Implement SM2 with RFC6979

赵嵘晖 202100460100

1 实验环境

编辑器：Visual Studio Code
操作系统:Windows11
编译语言:Python 3.10
CPU: 12th Gen Intel(R) Core(TM) i5-12500H 2.50 GHz

2 SM2 算法原理

2.1 参数选择

SM2 算法定义了 5 个默认参数，即有限域 F_q 的规模 q，椭圆曲线参数 a,b, 椭圆曲线的基点 G(x,y), 与 G 的阶 n。

椭圆曲线方程是定义在 F_q-256 上的 $y^2 = x^3 + ax + b$。

国密算法 SM2 给出了对应的默认值，如下所示。

q:FFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF

n:FFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123

a:FFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC

b:28E9FA9E9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93

G_x:32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7

G_y:BC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0

2.2 密钥生成

SM2 是非对称算法，拥有公钥与私钥，私钥 d 是符合要求的 256bit 随机数，公钥 (x,y), 实际一个坐标点，依据私钥 d 计算所得。

私钥: 随机选取一个数 d。公钥:$p = d \cdot G$。

2.3 加密

SM2 加密算法中的 hash 函数选择的是 SM3。加密公钥是 P_B，解密私钥是 d_B。流程图如下所示。

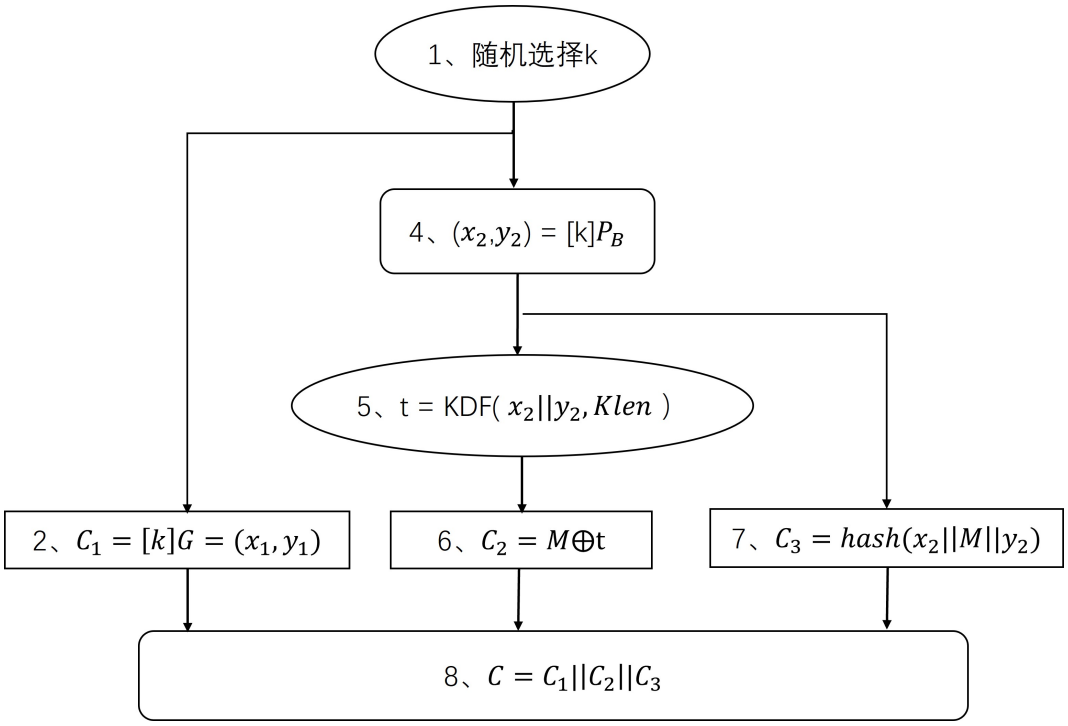

图 1: SM2 加密算法

算法如下所示。

Algorithm 1 SM2 Encryption

Input: M

Output: C

Compute the bit length of the M:klen
Select a random number k∈[1, n - 1]
Compute C_1=[k]G=(x_1,y_1)
Compute h=|E(F_q)|/n
Compute C_1=[k]G=(x_1,y_1)S=hP_B
if S is ◯ **then**
    return Error
end if
Compute kP_b=(x_2,y_2)
Compute t=KDF(x_2||y_2,Klen)
if t == 0 **then**
    Select a random number k again
end if
Compute C_2=M⊕t
Compute C_3=hash(x_2||M||y_2)
Compute C=C_1||C_2||C_3
return C

2.4 解密

流程图如下所示。

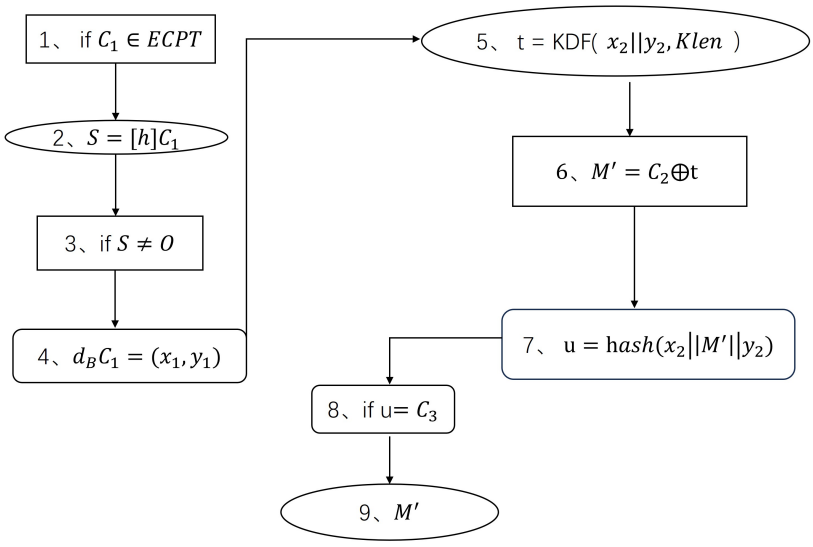

图 2: SM2 解密算法

算法如下所示。

Algorithm 2 SM2 Decryption

Input: C

Output: M′

if $C_1 \notin$ECPT **then**
  return ERROR
end if
Compute S=hC_1
if S=O **then**
  return ERROR
end if
Compute $d_B C_1$
Compute t=KDF($x_2||y_2$,Klen)
Compute M′=$C_2 \oplus$t
Compute u=hash($x_2||$M′$||y_2$)
if u≠C_3 **then**
  return ERROR
end if
return M′

2.5 签名

先预计算 Z_A=Hash($ENTL_A||ID_A||a||b||x_G||y_G||x_A||y_A$),$entl_A$ 是 ID_A 的长度,$ENTL_A$ 为 $entl_A$ 的两字节表示。HASH 函数采用 SM3。

流程图如下所示。

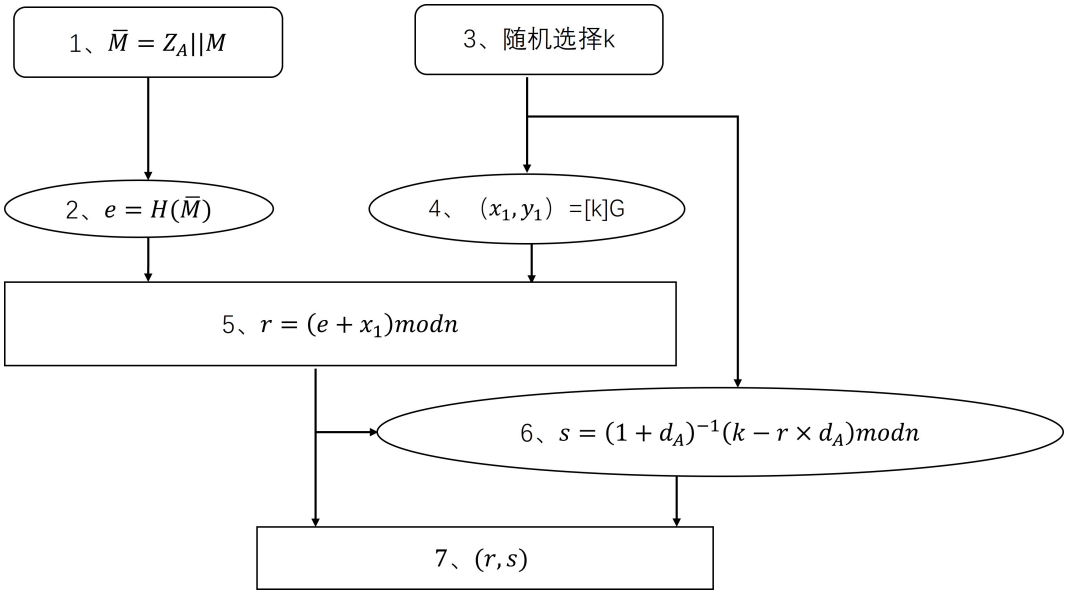

图 3: SM2 签名算法

算法如下所示。

Algorithm 3 SM2 Signature

Input: M

Output: (r,s)

Compute Z_A=Hash(ENTL$_A$||ID$_A$||a||b||x$_G$||y$_G$||x$_A$||y$_A$)
Compute $\overline{M}$=Z_A||M
Compute e=Hash($\overline{M}$)
Select a random number k∈[1, n - 1]
Compute KG=(x$_1$,y$_1$)
Compute r=(e+x$_1$)mod n
if r == 0 or r + k == n **then**
    return Select a random number k again
end if
Compute s=$(1+d_A)^{-1}$(k-rd_A)
if s == 0 **then**
    Select a random number k again
end if
return (r,s)

2.6 验证

流程图如下所示。

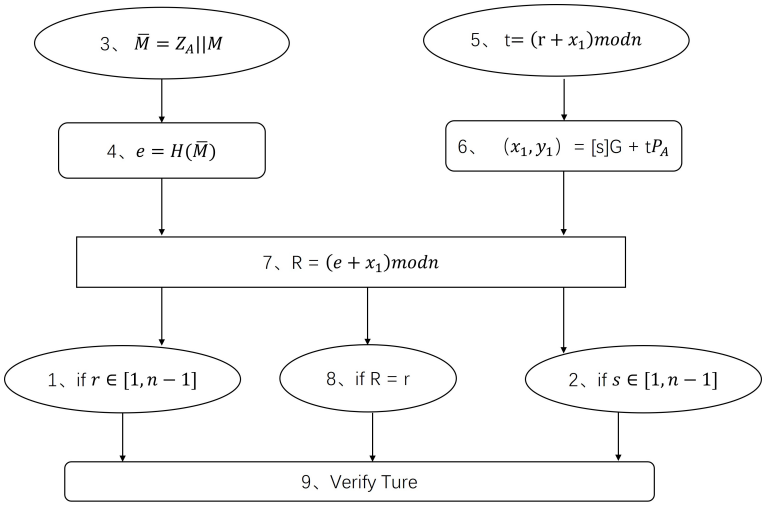

图 4: SM2 签名验证算法

算法如下所示。

Algorithm 4 SM2 Signature

Input: (r,s)

Output: False or TURE

Compute Z_A=Hash($ENTL_A$||ID_A||a||b||x_G||y_G||x_A||y_A)
Check r∈[1,n-1]
Check s∈[1,n-1]
Compute $\overline{M}$=Z_A||M
Compute e=hash($\overline{M}$)
Compute t=(r+x_1)mod n
Compute (x_1,y_1)=sG+tP_A
Compute R =(e+x_1)mod n
if R≠r **then**
  return FALSE
end if
return TURE

3 RFC6979

根据 RFC6979，最初的 ECDSA 概念中，每个签名都需要 256 位随机数据 。这会带来一个问题，因为当使用相同的随机输入签署两个不同的消息（即比特币交易）时，会泄露私钥。RFC6979 建议使用 HMAC-SHA256(private_key, message) 的输出来代替随机数据，从而消除了这种风险。

重点是 k 的生成。

前提：

qlen 是 q 的二进制字符串长度。

函数 bits2int() 将二进制字符串转换为 int 类型

函数 int2octets() 将 int 类型数据转换为多个 8bit 组成的字符串

函数 bits2octets() 将二进制字符串转换为多个 8bit 组成的字符串。

1、计算 h_1 = H(m)。m 是消息。

2、V = 0x01 0x01 0x01 0x01 0x01...0x01

3、K = 0x00 0x00 0x00 0x00 0x00...0x00

4、K = HMAC_K(V || 0x00 || int2octets(x) || bits2octets(h_1)), x 是私钥

5、V = HMAC_K(V)

6、K = HMAC_K(V || 0x01 || int2octets(x) || bits2octets(h_1))

7、V = HAMC_K(V)

8、T = " ",tlen 是序列 T 的长度，初始化为 0。

9、Loop: While tlen < qlen，计算 V = HAMC_K(V)，T = T||V，tlen = length(T)

10、计算 K = bits2int(T)

11、如果 K∈[1,q-1]，那么 K 正确生成。

12、如果 K∉[1,q-1]

13、Loop: while k >= q，K = HMAC_K(V || 0x00)，V = HMAC_K(V)，K = bits2int(T)

14、循环结束后，得到正确的 K。

4 实验思路

改变随机数 k 的生成方式，由 SM3 生成 k, 即 k = SM3(d,message)。其他的 SM2 实现方法依照之前的原理实现。

5 实验结果

5.1 实验结果正确性

SM2 加密函数与解密函数正确性验证。

图 5: SM2 加解密算法验证

SM2 签名算法正确性验证。

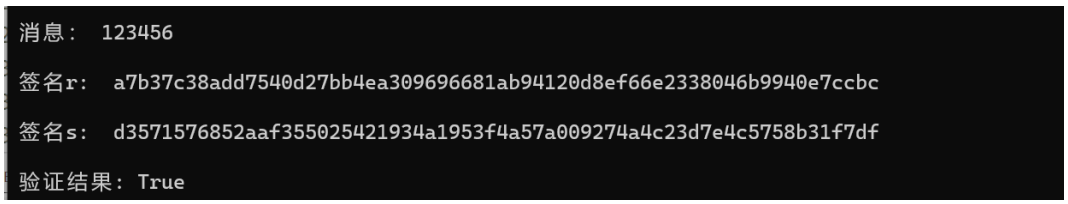

图 6: SM2 签名算法验证

5.2 实验效率

SM2 加密函数与解密函数时间效率测量。

图 7: SM2 加解密算法效率

如上图所示，一次加密与解密耗时为：0.005773189306259155s。
SM2 签名算法时间效率测量。

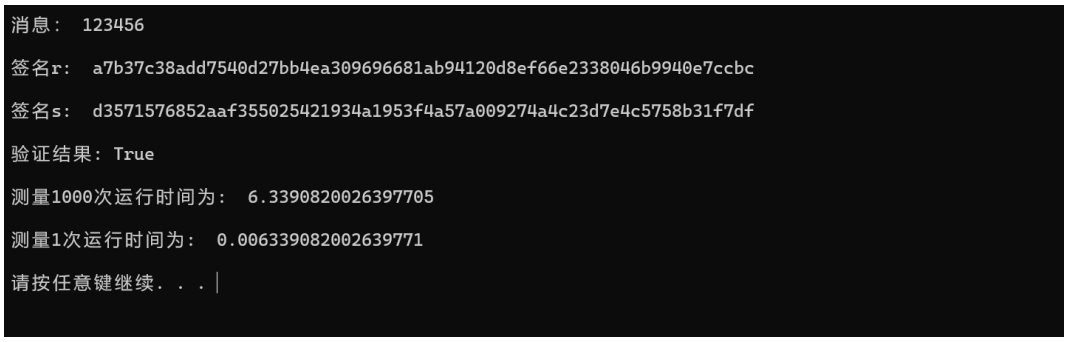

图 8: SM2 签名算法效率

如上图所示，一次签名及验证耗时为：0.006339082002639771s。

6 代码

如下是具体的代码。

6.1 sm3

```
def sm3(s: str) -> str:

    # 初始值
    IV = 0x7380166f4914b2b9172442d7da8a0600a96f30bc163138aae38dee4db0fb0e4e
    MAX_32 = 0xffffffff

    # 32位循环左移
    def left_shift(x: int, i: int) -> int:
        return ((x << (i % 32)) & MAX_32) + (x >> (32 - i % 32))

    # 常量T
    def T(j: int) -> int:
```

```
        if 0 <= j <= 15:
            return 0x79cc4519
        return 0x7a879d8a

    # 布尔函数
    def FF(j: int, x: int, y: int, z: int) -> int:
        if 0 <= j <= 15:
            return x ^ y ^ z
        return (x & y) | (x & z) | (y & z)

    # 布尔函数
    def GG(j: int, x: int, y: int, z: int) -> int:
        if 0 <= j <= 15:
            return x ^ y ^ z
        return (x & y) | (~x & z)

    # 置换函数
    def P0(x: int) -> int:
        return x ^ left_shift(x, 9) ^ left_shift(x, 17)

    # 置换函数
    def P1(x: int) -> int:
        return x ^ left_shift(x, 15) ^ left_shift(x, 23)

    # 消息填充函数，对长度为l(l < 2^64)比特的消息s，填充至长度为512比特的倍数
    def fill(s: str) -> str:
        v = 0
        for i in s:
            v <<= 8
            v += ord(i)
        msg = Int_bin(v).zfill(len(s) * 8)
        k = (960 - len(msg) - 1) % 512
        return hex(int(msg + '1' + '0' * k + Int_bin(len(msg)).zfill(64), 2))[2:]

    m = fill(s)

    # 迭代过程
    for i in range(len(m) // 128):

        # 消息扩展
        Bi = m[i * 128: (i + 1) * 128]
        W = []
        for j in range(16):
            W.append(int(Bi[j * 8: (j + 1) * 8], 16))
        for j in range(16, 68):
```

```
                W.append(P1(W[j - 16] ^ W[j - 9] ^ left_shift(W[j - 3], 15)) ^ left_shift(W[j - 13], 7) ^ W[j - 6])
            W1 = []
            for j in range(64):
                W1.append(W[j] ^ W[j + 4])
            A, B, C, D, E, F, G, H = [IV >> ((7 - i) * 32) & MAX_32 for i in range(8)]

            # 迭代计算
            for j in range(64):
                ss1 = left_shift((left_shift(A, 12) + E + left_shift(T(j), j)) & MAX_32, 7)
                ss2 = ss1 ^ left_shift(A, 12)
                tt1 = (FF(j, A, B, C) + D + ss2 + W1[j]) & MAX_32
                tt2 = (GG(j, E, F, G) + H + ss1 + W[j]) & MAX_32
                D = C
                C = left_shift(B, 9)
                B = A
                A = tt1
                H = G
                G = left_shift(F, 19)
                F = E
                E = P0(tt2)
            IV ^= ((A << 224) + (B << 192) + (C << 160) + (D << 128) + (E << 96) + (F << 64) + (G << 32) + H)
        return hex(IV)[2:].zfill(64)
```

6.2 生成 K

```
# 生成 K
q = int(str(p), 16)
qlen = 256
data1 = ''.join([chr(item) for item in range(256)])

def Bits2int(S):
    # 二进制字符串转int
    blen = len(S)
    if qlen < blen:
        S = S[:qlen]
    else:
        S = (qlen - blen) * "0" + S
    int_data = 0
```

```
14                      blen = len(S)
15                      while blen > 0:
16                              int_data *= 2
17                              int_data += data1.find(S[0])
18                              S = S[1:]
19                              blen = len(S)
20                      return int_data
21
22
23              def int2octets(S):
24                      oct_data = ""
25                      while S > 0:
26                              temp = S % 256
27                              oct_data += data1[temp]
28                              S = S // 256
29                      rlen = len(oct_data)
30                      if rlen >= 32:
31                              return oct_data
32                      oct_data += 'a' * (32 - rlen)
33                      return oct_data
34
35
36              def bits2octets(S):
37                      # 二进制字符串转8bit一组的字符串
38                      A = Bits2int(S) % q
39                      B = int2octets(A)
40                      return B
41
42
43              def generate_k(m):
44                      h1 = sm3(m.decode())
45                      v = '\x01' * 32
46                      k = '\x00' * 32
47                      k = hmac.new(k.encode(), (v + '\x00' + int2octets(private_key) + bits2octets(h1)).encode
    (), hashlib.sha256).digest()
48                      v = hmac.new(k, v.encode(), hashlib.sha256).digest()
49                      k = hmac.new(k, v + ('\x01' + int2octets(private_key) + bits2octets(h1)).encode(), hashlib
    .sha256).digest()
50                      v = hmac.new(k, v, hashlib.sha256).digest()
51                      tlen = 0
52                      T = b''
53                      while tlen < qlen:
54                              v = hmac.new(k, v, hashlib.sha256).digest()
55                              T = T + v
56                              tlen = len(T)
```

```
        T = T.decode(errors='ignore')
        K = Bits2int(T)
        if K >= 1 & K < q:
            return K
        else:
            while K >= q:
                k = hmac.new(k, v + ('\x00').encode(), hashlib.sha256).digest()
                v = hmac.new(k, v, hashlib.sha256).digest()
                K = Bits2int(T.decode(errors='ignore'))
        return K
```

6.3 SM2 加解密

```
# 定义SM2椭圆曲线参数
p = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF
a = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC
b = 0x28E9FA9E9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93
n = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123
Gx = 0x32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7
Gy = 0xBC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0

# 椭圆曲线点加
def Add(x1, y1, x2, y2):
    if x1 == x2 and y1 == p-y2:
        return False
    if x1 != x2:
        l = ((y2 - y1) * gmpy2.invert(x2 - x1, p)) % p
    else:
        l = (((3 * x1 * x1 + a) % p) * gmpy2.invert(2 * y1, p)) % p
    x3 = (l * l - x1 - x2) % p
    y3 = (l * (x1 - x3) - y1) % p
    return x3, y3

# 椭圆曲线点乘
def Mul_Add(x, y, n):
    n = Int_bin(n)
    qx, qy = x, y
    for i in range(1, len(n)):
        qx, qy = Add(qx, qy, qx, qy)
        if n[i] == '1':
            qx, qy = Add(qx, qy, x, y)
```

```
29                  return (qx, qy)
30
31          # 获取公私钥
32          private_key = 0x2359076592ACDF6423673737215635672EE127FD34386956
33          public_key = Mul_Add(Gx, Gy, private_key)
34
35          # 明文
36          data = "123456"
37
38          def KDF(xy, klen):
39                  t = ''
40                  count = 1
41                  for i in range(math.ceil(klen/256)):
42                          temp = (xy + '{:032b}'.format(count)).encode().hex()
43                          t = t + sm3(temp).encode().hex()
44                          count += 1
45                  x = int(t, 16)
46                  x = Int_bin(x)
47                  t = '0' * ((256-(len(x) % 256)) % 256) + x
48                  return t[:klen]
49
50
51          # 加密函数
52          def encrypt(m):
53                  plen = len(hex(p)[2:])
54                  m1 = m.encode().hex()
55                  m1 = int(m1, 16)
56                  m = '0' * ((4 - (len(Int_bin(m1)) % 4)) % 4) + Int_bin(m1)
57                  klen = len(m)
58                  k = generate_k(m.encode())
59                  x2, y2 = Mul_Add(public_key[0], public_key[1], k)
60                  if (len(hex(p)[2:]) * 4 == 192):
61                          x2, y2 = '{:0192b}'.format(x2), '{:0192b}'.format(y2)
62                  else:
63                          x2, y2 = '{:0256b}'.format(x2), '{:0256b}'.format(y2)
64                  t = KDF(x2 + y2, klen)
65                  x1, y1 = Mul_Add(Gx, Gy, k)
66                  x1, y1 = (plen - len(hex(x1)[2:])) * '0' + hex(x1)[2:], (plen - len(hex(y1)[2:])) * '0' +
    hex(y1)[2:]
67                  c1 = x1 + y1
68                  temp = int(m,2) ^ int(t, 2)
69                  temp = hex(temp)[2:]
70                  Tlen = len(temp)
71                  c2 = ((klen//4) - Tlen) * '0' + temp
72                  c3 = sm3(hex(int(x2 + m + y2, 2))[2:])
```

```
                return c1, c2, c3


        # 解密函数
        def decrypt(c1, c2, c3):
                x1, y1 = int(c1[:len(c1) // 2], 16), int(c1[len(c1) // 2:], 16)
                x2, y2 = Mul_Add(x1, y1, private_key)
                if (len(hex(p)[2:]) * 4 == 192):
                        x2, y2 = '{:0192b}'.format(x2), '{:0192b}'.format(y2)
                else:
                        x2, y2 = '{:0256b}'.format(x2), '{:0256b}'.format(y2)
                klen = len(c2) * 4
                t = KDF(x2 + y2, klen)
                if int(t, 2) == 0:
                        return False
                temp = int(c2, 16) ^ int(t, 2)
                temp = Int_bin(temp)
                Tlen = len(temp)
                m = '0' * (klen - Tlen) + temp
                u = sm3(hex(int(x2 + m + y2, 2))[2:])
                if u != c3:
                        return False
                message_dec = hex(int(m, 2))[2:]
                message_dec = binascii.a2b_hex(message_dec).decode()
                return message_dec
```

6.4 SM2 签名

```
        # 签名函数
        def sign(m, ID):
                k = generate_k(m.encode())
                ID = ID.encode().hex()
                ENTL='{:04X}'.format(len(ID) * 4)
                m1=ENTL + ID + '{:064X}'.format(a)+'{:064X}'.format(b)+'{:064X}'.format(Gx)+'{:064X}'.format(Gy)+'{:064X}'.format(public_key[0])+'{:064X}'.format(public_key[1])
                hash_m = hex(int(sm3(m1),16))[2:]
                m = hash_m + m
                e = sm3(m)
                x1,y1 = Mul_Add(Gx, Gy, k)
                r = (int(e, 16) + x1) % n
```

```
                s = (gmpy2.invert(1 + private_key, n) * (k - r * private_key)) % n
                r = hex(r)[2:]
                s = hex(s)[2:]
                return r, s

        # 验证函数
        def verify(r,s,ID,m):
                ID = ID.encode().hex()
                ENTL = '{:04X}'.format(len(ID) * 4)
                m1 = ENTL + ID +'{:064X}'.format(a)+'{:064X}'.format(b)+'{:064X}'.format(Gx)+'{:064
X}'.format(Gy)+'{:064X}'.format(public_key[0])+'{:064X}'.format(public_key[1])
                hash_m = hex(int(sm3(m1),16))[2:]
                if int(r, 16) not in range(1, n-1):
                        return False
                if int(s, 16) not in range(1, n-1):
                        return False
                m = hash_m + m
                        e = sm3(m)
                t=(int(r, 16) + int(s, 16)) % n
                if t == 0:
                        return False
                x1, y1 = Mul_Add(public_key[0], public_key[1], t)
                x2, y2 = Mul_Add(Gx, Gy, int(s, 16))
                x1, y1 = Add(x1, y1, x2, y2)
                R=(int(e, 16) + x1) % n
                if(hex(R)[2:] == r):
                        return True
                return False
```

参考文献

[1] https://blog.csdn.net/liranke/article/details/128124639

[2] https://blog.csdn.net/qq_53755809/article/details/121464982

[3] https://blog.csdn.net/samsho2/article/details/80772228

[4] https://datatracker.ietf.org/doc/html/rfc6979