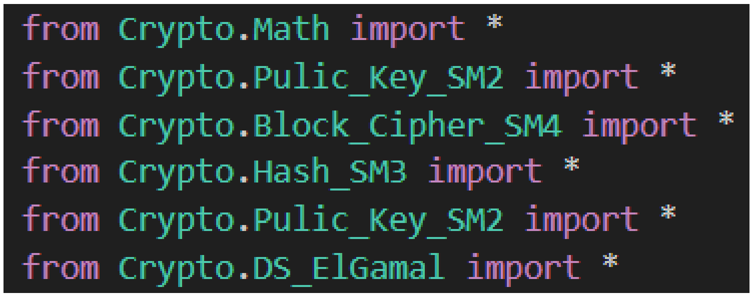
使用说明

1.密码库调用

直接使用from Crypto.xxx import *分别调用每个模块即可，如下图所示：



```
from Crypto.Math import *
from Crypto.Pulic_Key_SM2 import *
from Crypto.Block_Cipher_SM4 import *
from Crypto.Hash_SM3 import *
from Crypto.Pulic_Key_SM2 import *
from Crypto.DS_ElGamal import *
```

2.Math模块

各函数传入的参数及其类型以及函数的返回类型均已在源代码中标示：

1.`gcd(a: int, b: int)->int`

2.`gcdext(a: int, b: int)->tuple`

3.`invmod(a: int, m: int)->int`

4.`quick_pow(a: int, b: int, m: int)->int`

5.`isPrime(n: int)->bool`

6.`getPrime(n: int)->int`

7.`bytes_to_long(byte: bytes)->int`

8.`long_to_bytes(n: int)->bytes`

9.`hex_to_bin(h: 十六进制str)->二进制str`

10.`bin_to_hex(b: 二进制str)->十六进制str`

11.`bytes_to_bin(byte: bytes)->二进制str`

12.`bin_to_bytes(s: 二进制str)->bytes`

3.Block_Cipher_SM4模块

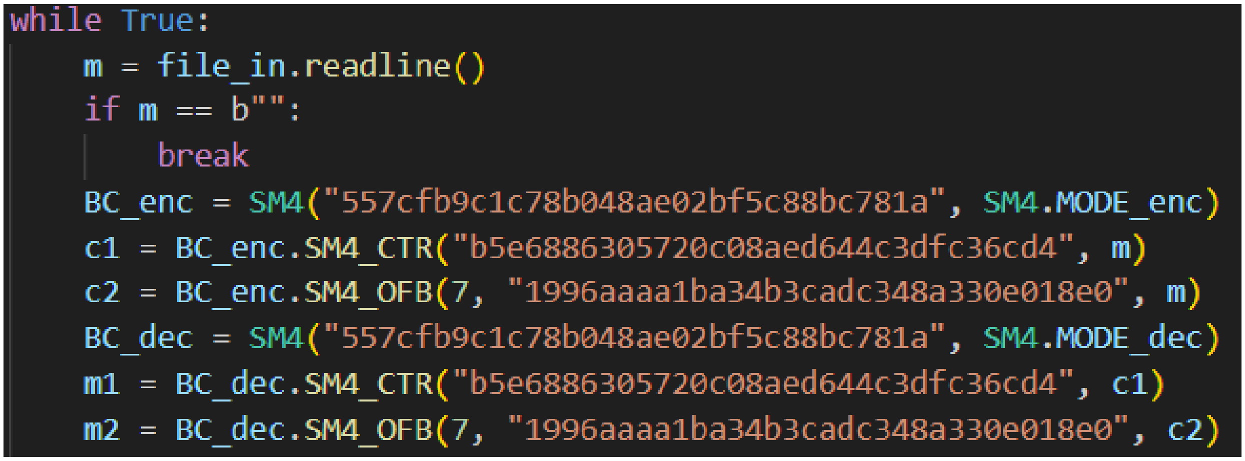
SM4类在创建对象时需要初始化密钥`key`（`key`为十六进制字符串且不含前缀`0x`）和加密/解密模式`op`（`op`为1或0，1为加密模式，0为解密模式）。

SM4包含CTR和OFB两种工作模式。

CTR模式还需初始化初始向量`IV`（`IV`为十六进制字符串且不含前缀`0x`）和明文/密文（均为字节串），得到的密文/明文也为字节串。

OFB模式还需要初始化反馈缓存器每次左移的字节位数`n`，初始向量`IV`（`IV`为十六进制字符串且不含前缀`0x`）和明文/密文（均为字节串），得到的密文/明文也为字节串。

示例：



```
while True:
    m = file_in.readline()
    if m == b"":
        break
    BC_enc = SM4("557cfb9c1c78b048ae02bf5c88bc781a", SM4.MODE_enc)
    c1 = BC_enc.SM4_CTR("b5e6886305720c08aed644c3dfc36cd4", m)
    c2 = BC_enc.SM4_OFB(7, "1996aaaa1ba34b3cadc348a330e018e0", m)
    BC_dec = SM4("557cfb9c1c78b048ae02bf5c88bc781a", SM4.MODE_dec)
    m1 = BC_dec.SM4_CTR("b5e6886305720c08aed644c3dfc36cd4", c1)
    m2 = BC_dec.SM4_OFB(7, "1996aaaa1ba34b3cadc348a330e018e0", c2)
```

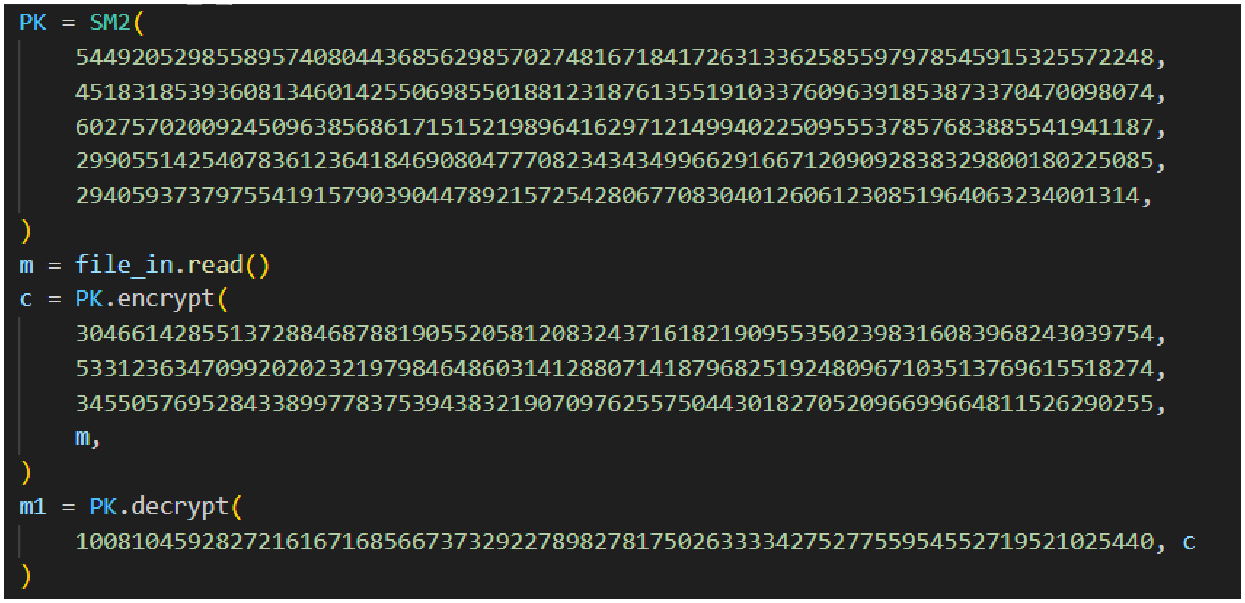
4.Public_Key_SM2模块

SM2类在创建对象时需要初始化椭圆曲线的两个参数`a`和`b`、参数中的大素数`p`、基点的横纵坐标`Gx`和`Gy`（上述5个参数均为整型）。

加密时还需要初始化公钥点`PB`的横纵坐标`PBx`和`PBy`、随机数`k`（上述3个参数均为整型）和明文（字节串类型），得到的密文也为字节串类型。

解密时还需要初始化私钥`dB`（整型）和密文（字节串类型），得到的明文也为字节串类型。

示例：



```
PK = SM2(
    54492052985589574080443685629857027481671841726313362585597978545915325572248,
    45183185393608134601425506985501881231876135519103376096391853873370470098074,
    60275702009245096385686171515219896416297121499402250955537857683885541941187,
    29905514254078361236418469080477708234343499662916671209092838329800180225085,
    2940593737975541915790390447892157254280677083040126061230851964063234001314,
)
m = file_in.read()
c = PK.encrypt(
    30466142855137288468788190552058120832437161821909553502398316083968243039754,
    53312363470992020232197984648603141288071418796825192480967103513769615518274,
    34550576952843389977837539438321907097625575044301827052096699664811526290255,
    m,
)
m1 = PK.decrypt(
    10081045928272161671685667373292278982781750263333427527755954552719521025440, c
)
```

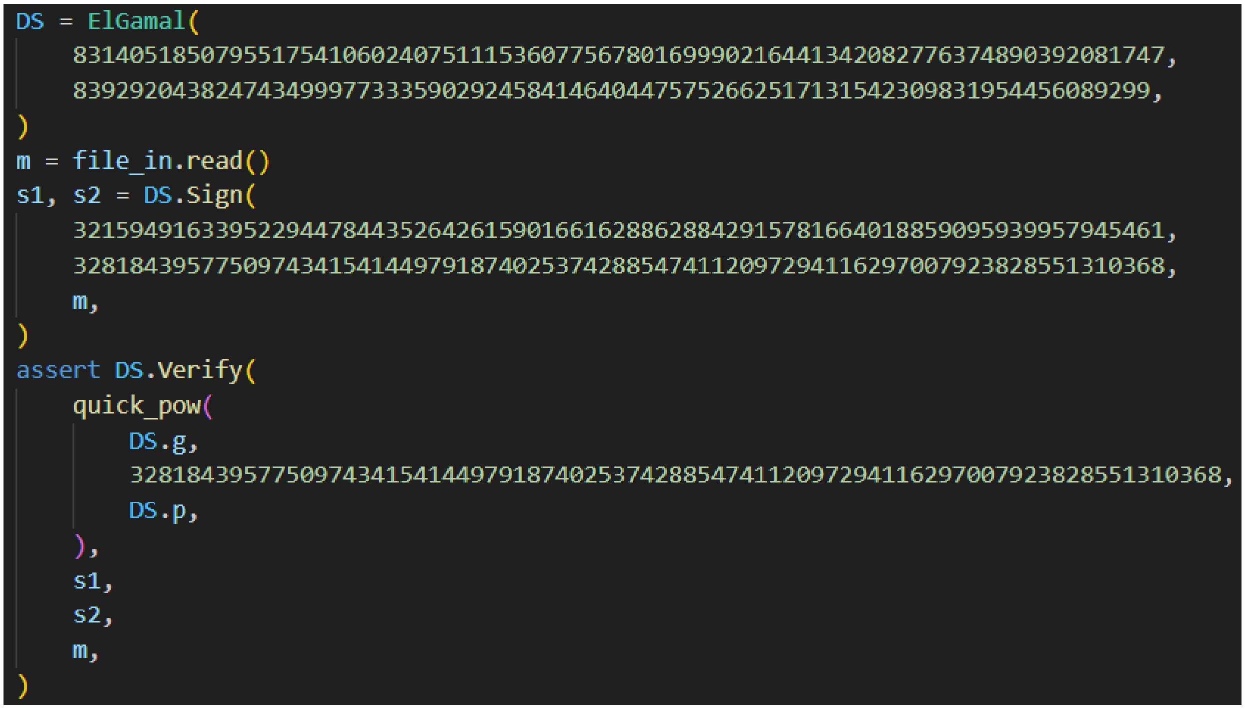
5.DS_ElGamal模块

ElGamal模块在创建对象时需要初始化大素数`p`和生成元`g`（上述2个参数均为整型）。

数字签名时还需要初始化随机数`k`、私钥`x`（上述两个参数均为整型）、消息（字节串），返回一个包含两个整型`s1、s2`的元组。

数字签名验签时还需要初始化公钥`y`、签名结果`s1、s2`（上述3个参数均为整型）和消息（字节串），返回一个bool值。

示例：

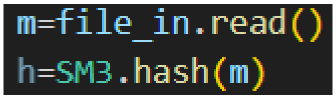


```
DS = ElGamal(
    83140518507955175410602407511153607756780169990216441342082776374890392081747,
    8392920438247434999773335902924584146404475752662517131542309831954456089299,
)
m = file_in.read()
s1, s2 = DS.Sign(
    32159491633952294478443526426159016616288628842915781664018859095939957945461,
    32818439577509743415414497918740253742885474112097294116297007923828551310368,
    m,
)
assert DS.Verify(
    quick_pow(
        DS.g,
        32818439577509743415414497918740253742885474112097294116297007923828551310368,
        DS.p,
    ),
    s1,
    s2,
    m,
)
```

6.Hash_SM3模块

直接调用`SM3.hash(M)`即可得到消息`M`的哈希值，其中要求`M`是字节串，返回的哈希值是十六进制字符串。

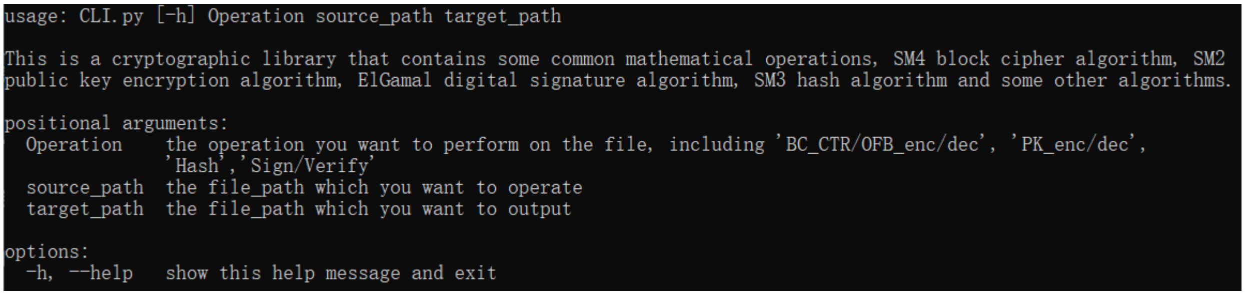
示例：



```
m=file_in.read()
h=SM3.hash(m)
```

7.CLI交互界面

命令行输入的结构为**python CLI.py [-h] Operation source_path target_path**，其中Operation代表需要执行的操作，包含“BC_CTR/OFB_enc/dec”, “PK_enc/dec”, “Hash”,“Sign/Verify”几种，source_path代表需要执行操作的文件的路径，target_path表示执行操作后的文件内容的存储路径。其中，对空文件亦能够执行操作。



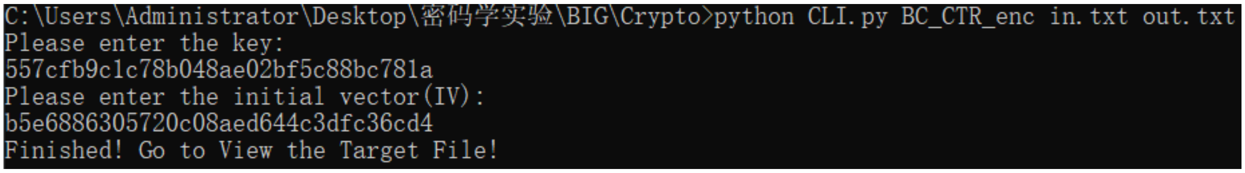
```
usage: CLI.py [-h] Operation source_path target_path

This is a cryptographic library that contains some common mathematical operations, SM4 block cipher algorithm, SM2
public key encryption algorithm, ElGamal digital signature algorithm, SM3 hash algorithm and some other algorithms.

positional arguments:
  Operation    the operation you want to perform on the file, including 'BC_CTR/OFB_enc/dec', 'PK_enc/dec',
               'Hash','Sign/Verify'
  source_path  the file_path which you want to operate
  target_path  the file_path which you want to output

options:
  -h, --help   show this help message and exit
```

（1）BC_CTR_enc/dec

输入密钥`key`→输入初始向量`IV`→执行操作

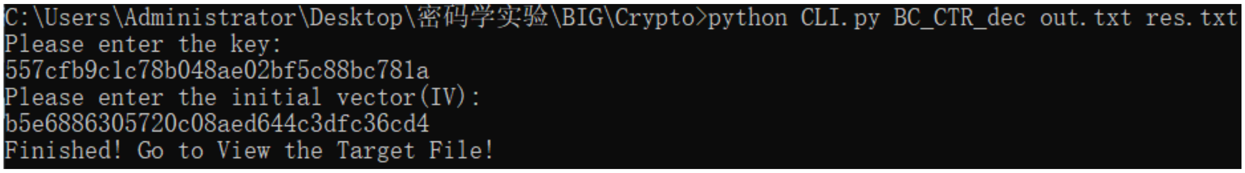
示例：

加密操作：



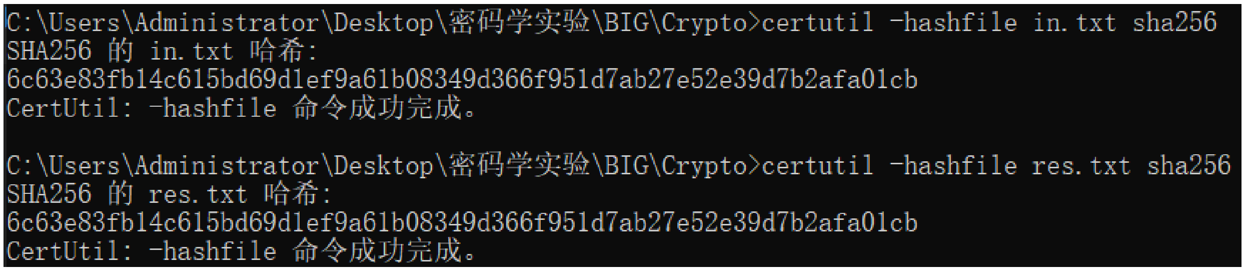
```
C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>python CLI.py BC_CTR_enc in.txt out.txt
Please enter the key:
557cfb9c1c78b048ae02bf5c88bc781a
Please enter the initial vector(IV):
b5e6886305720c08aed644c3dfc36cd4
Finished! Go to View the Target File!
```

解密操作：



```
C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>python CLI.py BC_CTR_dec out.txt res.txt
Please enter the key:
557cfb9c1c78b048ae02bf5c88bc781a
Please enter the initial vector(IV):
b5e6886305720c08aed644c3dfc36cd4
Finished! Go to View the Target File!
```

结果比对：



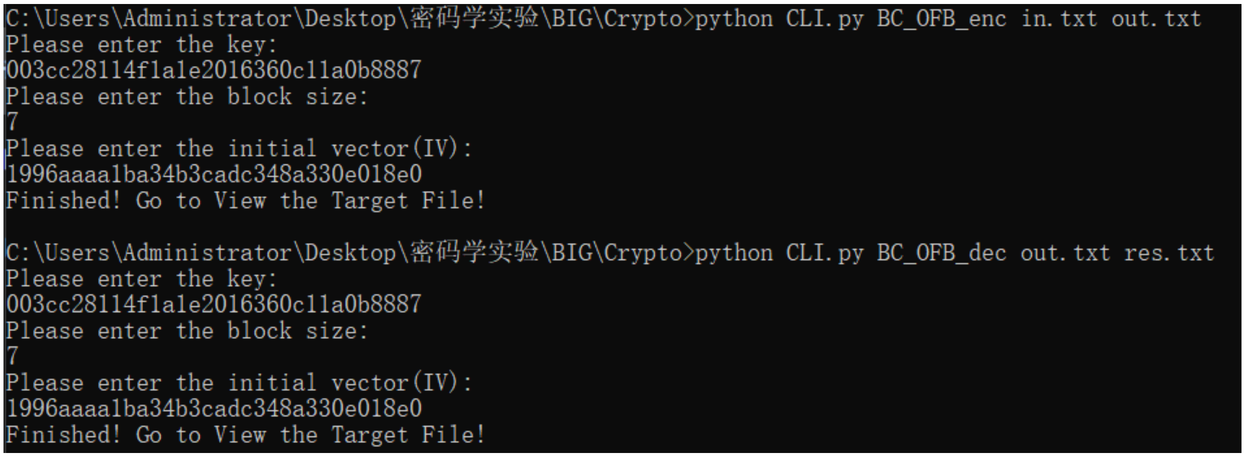
```
C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>certutil -hashfile in.txt sha256
SHA256 的 in.txt 哈希:
6c63e83fb14c615bd69d1ef9a61b08349d366f951d7ab27e52e39d7b2afa01cb
CertUtil: -hashfile 命令成功完成。

C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>certutil -hashfile res.txt sha256
SHA256 的 res.txt 哈希:
6c63e83fb14c615bd69d1ef9a61b08349d366f951d7ab27e52e39d7b2afa01cb
CertUtil: -hashfile 命令成功完成。
```

（2）BC_OFB_enc/dec

输入密钥`key`→输入移位字节位数`n`→输入初始向量`IV`→执行操作

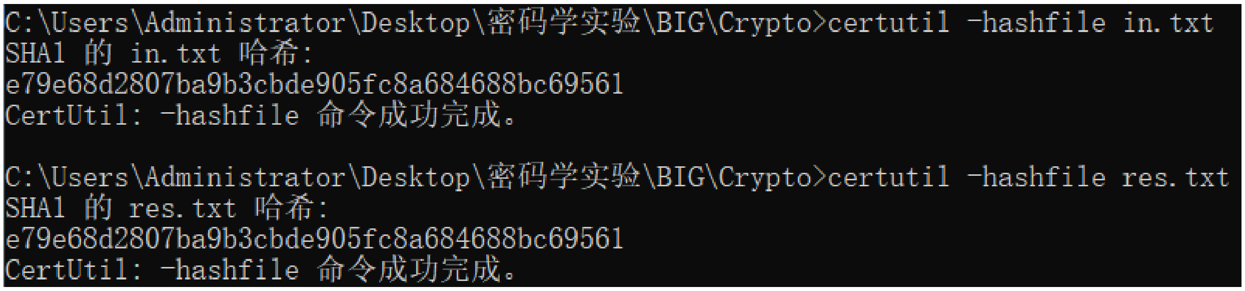
示例：

加解密操作：



```
C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>python CLI.py BC_OFB_enc in.txt out.txt
Please enter the key:
003cc28114f1a1e2016360c11a0b8887
Please enter the block size:
7
Please enter the initial vector(IV):
1996aaaa1ba34b3cadc348a330e018e0
Finished! Go to View the Target File!

C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>python CLI.py BC_OFB_dec out.txt res.txt
Please enter the key:
003cc28114f1a1e2016360c11a0b8887
Please enter the block size:
7
Please enter the initial vector(IV):
1996aaaa1ba34b3cadc348a330e018e0
Finished! Go to View the Target File!
```

结果比对：

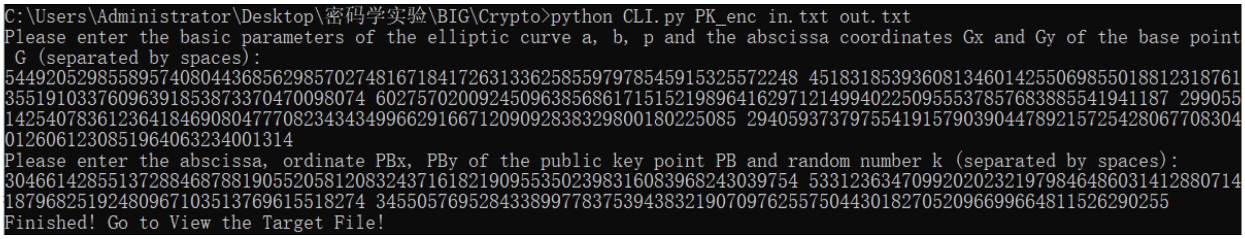


```
C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>certutil -hashfile in.txt
SHA1 的 in.txt 哈希:
e79e68d2807ba9b3cbde905fc8a684688bc69561
CertUtil: -hashfile 命令成功完成。

C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>certutil -hashfile res.txt
SHA1 的 res.txt 哈希:
e79e68d2807ba9b3cbde905fc8a684688bc69561
CertUtil: -hashfile 命令成功完成。
```

（3）PK_enc

输入参数`a, b, p, Gx, Gy`→输入公钥`PBx, PBy`、随机数`k`→执行操作

示例：

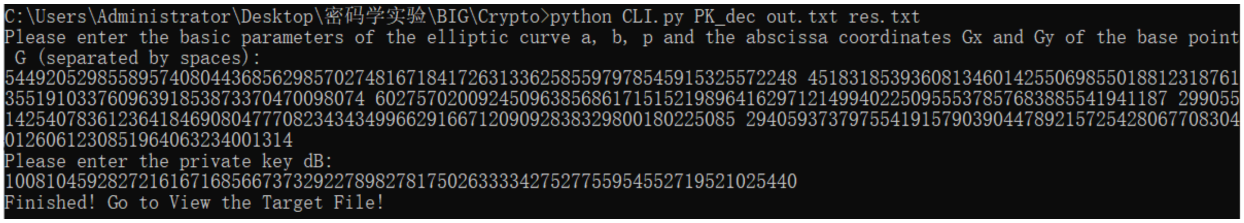


```
C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>python CLI.py PK_enc in.txt out.txt
Please enter the basic parameters of the elliptic curve a, b, p and the abscissa coordinates Gx and Gy of the base point
 G (separated by spaces):
54492052985589574080443685629857027481671841726313362585597978545915325572248 451831853936081346014255069855018812318761
35519103376096391853873370470098074 602757020092450963856861715152198964162971214994022509555378576838855419411 87 299055
14254078361236418469080477708234343499662916671209092838329800180225085 294059373797554191579039044789215725428067708304
0126061230851964063234001314
Please enter the abscissa, ordinate PBx, PBy of the public key point PB and random number k (separated by spaces):
30466142855137288468788190552058120832437161821909553502398316083968243039754 533123634709920202321979846486031412880714
18796825192480967103513769615518274 34550576952843389977837539438321907097625575044301827052096699664811526290255
Finished! Go to View the Target File!
```

（4）PK_dec

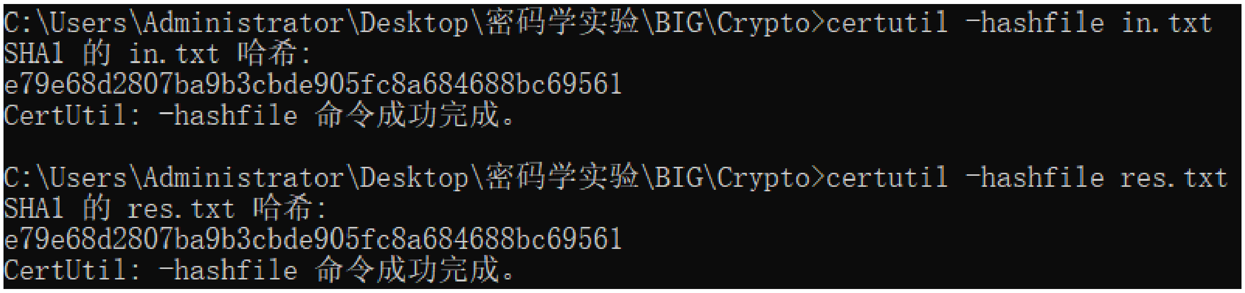
输入私钥`dB`→执行操作

示例：



```
C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>python CLI.py PK_dec out.txt res.txt
Please enter the basic parameters of the elliptic curve a, b, p and the abscissa coordinates Gx and Gy of the base point
 G (separated by spaces):
54492052985589574080443685629857027481671841726313362585597978545915325572248 451831853936081346014255069855018812318761
35519103376096391853873370470098074 602757020092450963856861715152198964162971214994022509555378576838855419411 87 299055
14254078361236418469080477708234343499662916671209092838329800180225085 294059373797554191579039044789215725428067708304
0126061230851964063234001314
Please enter the private key dB:
10081045928272161671685667373292278982781750263333427527755954552719521025440
Finished! Go to View the Target File!
```

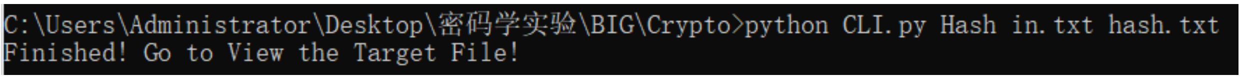
(3)(4)的结果比对：



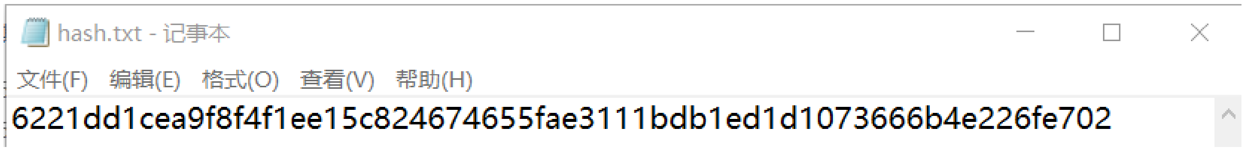
```
C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>certutil -hashfile in.txt
SHA1 的 in.txt 哈希:
e79e68d2807ba9b3cbde905fc8a684688bc69561
CertUtil: -hashfile 命令成功完成。

C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>certutil -hashfile res.txt
SHA1 的 res.txt 哈希:
e79e68d2807ba9b3cbde905fc8a684688bc69561
CertUtil: -hashfile 命令成功完成。
```

（5）Hash

直接对文件执行操作



```
C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>python CLI.py Hash in.txt hash.txt
Finished! Go to View the Target File!
```

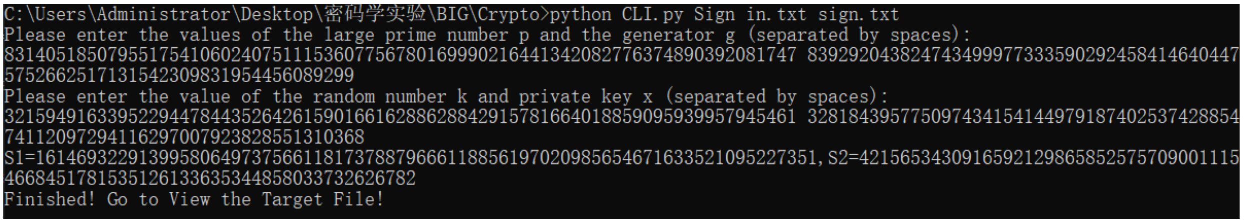


```
6221dd1cea9f8f4f1ee15c824674655fae3111bdb1ed1d1073666b4e226fe702
```

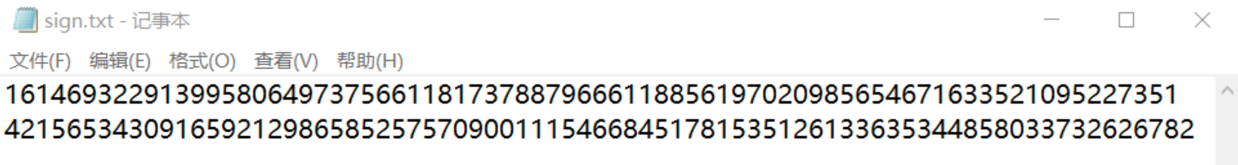
（6）Sign

输入大素数`p`、生成元`g`→输入随机数`k`、私钥`x`→执行操作

示例：



```
C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>python CLI.py Sign in.txt sign.txt
Please enter the values of the large prime number p and the generator g (separated by spaces):
83140518507955175410602407511153607756780169990216441342082776374890392081747 8392920438247434999773335902924584146404475752662517131542309831954456089299
Please enter the value of the random number k and private key x (separated by spaces):
32159491633952294478443526426159016616288628842915781664018859095939957945461 32818439577509743415414497918740253742885474112097294116297007923828551310368
S1=1614693229139958064973756611817378879666118856197020985654671633521095227351,S2=42156534309165921298658525757090011154668451781535126133635344858033732626782
Finished! Go to View the Target File!
```

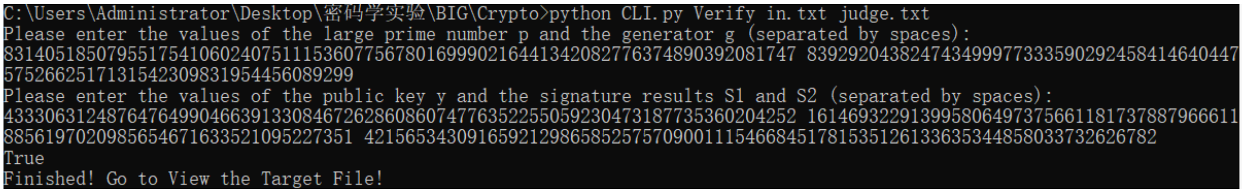


```
1614693229139958064973756611817378879666118856197020985654671633521095227351
42156534309165921298658525757090011154668451781535126133635344858033732626782
```

（7）Verify

输入公钥`y`、签名结果`S1、S2`→执行操作

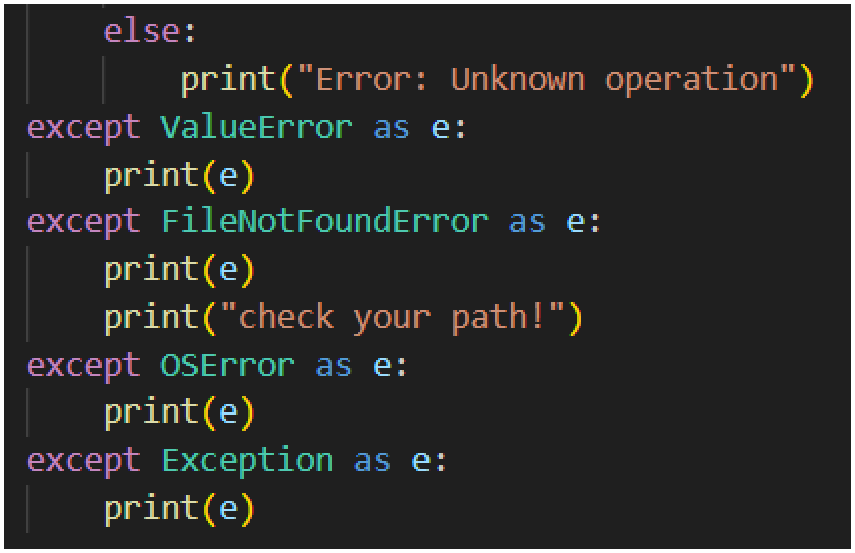
示例：



```
C:\Users\Administrator\Desktop\密码学实验\BIG\Crypto>python CLI.py Verify in.txt judge.txt
Please enter the values of the large prime number p and the generator g (separated by spaces):
83140518507955175410602407511153607756780169990216441342082776374890392081747 8392920438247434999773335902924584146404475752662517131542309831954456089299
Please enter the values of the public key y and the signature results S1 and S2 (separated by spaces):
43330631248764764990466391330846726286086074776352255059230473187735360204252 1614693229139958064973756611817378879666118856197020985654671633521095227351 42156534309165921298658525757090011154668451781535126133635344858033732626782
True
Finished! Go to View the Target File!
```

（8）错误处理

主要判断**ValueError**、**FileNotFoundError**、**OSError**、**Exception**几种错误。

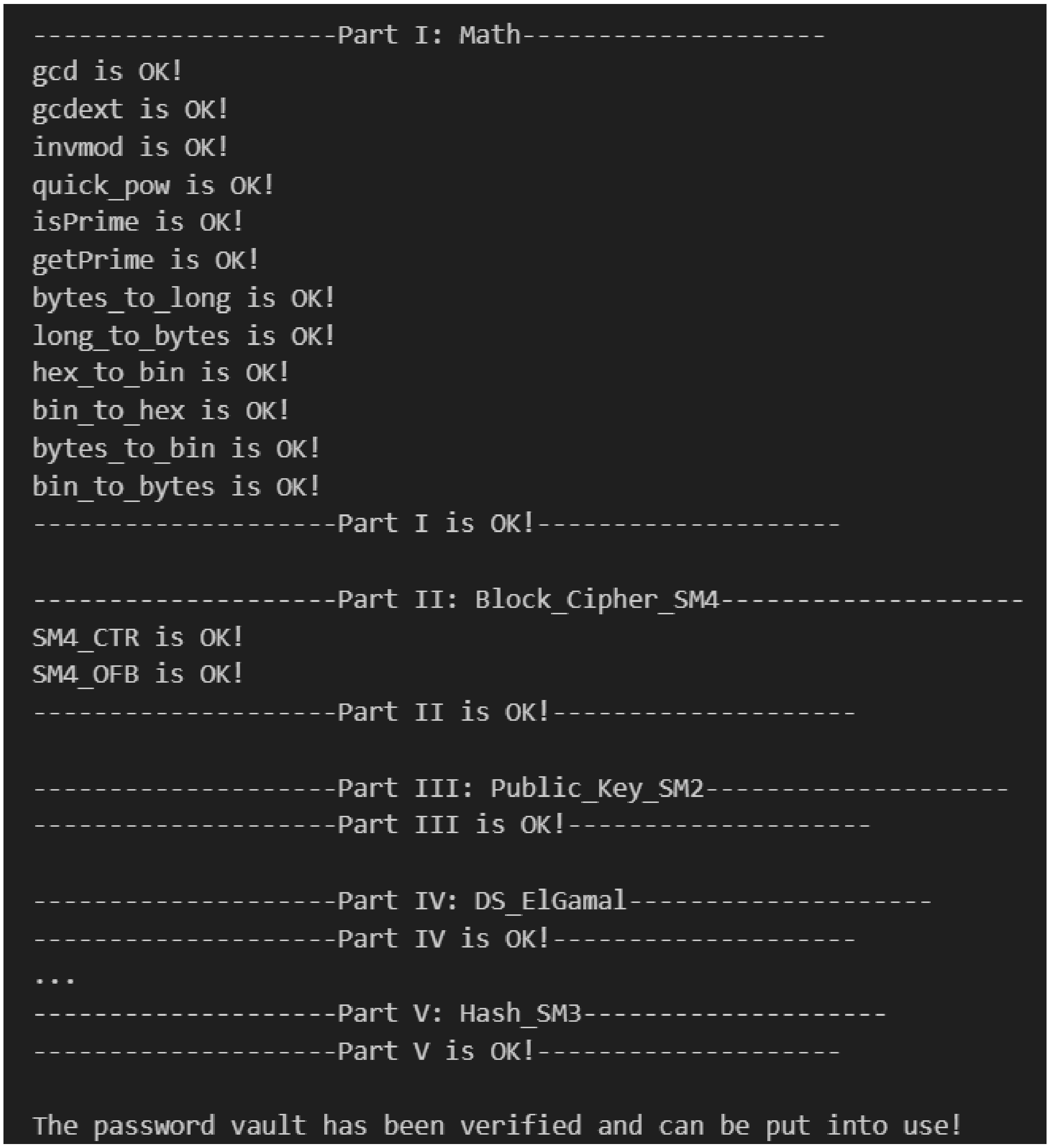


```
    else:
        print("Error: Unknown operation")
except ValueError as e:
    print(e)
except FileNotFoundError as e:
    print(e)
    print("check your path!")
except OSError as e:
    print(e)
except Exception as e:
    print(e)
```

8.test.py和Collection文件

直接运行即可，可生成函数调用关系图。

运行结果：



```
--------------------Part I: Math--------------------
gcd is OK!
gcdext is OK!
invmod is OK!
quick_pow is OK!
isPrime is OK!
getPrime is OK!
bytes_to_long is OK!
long_to_bytes is OK!
hex_to_bin is OK!
bin_to_hex is OK!
bytes_to_bin is OK!
bin_to_bytes is OK!
--------------------Part I is OK!--------------------

--------------------Part II: Block_Cipher_SM4--------------------
SM4_CTR is OK!
SM4_OFB is OK!
--------------------Part II is OK!--------------------

--------------------Part III: Public_Key_SM2--------------------
--------------------Part III is OK!--------------------

--------------------Part IV: DS_ElGamal--------------------
--------------------Part IV is OK!--------------------
...
--------------------Part V: Hash_SM3--------------------
--------------------Part V is OK!--------------------

The password vault has been verified and can be put into use!
```