SM2 2P Sign Report

姜舜天

2023 年 7 月 16 日

1 实现方式

基于 Python ecdsa 库实现的在实际网络中的 SM2 两方签名算法

总述

SM2 两方签名算法基于 SM2 签名算法，但在于公私钥的生成方式上有所区别

SM2 签名算法中私钥 d 为选取产生，公钥 $P = d \cdot G$

SM2 2P 签名算法中 $d = (d_1 d_2)^{-1} - 1, P = [(d_1 d_2)^{-1} G]$

验证算法与 SM2 算法中一致，具体流程如下

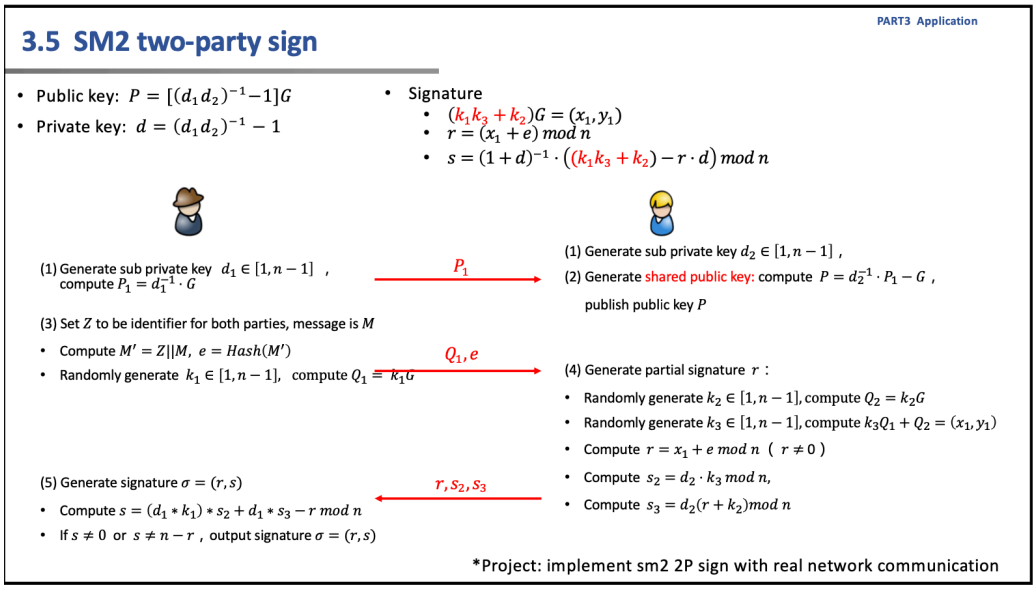

图 1: SM2 签名算法

下面将按照上图一步一步实现 SM2 2P 签名算法

1.**A (1)**

Alice 随机选取 $d_1 \in [1, n-1]$, 计算 $P_1 = d_1^{-1} \cdot G$, 并将 P_1 发送给 Bob

Listing 1: A (1)

```
def s_1_a():
    d_1 = random.randint(1, SM2_N)
    P_1 = mod_inverse(d_1, SM2_N) * G
    return P_1, d_1
```

2.**B (1)&(2)**

Bob 随机选取 $d_2 \in [1, n-1]$, 生成公钥 $P = [(d_1 d_2)^{-1} G]$

Listing 2: B (1)&(2)

```
def s_12_b(P_1):
    d_2 = random.randint(1, SM2_N)
    P = mod_inverse(d_2, SM2_N) * P_1 + (-1) * G
    return P, d_2
```

3.**A (3)**

设置 Z 作为两方的一个验证标识符，假定消息是 M（在这里设置为'identifier-'），计算 $M' = Z||M, e = Hash(M')$, 随机选取 $k_1 \in [1, n-1]$, 计算 $Q_1 = k_1 \cdot G$，将 Q_1, e 发送给 Bob

Listing 3: A (3)

```
def s_2_a(M):
    Z = 'identifier-'
    M_ = Z + M

    e = sm3.sm3_hash(list(M_.encode()))
    print(e)

    e = int(e, 16)  # 获取消息hash值
    k_1 = random.randint(1, SM2_N)
```

```
    Q_1 = k_1 * G
    return e, Q_1, k_1
```

4.**B (4)**

随机生成 $k_2 \in [1, n-1], k_3 \in [1, n-1]$, 计算 $Q_2 = k_2 \cdot G, k_3 \cdot Q_1 + Q_2 = (x_1, y_1)$, 接着计算 $r = x_1 + e \ mod \ n, s_2 = d_2 \cdot k_3 \ mod \ n, s_3 = d_2(r + k_2) \ mod \ n$, 将 r, s_2, s_3 发送给 Alice

Listing 4: B (4)

```
def s_4_b(e, Q_1, d_2):
    k_2 = random.randint(1, SM2_N)
    Q_2 = k_2 * G

    k_3 = random.randint(1, SM2_N)
    Q_3 = k_3 * Q_1 + Q_2

    r = pow(Q_3.x() + e, 1, SM2_N)
    s_2 = pow(d_2 * k_3, 1, SM2_N)
    s_3 = pow(d_2 * (r + k_2), 1, SM2_N)

    return r, s_2, s_3
```

5.**A (5)**

计算签名 $\sigma = (r, s)$, 其中 $s = (d_1 * k_1) \cdot s_2 + d_1 * s_3 - r \ mod \ n$, 如果 $s \neq 0 \ or \ s \neq n - r$, 输出 σ

Listing 5: A (5)

```
def s_5_a(r, s_2, s_3, d_1, k_1):
    s = pow((d_1 * k_1) * s_2 + d_1 * s_3 - r, 1, SM2_N)
    if s == 0 or s == SM2_N - r:
        print("Fail to generate")
    else:
        return [r, s]
```

2 实现效果

运行环境：MacBook air 2020 16GB Apple M1 Python 3.9

测试代码：

Listing 6: test bench of Alice

```
s = socket.socket()
host = socket.gethostname()
port = 9999
s.bind((host, port))

s.listen(5)
p1, d1 = s_1_a()

c, addr = s.accept()
print('连接地址为：', addr)
c.send(b'Here is pk')

serialized_point = pickle.dumps(p1)
c.send((serialized_point))
M = 'Sunshine Rainbow Pony SM2'

e, Q1, k1 = s_2_a(M)

serialized_Q1 = pickle.dumps(Q1)
c.send(serialized_Q1)
print(len(serialized_Q1))
print('Here is Q1:\n', Q1)

send_e = str(e)
c.send(send_e.encode())
print('Here is e:\n', e)
```

```
print(len(send_e.encode()))

seq = c.recv(256).decode()
len = []
len.append(int(seq[0:2]))
len.append(int(seq[2:4]))
len.append(int(seq[4:6]))
r = int(seq[6:6+len[0]])
s2 = int(seq[6+len[0]:6+len[0]+len[1]])
s3 = int(seq[6+len[0]+len[1]:6+len[0]+len[1]+len[2]])

print('Here is r:\n', r)
print('Here is s2:\n', s2)
print('Here is s3:\n', s3)
print('Now generate s')

s = s_5_a(r, s2, s3, d1, k1)
print('Here is s:\n',s)

c.close()
```

Listing 7: test bench of Bob

```
s = socket.socket()
host = socket.gethostname()
port = 9999
s.connect((host, port))
print(s.recv(1024))
p1 = s.recv(1024)
p1 = pickle.loads(p1)

print('P1 is :', p1)
```

```
10 print('Now generate P')
11 P, d2 = s_12_b(p1)
12
13 Q1 = s.recv(366)
14 Q1 = pickle.loads(Q1)
15 print('Q1 is :\n', Q1)
16
17 e = s.recv(100)
18 print(type(e))
19
20 print('e is :\n', int(e.decode()))
21
22 e = int(e)
23
24 print('Now generate r')
25 r, s2, s3 = s_4_b(e, Q1, d2)
26 print('Here is r:\n', r)
27 print('Here is s2:\n', s2)
28 print('Here is s3:\n', s3)
29 print(len(str(r)),len(str(s2)),len(str(s3)))
30
31
32 seq = str(len(str(r)))+str(len(str(s2)))+str(len(str(s3)))+str(r)
       +str(s2)+str(s3)
33
34
35 s.send(seq.encode())
36
37
38 s.close()
```

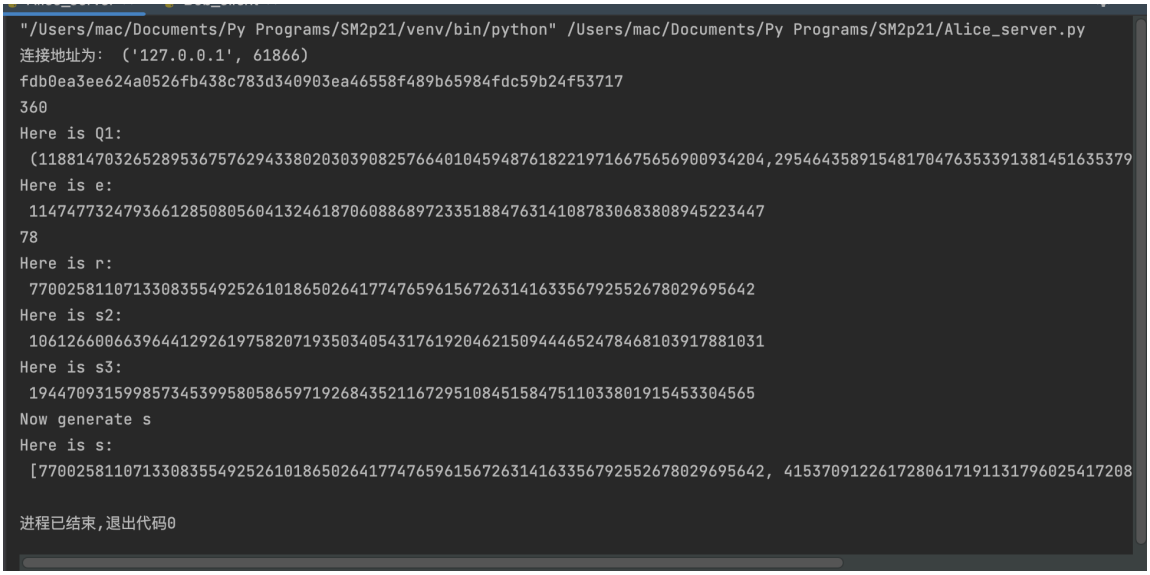

图 2: A 执行结果

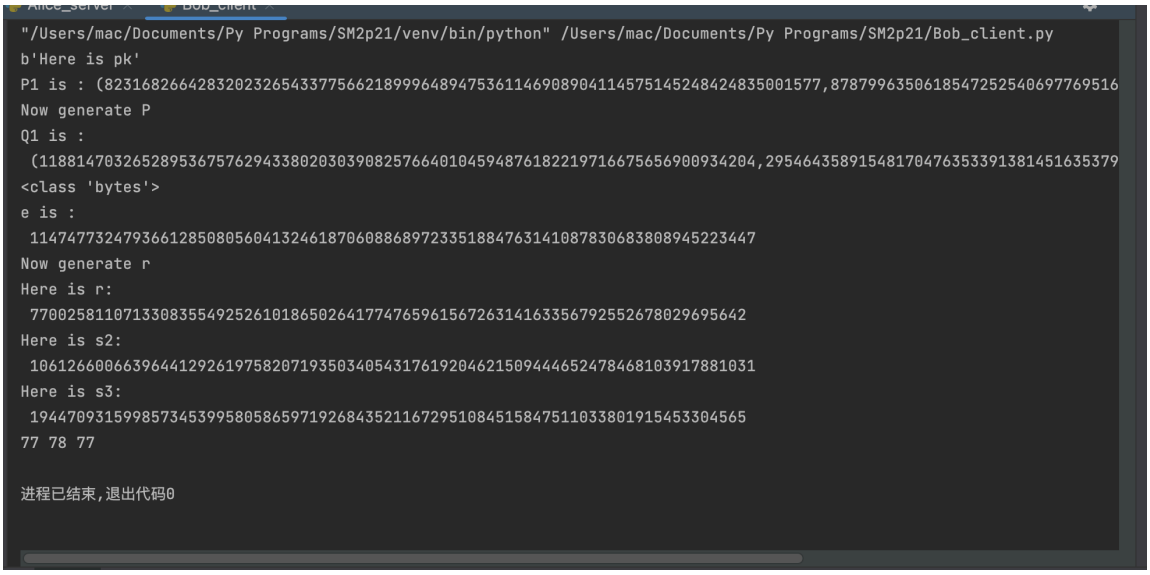

图 3: B 执行结果

Listing 8: testbench of sig verification

```
P_1 , d_1 = s_1_a()

P, d_2 = s_12_b(P_1)

test = pow((mod_inverse(d_1*d_2,SM2_N)-1),1,SM2_N)*G

print(P)
print(test)

e, Q_1, k_1 = s_2_a('sm22p')

r, s_2, s_3 = s_4_b(e,Q_1,d_2)

sig = s_5_a(r,s_2,s_3,d_1,k_1)

print('签名结果: \n',sig)

out = sm2_ver('sm22p',sig[0],sig[1],P)

print('验证结果: \n',out)
```

```
"/Users/mac/Documents/Py Programs/SM2p21/venv/bin/python" /Users/mac/Documents/Py Programs/SM2p21/SM22PSign.py
(25276493491907868469751306382293671116153378888490618864424130267019523916776,50741740966759240190754836174231117175 2
(25276493491907868469751306382293671116153378888490618864424130267019523916776,50741740966759240190754836174231117175 2
aaa49dfccd2e340ddde90e9da148b92cd9a8ebea389f57a778d05453b813f2b8
签名结果:
 [88626980221309019278975076582672944383632967580222909945006144599973509505373, 57412365376849288287009001324345858 74
aaa49dfccd2e340ddde90e9da148b92cd9a8ebea389f57a778d05453b813f2b8
88626980221309019278975076582672944383632967580222909945006144599973509505373
88626980221309019278975076582672944383632967580222909945006144599973509505373
验证结果:
 1

进程已结束,退出代码0
```

图 4: 签名验证测试结果

参考文献

[1] https://blog.csdn.net/Digquant/article/details/124429472

A 附录标题

Listing 9: SM2 参数设置

```
p= FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 FFFFFFFF
    FFFFFFFF

a= FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF 00000000 FFFFFFFF
    FFFFFFFC

b= 28E9FA9E 9D9F5E34 4D5A9E4B CF6509A7 F39789F5 15AB8F92 DDBCBD41
    4D940E93

n= FFFFFFFE FFFFFFFF FFFFFFFF FFFFFFFF 7203DF6B 21C6052B 53BBF409
    39D54123

Gx= 32C4AE2C 1F198119 5F990446 6A39C994 8FE30BBF F2660BE1715A4589
    334C74C7

Gy= BC3736A2 F4F6779C 59BDCEE3 6B692153 D0A9877C C62A4740 02
    DF32E5 2139F0A0
# 定义椭圆曲线参数
p = SM2_P
a = SM2_A
b = SM2_B
n = SM2_N
Gx = SM2_Gx
Gy = SM2_Gy
curve_sm2 = ellipticcurve.CurveFp(p, a, b)

# 定义生成器点G
G = ellipticcurve.Point(curve_sm2, Gx, Gy, n)
```