Programming Assignment 3.2 (Elgamal) Description

# Group number :

**two**

# Group member :

**方舟, 张帅, 任中杰, 邱磊, 乔鹏宇, 王天蔚**

# 1. Overview of this project

This project is an encryption and decryption implementation based on Elgamal secret strategy wihch uses the stream cipher strategy to encrypt the message we want to pass, and use public key and secure key system to encrypt the stream key final use the signature systeam to do the verification.

Before we show our code, we give a brief description of the Elgamal strategy with signature system (supposing that the sk keeper is Alice , and the pk keeper is Bob):

## Step 1(choose primes and generator) :

- Alice and Bob negotiate the p (2048bits primer) and g (the generator of multiplicative group),
- p , g are known for everyone

## Step 2 (generate sk, pk):

- Alice choose a **sk** 1 < sk < p-2
- compute the YA = $g^{sk}$ mod q
- pass the public key as the form of **pk = {q, g ,YA}** to Bob

## Step 3 (Alice sign):

- Alice choose a random k such that 1<k<p-1 and ged(k, p1)
- compute r $\equiv g^k$ mod p
- compute s $\equiv$ (H(m) - sk * r )$k^{-1}$ *mod (p-1) , if s =0 , continue (**the m is the pk to Bob***)
- pass the signature **pair (r, s)** to Bob

## Step 4 (Bob verify):

- receive the signature pair (r, s)
- if not 0 < r < p and 0 < s < p-1 then **verify fail**
- if not $g^{H(m)} \equiv y^r * r^s$ (mod p) then **verify fail**
- if pass the two condition then verify **successfully**

## Step 5(Bob generate stream key and encrypt message)

- input a clear text : M = message

- generate a stream key randomly as stream_key and the length should be same as message
- do M XOR stream_key we get the Cipher text
- **pass the Cipher Text to Alice**

### Step 6 (Bob use pk to encrypt stream key)

- **receive the public key {q, g ,YA}**
- get the message **M = Stream_key**
- choose a randon $k_2$ < p
- compute K = $(YA)^{k_2}$ mod p
- compute C1 = $g^{k_2}$ mod p
- compute C2 = K*M mod p
- **pass the cipher key as (C1, C2) to Alice**

### Step 7 (Alice use sk to decrypt Cipher key)

- receive the cipher key (C1, C2)
- compute K = $C1^{sk}$ mod p
- compute stream_key = (C2 * $K^{-1}$) mod p

### Step 8 (Alice recover the clear text)

- receive the cipher text CM
- do CM XOR stream_key (get in setp 7)
- finnaly : get the clear text

# Code

# file prestatement

- Coding File :
  - Alice_decrypy.py # use for doing the main logic
  - Bob_encrypy.py # use for doing the main logic
  - Big_number.py. # use for doing some big number compute
  - my_strong_treasure.py # use for doing some Hextype Hex to real Hex such as "0xab" => 0xab , and so on
  - Main.py # use for running the program

- import library:

  Since pyhton does not have a good mechanism for large numbers, we introduce third-party large-number arithmetic libraries for some large-number operations. For example, looking for prime numbers, inversion, etc.

  **We only use the library to do some basic big number operations, and we don't use the library to generate results directly.**

```
from rsa.prime import getprime  # 生成素数的函数
from rsa.common import inverse  # 求逆
from Crypto.PublicKey.ElGamal import generate # 生成元
import hashlib
import string
import random
```

# part 1

in the Alice side , we use to generate the pk, sk ,sign ,

- Alice_decrypy.py

```
"""
for : 密码学作业
消息的接受方
产生密钥
解密消息
"""
from my_strong_treasure import Decode
import random
import hashlib
from big_number import BigNumber
random_seed = 100  # 控制随机取的整数


class AliceDecrypt:

    def __init__(self):
        self.genskandpk()


    def get_pk(self):
        """
        外界获得公钥的方法
        和签名
        :return:
        """
        self.send_pk = hex(self.p)[2:] + "," + hex(self.g)[2:] + "," +
hex(self.Ya)[2:]
        self.send_sig = self.gen_sig(self.send_pk)
        return self.send_pk, self.send_sig


    def genskandpk(self):
        """
        开始计算公钥和私钥
        :return:
        """
```

```
        # 注意在类里 q, p 都是一个int类型 , 而非string
        # 获得 p ,g
        self.p, self.g = BigNumber.get_bigprime_generator(2048)
        self.sk = random.randint(2, random_seed)
        self.Ya = BigNumber.get_quick_mi_mod(self.g, self.sk, self.p)

    def decrypt(self, cipher_text, cipher_key):
        """
        :return: 明文
        """
        cipher_key_str = cipher_key.split(',')
        c1 = int(cipher_key_str[0], 16)
        c2 = int(cipher_key_str[1], 16)
        K = BigNumber.get_quick_mi_mod(c1, self.sk, self.p)
        # K = (c1 ** self.sk) % self.p
        stream_key = hex((c2 * BigNumber.get_inverse(K, self.p)) % self.p)
[2:]

        # 返回明文
        return Decode.StringToASCII(Decode.Two_string_OXR(stream_key,
cipher_text))

    def gen_sig(self, message):
        """
        对发送的消息message进行数字签名
        :return: (r, s) 的签名对
        """
        for i in range(2, self.p - 1):
            if BigNumber.gcd(i, self.p - 1) == 1:
                k = i
                r = BigNumber.get_quick_mi_mod(self.g, k, self.p)
                #
print(hashlib.sha256(message.encode("ascii")).hexdigest())
                s =
((int(hashlib.sha256(message.encode("ascii")).hexdigest(), 16) - self.sk *
r) * BigNumber.get_inverse(k, self.p)) % (self.p - 1)
                # s == 0 则重新选取
                if not s:
                    continue
                return hex(r)[2:] + "," + hex(s)[2:]
```

this class offer the following functions :

- Generate the p ,g
- Generate the pk ,sk
- Generate the signature
- decrypt the message

## part 2

in this side Bob can verify the signature, encrypt the clear and encrypt the stream key with pk

- Bob_encrypy.py

```
    """""""
from my_strong_treasure import Decode
from big_number import BigNumber
import hashlib
import random
import string
random_seed = 100  # 控制随机取的整数

class BobEncrypt:

    def __init__(self, public_key, signature):
        self.public_key = public_key
        pk = public_key.split(',')
        self.p = int(pk[0], 16)
        self.g = int(pk[1], 16)
        self.ya = int(pk[2], 16)
        sig = signature.split(',')
        self.sig_r = int(sig[0], 16)
        self.sig_s = int(sig[1], 16)
        if not self.verify_sig():
             print("verify error!!!")
             exit()
         else:
             print("verify pass!!!")


    def encrypt_origin_text(self, message):
        """
        :param message: 传递的明文 e.g. "fangzhou"
                        使用随机流密码加密密文
        """
        # this is the encrypt message
        self.message = message

        # this stream key is a hex string type
        self.stream_key =
Decode.get_hex_code(self.genRandomString(len(message)))
        # 流密码的十进制int类型
        self.stream_key_num_form = int(self.stream_key, 16)

        # cipher_messages is the hex type string, so when decrypt it does
not need to de transformed
```

```
        self.cipher_messges =
Decode.Two_string_OXR(Decode.get_hex_code(message),
                                                    self.stream_key)
        return self.cipher_messges

    def encrypt_stream_key(self):
        """
        利用私钥加密流密码的key
        elgamal的加密方式，密文由两个部分组成
        返回(c1, c2)
        :return:
        """
        random_k = random.randint(2, random_seed)
        k = BigNumber.get_quick_mi_mod(self.ya, random_k, self.p)
        self.cipher_key_c1 = BigNumber.get_quick_mi_mod(self.g, random_k,
self.p)
        self.cipher_key_c2 = (k * self.stream_key_num_form) % self.p
        self.cipher_key = hex(self.cipher_key_c1)[2:] + "," +
hex(self.cipher_key_c2)[2:]
        return self.cipher_key

    @staticmethod
    def genRandomString(slen=10):
        """
        :param slen: 生成的字符串的长度
        :return:  一个随机的字符串
        """
        return ''.join(random.sample(string.ascii_letters + string.digits,
slen))

    def verify_sig(self):
        """
        验证签名
        :return:
        """
        if not (0 < self.sig_r < self.p and 0 < self.sig_s < (self.p-1)):
            return False
        if pow(self.g,
int(hashlib.sha256(self.public_key.encode("ascii")).hexdigest(), 16),
self.p) != ((pow(self.ya, self.sig_r) * pow(self.sig_r, self.sig_s)) %
self.p):
            return False
        return True
```

we can see that, in the construct function of this class, we verify the signature , if fail we will exit the program.

we use a random string generator to generate the srteam key and then encrypt the clear text with XOR

then we can encrypt the stream key with the pk which is received from the Alice

Finnal we send the Cipher_key and Cipher_text back to Alice

## part 3

in the main function , we can see the process clearly.

- main.py

```
from Alice_decrypt import AliceDecrypt
from Bob_encrypt import BobEncrypt


# alice 产生pk & sk
my_alice = AliceDecrypt()
the_pk, sig = my_alice.get_pk()

# bob 加密text 和 stream_key
# message = "fangzhou Zhangshuai renzhongjie Qiaopengyu QiuLei WangTianWei"
message = input("please input the message you want to encrypt\n")


print("your signature to be verified is \n" + sig)
# 构造函数里面便有验证signature
my_bob = BobEncrypt(the_pk, sig)

# 到这一步说明验证通过
cipher_text = my_bob.encrypt_origin_text(message)
print("your cipher text is \n" + cipher_text)
cipher_key = my_bob.encrypt_stream_key()
print("you cipher_key is \n" + cipher_key)


# 解密
the_message = my_alice.decrypt(cipher_text, cipher_key)
print("your message is : \n" + the_message)
```

- First of all ,the Alice class generate the pk and sk , and do the sign with pk, then send the pk and signature to Bob class
- Secondly, Bob verify the signature first in the construct function , if pass , bob starts to encrypt the clear text : message , and encrypt the stream key with pk **then pass the Cipher Text and Cipher Key to Alice**
- Alice receive the Cipher Text and Cipher Key then do the decryption finally get the clear

text

## part 4

the extract big number functions and XOR functions

- big_number.py

```
"""
大数运算类
"""
from rsa.prime import getprime  # 生成素数的函数
from rsa.common import inverse
from Crypto.PublicKey.ElGamal import generate
import random
class BigNumber:

    @staticmethod
    def get_quick_mi_mod(a, b, mod):
        """
        返回 a**b % mod
        :param a:
        :param b:
        :param mod:
        :return:
        """
        return pow(a, b, mod)

    @staticmethod
    def get_primer(nbits):
        """
        返回一个nbits的素数
        """
        return getprime(nbits)

    @staticmethod
    def get_inverse(a, p):
        """
        p乘法群里取逆运算
        :param a:
        :param p:
        :return:
        """
        return inverse(a, p)

    @staticmethod
    def get_bigprime_generator(nbits):
        """
```

```
        获得一个nbits的素数， 和一个生成元
        :return:
        """
        obj = generate(nbits, randfunc=None)
        return int(obj.p), int(obj.g)

    @staticmethod
    def get_generator(primer):
        """
        找到一个素数的生成元
        :param primer:
        :return:
        """
        # Generate generator g
        while 1:
            # Choose a square residue; it will generate a cyclic group
of order q.
            g = pow(random.randint(2, primer), 2, primer)

            if g in (1, 2):
                continue

            # Discard g if it divides p-1 because of the attack
described
            if (primer - 1) % g == 0:
                continue

            # g^{-1} must not divide p-1 because of Khadir's attack
            ginv = BigNumber.get_inverse(g, primer)
            if (primer - 1) % ginv == 0:
                continue

            # Found
            return g

    @staticmethod
    def gcd(p, q):
        """Returns the greatest common divisor of p and q
        gcd(48, 180)
        12
        """
        while q != 0:
            (p, q) = (q, p % q)
        return p
```

this class used for doing some big number compute with high effectiveness

- my_strong_treasure.py

```
"""
for : 密码学作业

一个专门处理字符串ascii之间xor的类
可以实现
"hello" => ascII 十六进制的字符串
"abc" XOR "eee" => 十六进制进制字符串类型的XOR
...
"""

class Decode:

    @staticmethod
    def StringToASCII(StringText):
        """将string中的16进制 转化为16进制所对应的ascII字符"""
        length = len(StringText)
        index = 0
        AsciiString = ""
        tmp = ""
        HexFlag = 0
        while index < length:
            if not HexFlag:
                tmp = ""
                tmp += StringText[index]
                HexFlag += 1
            else:
                tmp += StringText[index]
                AsciiString += Decode.TwoBit_To_Hex(tmp)
                HexFlag = 0
            index += 1
        return AsciiString


    @staticmethod
    def TwoBit_To_Hex(stringBit):
        """将stringbit转为十六进制  'ab'==>0xab """
        FirstBit = stringBit[0]
        SecondBit = stringBit[1]
        firstHex = Decode.Bit_To_Hex(FirstBit) * 0x10
        SecondBit = Decode.Bit_To_Hex(SecondBit)
        return chr(firstHex+SecondBit)



    @staticmethod
    def Bit_To_Hex(Bit):
        """ 'a'==> 0xA  """
        if Bit == 'a':
            return 0xa
```

```
        elif Bit == 'b':
            return 0xb
        elif Bit == 'c':
            return 0xc
        elif Bit == 'd':
            return 0xd
        elif Bit == 'e':
            return 0xe
        elif Bit == 'f':
            return 0xf
        else:
            return int(Bit)

    @staticmethod
    def get_hex_code(StringText):
        """得到StringText的Ascii码的十六进制 StringToASCII的反过程"""
        length = len(StringText)
        ASCiiCode = ""
        index = 0
        while index < length:
            ASCiiCode += Decode.get_hex_by_bit(StringText[index])
            index += 1
        return ASCiiCode

    @staticmethod
    def get_hex_by_bit(OneChar):
        """已知OneChar为字符，得到他的ASCii码的十六进制"""
        return hex(ord(OneChar))[2:]

    @staticmethod
    def Two_string_OXR(StringOne, StringTwo):
        """两个字符串形式的十六进行异或 得到较短的"""
        length1 = len(StringOne)
        length2 = len(StringTwo)
        if length1 > length2:
            length = length2
        else:
            length = length1
        res = ""
        index = 0
        while index < length:
            res += Decode.XOR_Two_Bit(StringOne[index], StringTwo[index])
            index += 1
        return res

    @staticmethod
    def XOR_Two_Bit(BitOne, BitTwo):
        """异或两十六进制位 eg 'a' XOR '2'  """
```

```
        return hex(Decode.Bit_To_Hex(BitOne) ^ Decode.Bit_To_Hex(BitTwo))
[2:]
```

this class can do some transform between the string type hex XOR or the hex to ascii code and so on

## result

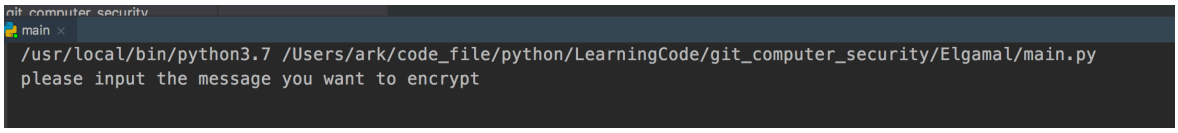
- the 2048-bits is too long for my computer to run, so i change the length into 512 to get the result:

```
# self.p, self.g = BigNumber.get_bigprime_generator(2048)
self.p, self.g = BigNumber.get_bigprime_generator(512)
# we can change the length easily !!!!
```

```
your signature to be verified is
4458147972436b2ff7ff39d5564eb274180dad6326f0670774adf005a0bd8c16ed3e2d
5a648e1186217777a457809e944f56f4583572fa9c3b436281725f48cb,28bdf658bc4
d6888353413e26e09107530e065e9af1abbe8539b6280135026bb3e726b1f21975ade2
95fc36b115da71b50a09682109536621c7e14799fab64f2
verify pass!!!

your cipher text is
3207160e2912271e142319255609215e37062f56022b2716002c220a1f5b2243662637
2a1a561d061b0175265c3f1b340d190739013d64285025253c24
you cipher_key is
34dcd5665d393a0929946221b55fd8c5468bd044c00b314a80fdd03f44977479d4a7d0
3664a4aea0fcaf67481010010dabd93d712c9d855aa6253fdb6ec60108,912a036b9a6
ce75c8639b528224d0cca4559ffc1113ab5a4c954447ed7a10b4a16c0be071730b729f
4564f23e1857600caca4c6016bf61f4c4192176833f1742
your message is :
fangzhou Zhangshuai renzhongjie Qiaopengyu QiuLei WangTianWei
```


- we can also input the massage to do the project





# conclusion

In this project, I learn how to use the **Elgamal Strategy** to implement the digital signature system, public key and private key system, stream cipher system, and figure out the mutual relationship Furthermore we use the konwleage which we learn in the class, and apply it into the program, which can increase our skill in programing and also give us second thinking about the knowledge, and give large improvement about our ability!