("\n");

    // Decrypt ciphertext
    SM4_Decrypt(key, ciphertext, decrypted);

    // Remove PKCS#7 padding from decrypted text
    int decrypted_length = 0;
    pkcs7_unpadding(decrypted, BLOCK_SIZE, plaintext,
&decrypted_length);
    plaintext[decrypted_length] = '\0'; // Null-terminate the
decrypted plaintext for printing

    printf("解密后的文本: %s\n", plaintext);

    return 0;
}
```

- PKCS#7填充的Python脚本代码

```
import sys

def pad_data(data, block_size=16):
    padding_length = block_size - (len(data) % block_size)
    if padding_length == 0:
        padding_length = block_size
    padding = bytes([padding_length] * padding_length)
    return data + padding

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: python3 pad.py <input_file> <output_file>")
        sys.exit(1)

    input_file = sys.argv[1]
    output_file = sys.argv[2]

    try:
        with open(input_file, 'rb') as f:
            data = f.read()

        padded_data = pad_data(data)

        with open(output_file, 'wb') as f:
            f.write(padded_data)

        print(f"Padded data written to {output_file}")
    except Exception as e:
        print(f"Error: {e}")
        sys.exit(1)
```

- 移除PKCS#7填充的Python脚本代码

```
import sys

def pkcs7_unpadding(data):
    # 获取最后一个字节的值，这个值表示填充的字节数
    padding_value = data[-1]
    # 解除填充
    return data[:-padding_value]

def main(input_file, output_file):
    try:
        # 读取输入文件
        with open(input_file, 'rb') as infile:
            padded_data = infile.read()

        # 解除PKCS#7填充
        unpadded_data = pkcs7_unpadding(padded_data)

        # 将解除填充后的数据写入输出文件
        with open(output_file, 'wb') as outfile:
            outfile.write(unpadded_data)

        print(f"解除填充成功！已将结果写入 '{output_file}'")

    except Exception as e:
        print(f"处理过程中出现错误：{e}")

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("用法：python remove_pkcs7_padding.py <输入文件名> <
输出文件名>")
        sys.exit(1)

    input_filename = sys.argv[1]
    output_filename = sys.argv[2]

    main(input_filename, output_filename)
```

- 验证过程中生成的密文一致

```
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# ls
SM4  SM4.c  SM4.h  ciphertext.bin  gmssl  key.bin  pad.py
padded_plaintext.bin  plaintext.txt
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# rm SM4.c
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# nano SM4.c
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# gcc SM4.c -o SM4 -lm
```

```
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# ./SM4
Usage: ./SM4 <plaintext>
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# ./SM4 xlm
自检成功!
生成的随机密钥: 31fbbe5b98a83c039b489f269340f2e5
明文 (xlm) 的密文: 772fc9106617fae43f654e30f577fc0d
解密后的文本: xlm
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# echo -n "xlm" >
plaintext.txt
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# python3 pad.py
plaintext.txt padded_plaintext.bin
Padded data written to padded_plaintext.bin
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# gmssl sm4_ecb -encrypt -key
31fbbe5b98a83c039b489f269340f2e5 -in padded_plaintext.bin -out
ciphertext.bin
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# xxd ciphertext.bin
00000000: 772f c910 6617 fae4 3f65 4e30 f577 fc0d  w/..f...?eN0.w..
```

- 验证解密结果一致

```
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# nano unpad.py
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# gmssl sm4_ecb -decrypt -key
31fbbe5b98a83c039b489f269340f2e5 -in ciphertext.
bin -out decrypted_padded.bin
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# python3 unpad.py
decrypted_padded.bin plaintext_unpadded.txt
Error: 'bool' object is not iterable
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# rm unpad.py
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# nano unpad.py
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# python3 unpad.py
decrypted_padded.bin plaintext_unpadded.txt
解除填充成功！已将结果写入 'plaintext_unpadded.txt'
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm4# cat plaintext_unpadded.txt
xlmroot@Youer:~/shiyan/shiyan02/shiyan2-1/sm4#
```

- SM2测试
  - MIRACL的安装与编译
    - 先在GitHub上查看其安装教程
    - 根据教程查看其Linux安装说明，将教程发给AI辅助理解

```
root@Youer:~/shiyan/MIRACL# cat linux.txt
RedHat Linux 6.0+ MIRACL i386 x86-32 installation

Also works OK for Solaris if its x386/Pentium based.

1. Unzip the MIRACL.ZIP file using the utility unzip, into an
empty directory

unzip -j -aa -L miracl.zip
```

```
The -j ignores the directory structure inside MIRACL.ZIP. The -aa
converts all
text files to Unix format, and -L ensures that all filenames are
lower-case.


2. Perform a tailored build of the MIRACL library by opening a
terminal
window, and typing

bash linux

3. All the MIRACL applications (except RATCALC) can then be built,
as
desired. Remember to link all C applications to the miracl.a
library.
C++ applications must be linked as well to one or more of big.o
zzn.o
ecn.o crt.o flash.o object files etc.
See the xxx.bat files for examples. Some applications that require
floating-point support may also require -lm in the compile command
line.


Make sure that your Linux PATH points to the current directory, so
that
executables can be run.

Some programs may require some small changes. For example in
schoof.cpp search
for the comment about "platforms".

Note that Linux already has (a rather pathetic) factor program. To
avoid name
clashes you might rename MIRACL's "factor" program to "facter", or
somesuch.



For a 64-bit build, on for example an AMD64 or a Core 2 processor
(x86-64), use

bash linux64


IMPORTANT

Some files might have to be changed from Windows File format, to
Unix file format
in order for programs to work correctly.
```

- 提前准备32位的环境

- 下载编译MIRACL的步骤（注意，由于代码在x86环境下运行，所以要编译32位的库：bash linux64改为bash linux）

```
root@Youer:~/shiyan# mkdir MIRACL
root@Youer:~/shiyan# cd MIRACL
root@Youer:~/shiyan/MIRACL# cp /mnt/d/xlm20/*.zip ./
root@Youer:~/shiyan/MIRACL# ls
MIRACL-master.zip
root@Youer:~/shiyan/MIRACL# mv MIRACL-master.zip miracl.zip
root@Youer:~/shiyan/MIRACL# unzip -j -aa -L miracl.zip
root@Youer:~/shiyan/MIRACL# bash linux
rm: cannot remove '*.exe': No such file or directory
rm: cannot remove 'miracl.a': No such file or directory
```

- 使用./pk-demo测试安装是否成功

```
root@Youer:~/shiyan/MIRACL# ./pk-demo
First Diffie-Hellman Key exchange ....

Alice's offline calculation
Bob's offline calculation
Alice calculates Key=
628193311933964507916727012456668690048412388675031629828376318549
788078675212119978001212962012815491611724550843583198751416895792
785831457420331911474835871318456852790769090064633348652194982871
229042164105260864750743258650766446089579122935525785967705810516
196128581343118114100769424009718264572844 78
Bob calculates Key=
628193311933964507916727012456668690048412388675031629828376318549
788078675212119978001212962012815491611724550843583198751416895792
785831457420331911474835871318456852790769090064633348652194982871
229042164105260864750743258650766446089579122935525785967705810516
196128581343118114100769424009718264572844 78
Alice and Bob's keys should be the same!

Lets try that again using elliptic curves....
Alice's offline calculation
Bob's offline calculation
Alice calculates Key=
148314637080074523495262414480507408456365955740429185 4909
Bob calculates Key=
148314637080074523495262414480507408456365955740429185 4909
Alice and Bob's keys should be the same! (but much smaller)

Testing El Gamal's public key method
Ciphertext=
140255713304004004668624720945909560572393838948991569929447772941
466994138516912896882002839080247711875606570491130030070634963102
430979675692734455673380681288145766312886268439457719765465813548
907360604954844942549174569085591090851846168483323192099687288583
```

```
653325214341544290477898912617460728384274563
601389634297563742481788208787902109816803843345483901020318718090
659095255641479443473095382612827554362575499980620939888818325070
314066737255647166427416982757255263140767226213719241645488258383
601334124096855184118289117991502240265638813719001724300322654595
005870477699134132092455912553933135310893 80
Plaintext=
MIRACL - Best multi-precision library in the World!


Now generating 512-bit random primes p and q
113403983748074141457125047640407720739728148216293548720373878762
179467351034734238875999443807740539671027032943389836374550543378
57774025489190903442169
114910870595307076233034599556217070085757583975137439633643887460
443654708110013048064757796342365314687757895516592294467226809834
3389204278636001406703 1
n = p.q =
130313505014672544175858966460143959752281046895918844621419284167
087756883685330891294585357728519313277894883126541858959795701010
001955662043493578178655558490078040180881467402818877404115378781
198940124305318141811635403340568315580177450282273501929747896464
766444067533526607935502701120618478998030239
Encrypting test string
Ciphertext=
300040780632969881410207317960994534804904631923707564685560727034
781614388649360120876934147085133055850331879753788684893207965551
919969160069071780793075584610196356690674867532183308216281182320
160128476391562382514556710402573792780308130472603253269634450809
538285539881964320740172339451567991022015 6
Decrypting test string
Plaintext=
MIRACL - Best multi-precision library in the World!

root@Youer:~/shiyan/MIRACL#
```

- 在miracl.h的第96行取消注释，避免找不到compare函数

```
/* To avoid name clashes - undefine this */

/* #define compare mr_compare */
```

- 使这个库可以在任意地方进行链接

```
root@Youer:~/shiyan/MIRACL# cp ./miracl.a /usr/lib/libmiracl_32.a
root@Youer:~/shiyan/MIRACL# mkdir /usr/include/miracl
root@Youer:~/shiyan/MIRACL# cp *.h /usr/include/miracl
```

- 使用说明：源代码中的与此相关的头文件都需要进行修改，修改为"miracl/XXX.h"即可
- 其他的参考对话、文章链接：
  - 与KIMI的对话
  - Linux下编译并使用miracl密码库
  - miracl里的compare不能调用的问题

- SM2加解密验证过程

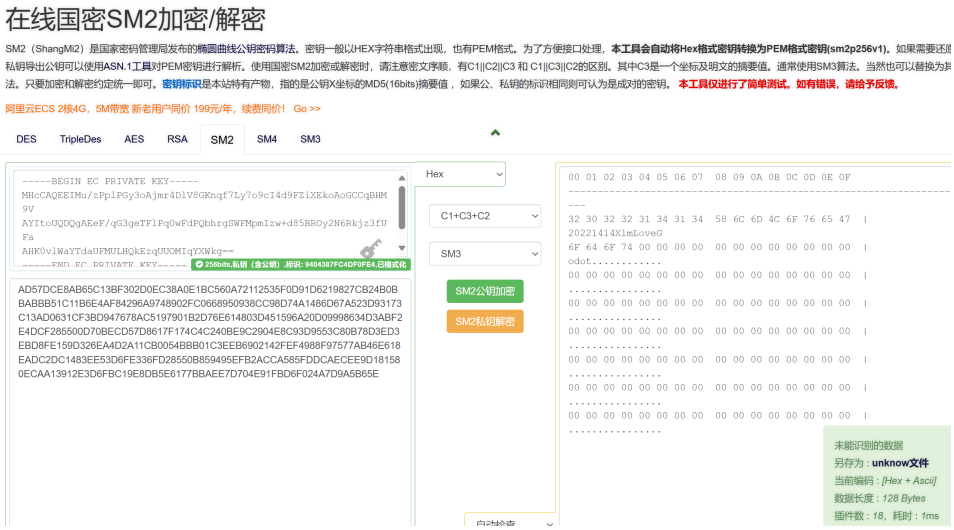
  - 说明：由于SM2算法加密依赖随机数生成器，一般不支持手动输入随机数，同时，不同随机数加密下密文不一致，所以我们只验证密文能否解密出正确的明文而不验证明文能否生成一样的密文。

  - 过程

    - 运行SM2_ENC，随机生成公私钥对，生成密文，验证程序是否正确运行

```
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm2/sm2_enc# nano
SM2_ENC.c
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm2/sm2_enc# gcc -m32
-o SM2_ENC SM2_ENC.c -lmiracl_32
In file included from SM2_ENC.c:4:
kdf.h: In function ‘CF’:
kdf.h:52:32: warning: result of ‘2055708042 << 16’ requires
48 bits to represent, but ‘int’ only has 32 bits [-Wshift-
overflow=]
52 | #define SM3_rotl32(x,n)  (((x) << n) | ((x) >> (32 -
n)))
    |                                ^~
kdf.h:161:29: note: in expansion of macro ‘SM3_rotl32’
161 |                         T = SM3_rotl32(SM3_T2, 16);
    |                             ^~~~~~~~~~
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm2/sm2_enc# ./SM2_ENC
SM2_ENC_SelfTest passed.
Private Key (hex):
CBBFCCFA653C6CB7A008E6AF80E557C18A9EA7FB2F2EE8F5C23877D159897
124
Print the
publicKey:785FEA1B781E4C594FAB4C0574F41B86B812585329988CF0F9D
F390513B2D8DE91923CF77D415A0072B4BE559A61375A5053142C7424133A
9450E308A985D692
Encryption and decryption successful!
Original Message: 20221414XlmLoveGodot
Ciphertext:
AD57DCE8AB65C13BF302D0EC38A0E1BC560A72112535F0D91D6219827CB24
B0BBABBB51C11B6E4AF84296A9748902FC0668950938CC98D74A1486D67A5
23D93173C13AD0631CF3BD947678AC5197901B2D76E614803D451596A20D0
9998634D3ABF2E4DCF285500D70BECD57D8617F174C4C240BE9C2904E8C93
D9553C80B78D3ED3EBD8FE159D326EA4D2A11CB0054BBB01C3EEB6902142F
EF4988F97577AB46E618EADC2DC1483EE53D6FE336FD28550B859495EFB2A
CCA585FDDCAECEE9D181580ECAA13912E3D6FBC19E8DB5E6177BBAEE7D704
E91FBD6F024A7D9A5B65E
Decrypted Message: 20221414XlmLoveGodot
```

- 通过在线网站进行验证(可以看到解密结果是20221414XlmLoveGodot)



- main函数代码 ```c #include <stdio.h> #include <string.h> #include <stdlib.h> #include <time.h>

```
#define MESSAGE_LEN 128 // Length of "xlm20221414"
#define CIPHER_LEN (SM2_NUMWORD * 3 + MESSAGE_LEN) // Length of
ciphertext

void printCiphertext(unsigned char* ciphertext, int len) {
    printf("Ciphertext: ");
    for (int i = 0; i < len; i++) {
        printf("%02X", ciphertext[i]); // Print each byte in
hexadecimal format
    }
    printf("\n");
}

// Function to print a 32-byte big private key
void printPrivateKey(big privateKey) {
    char keyBytes[32];
    int bytesWritten = big_to_bytes(32, privateKey, keyBytes,
TRUE);

    if (bytesWritten > 0) {
        printf("Private Key (hex): ");
        for (int i = 0; i < bytesWritten; i++) {
            printf("%02X", (unsigned char)keyBytes[i]);
        }
        printf("\n");
    }
    else {
        printf("Error converting big to bytes.\n");
    }
}
```

```
void handleDecryptionError(int errorCode) {
    switch (errorCode) {
    case -1:
        printf("Error: Invalid private key.\n");
        break;
    case -2:
        printf("Error: Ciphertext format is invalid.\n");
        break;
    case -3:
        printf("Error: Decryption failed due to other
reasons.\n");
        break;
    default:
        printf("Unknown decryption error.\n");
        break;
    }
}

int main() {

    // Define variables
    big privateKey;              // Private key
    epoint* publicKey;           // Public key
    unsigned char message[MESSAGE_LEN + 1] =
"20221414XlmLoveGodot"; // Message to encrypt
    unsigned char ciphertext[CIPHER_LEN]; // Buffer for ciphertext
    unsigned char decryptedMessage[MESSAGE_LEN + 1]; // Buffer for
decrypted message
    unsigned char randK[32];     // Random K value
    unsigned char priKeyBytes[SM2_NUMWORD]; // Random private key
bytes
    unsigned char kGxy[SM2_NUMWORD * 2] = { 0 };
    int result;
    result = SM2_ENC_SelfTest();
    if (result == 0) {
        printf("SM2_ENC_SelfTest passed.\n");
    }
    else {
        printf("SM2_ENC_SelfTest failed with error code: %d\n",
result);
    }
    // Initialize random seed
    srand(time(NULL));

    // Initialize MIRACL
    mip = mirsys(1000, 16); // Set up the MIRACL library with
enough space
    mip->IOBASE = 16;       // Set the base to hexadecimal
    big x = mirvar(0);
    big y = mirvar(0);
```

```
    // Allocate and initialize big numbers and epoints
    privateKey = mirvar(0);
    publicKey = epoint_init();

    // Generate a random private key
    for (int i = 0; i < 32; i++) {
        priKeyBytes[i] = rand() % 256; // Generate random bytes
for private key
    }
    bytes_to_big(32, priKeyBytes, privateKey);

    // Initialize SM2 parameters
    if (SM2_Init() != 0) {
        printf("SM2 initialization failed.\n");
        return -1;
    }

    // Generate public key from private key
    if (SM2_KeyGeneration(privateKey, publicKey) != 0) {
        printf("Key generation failed.\n");
        return -1;
    }

    // Generate a random K value
    for (int i = 0; i < 32; i++) {
        randK[i] = rand() % 256; // Generate random bytes for K
    }

    // Encrypt the message
    if (SM2_Encrypt(randK, publicKey, message, MESSAGE_LEN,
ciphertext) != 0) {
        printf("Encryption failed.\n");
        return -1;
    }

    // Print the privateKey
    printPrivateKey(privateKey);

    // Print the publicKey
    printf("Print the publicKey:");
    epoint_get(publicKey, x, y);
    big_to_bytes(SM2_NUMWORD, x, kGxy, 1);
    big_to_bytes(SM2_NUMWORD, y, kGxy + SM2_NUMWORD, 1);
    for (int i = 0; i < SM2_NUMWORD * 2; i++) {
        printf("%02X", (unsigned char)kGxy[i]);
    }
    printf("\n");

    // Decrypt the ciphertext
    result = SM2_Decrypt(privateKey, ciphertext, CIPHER_LEN,
decryptedMessage);
```

```
    if (result != 0) {
        printf("Decryption failed with error code: %d\n", result);
        handleDecryptionError(result);
        return -1;
    }

    // Null-terminate the decrypted message
    decryptedMessage[MESSAGE_LEN] = '\0';

    // Compare original and decrypted messages
    if (strcmp((char*)message, (char*)decryptedMessage) == 0) {
        printf("Encryption and decryption successful!\n");
        printf("Original Message: %s\n", message);
        printCiphertext(ciphertext, CIPHER_LEN);
        printf("Decrypted Message: %s\n", decryptedMessage);
    }
    else {
        printf("Decryption failed. Original and decrypted messages
do not match.\n");
    }

    // Clean up
    epoint_free(publicKey);
    mirkill(privateKey);
    mip = mirsys(0, 16); // Reset MIRACL memory

    return 0;
}
```

- SM2签名和验签验证过程
  - 程序运行过程（运行自检函数）

```
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm2/sm2_sv# gcc -m32 -o
SM2_sv SM2_sv.c -lmiracl_32
In file included from SM2_sv.c:26:
KDF.h: In function ‘CF’:
KDF.h:50:32: warning: result of ‘2055708042 << 16’ requires 48
bits to represent, but ‘int’ only has 32 bits [-Wshift-overflow=]
50 | #define SM3_rotl32(x,n)  (((x) << n) | ((x) >> (32 - n)))
    |                                ^~
KDF.h:159:29: note: in expansion of macro ‘SM3_rotl32’
159 |                         T = SM3_rotl32(SM3_T2, 16);
    |                             ^~~~~~~~~~
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm2/sm2_sv# ./SM2_sv
pass!
```

  - main函数

```
int main() {
    int result = SM2_SelfCheck();
    if (result == 0) {
        printf("pass!\n");
    }
    else {
        printf("fail!\n");
    }
}
```

- SM2密钥交换程序运行过程
  - 修复错误代码(SM2_EKY_EX.c)
    - 错误代码

```
memcpy(IDlen, &(unsigned char)ELAN + 1, 1);
memcpy(IDlen + 1, &(unsigned char)ELAN, 1);
```

    - 正确代码

```
memcpy(IDlen, (unsigned char*)&ELAN + 1, 1);
memcpy(IDlen + 1, (unsigned char*)&ELAN, 1);
```

  - 过程(自检函数运行返回值为0，验证成功)

```
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm2/SM2_KEY_EX# nano
SM2_KEY_EX.c
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm2/SM2_KEY_EX# gcc -m32 -o
SM2_KEY_EX SM2_KEY_EX.c -lmiracl_32
In file included from SM2_KEY_EX.c:3:
KDF.h: In function ‘CF’:
KDF.h:51:32: warning: result of ‘2055708042 << 16’ requires 48
bits to represent, but ‘int’ only has 32 bits [-Wshift-overflow=]
51 | #define SM3_rotl32(x,n)  (((x) << n) | ((x) >> (32 - n)))
    |                                ^~
KDF.h:160:29: note: in expansion of macro ‘SM3_rotl32’
160 |                         T = SM3_rotl32(SM3_T2, 16);
    |                             ^~~~~~~~~~
SM2_KEY_EX.c: In function ‘SM3_Z’:
SM2_KEY_EX.c:95:23: error: lvalue required as unary ‘&’ operand
95 |         memcpy(IDlen, &(unsigned char)ELAN + 1, 1);
    |                       ^
SM2_KEY_EX.c:96:27: error: lvalue required as unary ‘&’ operand
96 |         memcpy(IDlen + 1, &(unsigned char)ELAN, 1);
    |                           ^
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm2/SM2_KEY_EX# nano
SM2_KEY_EX.c
```

```
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm2/SM2_KEY_EX# gcc -m32 -o
SM2_KEY_EX SM2_KEY_EX.c -lmiracl_32
In file included from SM2_KEY_EX.c:3:
KDF.h: In function ‘CF’:
KDF.h:51:32: warning: result of ‘2055708042 << 16’ requires 48
bits to represent, but ‘int’ only has 32 bits [-Wshift-overflow=]
51 | #define SM3_rotl32(x,n)  (((x) << n) | ((x) >> (32 - n)))
    |                                ^~
KDF.h:160:29: note: in expansion of macro ‘SM3_rotl32’
160 |                         T = SM3_rotl32(SM3_T2, 16);
    |                             ^~~~~~~~~~
root@Youer:~/shiyan/shiyan02/shiyan2-1/sm2/SM2_KEY_EX#
./SM2_KEY_EX
pass!
```

- main代码

```c
int main() {
    int result = SM2_KeyEX_SelfTest();
    if (result == 0) {
        printf("pass!\n");
    }
    else {
        printf("fail!\n");
    }
}
```

2. 在密标委网站http://www.gmbz.org.cn/main/bzlb.html查找SM2，SM3，SM4相关标准，分析代码实现与标准的对应关系。（6分）

- 相关标准文件链接
  - SM3：SM3 密码杂凑算法
  - SM4: SM4分组密码算法
  - SM2: 五个标准文件如下所示
    - SM2 椭圆曲线公钥密码算法第1部分：总则
    - SM2 椭圆曲线公钥密码算法第2部分：数字签名算法
    - SM2 椭圆曲线公钥密码算法第3部分：密钥交换协议
    - SM2 椭圆曲线公钥密码算法第4部分：公钥加密算法
    - SM2 椭圆曲线公钥密码算法第5部分：参数定义
- 分析代码实现与标准的对应关系
  - SM3：
    - 常量一致
      - 代码中（SM3.h）中定义的常量SM3_T1、SM3_T2与文件中一致，IV常量（SM3_IVA到SM3_IVH）也和文件一致
      - 通过SM3_init函数将初始值设为标准的初始值
    - 过程一致
      - 包括填充方式(SM3_done)、消息扩散(BiToW)、压缩方式(CF&SM3_compress)等均与标准一致

        - 具体过程
    - 始终通过BigEndian函数保证存储是大端模式
- SM4：
    - 常数一致：固定参数(SM4_CK)、S盒(SM4_Sbox)、系统参数FK(SM4_FK)等均与标准文件一致
    - 算法结构一致：代码实现的轮函数结构和合成置换与标准一致
    - 算法过程一致: 其轮密钥生成过程、加解密过程与标准一致
- SM2：
    - 参数一致（第5部分：参数定义）
        - 曲线一致：曲线方程都是[y^2 = x^3 + ax + b]，素数、a、b、阶n、坐标xG和yG都一致
            - p = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF2B8044DEF3C3BC32DCFD7F6C2C2649
            - a = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF2B8044DEF3C3BC32DCFD7F6C2C2648
            - b = 0x5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B
            - n = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123
            - 生成元
                - Gx = 0x32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7
                - Gy = 0xBC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0
            - 对应的代码(sm2头文件中均有相关定义)：

```
unsigned char SM2_p[32] =
{
0xFF,0xFF,0xFF,0xFE,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF };
unsigned char SM2_a[32] =
{
0xFF,0xFF,0xFF,0xFE,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFC };
unsigned char SM2_b[32] =
{
0x28,0xE9,0xFA,0x9E,0x9D,0x9F,0x5E,0x34,0x4D,0x5A,0x9E,0x4B,0xCF,0x65,0x09,0xA7,
0xF3,0x97,0x89,0xF5,0x15,0xAB,0x8F,0x92,0xDD,0xBC,0xBD,0x41,0x4D,0x94,0x0E,0x93 };
unsigned char SM2_n[32] =
{
0xFF,0xFF,0xFF,0xFE,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
```

```
0x72,0x03,0xDF,0x6B,0x21,0xC6,0x05,0x2B,0x53,0xBB,0xF4,0
x09,0x39,0xD5,0x41,0x23 };
unsigned char SM2_Gx[32] =
{
0x32,0xC4,0xAE,0x2C,0x1F,0x19,0x81,0x19,0x5F,0x99,0x04,0
x46,0x6A,0x39,0xC9,0x94,
0x8F,0xE3,0x0B,0xBF,0xF2,0x66,0x0B,0xE1,0x71,0x5A,0x45,0
x89,0x33,0x4C,0x74,0xC7 };
unsigned char SM2_Gy[32] =
{
0xBC,0x37,0x36,0xA2,0xF4,0xF6,0x77,0x9C,0x59,0xBD,0xCE,0
xE3,0x6B,0x69,0x21,0x53,
0xD0,0xA9,0x87,0x7C,0xC6,0x2A,0x47,0x40,0x02,0xDF,0x32,0
xE5,0x21,0x39,0xF0,0xA0 };
```

- 测试用例数据一致
  - 可以对比标准中附录的测试数据和代码中自检函数的数据是一致的
  - 比如加解密中的自检函数与标准一致
    - 自检函数中的数据（SM2_ENC.c / SM2_ENC_SelfTest）

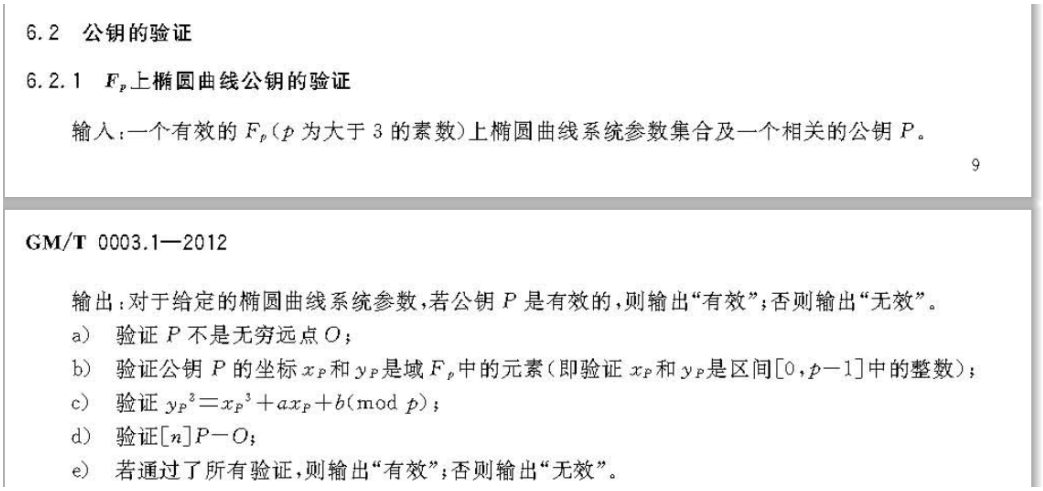
```
    unsigned char  std_priKey[32] =
    {
0x39,0x45,0x20,0x8F,0x7B,0x21,0x44,0xB1,0x3F,0x36,
0xE3,0x8A,0xC6,0xD3,0x9F,0x95,

0x88,0x93,0x93,0x69,0x28,0x60,0xB5,0x1A,0x42,0xFB,
0x81,0xEF,0x4D,0xF7,0xC5,0xB8 };
    unsigned char std_pubKey[64] =
    {
0x09,0xF9,0xDF,0x31,0x1E,0x54,0x21,0xA1,0x50,0xDD,
0x7D,0x16,0x1E,0x4B,0xC5,0xC6,

0x72,0x17,0x9F,0xAD,0x18,0x33,0xFC,0x07,0x6B,0xB0,
0x8F,0xF3,0x56,0xF3,0x50,0x20,

0xCC,0xEA,0x49,0x0C,0xE2,0x67,0x75,0xA5,0x2D,0xC6,
0xEA,0x71,0x8C,0xC1,0xAA,0x60,

0x0A,0xED,0x05,0xFB,0xF3,0x5E,0x08,0x4A,0x66,0x32,
0xF6,0x07,0x2D,0xA9,0xAD,0x13 };
    unsigned char std_rand[32] =
    {
0x59,0x27,0x6E,0x27,0xD5,0x06,0x86,0x1A,0x16,0x68,
0x0F,0x3A,0xD9,0xC0,0x2D,0xCC,

0xEF,0x3C,0xC1,0xFA,0x3C,0xDB,0xE4,0xCE,0x6D,0x54,
0xB8,0x0D,0xEA,0xC1,0xBC,0x21 };
    unsigned char std_Message[19] =
    {
0x65,0x6E,0x63,0x72,0x79,0x70,0x74,0x69,0x6F,0x6E,
0x20,0x73,0x74,0x61,0x6E, 0x64,0x61,0x72,0x64};
```

```
    unsigned char std_Cipher[115] =
    {
0x04,0xEB,0xFC,0x71,0x8E,0x8D,0x17,0x98,0x62,0x04,
0x32,0x26,0x8E,0x77,0xFE, 0xB6,

0x41,0x5E,0x2E,0xDE,0x0E,0x07,0x3C,0x0F,0x4F,0x64,
0x0E,0xCD,0x2E,0x14,0x9A,0x73,

0xE8,0x58,0xF9,0xD8,0x1E,0x54,0x30,0xA5,0x7B,0x36,
0xDA,0xAB,0x8F,0x95,0x0A,0x3C,

0x64,0xE6,0xEE,0x6A,0x63,0x09,0x4D,0x99,0x28,0x3A,
0xFF,0x76,0x7E,0x12,0x4D,0xF0,

0x59,0x98,0x3C,0x18,0xF8,0x09,0xE2,0x62,0x92,0x3C,
0x53,0xAE,0xC2,0x95,0xD3,0x03,

0x83,0xB5,0x4E,0x39,0xD6,0x09,0xD1,0x60,0xAF,0xCB,
0x19,0x08,0xD0,0xBD,0x87,0x66,

0x21,0x88,0x6C,0xA9,0x89,0xCA,0x9C,0x7D,0x58,0x08,
0x73,0x07,0xCA,0x93,0x09,0x2D,0x65,0x1E,0xFA};
```

- 哈希函数一致：代码用的哈希算法都是SM3算法(KDF.h中实现了SM3算法)，与标准要求的相同（标准第二、三、四部分的5.4.2）
- 测试数据一致
  - 比如验证sm2公钥有效性的函数与标准（第1部分：总则）是一致的
  - 具体代码(SM2_ENC.c / Test_PubKey)

```
int Test_PubKey(epoint* pubKey)
{

    big x, y, x_3, tmp;
    epoint* nP;
    x = mirvar(0);
    y = mirvar(0);
    x_3 = mirvar(0);
    tmp = mirvar(0);

    nP = epoint_init();

    //test if the pubKey is the point at infinity
    if (point_at_infinity(pubKey))// if pubKey is point at
```

```
infinity, return error;
        return ERR_INFINITY_POINT;
    //test if x<p   and  y<p  both hold
    epoint_get(pubKey, x, y);
    if ((compare(x, para_p) != -1) || (compare(y, para_p) !=
-1))
        return ERR_NOT_VALID_ELEMENT;

    if (Test_Point(pubKey) != 0)
        return ERR_NOT_VALID_POINT;

    //test if the order of pubKey is equal to n
    ecurve_mult(para_n, pubKey, nP);        // nP=[n]P
    if (!point_at_infinity(nP))           // if np is point
NOT at infinity, return error;
        return ERR_ORDER;
    return 0;
}
```



6.2 公钥的验证

6.2.1 F_p上椭圆曲线公钥的验证

输入：一个有效的 F_p（p 为大于 3 的素数）上椭圆曲线系统参数集合及一个相关的公钥 P。

9

GM/T 0003.1—2012

输出：对于给定的椭圆曲线系统参数，若公钥 P 是有效的，则输出“有效”；否则输出“无效”。

a) 验证 P 不是无穷远点 O；

b) 验证公钥 P 的坐标 x_P 和 y_P 是域 F_p 中的元素（即验证 x_P 和 y_P 是区间[0，$p-1$]中的整数）；

c) 验证 $y_P^2 = x_P^3 + ax_P + b \pmod p$；

d) 验证[n]$P=O$；

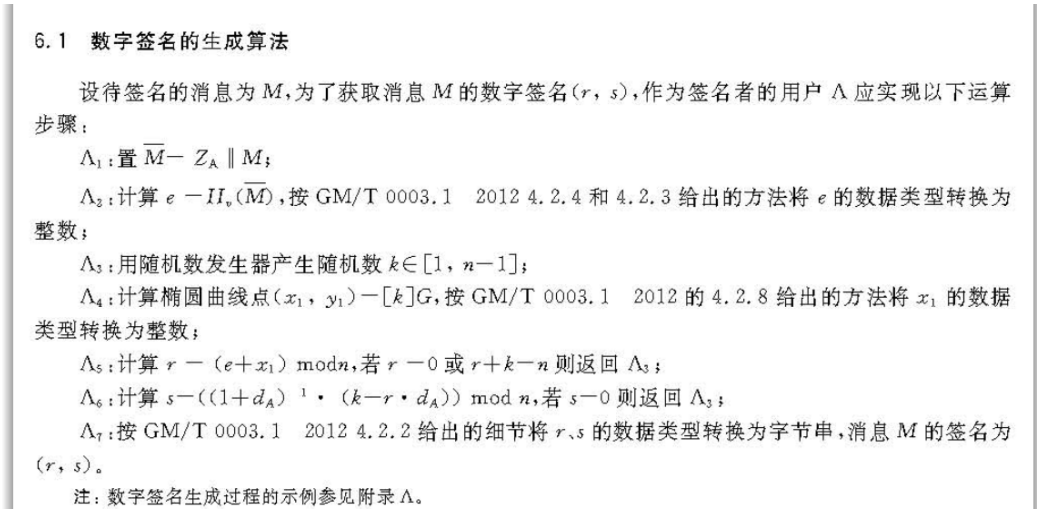
e) 若通过了所有验证，则输出“有效”；否则输出“无效”。

- 算法一致（以签名算法举例，对应的是第二部分的标准文件）
  - 签名代码

```
int SM2_Sign(unsigned char* message, int len, unsigned char
ZA[], unsigned char rand[], unsigned char d[], unsigned char
R[], unsigned char S[])
{
    unsigned char hash[SM3_len / 8];
    int M_len = len + SM3_len / 8;
    unsigned char* M = NULL;
    int i;
    big dA, r, s, e, k, KGx, KGy;
    big rem, rk, z1, z2;
    epoint* KG;

    i = SM2_Init();
    if (i)
        return i;
```

```
    //initiate
    dA = mirvar(0);
    e = mirvar(0);
    k = mirvar(0);
    KGx = mirvar(0);
    KGy = mirvar(0);
    r = mirvar(0);
    s = mirvar(0);
    rem = mirvar(0);
    rk = mirvar(0);
    z1 = mirvar(0);
    z2 = mirvar(0);

    bytes_to_big(SM2_NUMWORD, d, dA);//cinstr(dA,d);

    KG = epoint_init();

    //step1,set M=ZA||M
    M = (char*)malloc(sizeof(char) * (M_len + 1));
    memcpy(M, ZA, SM3_len / 8);
    memcpy(M + SM3_len / 8, message, len);

    //step2,generate e=H(M)
    SM3_256(M, M_len, hash);
    bytes_to_big(SM3_len / 8, hash, e);

    //step3:generate k
    bytes_to_big(SM3_len / 8, rand, k);

    //step4:calculate kG
    ecurve_mult(k, G, KG);

    //step5:calculate r
    epoint_get(KG, KGx, KGy);
    add(e, KGx, r);
    divide(r, n, rem);

    //judge r=0 or n+k=n? add(r,k,rk);
    if (Test_Zero(r) | Test_n(rk))
        return ERR_GENERATE_R;

    //step6:generate s
    incr(dA, 1, z1);
    xgcd(z1, n, z1, z1, z1);
    multiply(r, dA, z2);
    divide(z2, n, rem);
    subtract(k, z2, z2);
    add(z2, n, z2);
    multiply(z1, z2, s);
    divide(s, n, rem);

    //judge s=0?
    if (Test_Zero(s))
        return ERR_GENERATE_S;
```

```
    big_to_bytes(SM2_NUMWORD, r, R, TRUE);
    big_to_bytes(SM2_NUMWORD, s, S, TRUE);

    free(M);
    return 0;
}
```



6.1 数字签名的生成算法

设待签名的消息为 M，为了获取消息 M 的数字签名 (r, s)，作为签名者的用户 A 应实现以下运算步骤：

A_1：置 $\overline{M} = Z_A \| M$；

A_2：计算 $e = H_v(\overline{M})$，按 GM/T 0003.1 2012 4.2.4 和 4.2.3 给出的方法将 e 的数据类型转换为整数；

A_3：用随机数发生器产生随机数 $k \in [1, n-1]$；

A_4：计算椭圆曲线点 $(x_1, y_1) = [k]G$，按 GM/T 0003.1 2012 的 4.2.8 给出的方法将 x_1 的数据类型转换为整数；

A_5：计算 $r = (e + x_1) \bmod n$，若 $r = 0$ 或 $r + k = n$ 则返回 A_3；

A_6：计算 $s = ((1 + d_A)^{-1} \cdot (k - r \cdot d_A)) \bmod n$，若 $s = 0$ 则返回 A_3；

A_7：按 GM/T 0003.1 2012 4.2.2 给出的细节将 r、s 的数据类型转换为字节串，消息 M 的签名为 (r, s)。

注：数字签名生成过程的示例参见附录 A。

- 对应关系

3. 使用Rust完成SM2，SM3，SM4算法的实现（选做，10分）
4. 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

- gittee链接
- git log运行成果：

```
root@Youer:~/shiyan/shiyan02# git log
commit 5597341c77f05477aa42efa21b400a1777be3111 (HEAD -> master,
origin/master)
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sun Oct 27 11:30:09 2024 +0800

    shiyan2-1:finish sm2 key change code

commit 9e3f2dec7f102b14f7c393dc9484aee7bcddb4e3
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sun Oct 27 10:07:42 2024 +0800

    shiyan2-1:delete useless sm2_enc flie

commit c5062b0c65b4bbf31de34d7eef3e51ce6c4f929e
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Sat Oct 26 22:01:16 2024 +0800

    shiyan2-1:finish sm2_sv code

commit ef2c6667575c62aa55c974dd767d047f53c62526
Author: 徐鹿鸣 <xlm20040219@qq.com>
```

```
Date:   Fri Oct 25 18:18:31 2024 +0800

    finish sm2_enc run & fail hex to pem

commit 22d35b6852efbab24078f5d6486b980033a391b1
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 22 18:52:05 2024 +0800

    shiyan2-1:delete useless files

commit ffd89acdbb35fa50f2676e4d3e9c573c606aa266
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 22 18:50:40 2024 +0800

    shiyan2-1: finish sm4 test

commit 48e88aa69b71a073bd9e0677917c1b5eb5386c89
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Tue Oct 22 16:11:04 2024 +0800

    shiyan2-1: finish sm3 test

commit 59789be25f1bb42b435c279400ea8796d255eeeb
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Mon Oct 21 19:41:37 2024 +0800

    shiyan2-1: fail test sm3

commit d901fb34e852d0464e1057a088a4acecaa878817
Author: 徐鹿鸣 <xlm20040219@qq.com>
Date:   Mon Oct 21 19:36:21 2024 +0800

    first commit
(END)
```

5. 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
- 记录实验的问题与解决过程：
    - sm3的经验
        - 主要是pdf转word有很多错误，需要一一修改
        - 同时要理解自检函数的意义
            - 对于SM3算法，就是先预设好一个输入与其标准哈希值，再用自己实现的代码生成一个哈希值，通过判断两者是否相同判断实现是否正确

        - 在学习其他算法的实现时，这个函数也很值得先学习，它是自带的测试，自己构造数据、执行函数、验证结果
        - 这样便于我们来写测试代码

- sm4 密码模式判断与填充问题
- 首要的问题是理解加密模式
    - 可以看出其没有使用IV文件，不是娄老师使用的cbc加密模式。
    - 用AI判断这是ECB模式
- 同时其没有自动填充，所以要自己写填充函数。
    - 要确保代码中的填充模式和用命令实现的填充模式是一致的（这里都是PKCS#7填充）
- sm2运行环境问题
- 表现
    - 之前都在vs2022中运行代码，在那里要求以32位编译代码，因为MIRACL库是用32位做的
    - 在Ubuntu中，原先的MIRACL库是按64位进行编译的（因为我的Ubuntu就是64位）
    - 尽管在vs2022中运行良好，但只要我把代码移植到Ubuntu中，就会解密失败（报错代码：6）
    - 哪怕是自检函数都无法运行成功（报错代码：9）
    - 之前一直以为源代码与系统位数无关，并且编译的库的位数需要和系统位数一样，不然会报错，直到这时才意识到这可能与系统环境有关
- 解决问题的过程
    - 首先是如何在64位系统的编译32位代码
    - 按照AI的回复安装各种兼容的程序与C语言库
    - 修改编译命令为：gcc -m32 -o SM2_ENC SM2_ENC.c -lmiracl
    - 此时仍会报错，因为miracl是64位的，我们需要编译一个32位的miracl库
    - 编译32位的miracl库
    - 主要是生成一个miracl_32.a文件，因为头文件两者一般是一致的
    - 按照当初64位的编译方式编译，但改变“bash linux64”为“bash linux”
    - 将新生成的库复制到用户文件夹：cp ./miracl.a /usr/lib/libmiracl_32.a
    - 不需要复制新的头文件到用户文件夹中，同样，原先代码中的"miracl/miracl.h"等不需要改变，因为头文件不变
    - 修改编译命令为：gcc -m32 -o SM2_ENC SM2_ENC.c -lmiracl_32
- 一些原理的解释
    - 编译库的位数与系统位数的关系
    - 为什么头文件不变
    - 解释编译命令中的"-lmiracl_32"
- 经验教训
    - 像这种处理字节的代码对系统位数比较敏感，要额外注意其适用于什么操作系统

        - 重视自检函数的作用，优先判断其自检函数能否运行，学习将自检函数一步步转换成自己的测试函数

    - sm2密钥交换报错：

        - 报错代码：

```
    memcpy(IDlen, &(unsigned char)ELAN + 1, 1);
    memcpy(IDlen + 1, &(unsigned char)ELAN, 1);
```

        - 报错信息

```
SM2_KEY_EX.c: In function ‘SM3_Z’:
SM2_KEY_EX.c:95:23: error: lvalue required as unary ‘&’ operand
95 |         memcpy(IDlen, &(unsigned char)ELAN + 1, 1);
   |                       ^
SM2_KEY_EX.c:96:27: error: lvalue required as unary ‘&’ operand
96 |         memcpy(IDlen + 1, &(unsigned char)ELAN, 1);
   |                           ^
```

        - 分析问题：要抓住问题的本质
            - 第一步：明确代码是干什么的
            - 第二步：问有没有合法的其他有效表达
            - 具体过程
        - 最终方法：将问题代码改成正确代码：

```
    memcpy(IDlen, (unsigned char*)&ELAN + 1, 1);
    memcpy(IDlen + 1, (unsigned char*)&ELAN, 1);
```

        - 为什么vs2022可以运行原本的代码，Linux中却不行
            - 不同环境下C语言实现有差异。vs2022更加宽松一些。

    - 未解决的问题：理论上讲，我应该将hex格式的私钥转换成PEM格式的私钥，然后用gmssl命令进行解密来验证SM2，但这一过程遇到了解决不了的困难

        - 一：解密的命令刚需密码。需要将无密码的密钥改为有密码的密钥
        - 二：无法用代码将hex格式的密钥转换成可读的PEM格式的密钥（openssl命令验证失败）
        - 三：哪怕使用从在线网站生成的密钥也无法读取
        - 由此，所以sm2的验证过程都没能用gmssl命令验证
        - 可能的解决方法：
            - Openssl命令行:如何获取十六进制公钥的PEM，224位曲线?

- 经验教训

    - 先易后难：先做SM3，再做SM4，再做SM2。难度曲线非常平滑。

- 抓住问题本质：在解决“sm2密钥交换报错”报错上，理解代码，并尝试转换成其他合法表达，而不是直接去一步步调试