user's Guide

1. Introduction

This is a an user's Guide of my Crypto Library. In this article, there are some introductions and precautions about how to use this Algorithm library, for example, I will show you what you should input and what you may get from output.

2. Illustration

1)Math_Crypto.py

Firstly, you should to import `Math_Crypto.py` ,then you can use any functions in this file. What have to be aware of is that if you wanna use the function: fastpower, you need to understand the meaning of these three parameters: calculate $a^b\%c$.

This part is not object-oriented.

```
import Math_Crypto.py
a = 6
b = 7
c = 8
print(fastpower(a, b, c))
# the result is 0
```

2)SM2.py

SM2 is a digital signature function. There are two functions: sign and verify.

Whatever you want to do, first create an object.

```
from SM2 import SM2
sm2 = SM2(a, b, p, G, n)
```

five init parameters: a, b, p, G, n
a, b, p are parameters of the elliptic curve
G is the base point of the elliptic curve
n is the order of the base point

```
result = sm2.sign(SM2_sign_ID_A, SM2_sign_P_A, SM2_sign_M, SM2_sign_d_A,
SM2_sign_k)
```

the parameters of sign() you should input are:ID_A, P_A, M, d_A, k
ID_A: the ID of the signer (a string)
P_A: the public key of the signer (a point on the elliptic curve)
M: the message you wanna sign (a string)
d_A: the private key of the signer (a number)
k: a random number (a number)

return: Signature of A (bytes type)

Steps to use Veriry function is the same as sign function.

```
result = sm2.verify(SM2_verify_ID_A, SM2_verify_P_A, SM2_verify_M, SM2_verify_r, SM2_verify_s)
```

the parameters of verify() you should input are:ID_A, P_A, M, r, s
ID_A: the ID of the signer (a string)
P_A: the public key of the signer (a point on the elliptic curve)
M: the message you wanna check signature (a bytes)
r: the first part of the signature (a number)
s: the second part of the signature (a number)

return: True if the signature is valid, False otherwise

3)SM3.py

First thing is to create an objection.

```
sm3 = SM3()
```

you dont need to input any inital parameters. Just pass parameters to this objection has been created .

```
result = sm3.hash_sm3(SM3_MSG)
```

the parameters of hash() you should input are: msg
msg: the message you wanna hash (a string or bytes)

4)SM4.py

Whatever you want to do, first thing you should do is to create an objection.

```
sm4 = SM4(key)
```

The parameter 'key' a hexadecimal number starting with '0x', the length is 34.

i.simply encryption or decryption

if you just want to simply encrypt/decrypt, please input: SM4_encrypt/SM4_decrypt
the parameter is: SM4_encrypt/SM4_decrypt [plaintext/ciphertext]

```
result = sm4.SM4_encrypt(SM4_plain)
```

plain_text or ciphertext: 128bit int

ii. CTR mode

if you want to use CTR mode, please input: SM4_CTR(file_path, IV, mode)
the parameter is: SM4_CTR [file_path] [IV] [mode]
mode is 0 or 1, 0 means encrypt, 1 means decrypt
IV is a 16 bytes hex format string, like 0x0123456789abcdef, it is the initial vector
file_path is the file you want to encrypt/decrypt

If you use encryption mode, program will return a new file created in current folder and its name will be file_path + '.SM4_CTR'. And if you use decryption mode, program will return a new file created in current folder and its name will be file_path without '.SM4_CTR'

Example:

```
# encryption mode
file_path_SM4CTR = 'test data/SM4 CTR data.txt'
SM4_encryption_CTR_key = '0x557cfb9c1c78b048ae02bf5c88bc781a'
IV_SM4_CTR = 0xb5e6886305720c08aed644c3dfc36cd4
sm4 = SM4(SM4_encryption_CTR_key)
result = sm4.SM4_CTR(file_path_SM4CTR, IV_SM4_CTR, 1)

# decryption mode
file_path_SM4CTR = 'test data/SM4 CTR data.txt..SM4_CTR'
SM4_encryption_CTR_key = '0x557cfb9c1c78b048ae02bf5c88bc781a'
IV_SM4_CTR = 0xb5e6886305720c08aed644c3dfc36cd4
sm4 = SM4(SM4_encryption_CTR_key)
result = sm4.SM4_CTR(file_path_SM4CTR, IV_SM4_CTR, 0)
```

iii.CFB mode

the parameter is: SM4_CFB [file_path] [IV] [mode]
mode is 0 or 1, 0 means encrypt, 1 means decrypt')
IV is a 16 bytes hex format string, like 0x0123456789abcdef, it is the initial vector
file_path is the file you want to encrypt/decrypt

If you use encryption mode, program will return a new file created in current folder and its name will be file_path + '.SM4_CFB'. And if you use decryption mode, program will return a new file created in current folder and its name will be file_path without '.SM4_CFB'

Example:

```
# encryption mode
file_path_SM4CFB = 'test data/SM4 CFB data.txt'
SM4_encryption_CFB_key = '0x04ab5f1f059edc1d283fb746004847d2'
IV_SM4_CFB = '0xcfd5e738c3887d647181484813ebf90e'
sm4 = SM4(SM4_encryption_CFB_key)
result = sm4.SM4_CFB(file_path_SM4CFB, IV_SM4_CFB, 7, 1)

# decryption mode
file_path_SM4CFB = 'test data/SM4 CFB data.txt.SM4_CFB'
SM4_encryption_CFB_key = '0x04ab5f1f059edc1d283fb746004847d2'
IV_SM4_CFB = '0xcfd5e738c3887d647181484813ebf90e'
sm4 = SM4(SM4_encryption_CFB_key)
result = sm4.SM4_CFB(file_path_SM4CFB, IV_SM4_CFB, 7, 0)
```

5)RSA.py

Whichever you choose, first step is to create an objection.

```
from RSA import RSA
rsa = RSA(e, N)
```

parameter e is the Decryption Index or Encryption Index. And N is modulus.

i. generate a big prime number

you can use keyGenerateGivenBitLength(N). It will return a pair of public keys: e and n.

The parameter N is the bitLength of p and q. You should input a number.

ii. simple encryption and decryption

if you wanna do RSA encryption or decryption, you should use the following function
encrypt(m) -- m is the plaintext, and the return value is the ciphertext
what should be noted is that the plaintext should be a number, not a string
decrypt(c) -- c is the ciphertext, and the return value is the plaintext
what should be noted is that the ciphertext should be a number, not a string

Example:

```
from RSA import RSA
e =
29204095224029265976271914881206362445387242700229981669810375836941828897450721
53997951709132903469153746154058782302049361904975953920415583179235627460458533
89782245078373000946801740677894785513463 5
p =
90847652794599737227031051225784344635704605484460117827955310966316087113867580
39869846624407809636935725454210198201714484752683287324692998717028735972189703
00686463533318033838476581283016731917331 9
q =
65809503921945612010821222237983223854841427225462272741260198683353072550956765
62482247070488705415319999541622125183113185633569436801807262284795751129437784
57627584304132525988962784939253169192487
N = p * q
msgRSA =
82671889971987093831022162955288618670504296687091223813624817977421146352977 50
rsa = RSA(e, N)
result = rsa.encrypt(msgRSA)
```

iii. RSA-OAEP encryption and decryption

if you wanna do OAEP encryption or decryption, you should use the following function
OAEP_Encryption(m, L) -- m is the plaintext, L is the label
what should be noted is that the plaintext should be a hex number, not a string so do the other params
OAEP_Decryption(c, L) -- c is the ciphertext, L is the label
what should be noted is that the ciphertext should be a hex number, not a string so do the other params

Example:

```
# encryption mode
k = 128
NN =
0xc32a1dbd27a8b57617cde48c424ff5936d6415a679a019605deb0f1a4a3b29c73770d77fcb4f3e
d36c4f2c94fb17e8c32feb0674b11d340c7734ffc4ca19401b00edca55b34234d6e6459022fe7b31
8e222b35435aaffcc430990acda67199035118ae140f80fc5b0dafaa3df9f9300a03163404039082
78cfe3b9aca2aa6bf1
ee = 0x10001
```

```
m =
'0xb96af050d20a2f68f1b1b68a90139ff3ef62c2c5527d41122bc6c2bfa59c300dea4791e2e9d9b
74518042bc860'
L = '0x'
seed = '0xf72f68a0abb2c333c489800626349e8ea7ecd4da'
rsa = RSA(ee, NN)
result = rsa.OAEP_encryption(k, m, L, seed)

# decryption mode
k = 512
NN =
0xd561ae21b504e8b34f96611a2afea71a927cd2f80a763b6f116d619b1c414f74cf26b50255ef11
1fdb1a169b3f1388766a36bd9dd42312fe90b1698247e0336b2d73992b1d6f236ceb049e2ebabdbd
b66f5bf956fba33dda1ddfeec40f878dd318a254533cb123e62cd22401915e811a0420714b3cb488
6734ce5869f754e85a82842e7ce3f8b1ab37ea36b625d4f2370b6cfe93af965562c01e33968dac31
0b305bd9aadca3a99bd1c6e1b34905762cd434f5a8bc76891e65ecf8b8d22b5803f84795cfa400e8
a7da52cafd53f0634fde71b206cbece56785e48da5056b8a6fee4a5df8549f4c85f5c378862e7c33
9852c25ec76122ab8617f2ce87003be2815d88d5464ab4fda04a63265aa579b3ad434a779ee35647
0aefcd638469e9e7e07d47f518e32f609976ba2b55230a4c80bb3ff7de65a6fd5b1b6bb454b98e04
d7baba9e23c8be59376c96f27c2fddc079a720ecb71508c25ec7af6c89ed1cfbcbf28caaf7d0ea23
44d1545967ef0dcd9f83794c57b333eb9fa5f3a1ebf757985289fe9f7286f640d59b8b687cbf1fb8
ef194f9279f6684b69d44014ce4dc581d2c4571c94026748d71c6025290885528b1c9842b58a7400
1022ed8d88082403d2c0379d289c44fe1223041fc7e74ff13e392e8e63a86d0d67397f45d62b3e58
d6bbaff9cabe5e748a37fdb6bad47e2245f9d9484c044be0fdce09cf94d89da60f
dd =
0x57c9e74d60df5329aa9dd0f76626fcf316bf9a4088755a4078e80a973949b979f25c176e8d925c
f6413b7b7f85b4f098df5c04293204f82499bd140e90a9b21def4637072105ff8a292aee358588d8
cb7d07570f2d5a5cf127076181726e2fb060db305c4a850aa3015fb786999b4afc3c2468d82e728c
080216bf1361b9a21ad2c5b52696d853a880c9b096c1fbd9674221269ff646a07c8f541d9f5c4c32
a54ad10f3d97a711f9d495c177f49bf5a1bdf0c7fd1304a5a74701f6d67b496fddba4121669b2ff0
88566f67aa4fe57fc6755b4b8636521193c4e7624b396ea43726cf9bd7b6610fa7b194293af35710
17243d69a717d49ec2fd895ce4f7f8ab58132cb592101faa79105573603b2785565f44df34a03780
d33124f635fd1a6bd3b8cefbcfbbed603b29aa0c777e933c9bccec44dea9b86ccd143234db686dcf
4216f3b7b6014e06b9e366423e832c40b6b1f7aac6ec83d8d45ae71ad97b9fc084c386594b50666c
6cba1207e6149e848af16296ba674949679e811b0ba899380435aa7226147be8236e1866dd6bd7fe
fb1c41537111ae1b54c9e30444ae4497b36f13789482bccd3d3ef72f395a20a216ac95b191bcb8a4
aec0ac48506155863ae7812287330c90618c09810fc961ef750759b55188859caf43b6bd11d30b38
911b2c9b4d9e288d97ffd69de28c560da8d5a09942488c0a2c068200ba0131afe1
c =
'0x5cb545b00acadd8a12088ee58da7a25279e6b1b4347c986cee5f0710fe361864376c6e8264812
1e2e57b40b38f969f47c905211d6cb28fb88aaafcecbf6655b891f70e10d7efa6dca3b58b249ab2a
ff3d7cf8f2509bcea00f5ba43f9c6124a07fc5e938654d0d70d709b73ab1013a34d969c950e2ebce
1279364a0df88a89e74b27fc9f8cfee6423b5b61a3a08aba385991835ff749a7b4098cbb7ca0e0f6
a5255f7e3174d9dcfccc6fc8c7e834bbd14f2c58ac33094b277413b84620a5296ca9a2c8486d152e
a3fb1025f2d8ec88a467d277650b04166e761649eb833b9083c09c8940a2f6c1981bc0839bbbbc60
8cb41e72d56671648d9bac0eca47240cd4ccadbd9c100f01913af27457a40dc25dad53c90c297ebb
1e7dfcdf7c53cd24cc4f68f034b77f583c8e58892d44113a4a73d565df68fc69e15fbfaec4643d22
f14f376ea8f4edc3902aa68f7b39d42c0ba34b6eaaa0872ea1a739a40e26bdf34b58430e159440c6
72a1a9da48f0f1eb5c328eea541e6bcda9d85eac47f2185616fabc0025f25529b27ac36a880a7cae
088942f489627fd2f51c13f5d9713d3d00ae5915da6ac3d611a0bd891103249ebb2e8b5603d7e3ad
e797fc92378115fadad98b95203770db7b96c0538b191afc82a2928e163b8b0852534f3e5905b5476
55707695b5b986db502e1737e1c813b76409855bf966f6429d171e31cfa11d733e4'
L = '0x'
rsa = RSA(dd, NN)

result = rsa.OAEP_decryption(k, c, L)
```

- **k**: An integer k, representing the security parameter of the RSA algorithm (such as k=1024 / 8 = 128 given for RSA-1024)
- **m**: A hexadecimal number, starting with 0x, representing plaintext; the length does not exceed k - 2hLen - 2
- **c**:A hexadecimal number, starting with 0x, representing ciphertext; the length does not exceed k - 2hLen - 2
- **L**: A hexadecimal number, starting with 0x, representing the label; the length does not exceed 2^61 - 1
- **seed**: A hexadecimal number, starting with 0x, representing a random number; the length is hLen
- **return**: A hexadecimal number, starting with 0x, representing ciphertext or plaintext

The End

Thank you for using my CryptoLibrary. If possible, please give me a star on **[Crypto_Lab_Final_Project]** (GitHub link for this project), thank you very much.

By the way, all the usage of functions and classes could be viewed in method: help().