

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;

其中一个为无穷远点时则结果等于另一个点;

否则就有公式如下

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

$$\begin{aligned} Rx &= \delta^2 - Px - Qx(\text{mod } p) \\ Ry &= \delta \times (Px - Qx) - Py(\text{mod } p) \end{aligned}$$

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:

```

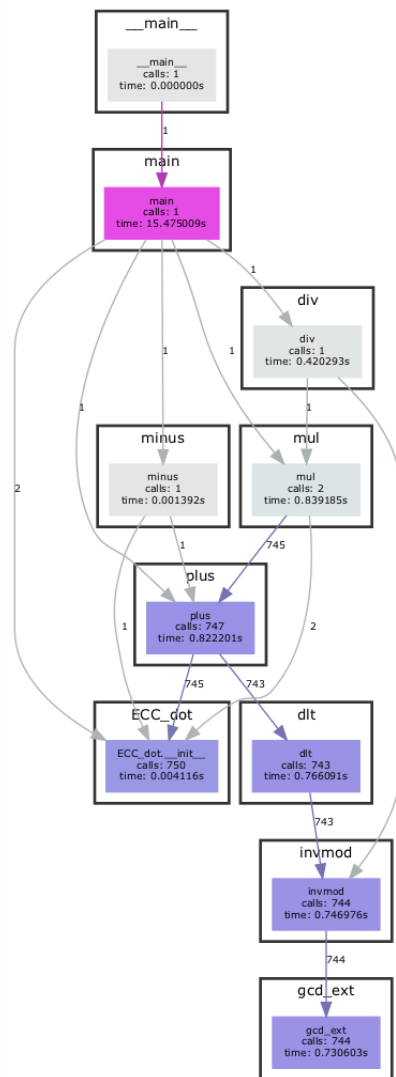
1  def mul(P, p, a, b, k):
2      bin = []
3      ind = k
4      while ind:
5          bin.append(ind % 2)
6          ind = ind // 2
7      c = ECC_dot(0, 0)
8      for i in range(len(bin) - 1, -1, -1):
9          c = plus(c, c, p, a, b)
10         if bin[i] == 1:
11             c = plus(P, c, p, a, b)
12     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



Generated by Python Call Graph v1.0.1
http://pycallgraph.slowchop.com

图1 ECC四则运算函数调用图

1.2 测试结果

← ECC - 四则运算

题目描述我的提交

✔ 您已通过本题!

查看历史提交

评测编号 ↓	提交时间	提交状态	代码语言
22207	2022-05-18 14:41:47	Accepted	Python

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 $E_p(a,b)$ ，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 $E_p(a,b)$ 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 $E_p(a,b)$ 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：

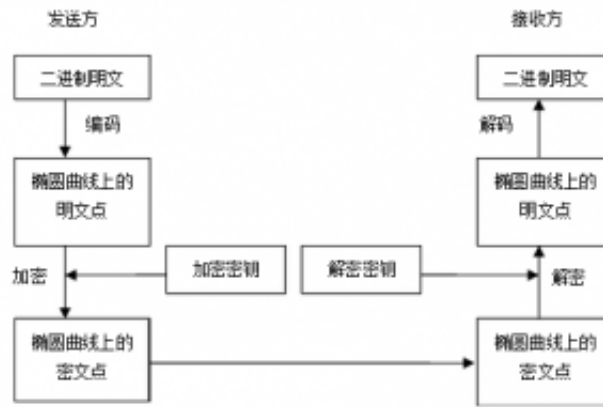


图 1 椭圆曲线密码算法的加解密流程图

函数调用图如下：

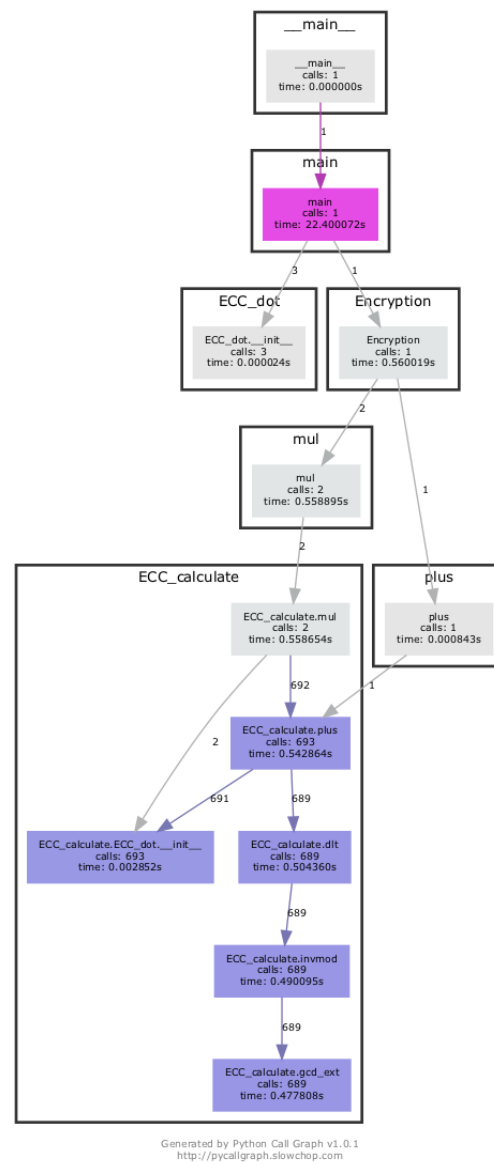


图4 ECC加密函数调用

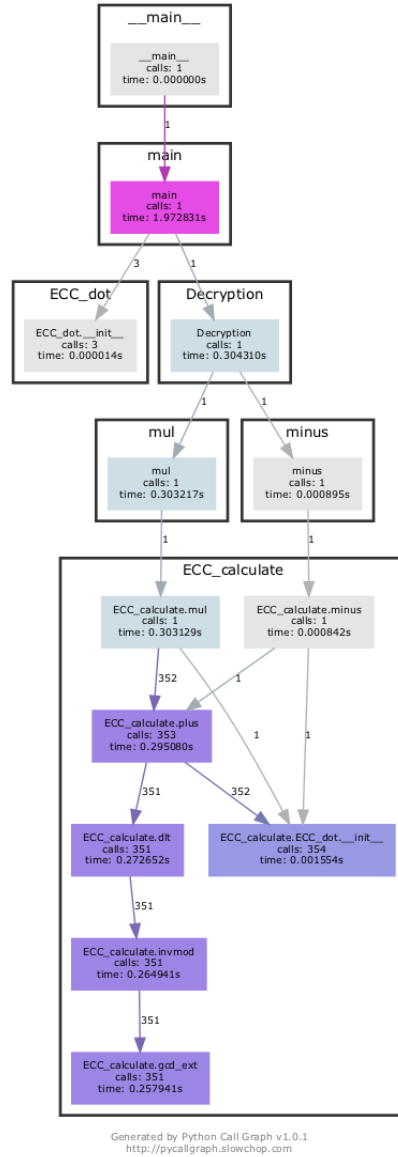


图5 ECC解密函数调用

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: $(C1, C2)/Pm$

```

1: function ECC( $p, a, b, G, op, (Pm, k, Pb)/(C1, C2, nb)$ )
2:   if  $op == 1$  then
3:      $C1 = kG$ 
4:      $C2 = Pm + kPb$ 
5:     return  $(C1, C2)$ 
6:   else
7:      $M = C2 - nbC1$ 
8:     return  $M$ 
9:   end if
10: end function

```

图6 ECC加解密伪代码

2.2 测试结果

← ECC - 公钥加密

题目描述我的提交

✔ 您已通过本题!

查看历史提交

评测编号 ↓	提交时间	提交状态	代码语言	最大运行时间	最大运行内存
22212	2022-05-18 15:11:08	Accepted	Python	93ms	9128KB

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：

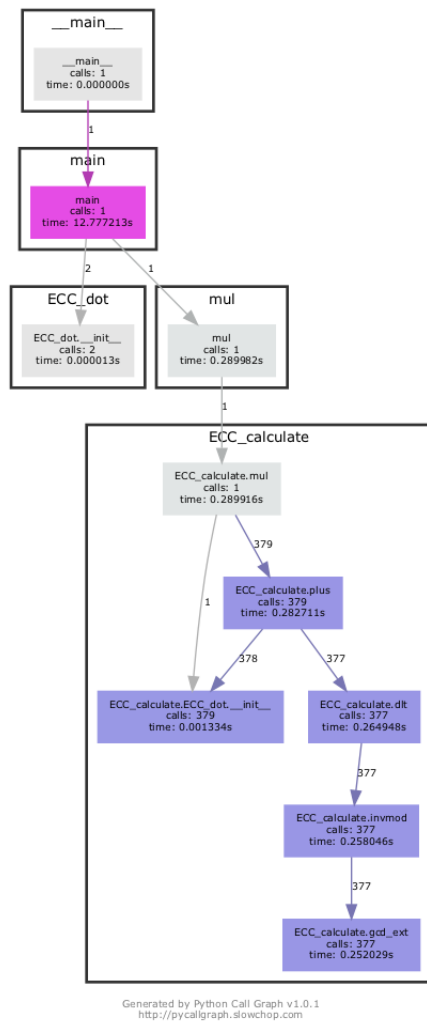


图8 ECCDH的函数调用

3.2 测试结果

← ECC - DH密钥交换协议

题目描述

我的提交

✓

您已通过本题！

查看历史提交

评测编号 ↓	提交时间	提交状态	代码语言	最大运行时间	最大运行内存
22214	2022-05-18 15:19:47	Accepted	Python	87ms	9088KB

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

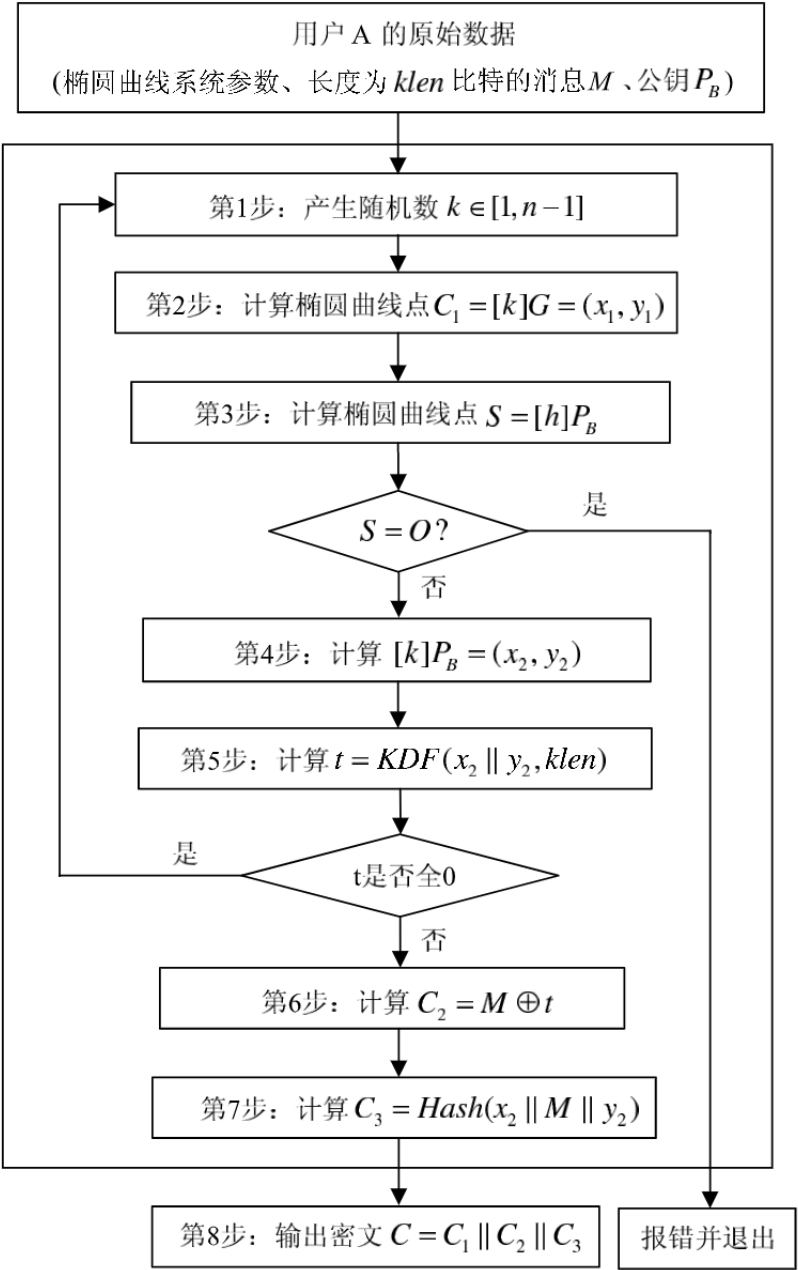


图10 SM2算法加密流程

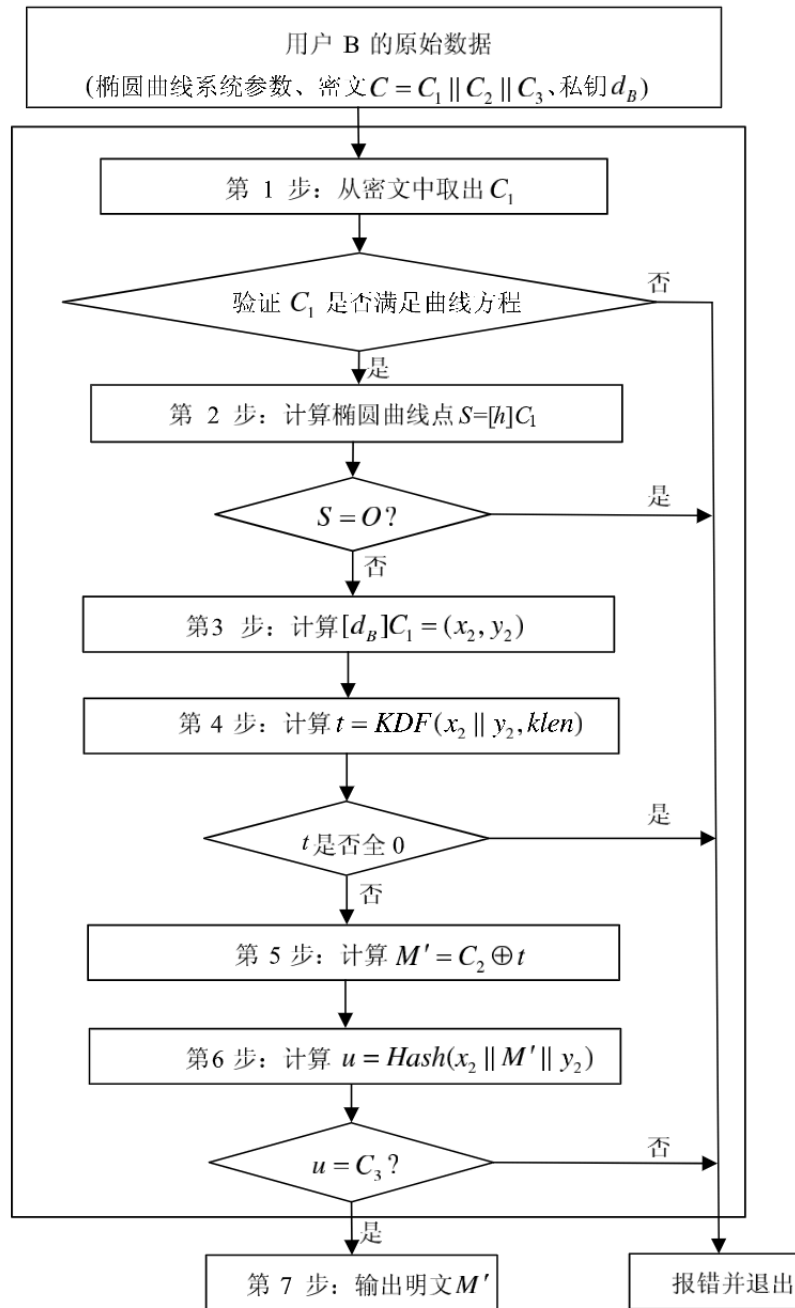


图11 SM2解密流程

算法 2 SM2

输入: $p, a, b, G, op, M/C, (Pb, k)/db$

输出: M/C

```
1: function SM2( $p, a, b, G, op, M/C, (Pb, k)/db$ )
2:   if  $op == 1$  then
3:      $C1 = kG$ 
4:      $tmp(x_2, y_2) = kPb$ 
5:      $t = KDF(x_2 || y_2, klen)$ 
6:      $C2 = M \oplus t$ 
7:      $C3 = hash(x_2 || M || y_2)$ 
8:     return  $C1 || C2 || C3$ 
9:   else
10:     $C1 = M[: 2 * l + 1]$ 
11:     $tmp(x_2, y_2) = dbC1$ 
12:     $t = KDF(x_2 || y_2, klen)$ 
13:     $C2 = M[2 * l + 1 : 2 * l + 1 + (klen//8)]$ 
14:     $M = C2 \oplus t$ 
15:    return  $M$ 
16:   end if
17: 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

```
1: function SM2( $Z, klen$ )
2:    $ct = 0x00000001$ 
3:    $par = ceil(klen/v)$ 
4:   for  $i = 1 \rightarrow ceil(klen/v)$  do
5:      $Ha_i = H_v(Z || t)$ 
6:      $ct++$ 
7:   end for
8:   if  $klen$  is integer then
9:      $Ha!_{par} = Ha_{par}$ 
10:  else
11:    the left  $klen - v * floor(klen/v)$  bits of  $Ha_{par}$ 
12:  end if
13:   $K = Ha_1 || Ha_2 || \dots || Ha_{par-1} || Ha!_{par}$ 
14:  return  $K$ 
15: end function
```

图13 KDF函数伪代码

4.1.3 函数调用

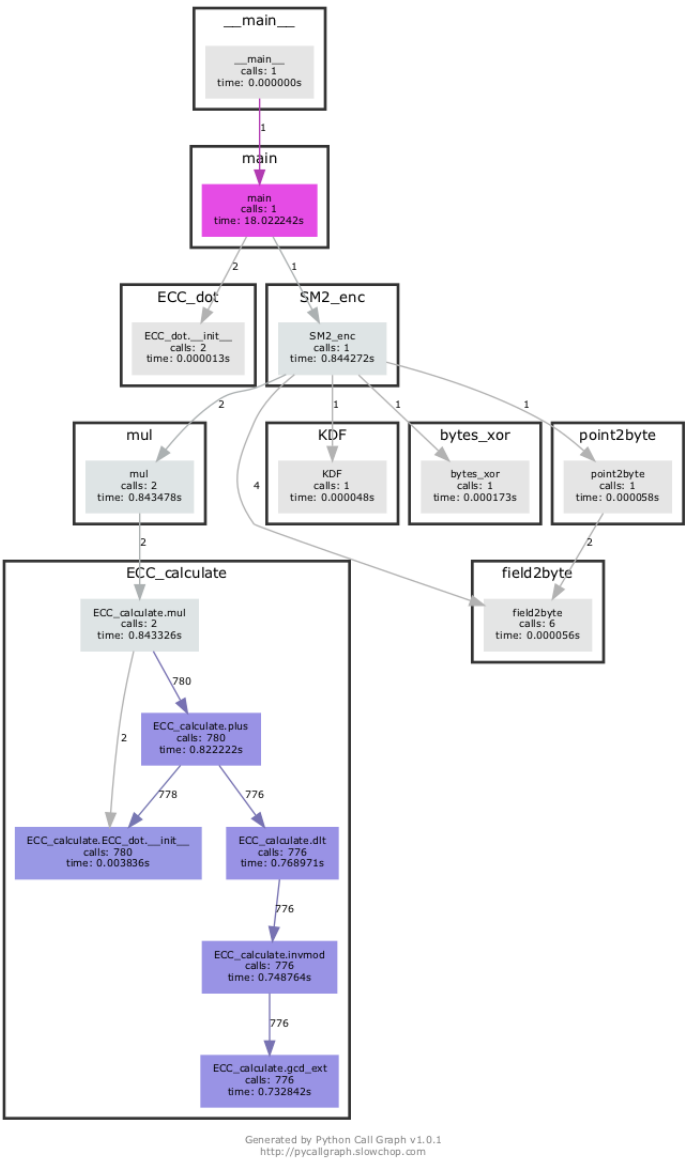


图14 SM2加密函数调用

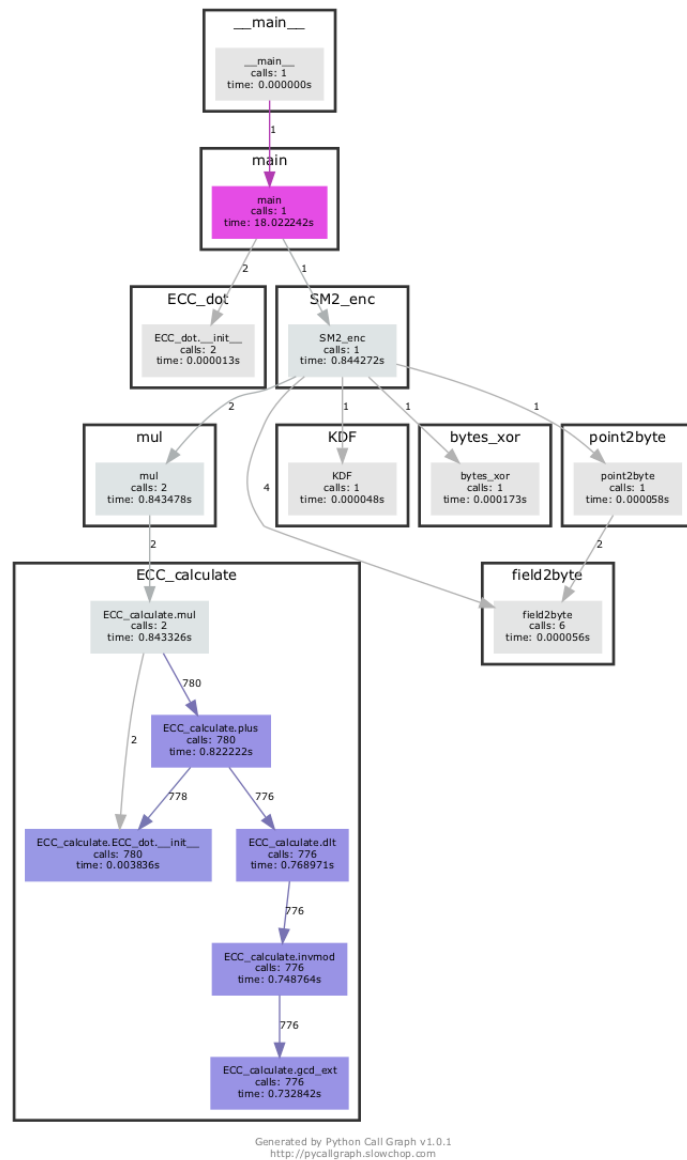


图15 SM2解密函数调用

4.2 测试结果

←

SM2 - 公钥加密

题目描述

我的提交

✓

您已通过本题!

查看历史提交

评测编号 ↓	提交时间	提交状态	代码语言	最大运行时间
22248	2022-05-18 16:53:16	Accepted	Python	87ms

图16 SM2测试结果

5 SM2-DH

5.1 算法流程

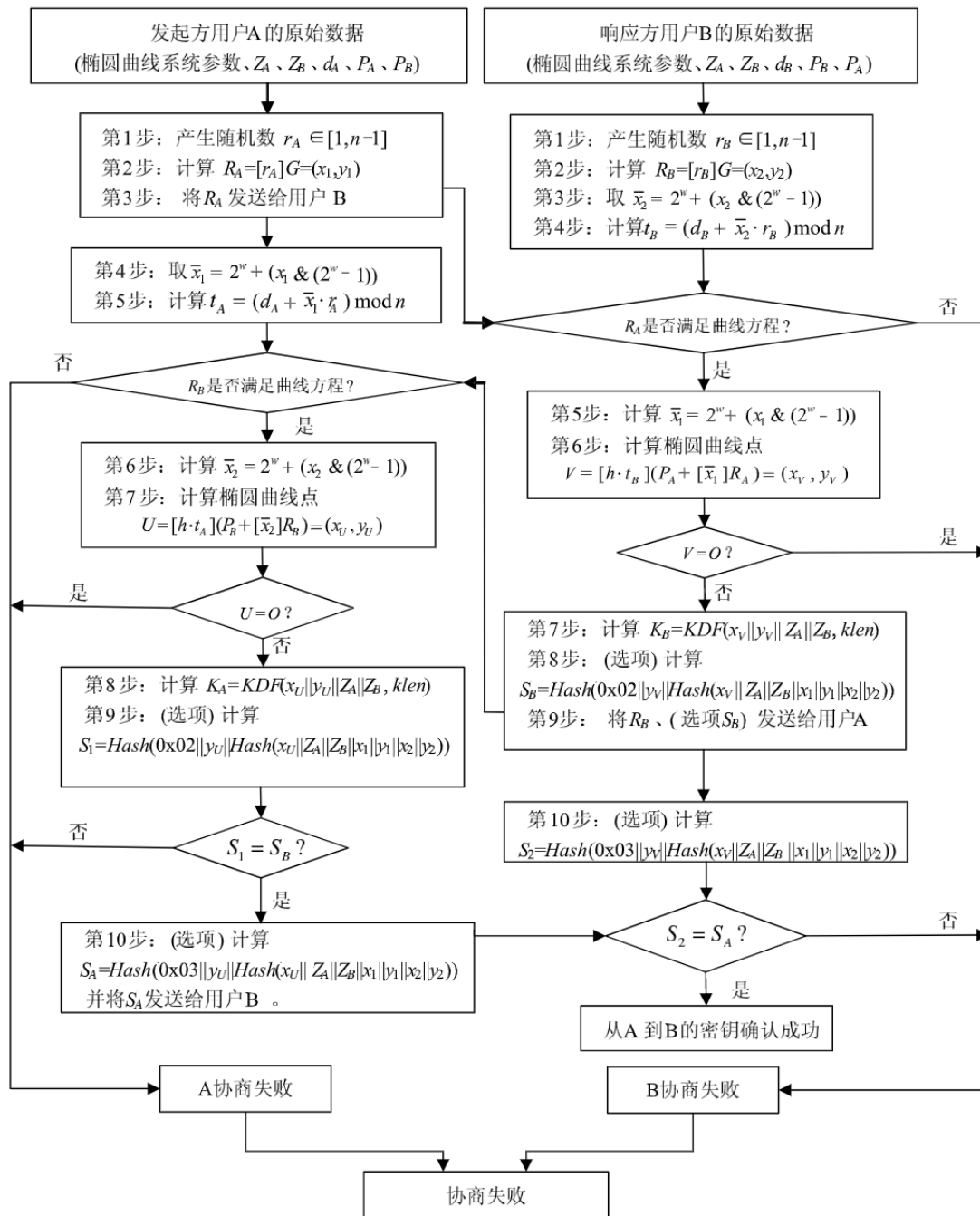


图17 SM2DH流程图

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。

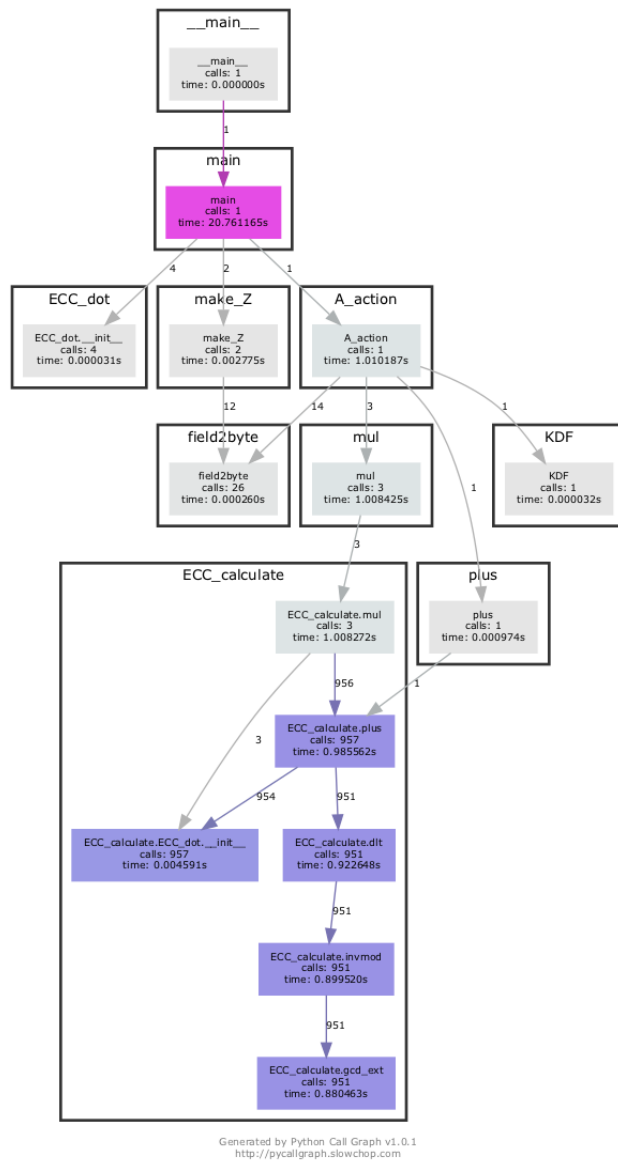


图18 A的行为函数调用

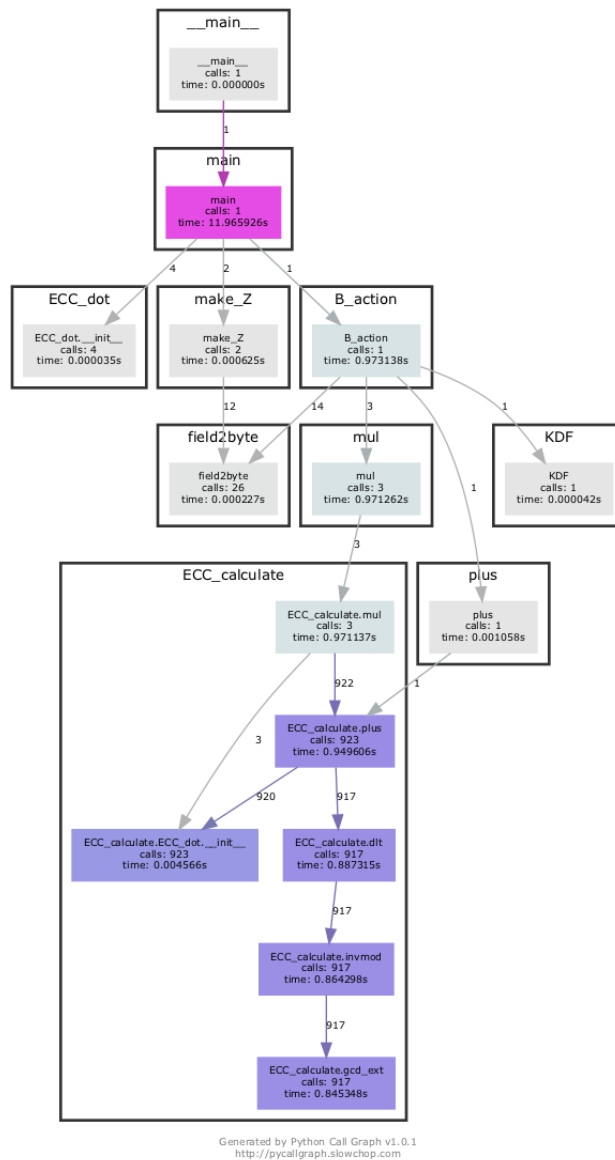


图19 B的行为函数调用

5.2 测试结果

←
【选做】SM2 - 密钥交换协议

题目描述

我的提交

✓ 您已通过本题!

查看历史提交

评测编号 ↓	提交时间	提交状态	代码语言	最大运行时间
22342	2022-05-18 21:48:18	Accepted	Python	109ms

图20 SM2DH测试结果

5.3 完整行为与协商失败考虑

```
1 def B_action(p, a, b, G, n, ZA, ZB, d, Pa, Pb, r, R, w):
2     """
3     密钥交换B的行为
4     :return: 协商密钥KB与两个校验数SB, S2
5     """
6     ...
7     A1: rA = random.randint(1, n-1)
8     A2: Ra = mul(G, p, a, b, r)
9         x1, y1 = Ra.x, Ra.y
10    A3: A send Ra to B
11    ...
12    Rb = mul(G, p, a, b, r)
13    x2, y2 = Rb.x, Rb.y
14    # print(hex(x2), hex(y2))
15    x2_ba = 2 ** w + (x2 & (2 ** w - 1))
16    # print(hex(x2_ba))
17    tb = (d + x2_ba * r) % n
18    # print(hex(tb))
19    x1, y1 = R.x, R.y
20    if (y1 ** 2) % p != (x1 ** 3 + a * x1 + b) % p:
21        print('Error: Ra is invalid')
22        return
23    else:
24        x1_ba = 2 ** w + (x1 & (2 ** w - 1))
25        V = mul(plus(Pa, mul(R, p, a, b, x1_ba), p, a, b), p, a, b, h * tb)
26        xv, yv = V.x, V.y
27        # print(hex(xv), hex(yv)) : up to here is right
28        Kb_tmp = KDF(field2byte(xv, p) + field2byte(yv, p) + ZA + ZB, 128)
29        KB = Kb_tmp[0:klen // 8]
30        SB = sha256(b'\x02' + field2byte(yv, p) + sha256(
31            field2byte(xv, p) + ZA + ZB + field2byte(x1, p) + field2byte(y1, p)
32            + field2byte(x2, p) + field2byte(y2,
33
34                p)).digest()).digest()
35        ...
36        # B send Rb, SB to A
37        A: x1_ba = 2 ** w + (x1 & (2**w-1))
38            ta = (d + x1_ba * r) % n
39            # A check if R is on Ep(a, b)
40            if (R.y ** 2) % p != (R.x ** 3 + a * R.x + b) % p:
41                print('Error: Rb is invalid')
42                return
43            x2, y2 = R.x, R.y
44            x2_ba = 2 ** w + (x2 & (2 ** w - 1))
45            U = mul(plus(Pb, mul(R, p, a, b, x2_ba), p, a, b), p, a, b, h*ta)
46            # A ensure U is not 0(0,0)
47            if U.x == 0 and U.y == 0:
48                print('Error: U is 0!')
```



```

48         xu, yu = U.x, U.y
49         Ka_tmp = KDF(field2byte(xu, p)+field2byte(yu, p)+ZA+ZB, 128)
50         KA = Ka_tmp[0:klen//8]
51         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()
52         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()
53         # A send SA to B
54         '''
55         S2 = sha256(b'\x03' + field2byte(yv, p) + sha256(
56             field2byte(xv, p) + ZA + ZB + field2byte(x1, p) + field2byte(y1, p)
+ field2byte(x2, p) + field2byte(y2,
57
p)).digest()).digest()
58         '''
59         B check if S2 is equal to SA:
60         if S2 != SA:
61             print('Error: Wrong conference!')
62             return
63         '''
64         return sha256(
65             ENTL + ID1 + field2byte(a, p) + field2byte(b, p) + field2byte(xG, p) +
field2byte(yG, p) + field2byte(x,
66
p) + field2byte(
67             y, p)).digest()
68
69
70 def A_action(p, a, b, G, n, ZA, ZB, d, Pa, Pb, r, R, w):
71     '''
72     A在密钥交换中的行为
73     :return: 协商好的密钥KA与产生的校验数S1, SA
74     '''
75     '''
76     A1: rA = random.randint(1, n-1)
77     '''
78     # A2:
79     Ra = mul(G, p, a, b, r)
80     x1, y1 = Ra.x, Ra.y
81     # A3: A send Ra to B
82     '''
83     B1: rB = random.randint(1, n-1)
84     B2: Rb = mul(G, p, a, b, r)
85     x2, y2 = Rb.x, Rb.y
86     B3: x2_ba = 2 ** w + (x2 & (2 ** w - 1))
87     B4: tb = (d + x2_ba * r) % n
88     # B check if Ra got from A satisfies  $y^2 = x^3 + ax + b$ .
89     if (R.y ** 2) % p != (R.x ** 3 + a * R.x + b) % p:
90         print('Error: Ra is invalid')
91         return

```

```

92     B5: x1, y1 = R.x, R.y
93         x1_ba = 2 ** w + (x1 & (2 ** w - 1))
94     B6: V = mul(plus(Pa, mul(R, p, a, b, x1_ba), p, a, b), p, a, b, h * tb)
95         xv, yv = V.x, V.y
96         # B check if V is infinity point 0(0,0)
97         if V.x == 0 and V.y == 0:
98             print('Error: V is 0!')
99             return
100     B7: Kb_tmp = KDF(field2byte(xv, p) + field2byte(yv, p) + ZA + ZB, 128)
101     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()
102     # B send Rb, SB to A
103     '''
104     x1_ba = 2 ** w + (x1 & (2 ** w - 1))
105     ta = (d + x1_ba * r) % n
106     if (R.y ** 2) % p != (R.x ** 3 + a * R.x + b) % p:
107         print('Error: Rb is invalid')
108         return
109     else:
110         x2, y2 = R.x, R.y
111         x2_ba = 2 ** w + (x2 & (2 ** w - 1))
112         U = mul(plus(Pb, mul(R, p, a, b, x2_ba), p, a, b), p, a, b, h * ta)
113         '''
114         # A ensure that U is not infinity point 0(0,0)
115         if U.x == 0 and U.y == 0:
116             print('Error: U is 0!')
117             return
118         '''
119         xu, yu = U.x, U.y
120         Ka_tmp = KDF(field2byte(xu, p) + field2byte(yu, p) + ZA + ZB, 128)
121         KA = Ka_tmp[0:klen // 8]
122         S1 = sha256(b'\x02' + field2byte(yu, p) + sha256(
123             field2byte(xu, p) + ZA + ZB + field2byte(x1, p) + field2byte(y1, p)
+ field2byte(x2, p) + field2byte(y2,
124
p)).digest()).digest()
125         '''
126         # A check if S1 is equal to SB
127         if S1 != SB:
128             print('Error: Wrong conference!')
129             return
130         '''
131         SA = sha256(b'\x03' + field2byte(yu, p) + sha256(
132             field2byte(xu, p) + ZA + ZB + field2byte(x1, p) + field2byte(y1, p)
+ field2byte(x2, p) + field2byte(y2,
133
p)).digest()).digest()
134         '''
135         # A send SA to B

```

```

136         B9: S2 = sha256(b'\x03' + field2byte(yv, p) + sha256(field2byte(xv, p) +
137         ZA + ZB + field2byte(x1, p) + field2byte(y1, p) + field2byte(x2, p) +
138         field2byte(y2, p)).digest()).digest()
139         # B check if SA is equal to S2
140         if S2 != SA:
141             print('Error: Wrong conference!')
142             return
143         '''
144     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：

```

1  #输出加法链
2  def Print(x,num):
3      for i in range(1,num+1):
4          print(x[i],end=" ")
5      print()
6  #回退到上一个元素
7  def Back(x,Sum,Add,num):
8      #x[num]=0
9      Sum-=x[Add[num]]#减去上一步添加的元素
10     Add[num+1]=num+1#恢复Add[num]
11     num=num-1
12     return x,Sum,Add,num
13 def ADD(n):
14     x=[0 for i in range(n+1)]#当前加法链
15     #加法链的第一个元素是1,第二个一定是2
16     x[1]=1
17     x[2]=2
18     Sum=2#当前计算的和
19     num=2#当前加法链的长度为1
20     bestx=[0 for i in range(n+1)]#最优加法链
21     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
22     Add[2]=1
23     bestnum=n#最优加法链的长度最大为n
24     i=2#下一个要加的加数的下标
25     while True:

```

```

26     #找加法链上的第num+1个元素
27     while Sum+x[i]<=n and num+1<=bestnum:
28         Sum+=x[i]
29         num+=1
30         Add[num]=i
31         i=Add[num+1]-1#为下一次尝试做准备
32         x[num]=Sum
33     if Sum==n:
34         bestx=list(x)
35         bestnum=num
36         #Print(bestx,bestnum)
37         #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
38         x,Sum,Add,num=Back(x,Sum,Add,num)
39         x,Sum,Add,num=Back(x,Sum,Add,num)
40         x,Sum,Add,num=Back(x,Sum,Add,num)
41         i=Add[num+1]-1
42         Add[num+1]=i
43     else:
44         #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
45         if num+1>bestnum:
46             x,Sum,Add,num=Back(x,Sum,Add,num)
47             x,Sum,Add,num=Back(x,Sum,Add,num)
48             i=Add[num+1]-1
49             Add[num+1]=i#更新加数的下标
50     while i<1:#第num+1个元素的所有可加项都尝试过
51         x,Sum,Add,num=Back(x,Sum,Add,num)#回退到上一个元素
52         i=Add[num+1]-1
53         Add[num+1]=i
54     if num<2:#所有元素的所有可加项都尝试过
55         return bestx,bestnum
56
57
58 # 输出结果用于验证
59 bestx, bestnum=ADD(98)
60 print("最短链长:", bestnum-1)
61 print("链中的元素:")
62 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

✓ 您已通过本题!

查看历史提交

评测编号 ↓	提交时间	提交状态	代码语言	最大运行时间
22375	2022-05-19 00:44:19	Accepted	Python	4164ms

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。