第八次实验报告

赵碧琪 20373669

【实验目的】

1.掌握椭圆曲线上的运算和常见的椭圆曲线密码算法；

2.了解基于ECC的伪随机数生成算法和基于椭圆曲线的商用密码算法。

【实验环境】

本次实验采用Pycharm编译器与Python3.9环境编写。

1 ECC四则运算

1.1 算法流程

1.1.1 ECC加法减法

对于点P，Q，如果两点横坐标相同但纵坐标相反，那么相加结果为无穷远点；

其中一个是无穷远点时则结果等于另一个点；

否则就有公式如下

$$\delta = \begin{cases} (3Px^2 + a)(mod\ p) \times invmod(2Py, p)(mod\ p) \\ (Py - Qy)(mod\ p) \times invmod((Px - Qx), p))(mod\ p) \end{cases}$$

$$Rx = \delta^2 - Px - Qx(mod\ p)$$
$$Ry = \delta \times (Px - Qx) - Py(mod\ p)$$

R即为结果。

对于减法则只需要将点Q点坐标改为负数即可。

1.1.2 ECC乘法除法

即整数k与点坐标的乘法，意为倍点。这里需要解决的是快速求倍点的问题。继续沿用第一次实验中的快速模幂算法，将整数k化为二进制，将模乘运算变为点加即可，代码如下：

```python
def mul(P, p, a, b, k):
    bin = []
    ind = k
    while ind:
        bin.append(ind % 2)
        ind = ind // 2
    c = ECC_dot(0, 0)
    for i in range(len(bin) - 1, -1, -1):
        c = plus(c, c, p, a, b)
        if bin[i] == 1:
            c = plus(P, c, p, a, b)
    return c
```

这里的ECC_dot是定义的一个椭圆曲线点类，传入坐标x,y存储为c(x,y)

对于除法，情况也类似。即求出k的关于模数p的逆元，将其作为整数参数传入乘法函数即可。

1.1.3 函数调用图

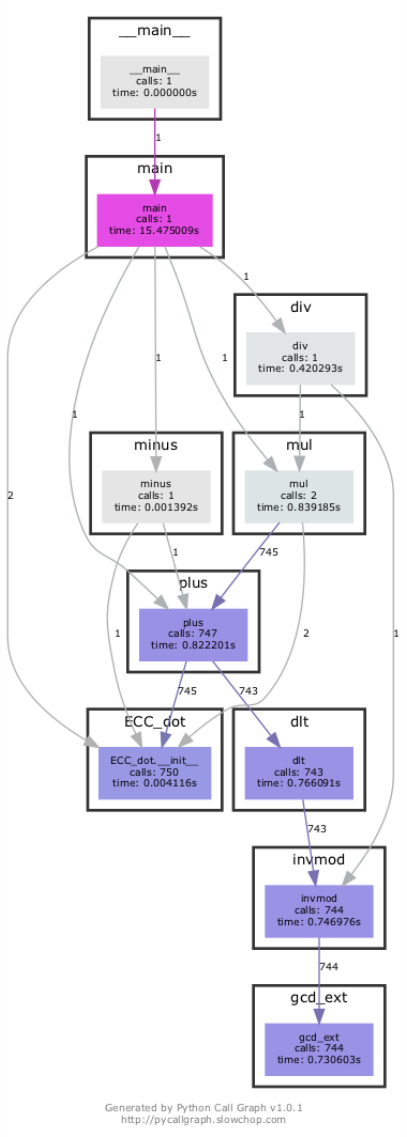

图1 ECC四则运算函数调用图

1.2 测试结果

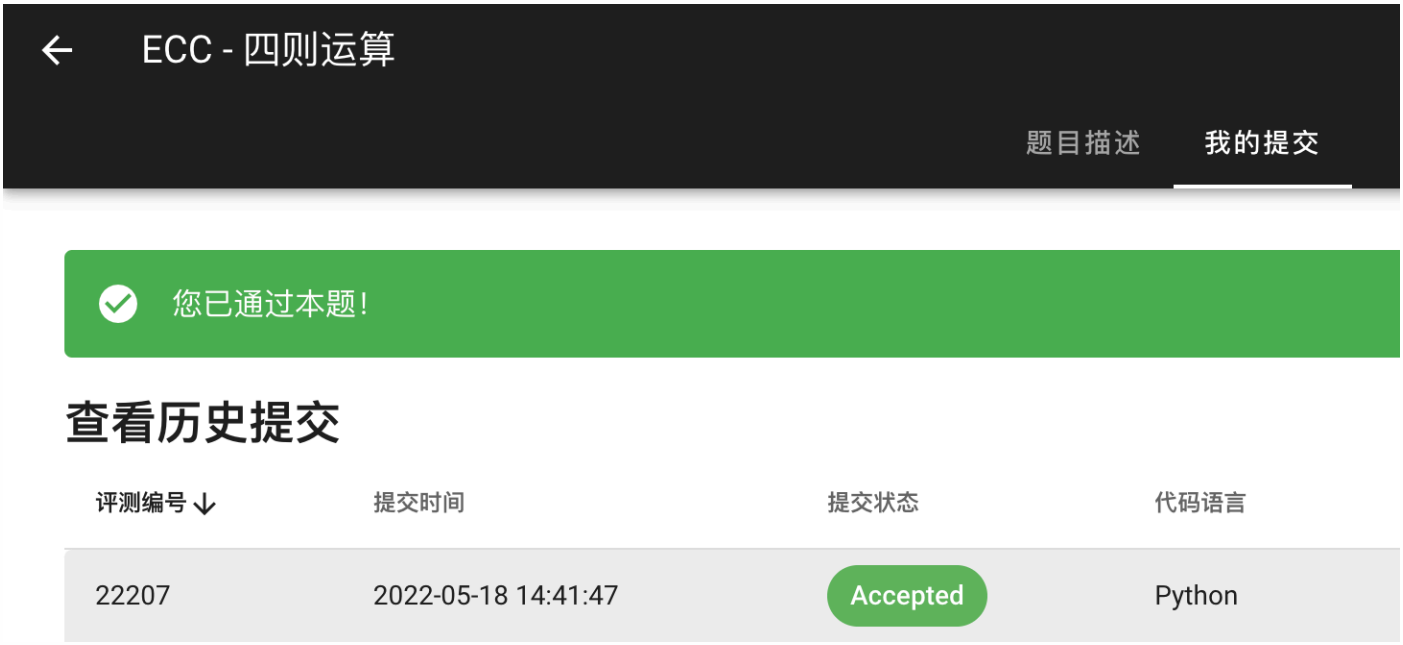

图2 ECC四则运算测试结果

2 ECC加解密

2.1 算法流程

下面是利用椭圆曲线进行加密通信的过程：

> 1、用户A选定一条椭圆曲线Ep(a,b)，并取椭圆曲线上一点，作为基点G。
>
> 2、用户A选择一个私有密钥k，并生成公开密钥K=kG。
>
> 3、用户A将Ep(a,b)和点K，G传给用户B。
>
> 4、用户B接到信息后，将待传输的明文编码到Ep(a,b)上一点M（编码方法很多，这里不作讨论），并产生一个随机整数r（r<n）。
>
> 5、用户B计算$C_2 = M + rK, C_1 = rG$。
>
> 6、 用户B将（C_1, C_2）传输给A。
>
> 7、A在接收到B的密文信息后，计算$C_2 - kC_1 = M$得到明文，解码即可得到消息。

算法流程图如下：

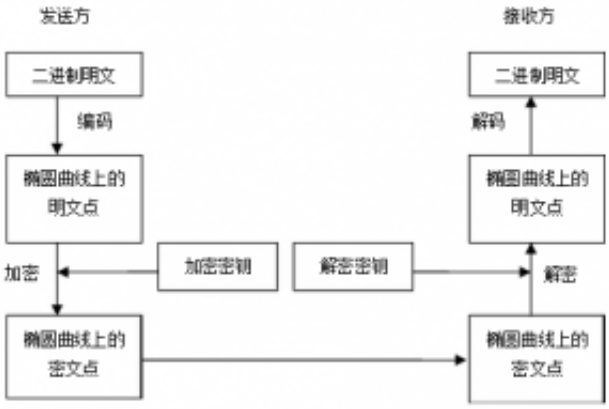

图 1　椭圆曲线密码算法的加解密流程图

函数调用图如下：

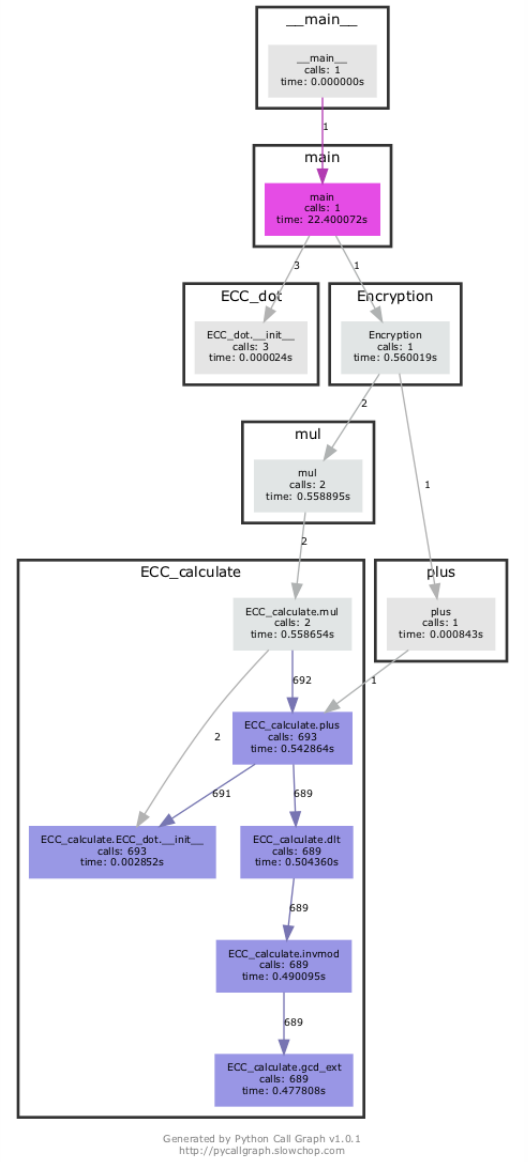

图4 ECC加密函数调用

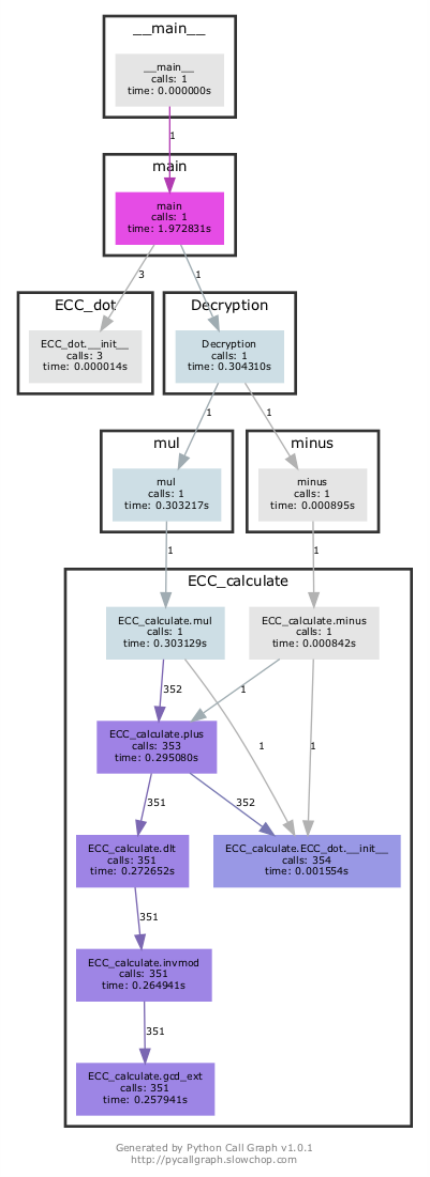

图5 ECC解密函数调用

图中加解密函数对应`Encryption()`,`Decryption()`，其中ECC_calculate部分为实验内容1的四则运算的代码文件。

伪代码如下：

算法 1 ECC加解密

输入： $p, a, b, G, op, (Pm, k, Pb)/(C1, C2, nb)$

输出： $(C1, C2)/Pm$

function ECC$(p, a, b, G, op, (Pm, k, Pb)/(C1, C2, nb))$
  if $op == 1$ **then**
    $C1 = kG$
    $C2 = Pm + kPb$
    return $(C1, C2)$
  else
    $M = C2 - nbC1$
    return M
  end if
end function

图6 ECC加解密伪代码

2.2 测试结果

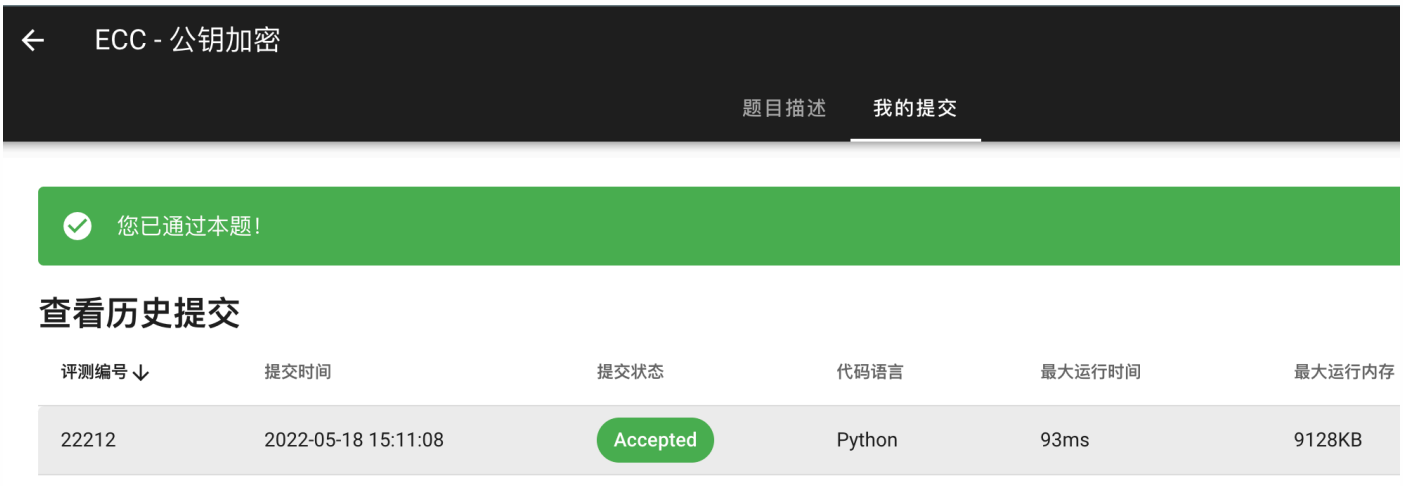

图7 ECC公钥加密测试结果

3 ECCDH密钥交换协议

3.1 算法流程

过程与Diffie-Hellman协议一致，这里一方将自己产生的随机数与基点相乘后传输给另一方，另一方再将该点与自己的随机数相乘。双方的密钥就都是$K = Xa \times Xb \times G$ 。

下面是函数调用图：

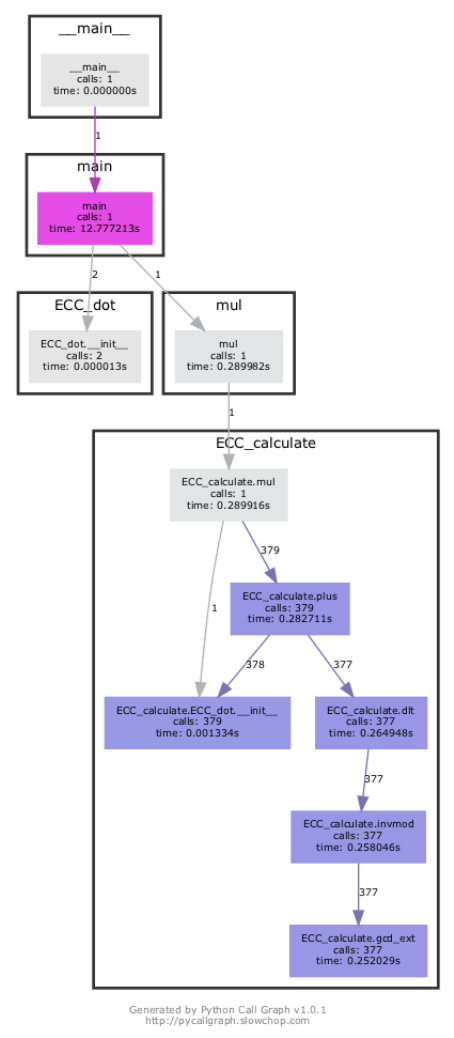

图8 ECCDH的函数调用

3.2 测试结果

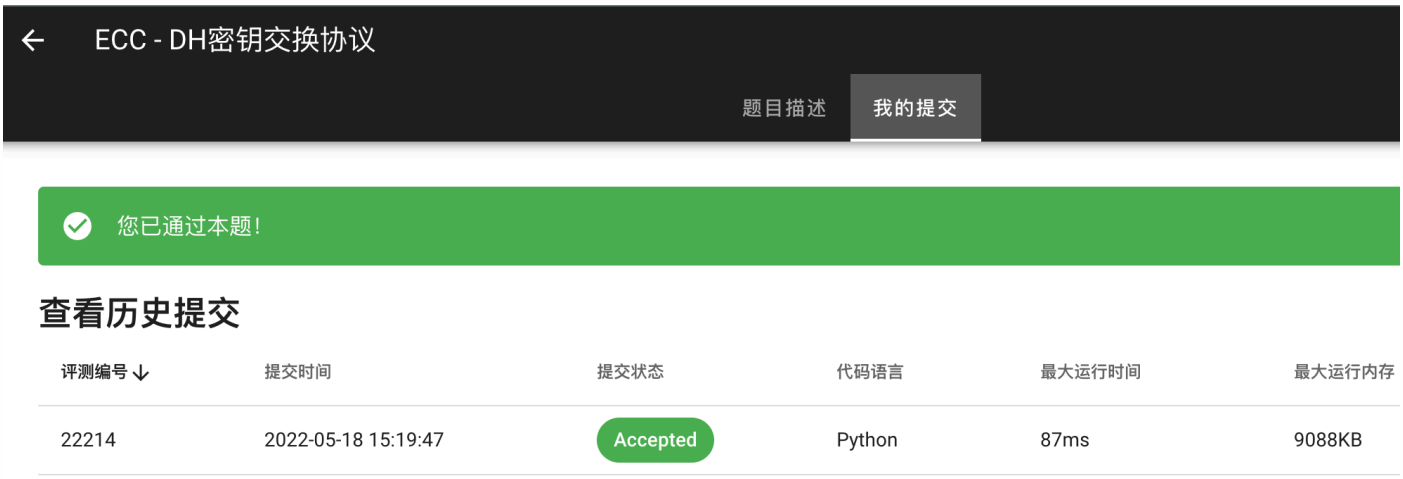

图9 ECC密钥交换测试结果

4 SM2

4.1 算法流程

4.1.1 算法总览

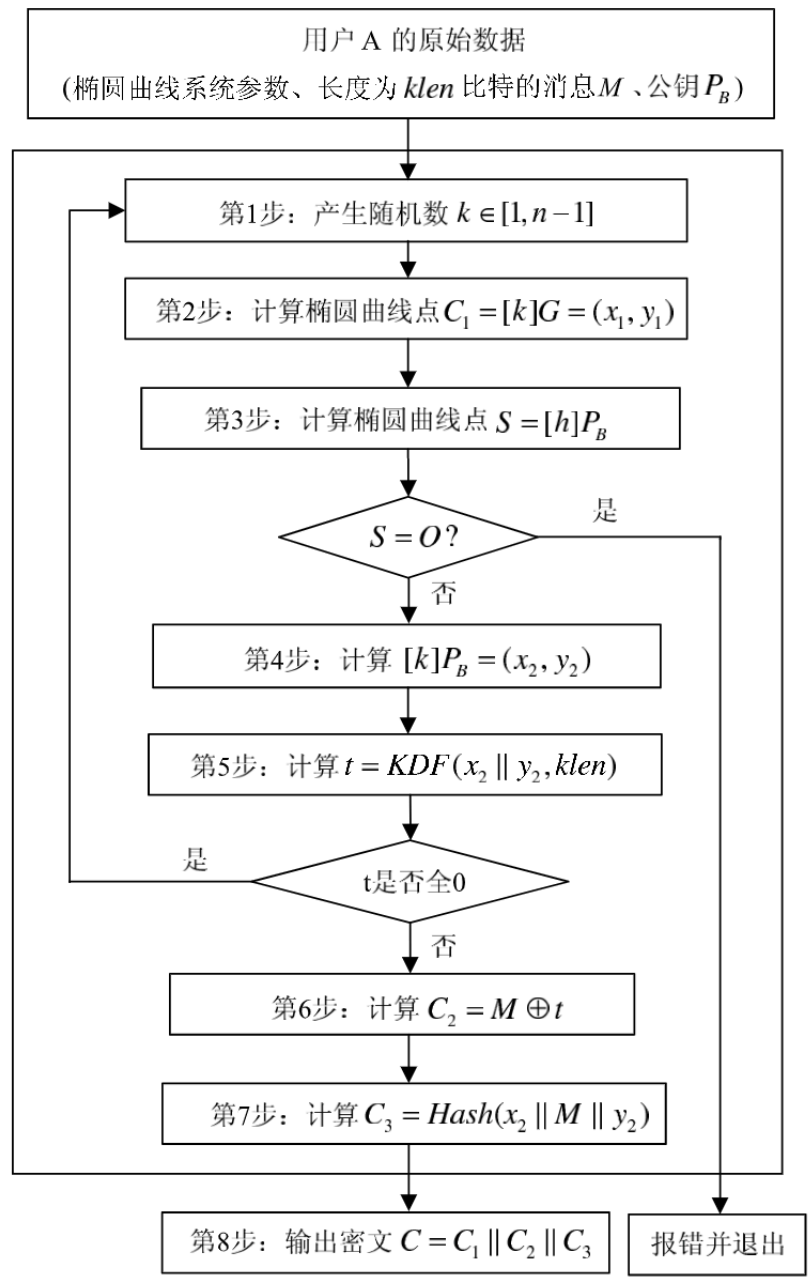

图10 SM2算法加密流程

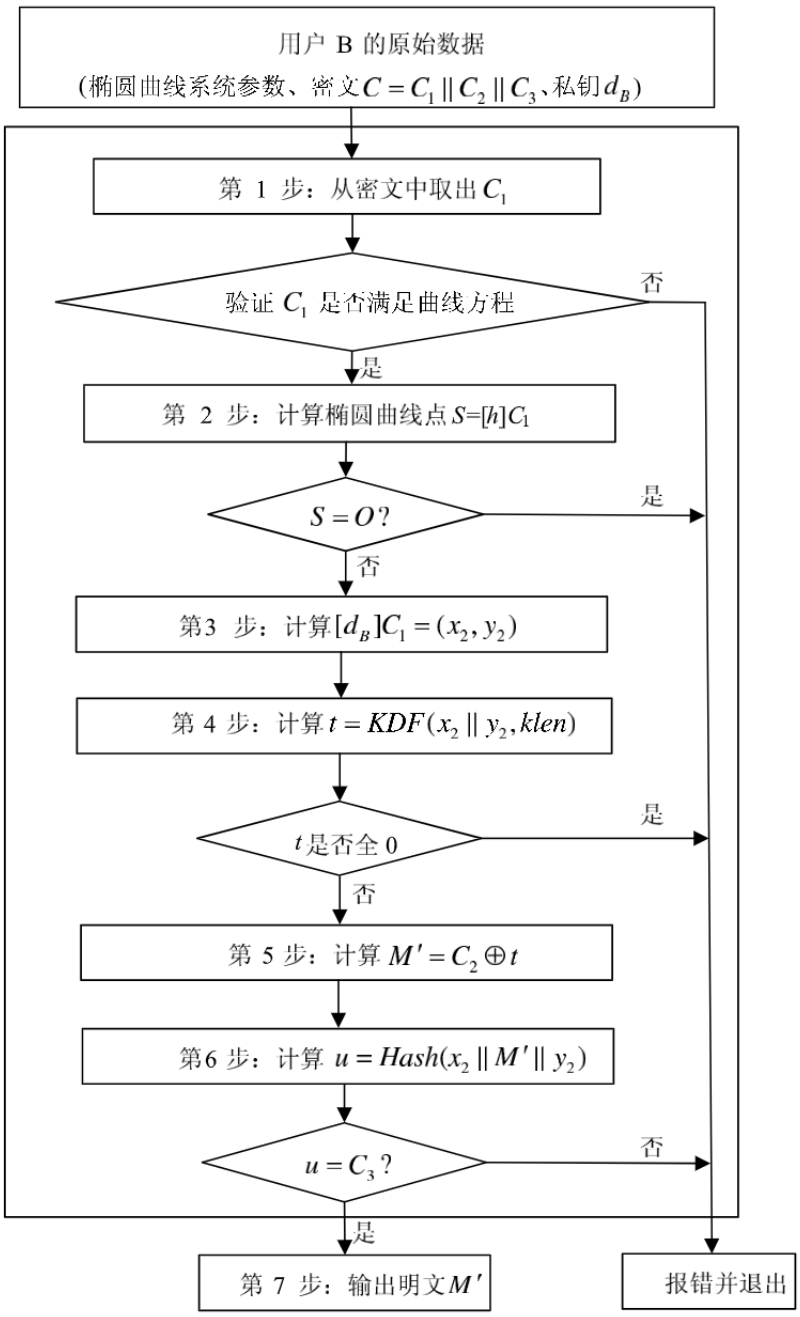
用户 B 的原始数据
(椭圆曲线系统参数、密文$C=C_1\|C_2\|C_3$、私钥d_B)
第 1 步：从密文中取出C_1
验证 C_1 是否满足曲线方程
否
是
第 2 步：计算椭圆曲线点$S=[h]C_1$
$S=O$?
是
否
第3 步：计算$[d_B]C_1=(x_2,y_2)$
第 4 步：计算$t=KDF(x_2\|y_2,klen)$
t是否全 0
是
否
第 5 步：计算$M'=C_2\oplus t$
第6 步：计算 $u=Hash(x_2\|M'\|y_2)$
$u=C_3$?
否
是
第 7 步：输出明文M'
报错并退出

图11 SM2解密流程

算法 2 SM2

输入： $p, a, b, G, op, M/C, (Pb, k)/db$

输出： M/C

function SM2$(p, a, b, G, op, M/C, (Pb, k)/db)$
  if $op == 1$ **then**
    $C1 = kG$
    $tmp(x_2, y_2) = kPb$
    $t = KDF(x_2||y_2, klen)$
    $C2 = M \oplus t$
    $C3 = hash(x2||M||y2)$
    return $C1||C2||C3$
  else
    $C1 = M[: 2 * l + 1]$
    $tmp(x_2, y_2) = dbC1$
    $t = KDF(x_2||y_2, klen)$
    $C2 = M[2 * l + 1 : 2 * l + 1 + (klen//8)]$
    $M = C2 \oplus t$
    return M
  end if
end function

图12 SM2加解密伪代码

4.1.2密钥派生函数KDF

密钥派生函数需要调用杂凑函数，设为H，输出长度为v比特的串，输入不定长度的Z与要获得的密钥数据的比特长度klen，输出数据比特串K。具体过程如伪代码所示。

算法 3 KDF

输入： $Z, klen$

输出： K

function SM2$(Z, klen)$
  $ct = 0x00000001$
  $par = heil(klen/v)$
  for $i = 1 \rightarrow heil(klen/v)$ **do**
    $Ha_i = H_v(Z||t)$
    $ct++$
  end for
  if $klen\ is\ integer$ **then**
    $Ha!_{par} = Ha_{par}$
  else
    $the\ left\ klen - v * floor(klen/v)\ bits\ of\ Ha_{par}$
  end if
  $K = Ha1||Ha2||...||Ha_{par-1}||Ha!_{par}$
  return K
end function

图13 KDF函数伪代码

4.1.3 函数调用

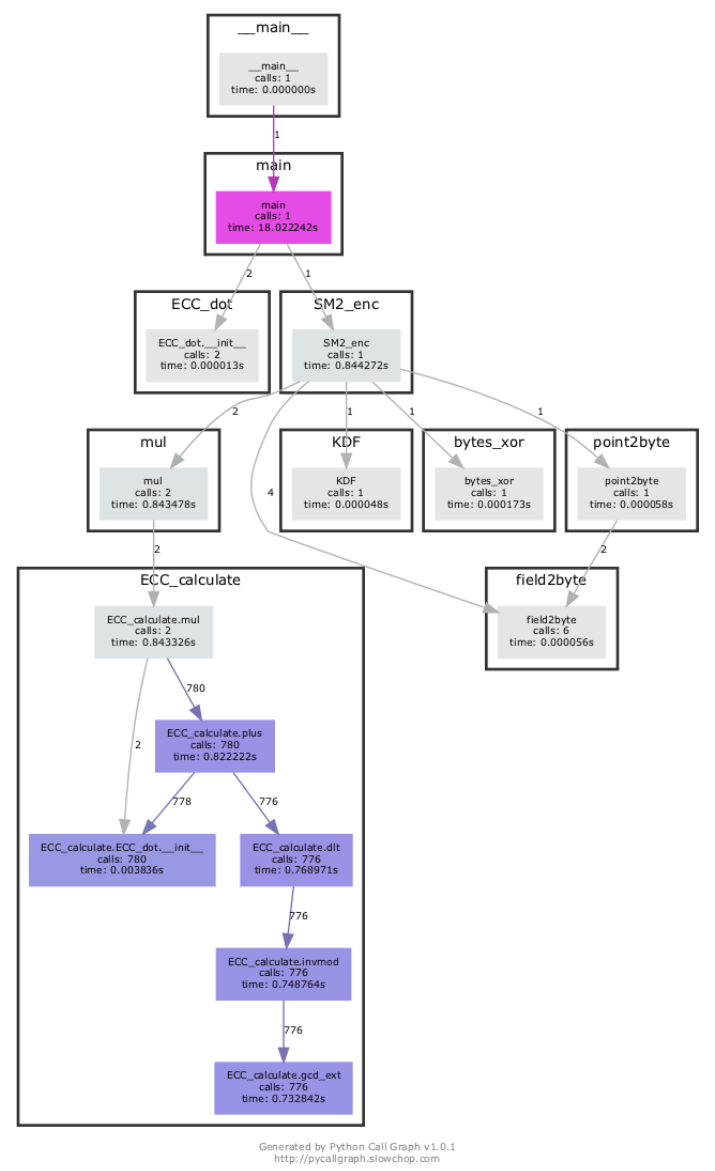

图14 SM2加密函数调用

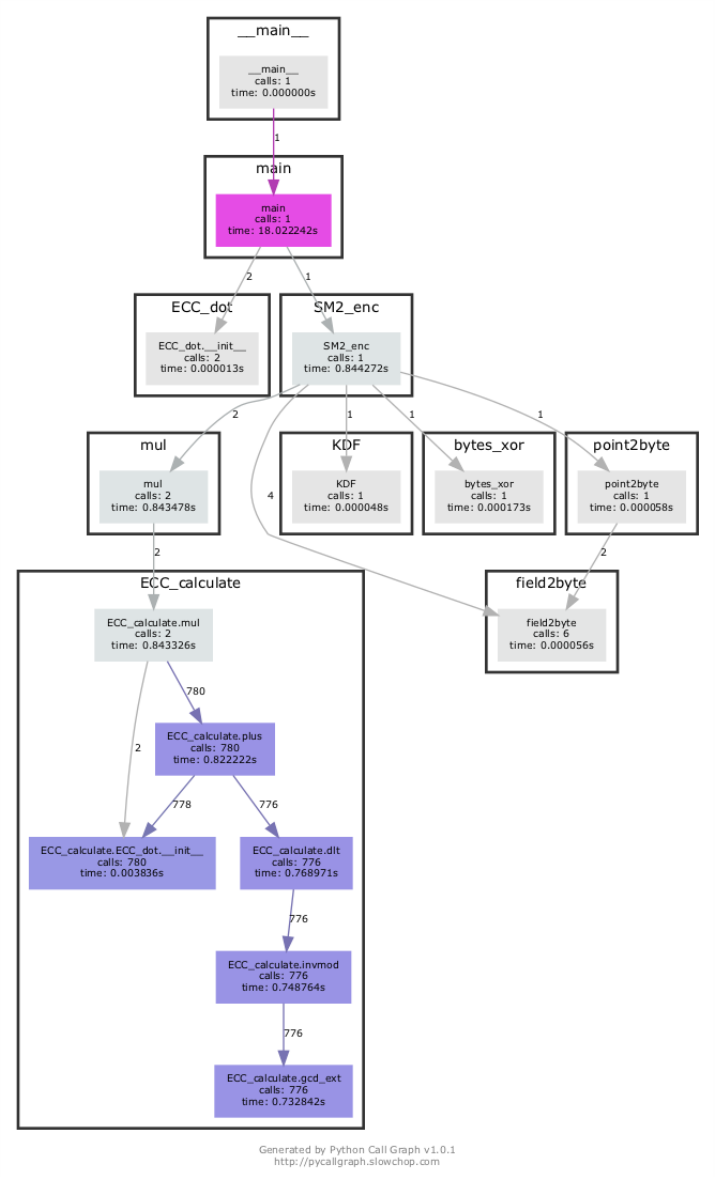

图15 SM2解密函数调用

4.2 测试结果

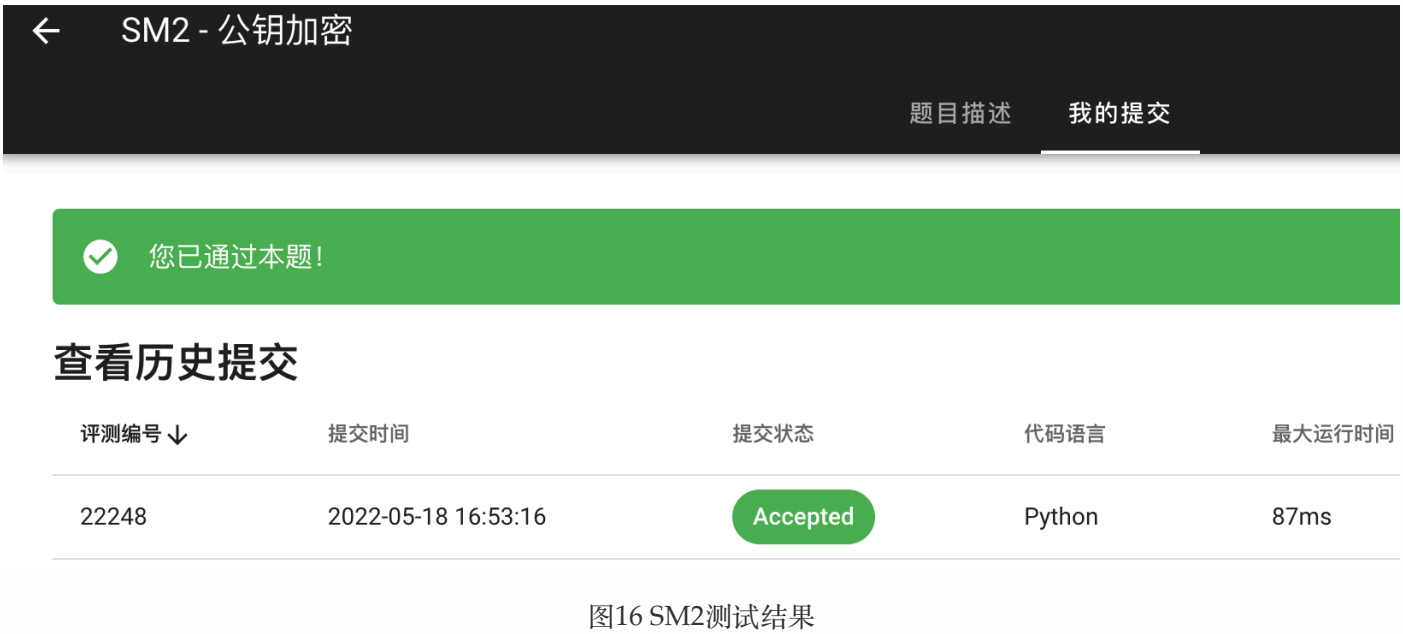

图16 SM2测试结果

5 SM2-DH

5.1 算法流程

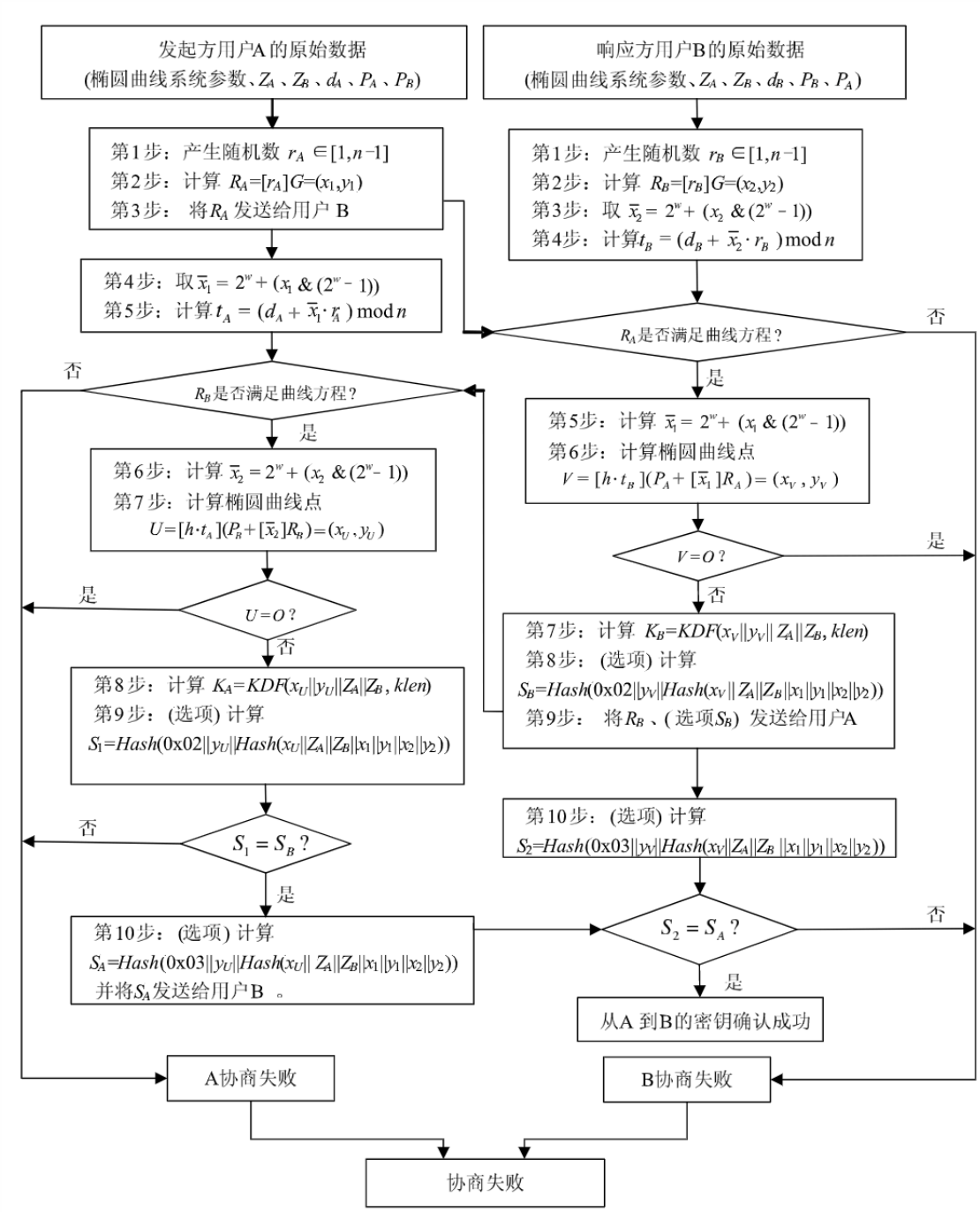

图17 SM2DH流程图

由于本题只需要完成一个参与者并且不考虑协商失败，故下面分别给出AB两人的行为操作调用。并且完整过程在5.3中会给出。

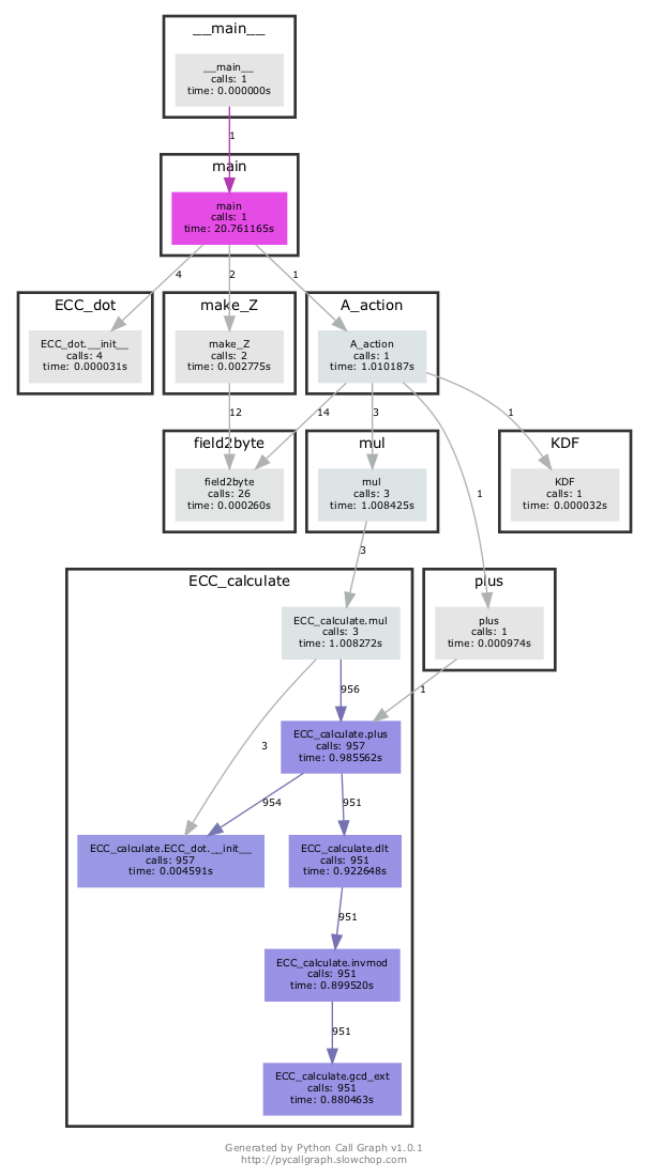

图18 A的行为函数调用

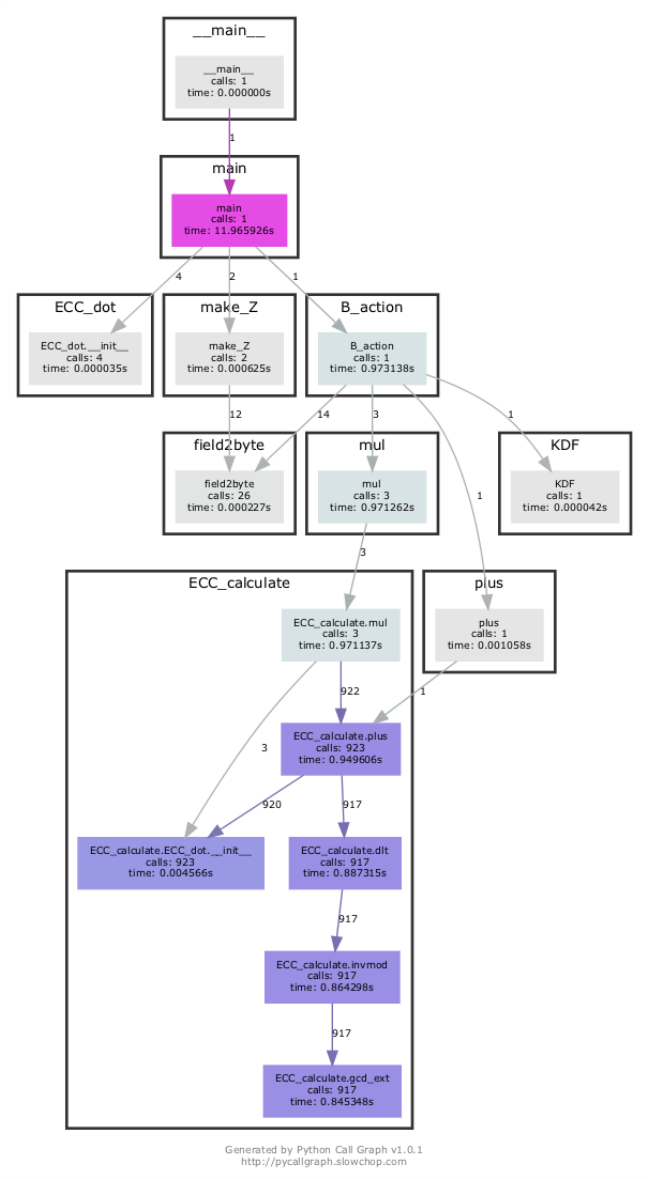

图19 B的行为函数调用

5.2 测试结果

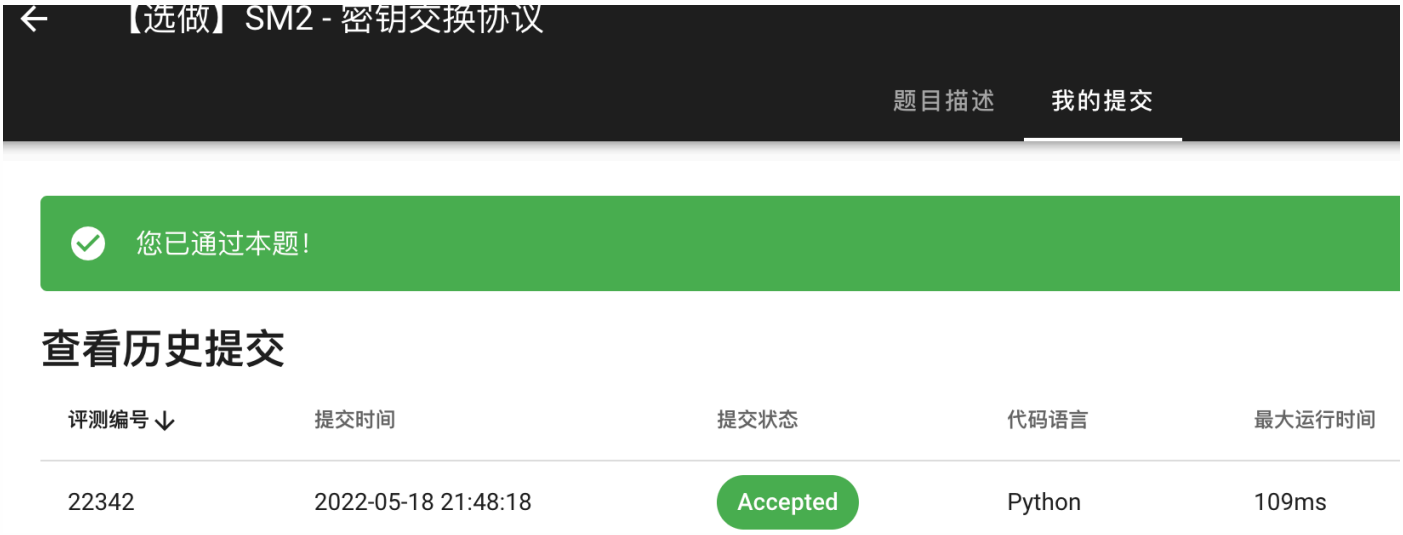

图20 SM2DH测试结果

5.3 完整行为与协商失败考虑

```
def B_action(p, a, b, G, n, ZA, ZB, d, Pa, Pb, r, R, w):
    """
    密钥交换B的行为
    :return: 协商密钥KB与两个校验数SB, S2
    """
    '''
    A1: rA = random.randint(1, n-1)
    A2: Ra = mul(G, p, a, b, r)
        x1, y1 = Ra.x, Ra.y
    A3: A send Ra to B
    '''
    Rb = mul(G, p, a, b, r)
    x2, y2 = Rb.x, Rb.y
    # print(hex(x2), hex(y2))
    x2_ba = 2 ** w + (x2 & (2 ** w - 1))
    # print(hex(x2_ba))
    tb = (d + x2_ba * r) % n
    # print(hex(tb))
    x1, y1 = R.x, R.y
    if (y1 ** 2) % p != (x1 ** 3 + a * x1 + b) % p:
        print('Error: Ra is invalid')
        return
    else:
        x1_ba = 2 ** w + (x1 & (2 ** w - 1))
        V = mul(plus(Pa, mul(R, p, a, b, x1_ba), p, a, b), p, a, b, h * tb)
        xv, yv = V.x, V.y
        # print(hex(xv), hex(yv)) : up to here is right
        Kb_tmp = KDF(field2byte(xv, p) + field2byte(yv, p) + ZA + ZB, 128)
        KB = Kb_tmp[0:klen // 8]
        SB = sha256(b'\x02' + field2byte(yv, p) + sha256(
            field2byte(xv, p) + ZA + ZB + field2byte(x1, p) + field2byte(y1, p)
+ field2byte(x2, p) + field2byte(y2,

                                p)).digest()).digest()
        '''
        # B send Rb, SB to A
        A:  x1_ba = 2 ** w + (x1 & (2**w-1))
            ta = (d + x1_ba * r) % n
            # A check if R is on Ep(a, b)
            if (R.y ** 2) % p != (R.x ** 3 + a * R.x + b) % p:
                print('Error: Rb is invalid')
                return
            x2, y2 = R.x, R.y
            x2_ba = 2 ** w + (x2 & (2 ** w - 1))
            U = mul(plus(Pb, mul(R, p, a, b, x2_ba), p, a, b), p, a, b, h*ta)
            # A ensure U is not O(0,0)
            if U.x == 0 and U.y == 0:
                print('Error: U is O!')
                return
```

```
            xu, yu = U.x, U.y
            Ka_tmp = KDF(field2byte(xu, p)+field2byte(yu, p)+ZA+ZB, 128)
            KA = Ka_tmp[0:klen//8]
            S1 = sha256(b'\x02'+field2byte(yu, p)+sha256(field2byte(xu,
p)+ZA+ZB+field2byte(x1, p)+field2byte(y1, p)+field2byte(x2, p)+field2byte(y2,
p)).digest()).digest()
            SA = sha256(b'\x03'+field2byte(yu, p)+sha256(field2byte(xu,
p)+ZA+ZB+field2byte(x1, p)+field2byte(y1, p)+field2byte(x2, p)+field2byte(y2,
p)).digest()).digest()
            # A send SA to B
        '''
        S2 = sha256(b'\x03' + field2byte(yv, p) + sha256(
            field2byte(xv, p) + ZA + ZB + field2byte(x1, p) + field2byte(y1, p)
+ field2byte(x2, p) + field2byte(y2,

                                p)).digest()).digest()
        '''
        B check if S2 is equal to SA:
        if S2 != SA:
            print('Error: Wrong conference!')
            return
        '''
        return sha256(
        ENTL + ID1 + field2byte(a, p) + field2byte(b, p) + field2byte(xG, p) +
field2byte(yG, p) + field2byte(x,

                             p) + field2byte(
            y, p)).digest()


def A_action(p, a, b, G, n, ZA, ZB, d, Pa, Pb, r, R, w):
    """
    A在密钥交换中的行为
    :return: 协商好的密钥KA与产生的校验数S1, SA
    """
    '''
    A1: rA = random.randint(1, n-1)
    '''
    # A2:
    Ra = mul(G, p, a, b, r)
    x1, y1 = Ra.x, Ra.y
    # A3: A send Ra to B
    '''
    B1: rB = random.randint(1, n-1)
    B2: Rb = mul(G, p, a, b, r)
        x2, y2 = Rb.x, Rb.y
    B3: x2_ba = 2 ** w + (x2 & (2 ** w - 1))
    B4: tb = (d + x2_ba * r) % n
        # B check if Ra got from A satisfies y^2 = x^3 + a*x + b.
        if (R.y ** 2) % p != (R.x ** 3 + a * R.x + b) % p:
            print('Error: Ra is invalid')
            return
```

```
    B5: x1, y1 = R.x, R.y
        x1_ba = 2 ** w + (x1 & (2 ** w - 1))
    B6: V = mul(plus(Pa, mul(R, p, a, b, x1_ba), p, a, b), p, a, b, h * tb)
        xv, yv = V.x, V.y
        # B check if V is infinity point O(0,0)
        if V.x == 0 and V.y == 0:
            print('Error: V is O!')
            return
    B7: Kb_tmp = KDF(field2byte(xv, p) + field2byte(yv, p) + ZA + ZB, 128)
    B8: SB = sha256(b'\x02' + field2byte(yv, p) + sha256(field2byte(xv, p) + ZA
+ ZB + field2byte(x1, p) + field2byte(y1, p) + field2byte(x2, p) +
field2byte(y2, p)).digest()).digest()
    # B send Rb, SB to A
    '''
    x1_ba = 2 ** w + (x1 & (2 ** w - 1))
    ta = (d + x1_ba * r) % n
    if (R.y ** 2) % p != (R.x ** 3 + a * R.x + b) % p:
        print('Error: Rb is invalid')
        return
    else:
        x2, y2 = R.x, R.y
        x2_ba = 2 ** w + (x2 & (2 ** w - 1))
        U = mul(plus(Pb, mul(R, p, a, b, x2_ba), p, a, b), p, a, b, h * ta)
        '''
        # A ensure that U is not infinity point O(0,0)
        if U.x == 0 and U.y == 0:
            print('Error: U is O!')
            return
        '''
        xu, yu = U.x, U.y
        Ka_tmp = KDF(field2byte(xu, p) + field2byte(yu, p) + ZA + ZB, 128)
        KA = Ka_tmp[0:klen // 8]
        S1 = sha256(b'\x02' + field2byte(yu, p) + sha256(
            field2byte(xu, p) + ZA + ZB + field2byte(x1, p) + field2byte(y1, p)
+ field2byte(x2, p) + field2byte(y2,

                                p)).digest()).digest()
        '''
        # A check if S1 is equal to SB
        if S1 != SB:
            print('Error: Wrong conference!')
            return
        '''
        SA = sha256(b'\x03' + field2byte(yu, p) + sha256(
            field2byte(xu, p) + ZA + ZB + field2byte(x1, p) + field2byte(y1, p)
+ field2byte(x2, p) + field2byte(y2,

                                p)).digest()).digest()
        '''
        # A send SA to B
```

```
        B9: S2 = sha256(b'\x03' + field2byte(yv, p) + sha256(field2byte(xv, p) + ZA + ZB + field2byte(x1, p) + field2byte(y1, p) + field2byte(x2, p) + field2byte(y2, p)).digest()).digest()
            # B check if SA is equal to S2
                if S2 != SA:
                    print('Error: Wrong conference!')
                    return
        '''
        return KA, S1, SA

```

6 ECC快速加密

在本题中我尝试了两种方式，分别是蒙哥马利算法与加法链。

6.1 蒙哥马利

蒙哥马利算法的本质就是快速模幂算法，在尝试更换进制时发现并未快速提高乘法运算的速度，故保留使用2进制，并对原有的乘法进行了优化。

6.2 加法链

加法链可以有效减少乘法中的加法次数，所以可以使用加法链来优化。

我采用了最短加法链的想法，代码如下：

```python
#输出加法链
def Print(x,num):
    for i in range(1,num+1):
        print(x[i],end=" ")
    print()
#回退到上一个元素
def Back(x,Sum,Add,num):
    #x[num]=0
    Sum-=x[Add[num]]#减去上一步添加的元素
    Add[num+1]=num+1#恢复Add[num]
    num=num-1
    return x,Sum,Add,num
def ADD(n):
    x=[0 for i in range(n+1)]#当前加法链
    #加法链的第一个元素是1,第二个一定是2
    x[1]=1
    x[2]=2
    Sum=2#当前计算的和
    num=2#当前加法链的长度为1
    bestx=[0 for i in range(n+1)]#最优加法链
    Add=[i for i in range(n+1)]#Add[i]记录当前链第i个元素的加数的下标
    Add[2]=1
    bestnum=n#最优加法链的长度最大为n
    i=2#下一个要加的加数的下标
    while True:

```

```
        #找加法链上的第num+1个元素
        while Sum+x[i]<=n and num+1<=bestnum:
            Sum+=x[i]
            num+=1
            Add[num]=i
            i=Add[num+1]-1#为下一次尝试做准备
            x[num]=Sum
        if Sum==n:
            bestx=list(x)
            bestnum=num
            #Print(bestx,bestnum)
            #当找到一种解之后，只倒退一步或者两步时是不会再出现链长更短的解的
            x,Sum,Add,num=Back(x,Sum,Add,num)
            x,Sum,Add,num=Back(x,Sum,Add,num)
            x,Sum,Add,num=Back(x,Sum,Add,num)
            i=Add[num+1]-1
            Add[num+1]=i
        else:
            #第num+1个元素加上之后链长大于最优解时回退到上一个元素
            if num+1>bestnum:
                x,Sum,Add,num=Back(x,Sum,Add,num)
                x,Sum,Add,num=Back(x,Sum,Add,num)
            i=Add[num+1]-1
            Add[num+1]=i#更新加数的下标
        while i<1:#第num+1个元素的所有可加项都尝试过
            x,Sum,Add,num=Back(x,Sum,Add,num)#回退到上一个元素
            i=Add[num+1]-1
            Add[num+1]=i
        if num<2:#所有元素的所有可加项都尝试过
            return bestx,bestnum


# 输出结果用于验证
bestx, bestnum=ADD(98)
print("最短链长:", bestnum-1)
print("链中的元素:")
Print(bestx,bestnum)
```

问题在于，成功实现了最短加法链后，依然速度不高，是因为要找到一个极大的整数k的最短加法链是极其困难的，时间复杂度很大，故最短加法链不适用，所以我并没有实现，因为当k=999时就已经需要等很长时间来得到加法链结果了。但是使用加法链应该也是可以的，我还没有继续探索完毕。

6.3 测试结果

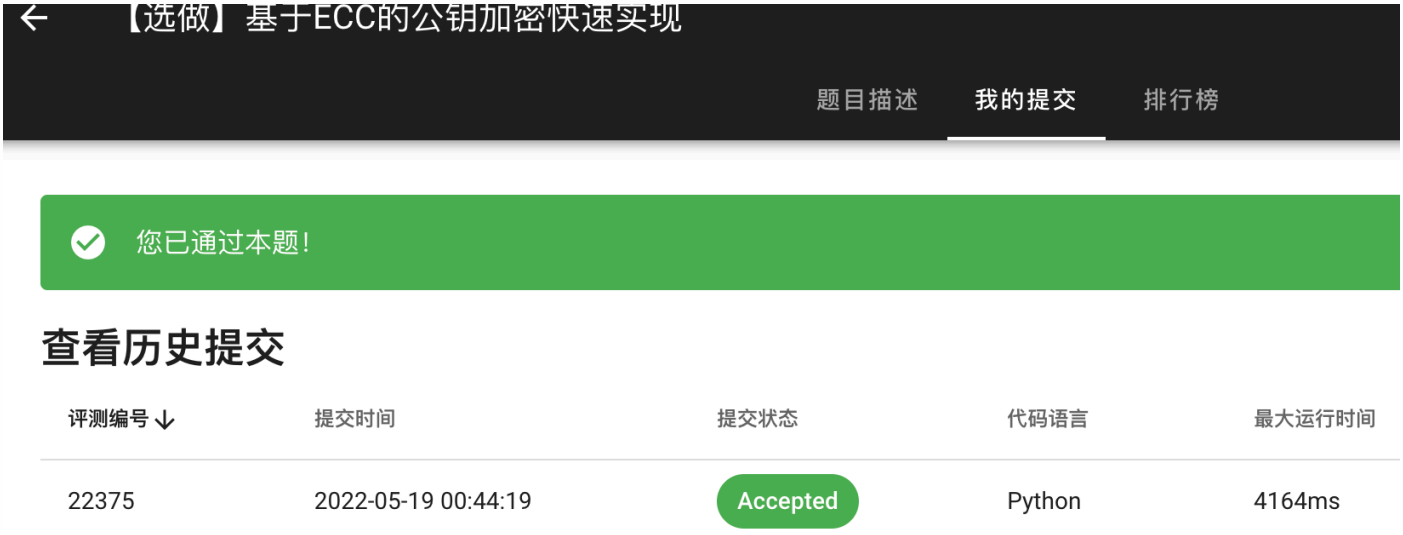

图21 ECC快速算法测试

7 讨论与思考

ECC算法抗攻击性强，CPU占用少，内容使用少，网络消耗低，加密速度快，更高的扩展性。而RSA的数学原理简单，在工程应用中易于实现，且已经较为成熟。

8 总结与感悟

在本次实验中完成了ECC与SM2算法的实现，感受颇深，在之后的学习生活中也会继续体会椭圆曲线加密的奥妙。