# PGP with SM2

姜舜天

2023 年 7 月 25 日

## 目录



## 1 原理

**Pretty Good Privacy (PGP)** 可用于发送机密消息。PGP 结合了对称密钥加密和公钥加密。PGP 使用对称加密算法对消息进行加密，该算法需要对称密钥。每个对称密钥也称为

会话密钥仅被使用一次。消息及其会话密钥被发送到接收方。会话密钥必须被发送给接收方以解密消息，但是为了在传输过程中保护它，它使用接收方的公钥进行加密。只有属于接收方的私钥才能解密会话密钥

## 1.1 PGP

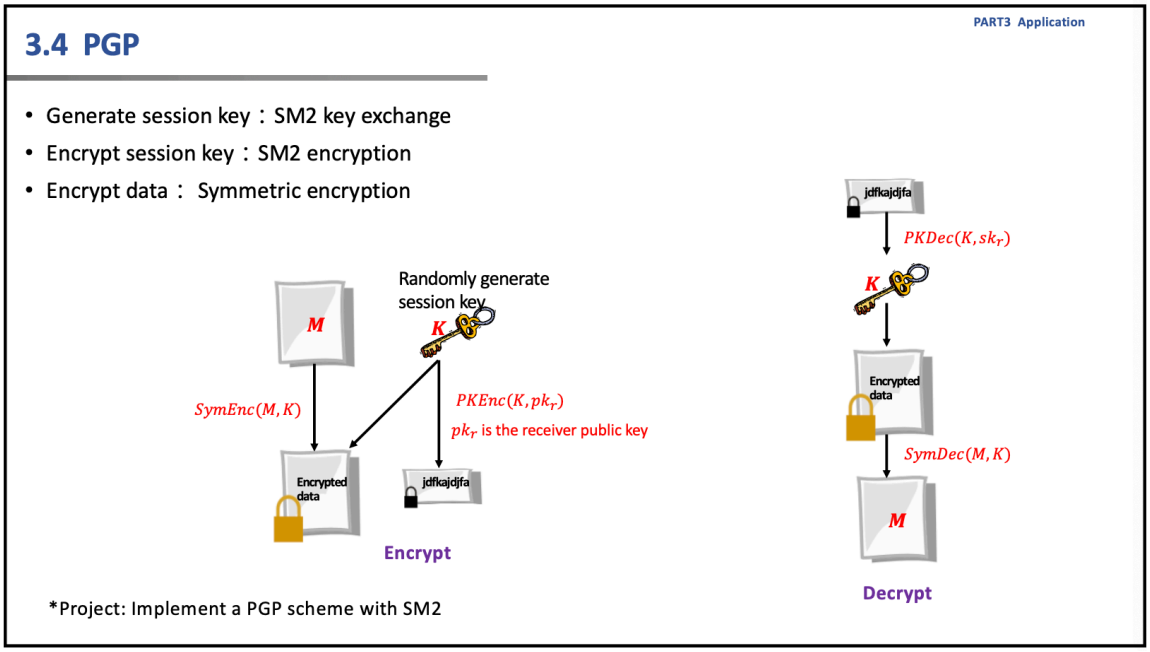


图 1: PGP 协议流程

PGP 协议示意图如上，使用 SM2 完成的 PGP 协议大致分为如下三步：

1. 通过 SM2 密钥交换算法协商生成会话密钥 $sessionkey$。
2. 通过 SM2 加密将协商好的回话密钥发送给对方
3. 通过会话密钥进行对称加密进而安全发送数据

## 1.2 SM2 key-exchange

SM2 密钥交换协议是基于 SM2 算法的一种密钥交换协议，用于在不安全的通信环境中进行密钥交换和密钥协商的安全协议，采用椭圆曲线密码学的方式实现。它具有高安全性、高效性和保密性的特点，在信息安全领域得到广泛应用。

**Algorithm 1** SM2 key-exchange

设用户 A 和 B 协商获取密钥数据的长度为 klen 比特 $w = [[log2(n)]/2] - 1$（注：此处的 [] 指的是顶函数）

用户 A:

A1: 随机选取 $d_1 \in [1, n-1]$;

A2: 计算椭圆曲线点 $P_1 = d_1 \cdot G = (x1, y1)$;

A3: 将 $P_1$ 发送给用户 B。

用户 B:

B1: 随机选取 $d_2 \in [1, n-1]$;

B2: 计算椭圆曲线点 $P_2 = d_2 \cdot G = (x2, y2)$;

B3: 从 $P_2$ 中取出元素 $x2$，计算 $x2' = 2^w + (x2\&(2^w - 1))$;

B4: 计算 $t_B = (dB + x2' \cdot rB) mod\ n (PB, dB$ 分别为 B 的公钥私钥，$PA, d_A$ 以此类推);

B5: 验证 $P_1$ 是否满足椭圆曲线的方程，若不满足则协商失败；否则从 $P_1$ 中取出元素 $x1$; 计算 $x1' = 2^w + (x1\&(2^w - 1))$;

B6: 计算点 $V = (h \cdot tB)(PA + (x1' \cdot P_1)) = (xv, yv)$，若 $V$ 是无穷远点，则 B 协商失败;

B7: 计算 $KB = KDF(xv||yv||ZA||ZB\ klen)$;

用户 A:

A4: 从 $P_1$ 中取出元素 $x1$，计算 $x1' = 2^w + (x1\&(2^w - 1))$;

A5: 计算 $tA = (dA + x1' \cdot rA)\ mod\ n$;

A6: 验证 $P_2$ 是否满足椭圆曲线的方程，若不满足则协商失败；否则从从 $P_2$ 中取出元素 $x2$, 计算 $x2' = 2^w + (x2\&(2^w - 1))$;

A7: 计算点 $U = (h \cdot tA)(PB + (x2' \cdot RB)) = (xu, yu)$，若 $U$ 是无穷远点，则 A 协商失败;

A8: 计算 $KA = KDF(xu||yu||ZA||ZB, klen)$;

上述算法核心代码如下:

Listing 1: Alice SM2 key-exchange

```
1 def key_exchange(soc):
2     print('Begin to key exchange')
3     d_1 = random.randint(2 ** 160, 2 ** 193)
4     P_1 = d_1 * G
5     # 将P_1发送给Bob
6     serial_P_1 = pickle.dumps(P_1)
7     soc.send(serial_P_1)
```

```
8
9       P_2 = soc.recv(1024)
10      P_2 = pickle.loads(P_2)
11
12      x1 = P_1.x()
13      x_1 = 2 ** w + (x1 & (2 ** w - 1))
14      x2 = P_2.x()
15      x_2 = 2 ** w + (x2 & (2 ** w - 1))
16
17      tA = (dA + x_1 * d_1) % SM2_N
18      # 验证是否满足椭圆曲线方程
19      # 可选计算
20      U = (h * tA) * (PB + x_2 * P_2)
21      xu, yu = U.x(), U.y()
22      z = str(hex(xu))[2:] + str(hex(yu))[2:] + ZA + ZB
23      K_A = sm3.sm3_kdf(z.encode(), 16)
24      print('协商结果: ', K_A)
25      return K_A
```

Listing 2: Bob SM2 key-exchange

```
1  def key_exchange(soc):
2      print('Begin to key exchange')
3      op = '1'
4      b.send(op.encode())
5
6      P_1 = soc.recv(1024)
7      P_1 = pickle.loads(P_1)
8
9      d_2 = random.randint(2 ** 160, 2 ** 193)
10     P_2 = (d_2 * G)
11     x2 = P_2.x()
12
13     x_2 = 2 ** w + (x2 & (2 ** w - 1))
14     tB = (dB + x_2 * d_2) % SM2_N
```

```
    # 验证是否满足椭圆曲线方程

    x1 = P_1.x()
    x_1 = 2 ** w + (x1 & (2 ** w - 1))

    V = (h * tB) * (PA + x_1 * P_1)
    xv, yv = V.x(), V.y()
    z = str(hex(xv))[2:] + str(hex(yv))[2:] + ZA + ZB
    K_B = sm3.sm3_kdf(z.encode(), 16)
    # SB 可选就不生成了，少掉两根头发
    serial_P_2 = pickle.dumps(P_2)
    soc.send(serial_P_2)
    print('协商结果: ',K_B)
    return K_B
```

### 1.3 SM2 encrypt

SM2 加密数据使用公钥进行加密，加密结果为 $C_1, C_2, C_3$ 密文中各个部分实际含义如下:

$C_1$: 随机数 K 与 G(x,y) 的多倍点运算结果，结果也是一个点，记录为 $(kx, ky)$

$C_2$: 实际密文值

$C_3$: 使用 SM3 对 $kx||data||ky$ 的 hash 值，在解密时校验解密结果是否正确

**Algorithm 2** SM2 encrypt

1. 随机选取 $k \in [1, n-1]$, 计算 $C_1 = k \cdot G$;

2. 计算 $k \cdot pk = (kpx, kpy)$, 计算密钥流 $KDF = kdf(kpx||kpy, klen)$, 这里的 $pk, kdf$ 分别是公钥与密钥派发函数,$klen$ 表示密钥长度;

3. 计算 $C_2 = M \oplus KDF$;

4. 计算 $C_3 = HASH(kpx||data||kpy)$, 其中 HASH 为 SM3。

上述算法核心代码如下:

Listing 3: SM2 encrypt

```
def encrypt(message,pk):
    k = random.randint(1, SM2_N)
    C_1 = k * G
```

```
    kP = k * pk
    KDF = str(hex(kP.x()))[2:] + str(hex(kP.y()))[2:]

    t = sm3.sm3_kdf(KDF.encode(), 64)
    M = binascii.b2a_hex(message)


    HASH = str(hex(kP.x()))[2:] + str(M) + str(hex(kP.y()))[2:]

    M = int(M, 16)
    C_2 = M ^ int(t, 16)
    C_3 = int(sm3.sm3_hash(list(HASH.encode())),16)
    #print(C_3)
    return [C_1, C_2, C_3]
```

### 1.4 SM2 decrypt

SM2 的解密流程实际是根据 $C_1$ 计算出加密时使用的密钥流，使用密文数据与密钥流进行异或得到数据明文，后续在确认计算出的摘要值与密文中 $C_3$ 是否一致。

**Algorithm 3** SM2 decrypt

1. 计算 $d \cdot C_1 = (cx, cy)$
2. 计算密钥流 $KDF' = kdf(cx||cy, klen)$
3. 计算明文 $M = KDF' \oplus C_2$
4. 计算 $SM3(cx||M||cy)$ 并与 $C_3$ 对比计算结果, 一致则解密成功

上述算法核心代码如下：

Listing 4: SM2 decrypt

```
def decrypt(C_1,C_2,C_3,sk):
    x,y = C_1.x(),C_1.y()
    cx,cy = (sk*C_1).x(),(sk*C_1).y()
    kdf = str(hex(cx))[2:]+str(hex(cy))[2:]
    KDF = sm3.sm3_kdf(kdf.encode(),64)
    M = C_2^int(KDF,16)
```

```
    data = str(hex(M))[2:].encode()
    data_1 = binascii.a2b_hex(data)

    HASH = str(hex(cx))[2:] + str(data) + str(hex(cy))[2:]
    u = int(sm3.sm3_hash(list(HASH.encode())),16)
    if u == C_3:
        print('yes')
        print('datax = ',data_1)
    return data_1
```

## 2 具体实现

按照 CS 模式实现，由 Alice 方担任服务器监听 Bob 方发来的指令请求

| 操作序号 | Alice 对应操作功能 | Bob 对应操作功能 |
|---|---|---|
| 0 | 退出客户端 | 退出客户端 |
| 1 | 协商轮密钥 | 协商轮密钥 |
| 2 | 使用对称加密加密数据并发送给 Bob | 使用轮密钥解密数据 |
| 3 | 使用轮密钥解密数据 | 使用对称加密加密数据并发送给 Alice |

关键代码如下

Listing 5: Alice main

```
while True:
    try:
        op = b.recv(1024).decode()
        if op == '0':
            print('Operation is 0')
            break
        elif op == '1':
            print('Operation is 1')
            KA = key_exchange(b)
            C = encrypt(KA.encode(), PB)
            serial_1 = pickle.dumps(C[0])
```

```
            b.send(serial_1)
            b.send(str(C[1]).encode())
            response = b.recv(16).decode()
            if response == '1':
                print('OK')
                session = base64.b64encode(KA.encode())
        elif op == '2':
            print('Operation is 2')
            data = input('请输入要发送的数据: ')
            send_data(data, session, b)
        elif op == '3':
            print('Operation is 3')
            data = receive_data(session, b)
            print('收到数据: ', data.decode())
    except:
        break
```

Listing 6: Bob main

```
while True:
    op = input('请输入你要执行的操作: ')
    if op == '0':
        print('Operation is 0')
        b.send(op.encode())
        break
    elif op == '1':
        print('Operation is 1')
        KB = key_exchange(b)
        C_1 = b.recv(1024)
        C_1 = pickle.loads(C_1)
        print('receive C_1:',C_1)
        C_2 = int(b.recv(160).decode())
        print('receive C_2:',C_2)
        KA = decrypt(C_1,C_2,0,dB).decode()
        if KA == KB:
```

```
17                print("OK")
18                b.send('1'.encode())
19                session = base64.b64encode(KB.encode())
20        elif op == '2':
21            print('Operation is 2')
22            b.send('2'.encode())
23            data = receive_data(session,b)
24            print('收到数据: ',data.decode())
25        elif op == '3':
26            print('Operation is 3')
27            b.send('3'.encode())
28            data = input('请输入要发送的数据: ')
29            send_data(data, session, b)
```

# 3 实现效果

测试代码请见附录

## 3.1 测试流程

测试流程如下:

1. 运行服务器 Alice.py，此时 Alice 端会进行随机 ID 的生成以及 socket 的绑定，同时等待 Bob 接入。

2. 运行 Bob.py,, Bob 端会生成随机 ID 并连接 socket, 此时服务器会提示“连接地址:(xxx，xxx，xxx，xxx)”，并向 Bob 发送自己的 ID，Bob 收到后也会回复自己的 ID，接着 Alice 和 Bob 端都进行 SM2 密钥生成，并将生成的公私钥显示出来，并分别执行获取对方公钥的函数获得对方当前公钥，然后双方生成 ZA，ZB，完成初始化工作

3. 此时 Bob 端会提示“请输入你要执行的操作”，这里键盘输入‘1’回车 Alice 端和 Bob 端会立刻提示“Operation is 1”和“Begin to key exchange”，此时双方进行密钥协商，协商后双方会提示“协商结果: xxxxx”，此时 Alice 再使用 Bob 的公钥加密协商结果发送给 Bob 进行比对，比对完后如果正确 Bob 进行确认，Alice 端与 Bob 端都会提示“OK”。

3.Bob 端接着输入‘2’，提示当前的发送的指令“Operation is 2”，接着 Alice 端提示“Begin to send data:”，Bob 端提示“Begin to receive data”，Alice 端开始输入数据，然后计算使用轮密钥进行对称加密的密文并发送给 Bob，Bob 收到密文后进行解密，最后会提示“收到数据: xxx”。

4.Bob 端接着输入‘3’，提示当前的发送的指令“Operation is 3”，接着 Bob 端提示“Begin to send data：”，Alice 端提示“Begin to receive data”，Bob 端开始输入数据，然后计算使用轮密钥进行对称加密的密文并发送给 Alice，Alice 收到密文后进行解密，最后会提示“收到数据：xxx”。

5. 接着输入‘0’，提示“Operation is 0“，两方进程结束。

## 3.2 执行结果

执行结果如下：

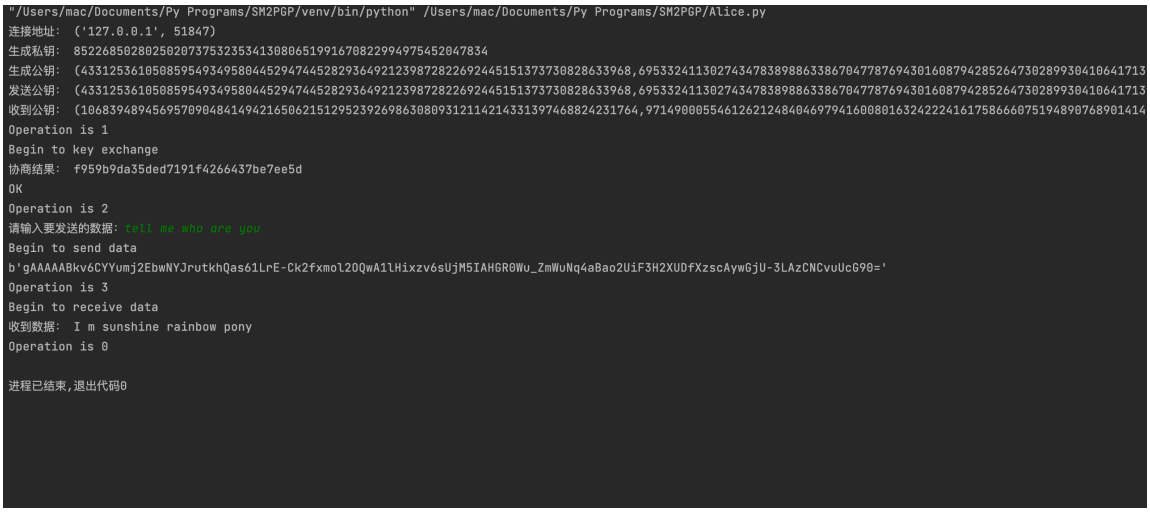


图 2: Alice 端执行结果


图 3: Bob 端执行结果

# 参考文献

[1] https://blog.csdn.net/come_sky/article/details/125952228

[2] https://blog.csdn.net/weixin_42369053/article/details/119036995

[3] https://blog.csdn.net/qq_30866297/article/details/51194236

[4] https://zh.wikipedia.org/wiki/PGP

[5] http://t.csdn.cn/3ohaO

# A 附录

## A.1 调用库以及参数定义

Listing 7: Setting

```
import base64
import binascii
from ecdsa import ellipticcurve, ecdsa, numbertheory
from gmssl import sm3
```

```
5  from cryptography.fernet import Fernet
6  import random
7  import socket
8  import pickle
9
10 SM2_A = 0
       xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC
11 SM2_B = 0
       x28E9FA9E9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93
12 SM2_P = 0
       xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF
13 SM2_N = 0
       xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123
14 SM2_Gx = 0
       x32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7
15 SM2_Gy = 0
       xBC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0
16
17 # 定义椭圆曲线参数
18
19 curve_sm2 = ellipticcurve.CurveFp(SM2_P, SM2_A, SM2_B)
20
21 # 定义生成器点G
22 G = ellipticcurve.Point(curve_sm2, SM2_Gx, SM2_Gy, SM2_N)
23
24 w = int(256 / 2 - 1)
25
26 h = 1  # sm2推荐余因子1
```

## A.2 工具函数

Listing 8: Z_get

```
def Z_get(ID, pk):
    entlen = len(ID)
    ENTL = str(entlen)
    has = ENTL + ID + str(SM2_A) + str(SM2_B) + str(G.x()) + str(G.
        y()) + str(pk.x()) + str(pk.y())
    has = list(has.encode())
    Z = sm3.sm3_hash(has)
    return Z
```

Listing 9: pk_get

```
def pk_get(loc, pk):
    print('发送公钥: ', pk)
    serial_pk = pickle.dumps(pk)
    loc.send(serial_pk)
    PA = loc.recv(1024)
    PA = pickle.loads(PA)
    print('收到公钥: ', PA)
    return PA

def pk_get(loc, pk):
    print('发送公钥: ', pk)
    serial_pk = pickle.dumps(pk)
    loc.send(serial_pk)
    PB = loc.recv(1024)
    PB = pickle.loads(PB)
    print('收到公钥: ', PB)
    return PB
```

## A.3 通信使用函数

Listing 10: Send&Receive

```
def send_data(data, sk, soc):
    print('Begin to send data')
    cipher = Fernet(sk)
    en_data = cipher.encrypt(data.encode())
    print(en_data)
    soc.send(en_data)

def receive_data(sk, soc):
    print('Begin to receive data')
    cipher = Fernet(sk)
    en_data = soc.recv(1024)
    de_data = cipher.decrypt(en_data)
    return de_data
```

## A.4 SM2 相关函数

Listing 11: Example Code

```
def SM2_Key_Generate():
    d = random.randint(2 ** 160, 2 ** 193)
    public_key = d * G
    secret_key = d
    # pair = [secret_key, public_key]
    return secret_key, public_key


def encrypt(message, pk):
    k = random.randint(1, SM2_N)
    C_1 = k * G
    kP = k * pk
    KDF = str(hex(kP.x()))[2:] + str(hex(kP.y()))[2:]
```

```
14
15      t = sm3.sm3_kdf(KDF.encode(), 64)
16      M = binascii.b2a_hex(message)
17      HASH = str(hex(kP.x()))[2:] + str(M) + str(hex(kP.y()))[2:]
18      M = int(M, 16)
19      C_2 = M ^ int(t, 16)
20      C_3 = int(sm3.sm3_hash(list(HASH.encode())), 16)
21      # print(C_3)
22      return [C_1, C_2, C_3]
23
24
25  def decrypt(C_1, C_2, C_3, sk):
26      x, y = C_1.x(), C_1.y()
27      cx, cy = (sk * C_1).x(), (sk * C_1).y()
28      kdf = str(hex(cx))[2:] + str(hex(cy))[2:]
29      KDF = sm3.sm3_kdf(kdf.encode(), 64)
30      M = C_2 ^ int(KDF, 16)
31      data = str(hex(M))[2:].encode()
32      data_1 = binascii.a2b_hex(data)
33      print(data_1)
34      return data_1
35
36
37  def key_exchange(soc):#Alice端
38      print('Begin to key exchange')
39      d_1 = random.randint(2 ** 160, 2 ** 193)
40      P_1 = d_1 * G
41      # 将P_1发送给Bob
42      # print(P_1)
43      serial_P_1 = pickle.dumps(P_1)
44      # print(serial_P_1)
45      soc.send(serial_P_1)
46
47      P_2 = soc.recv(1024)
```

```
48      P_2 = pickle.loads(P_2)
49
50      x1 = P_1.x()
51      x_1 = 2 ** w + (x1 & (2 ** w - 1))
52      x2 = P_2.x()
53      x_2 = 2 ** w + (x2 & (2 ** w - 1))
54
55      tA = (dA + x_1 * d_1) % SM2_N
56      # 验证是否满足椭圆曲线方程
57      # 可选计算
58      U = (h * tA) * (PB + x_2 * P_2)
59      xu, yu = U.x(), U.y()
60      z = str(hex(xu))[2:] + str(hex(yu))[2:] + ZA + ZB
61      K_A = sm3.sm3_kdf(z.encode(), 16)
62      print('协商结果: ', K_A)
63      return K_A
64
65  def key_exchange(soc):#Bob端
66      print('Begin to key exchange')
67      op = '1'
68      b.send(op.encode())
69
70      P_1 = soc.recv(1024)
71      #print(P_1)
72      P_1 = pickle.loads(P_1)
73
74      d_2 = random.randint(2 ** 160, 2 ** 193)
75      P_2 = (d_2 * G)
76      x2 = P_2.x()
77
78      x_2 = 2 ** w + (x2 & (2 ** w - 1))
79      tB = (dB + x_2 * d_2) % SM2_N
80      # 验证是否满足椭圆曲线方程
81
```

```
    x1 = P_1.x()
    x_1 = 2 ** w + (x1 & (2 ** w - 1))

    V = (h * tB) * (PA + x_1 * P_1)
    xv, yv = V.x(), V.y()
    z = str(hex(xv))[2:] + str(hex(yv))[2:] + ZA + ZB
    K_B = sm3.sm3_kdf(z.encode(), 16)
    # SB 可选就不生成了, 少掉两根头发
    serial_P_2 = pickle.dumps(P_2)
    soc.send(serial_P_2)
    print('协商结果: ',K_B)
    return K_B
```

### A.5 初始化代码

Listing 12: Initial

```
#Alice Part
IDA = str(hex(random.randint(2 ** 64, 2 ** 65)))
a = socket.socket()
host = socket.gethostname()
port = 9966
a.bind((host, port))

a.listen(5)
b, addr = a.accept()
print('连接地址: ', addr)

b.send(IDA.encode())
IDB = b.recv(1024).decode()

dA, PA = SM2_Key_Generate()
print('生成私钥: ', dA)
print('生成公钥: ', PA)
```

```
18
19 PB = pk_get(b, PA)
20 ZA = Z_get(IDA, PA)
21 ZB = Z_get(IDB, PB)
22 KA = 0
23 session = 0
24
25 #Bob Part
26 IDB = str(hex(random.randint(2**64,2**65)))
27 b = socket.socket()
28 host = socket.gethostname()
29 port = 9966
30 b.connect((host, port))
31
32 IDA = b.recv(1024).decode()
33 b.send(IDB.encode())
34
35 dB, PB = SM2_Key_Generate()
36 print('生成私钥: ', dB)
37 print('生成公钥: ', PB)
38 PA = pk_get(b, PB)
39
40 ZA = Z_get(IDA, PA)
41 ZB = Z_get(IDB, PB)
42 KB = 0
43 session = 0
```