实验报告

【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】

1. 语言：C
2. 平台：clion 2021.2 版本

【实验内容】

一、ECC - 四则运算

1. **算法流程**

   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：

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

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

      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。

   2. ECC乘法除法

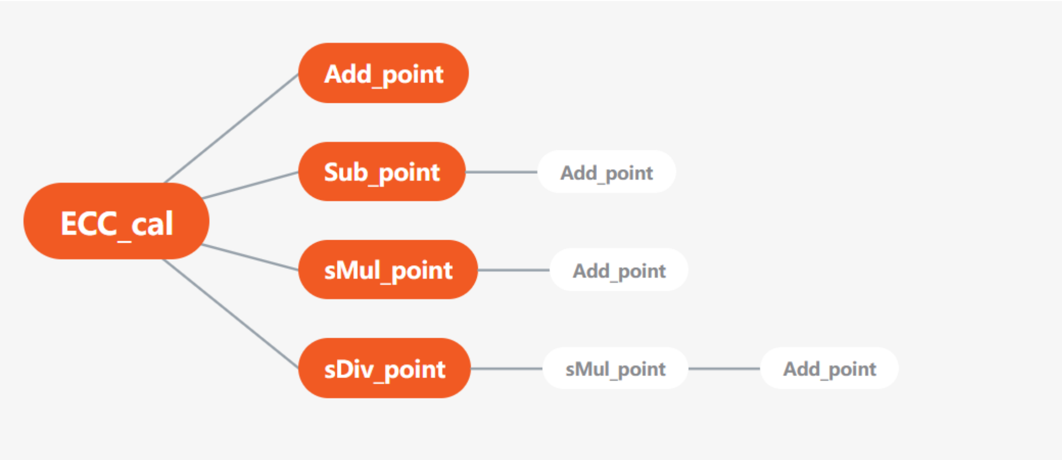
      即整数k与点坐标的乘法，意为倍点。这里需要解决的是快速求倍点的问题。继续沿用第一次实验中的快速模
      幂算法，将整数k化为二进制，将模乘运算变为点加即可，伪代码如下：

      Algorithm 1 ECC的数乘运算

      ```
      Input: point p, mpz_t k, ECC_curve C
      Output: point ans
      if k%2 == 1 then
          ans = p
      end if
      k = k/2
      for k != 0 do
          p = p + p
          if k% 2== 1 then
              ans+ = p
          end if
          k = k/2
      end for
      ```

      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。

- ### 函数调用图：



2. ## 测试样例及结果截图：

本地测试样例的运行结果如下所示：

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_cal.exe"
115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981827955639221458448578012075254857346196103069175443
64901889550129866513443884082574452575157116031103742365434905633820925813192
 84553412528427919723206133858954594911213526647800598970633596412071681640913
64901889550129866513443884082574452575157116031103742365434905633820925813192
 84553412528427919723206133858954594911213526647800598970633596412071681640913
10909953364083364362993238420745593771736340354988558975872500853194729556
91829719240076595600910287219737299259627413891073174690491219092963035830325 31474822276849859104123114646070976974921401394140157637420547181522913249875
0 0
57373321148051506091723093715148369162664271829435462559258806045759894672711 47810523934102992307219225430955925621395444436399395870291687870429993066087
65262872771489155421945763814501787279403072416537416049831455048807745490812 85255444392472600584215820646188443364214632091202947623642966545271132050758
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_cal.exe"
115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981827955639221458448578012075254857346196103069175443
30475480087869614443470089127706983386675937706263121223619594839405535269140
 90023601239643517158939790510908951707001876773487950710252404854609212360654
14488774906741259066195719087442340025866377506648801280501236926101212902354
 51288710117138148532316896987145857741690421850918580077708238138429775187679
617136796639221155023273542791931252086147246828731258960128457053519788
76418670396892355653243957783665229567941867955910264199035186565505072408164 74013006155000694365338089499021337468330255049509582708193288388722549065 72
78895673407062940374715955825638547545005250291461468259382893293918199494264 22043159006412747240268106565053053926535081007178340513578111610711191864756
8782182299491035655407366522213245608339654612217932552302421142270846681994 78144468428648451720499658567174158897803403506995461546832258623034756259383
13358924127889012189855145261361973853304174397400758918867054805888167102912 15380847576836099072652300271027820007537913344843337231645846731885387715563
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_cal.exe"
115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981827955639221458448578012075254857346196103069175443
41480030566370175534726755026526703642210603688461015843588743859313531189573
 40001835605017208139957636277957072656151173077904702162748249716709420042989
48676605738604584945904802571952219154853024319155192479396067314569782342962
 26014847953100858545226555385373319892955338660822977861066649406484312182114
1651695124807582862021267793079133691171212477804156362891432494863976849
15139384982191900487591983788353708440908316366057517132419670935744279947431121938196345632583614385494949414167041864937689326132933650053264392471602
11298042905298194579280680536175001764983853315928768898923405802682170476432119896334991812450652080092926702658620188688821228463595414901333673743909614
12637770518718272361069030959675909011950070361104001676990341902418348821166 40916222875094440749705560971391188636355520322606034728049992680090883970559
72519289070001971517117073049519824288518639052252472087812461978663312943158 97782122584291753595891393418750021218406145461157584281766145451975450555990
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_cal.exe"
115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981827955639221458448578012075254857346196103069175443
33552156979087533363935906384612651321612502271075937256856306466962780538197
 27103113344765526834401013441646545772836899432328515837453878169124736604412
89819812547535870130170737631328978849637371084031844919012473274996498495969
 113625264591997387731342929687010016579774032982218189725700208934506087897272
740099031770021203953025989803381877154685456201384716786430294521974383
46416434919624558033056716946399905725184054194477621641981818870586970614985 870905641694899167862170143314564693880074
30460863965606250437495188833946443
104673815949190848064823792929504530617300375265159409007922482354341802926247 49720414426250787946558522178489712574956
910909864851167733778132342334607845
12155434101822315229785474237673275927256428374157073820978014707268450290347 592607920940730025477492132760973509069057
09657685638377740494081362239475981
49594380326687913930629211348076681153717837731434921282465624553827847405191 102275878924654171059450634351933573693661
315401180203533760541918655258792286
Process finished with exit code 0
```

二、ECC - 公钥加解密

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：

- **算法流程图：**

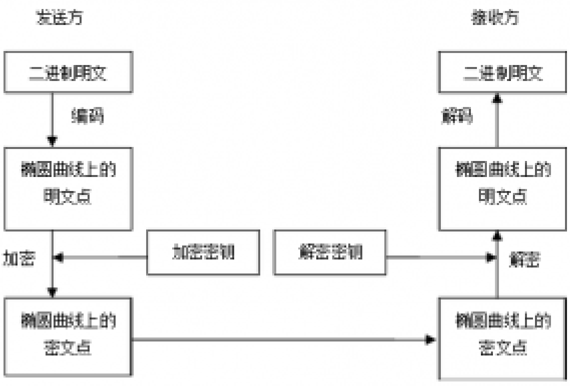


图 1　椭圆曲线密码算法的加解密流程图

- **伪代码如下：**

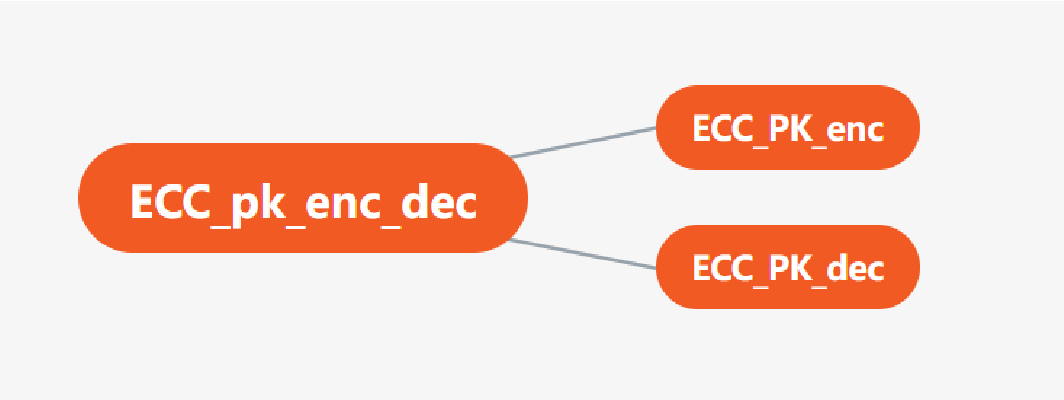
算法 1 ECC加解密

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

输出: $(C1, C2)/Pm$

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

- **函数调用图如下:**



2. **测试样例及结果截图:**

本地测试样例的运行结果如下所示:

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_pk_enc_dec.exe"
115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981827955639221458448578012075254857346196103069175443
22963146547237050559479531362550074578802567295341616970375194840604139615431
 85132369209828568825618990617112496413088388631904505083283536607588877201568
1
62220385967324995556943859594383843316990201421105076884209933343560809233015
 9063699494246492215875763452283052636231938003241827298178711426179690419737
61439397294905372254772835168938400931071745967362193779666947296905580
88698242911255079819830709760181888251691466959465004540024108127387463566707
 46820188532500665427514921969913203667116864579000112361892669654799897418874
89583431844721923129778721014009509006972090847660911645111898104009749186163 269307114890302101141102411704530064878973
95422517206355196890099778264510599
103440095822431431181554656770186818550669543322647226082461390490749218093950 20457019858049175615069165493796980004756
58851172605201819677190498475595 7123
Process finished with exit code 0
```

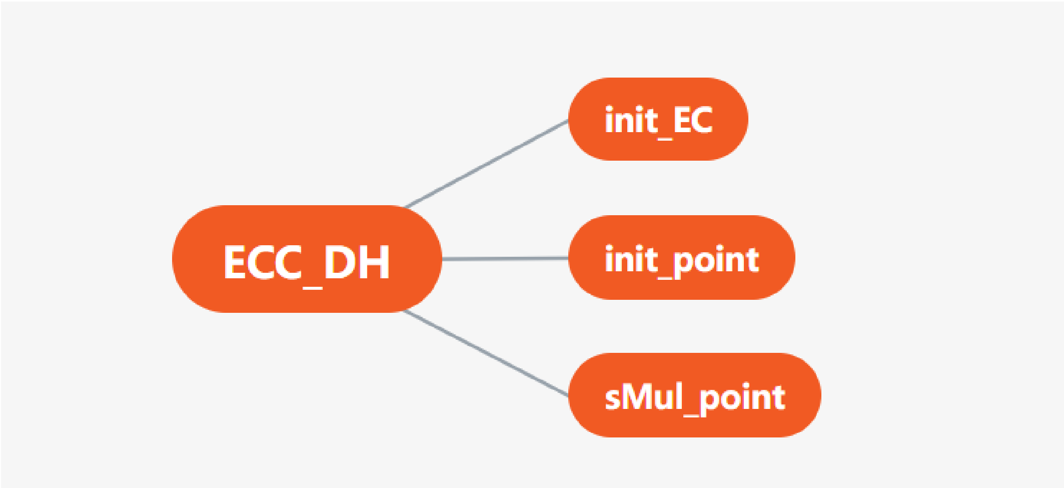
```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_pk_enc_dec.exe"
115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981827955639221458448578012075254857346196103069175443
22963146547237050559479531362550074578802567295341616970375194840604139615431
 85132369209828568825618990617112496413088388631904505083283536607588877201568
0
6248961784231464103192743400187174760735397105250758735810263385327672540480
 91738081692867197137958635646545231615668996567283180053935793748253818895161
87426908955167008979127300297989284851464893201680163779675531372419523846423
 18112318565015281508380063949435814123120972625131223552874941802645266398570
62850942488285990642567364045064888609861641279362042129880709558414313
93511574072901433740748722328989118840082983322674755679547376448446874112389 855600490203970259106461156246452983425447
12923074463923232161292330016598209
Process finished with exit code 0
```

三、ECC - DH密钥交换协议

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$K = X_a * X_b * G$。

- **函数调用图：**



2. 测试样例及结果截图：

本地测试样例的运行结果如下所示：

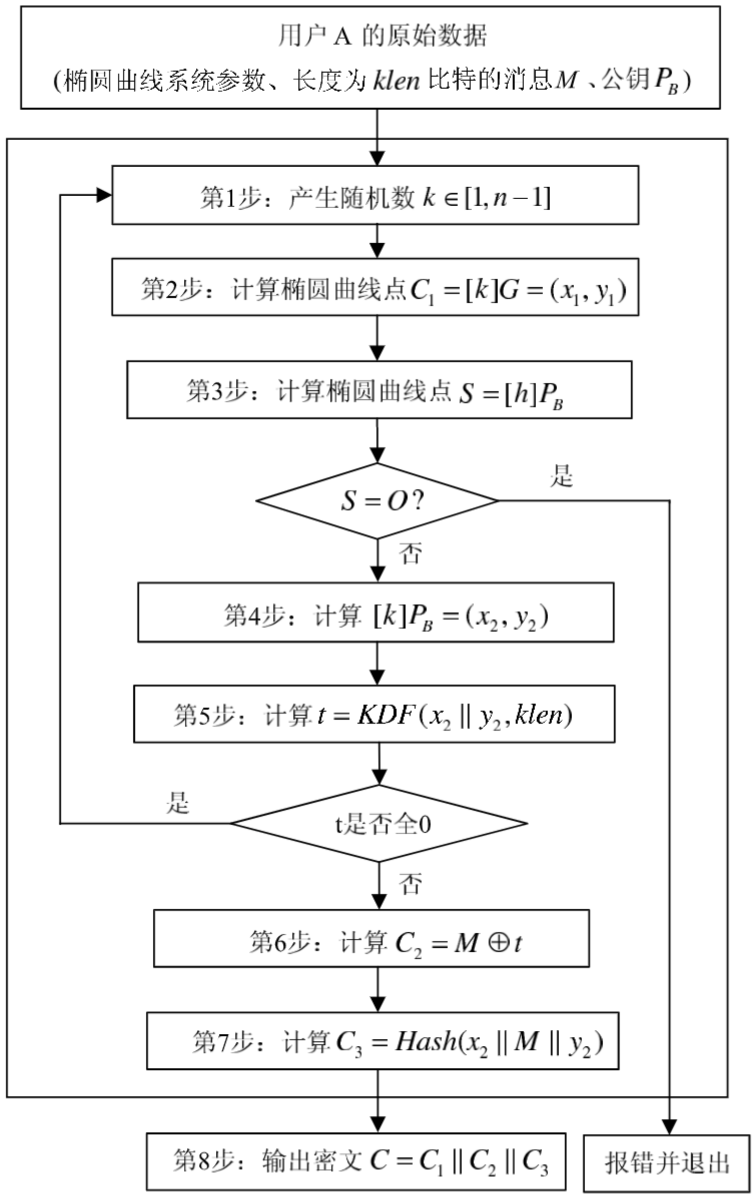
```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_DH.exe"
115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981827955639221458448578012075254857346196103069175443
22963146547237050559479531362550074578802567295341616970375194840604139615431
 85132369209828568825618990617112496413088388631904505083283536607588877201568
81198104618913722578604425355370616100415192131930555845711809030425859347
109988095266094733830946895666108835209722525233241660151830323414352837844396
 29610320893061341239531393642156416470641927512692726941625588654346048337 29
17563276920628041528213370371379123587681909252989227496921943682543883264878 162907300695687139410749046539969910089816
85386084447436564101776231176534603
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_DH.exe"
115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981827955639221458448578012075254857346196103069175443
22963146547237050559479531362550074578802567295341616970375194840604139615431
 85132369209828568825618990617112496413088388631904505083283536607588877201568
20165770951009370481409880442007347350569267408126481498660544804169638482 9
32440691193999261301314480034434061837285304746039122516928973565870356825367
 44304850600701313197998383399207915664655330089170910907161718276282552540944
12512346806273286597779399744481781487246936530763666061969314356007739153319 96522631307920229937302226944281235271224 8
04951260748197377933029315574510741
Process finished with exit code 0
```

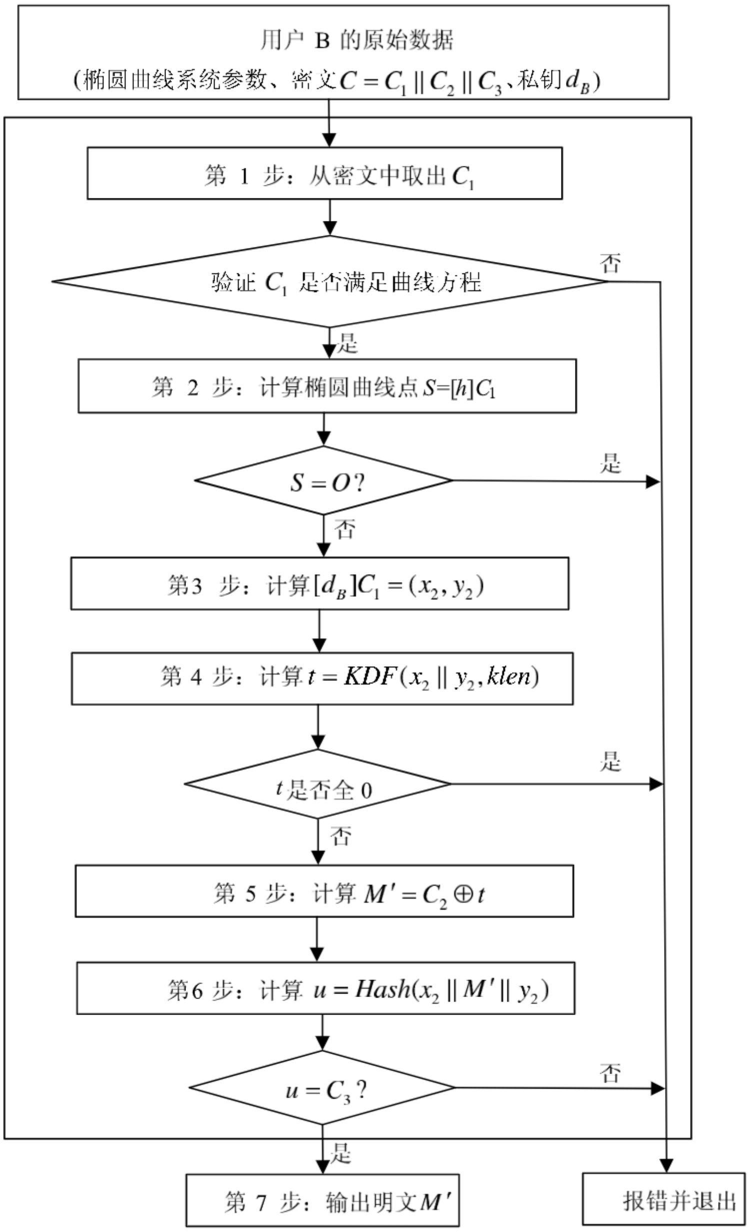
四、SM2 - 公钥加密

1. 算法流程

- **原理：**



用户A的原始数据
(椭圆曲线系统参数、长度为 $klen$ 比特的消息 M 、公钥 P_B)
第1步：产生随机数 $k \in [1, n-1]$
第2步：计算椭圆曲线点 $C_1 = [k]G = (x_1, y_1)$
第3步：计算椭圆曲线点 $S = [h]P_B$
$S = O$?
是
否
第4步：计算 $[k]P_B = (x_2, y_2)$
第5步：计算 $t = KDF(x_2 \| y_2, klen)$
t是否全0
是
否
第6步：计算 $C_2 = M \oplus t$
第7步：计算 $C_3 = Hash(x_2 \| M \| y_2)$
第8步：输出密文 $C = C_1 \| C_2 \| C_3$
报错并退出



- **伪代码：**

1. **伪代码：**

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

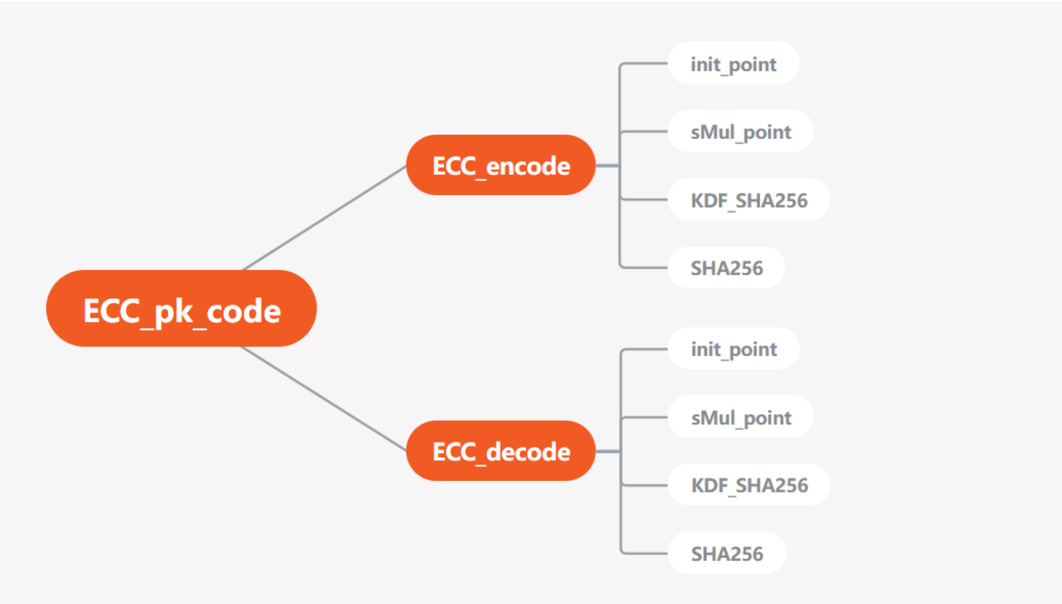
算法 3 KDF

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

- **函数调用图：**



2. 测试样例及结果截图：

本地测试样例的运行结果如下所示：

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_pk_code.exe"
60275702009245096385686171515219896416297121499402250955537857683885541941187
54492052985589574080443685629857027481671841726313362585597978545915325572248
45183185393608134601425506985501881231876135519103376096391853873370470098074
29905514254078361236418469080477708234343499662916671209092838329800180225085
 29405937379755419157903904478921572542806770830401260612308519640632340013l4
256
1
0x656e6372797074696f6e207374616e64617264
30466142855137288468788190552058120832437161821909553502398316083968243039754
 53312363470992020232197984648603141288071418796825192480967103513769615518274
34550576952843389977837539438321907097625575044301827052096699664811526290255
0x04245c26fb68b1ddddb12c4b6bf9f2b6d5fe60a383b0d18d1c4144abf17f6252e776cb9264c2a7e88e52b19903fdc47378f605e36811f5c07423a2
4b84400f01b8229e6c9aee2bb92cad649fe2c035689785da33be89139d07853100efa763f60cbe30099ea3df7f8f364f9d10a5e988e3c5aafc
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_pk_code.exe"
4651790315172547324421427488787163462617155070424625206559
4598862935839736669809089888524020718089890230495750986117
5965952540593547264152942164910691212883782209899173687 69
1834968487600647824514410140269064362459199060214757221952 6697260345500518269177832053784411778651972200118689173 0
192
0
0x0423fc680b124294dfdf34dbe76e0c38d883de4d41fa0d4cf570cf14f20daf0c4d777f738d16b16824d31eefb9de31ee1fbc104da135491c409810c55bd04ffb91a67b51a846266
 4fe2e1ed74514ee1f75b422d1a0ade2a367a09ab5b6d91c432ee40f5c
2170891990532084219830703229170485512351379721740652731133
0x1145141919810ab19260817cd947866efedcba

Process finished with exit code 0
```

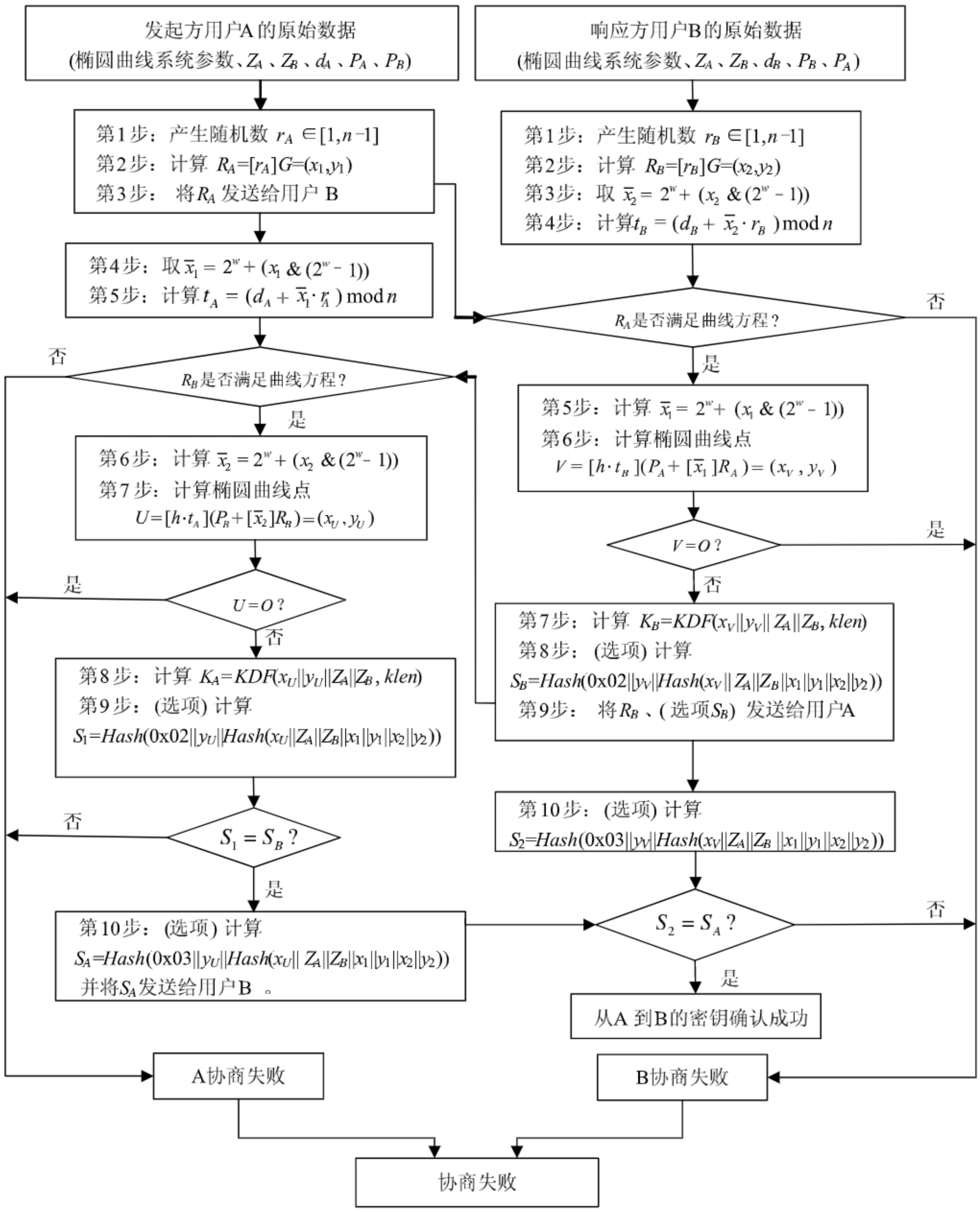
```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_pk_code.exe"
60275702009245096385686171515219896416297121499402250955537857683885541941187
54492052985589574080443685629857027481671841726313362585597978545915325572248
45183185393608134601425506985501881231876135519103376096391853873370470098074
29905514254078361236418469080477708234343499662916671209092838329800180225085
 2940593737975541915790390447892157254280677083040126061230851964063234001314
256
0
0x04245c26fb68b1ddddb12c4b6bf9f2b6d5fe60a383b0d18d1c4144abf17f6252e776cb9264c2a7e88e52b19903fdc47378f605e36811f5c07423a24b84400f01b856b51bf18edac
 7f4506a3eed6d13809d1a74ed74a0e5ae6841875ee6e77615c7d790be829a6ee44e546932e66a7970bd304540
10081045928272161671685667373292278982781750263333427527755954552719521025440
0x1145141919810ab19260817cd947866efedcba

Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_pk_code.exe"
4651790315172547324421427488787163462617155070424625206559
4598862935839736669809089888524020718089890230495750986117
596595254059354726415294216491069121288378220989917368769
1834968487600647824514410140269064362459199060214757221952 66972603455005182691778320537844117786519722001186891730
192
1
0x656e6372797074696f6e207374616e64617264
2989962154103254810804845704859775855103279169487815094090 4271922179579769876189437850048448302473410107141046597906
1380700738017179424849792545563870218395108139972515540149
0x0423fc680b124294dfdf34dbe76e0c38d883de4d41fa0d4cf570cf14f20daf0c4d777f738d16b16824d31eefb9de31ee1fc83b3aca55b86298651e
64547d69139b39d58f03a0017c01434a4d803ca01fd5736821047292b1615bc0f71e0e6270edcee7b3
Process finished with exit code 0
```

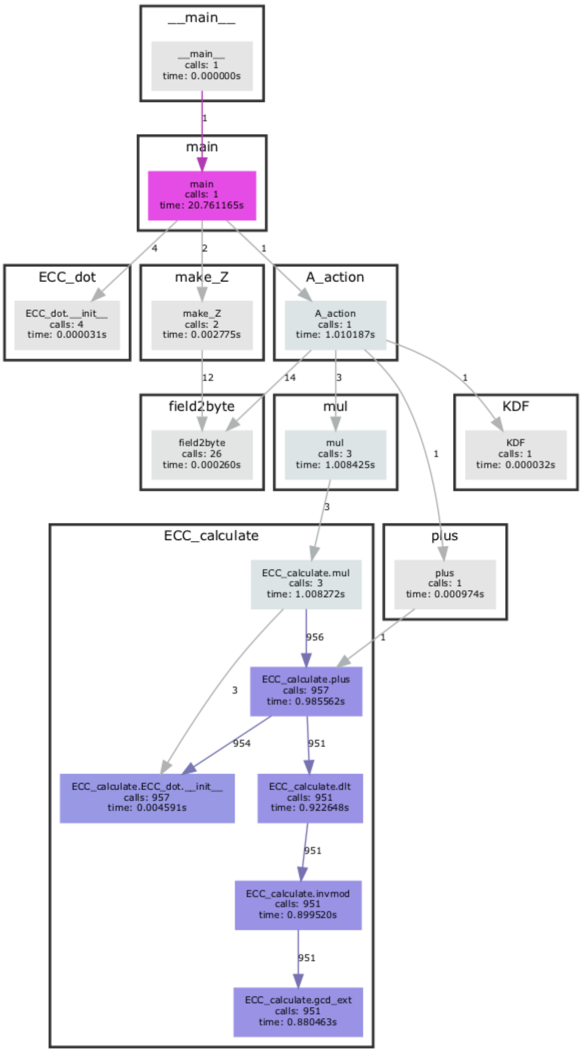
五、【选做】SM2 - 密钥交换协议

1. **算法流程**

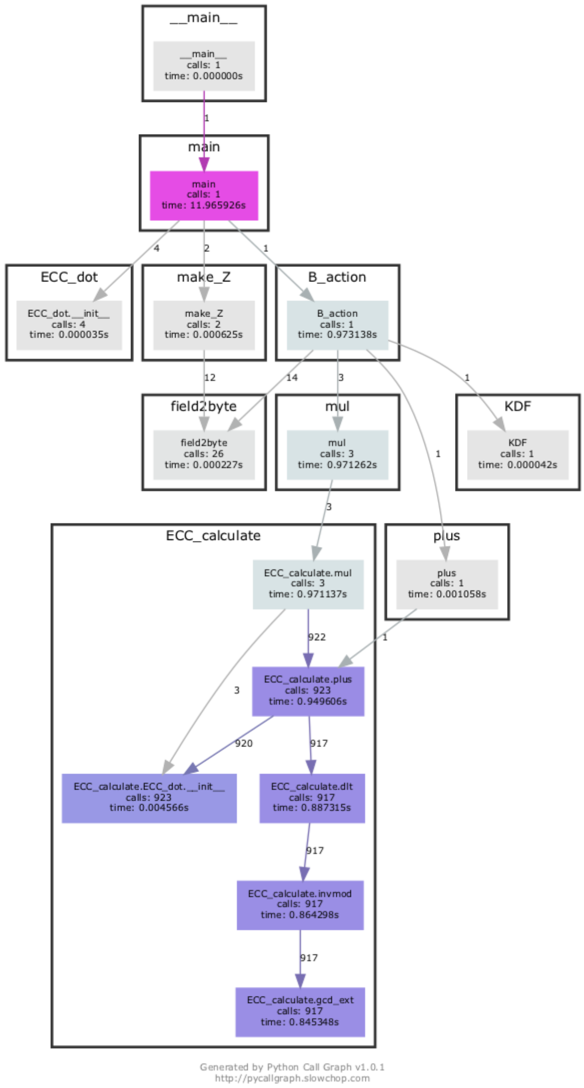


发起方用户A的原始数据
(椭圆曲线系统参数、Z_A、Z_B、d_A、P_A、P_B)
响应方用户B的原始数据
(椭圆曲线系统参数、Z_A、Z_B、d_B、P_B、P_A)
第1步：产生随机数 $r_A \in [1,n-1]$
第2步：计算 $R_A=[r_A]G=(x_1,y_1)$
第3步：将R_A发送给用户 B
第4步：取 $\bar{x}_1 = 2^w + (x_1 \& (2^w - 1))$
第5步：计算 $t_A = (d_A + \bar{x}_1 \cdot r_A) \bmod n$
第1步：产生随机数 $r_B \in [1,n-1]$
第2步：计算 $R_B=[r_B]G=(x_2,y_2)$
第3步：取 $\bar{x}_2 = 2^w + (x_2 \& (2^w - 1))$
第4步：计算$t_B = (d_B + \bar{x}_2 \cdot r_B) \bmod n$
R_A是否满足曲线方程？
否
是
第5步：计算 $\bar{x}_1 = 2^w + (x_1 \& (2^w - 1))$
第6步：计算椭圆曲线点
$V = [h \cdot t_B](P_A + [\bar{x}_1]R_A) = (x_V, y_V)$
$V=O$？
是
否
第7步：计算 $K_B=KDF(x_V||y_V||Z_A||Z_B, klen)$
第8步：(选项) 计算
$S_B=Hash(0x02||y_V||Hash(x_V||Z_A||Z_B||x_1||y_1||x_2||y_2))$
第9步：将R_B、(选项S_B) 发送给用户A
第10步：(选项) 计算
$S_2=Hash(0x03||y_V||Hash(x_V||Z_A||Z_B||x_1||y_1||x_2||y_2))$
$S_2 = S_A$？
否
是
从A 到B的密钥确认成功
B协商失败
否
R_B是否满足曲线方程？
是
第6步：计算 $\bar{x}_2 = 2^w + (x_2 \& (2^w - 1))$
第7步：计算椭圆曲线点
$U=[h \cdot t_A](P_B+[\bar{x}_2]R_B)=(x_U, y_U)$
是
$U=O$？
否
第8步：计算 $K_A=KDF(x_U||y_U||Z_A||Z_B, klen)$
第9步：(选项) 计算
$S_1=Hash(0x02||y_U||Hash(x_U||Z_A||Z_B||x_1||y_1||x_2||y_2))$
否
$S_1 = S_B$？
是
第10步：(选项) 计算
$S_A=Hash(0x03||y_U||Hash(x_U||Z_A||Z_B||x_1||y_1||x_2||y_2))$
并将S_A发送给用户B 。
A协商失败
协商失败

- **函数调用图：**

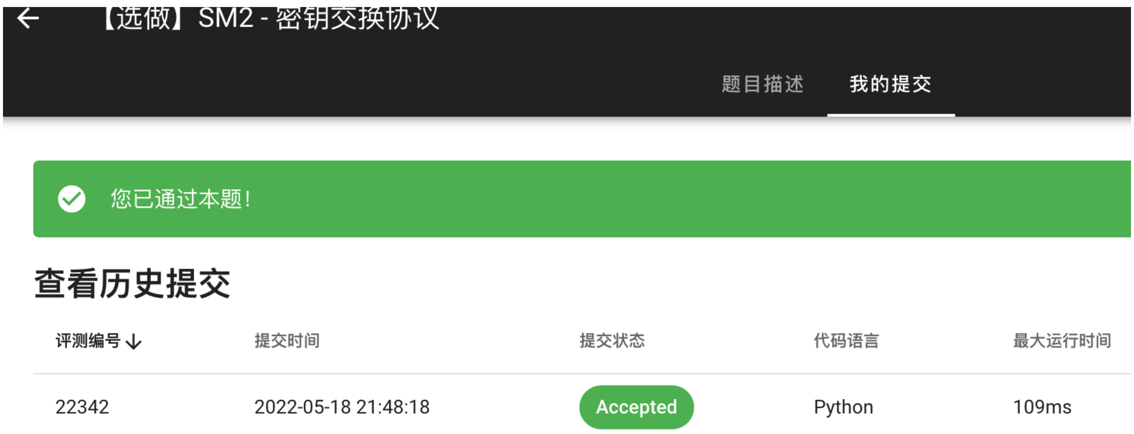


Generated by Python Call Graph v1.0.1
http://pycallgraph.slowchop.com



2. **测试样例及结果截图：**

本地测试样例的运行结果如下所示：



六、【选做】基于ECC的公钥加密快速实现

1. 算法流程

在本题中我尝试了蒙哥马利算法。

- **原理：**

  1. **蒙哥马利算法：**

蒙哥马利算法（MontgomeryAlgorithm）蒙哥马利约简、模乘、模幂

Montgomery Algorithm(蒙哥马利算法)

蒙哥马利算法分为3种，蒙哥马利模乘，蒙哥马利约简，蒙哥马利模幂

1、从蒙哥马利模乘说起

模乘是为了计算$ab \pmod N$。普通算法中，在计算模N时，利用的是带余除法，除法运算需要太多次乘法，计算复杂度较高，蒙哥马利算法的思想就是利用进制表示简化除法运算，转化成位运算。

- 蒙哥马利形式：为了计算$ab \pmod N$，找一个R，然后使得$a' \equiv aR \pmod N, b' \equiv bR \pmod N$，这就是蒙哥马利形式。
- 这个R不是随便一个数，需要满足两个条件：1:$R = 2^k > N$,其中k是满足条件的最小的正数，这个取法能够保证除以R就相当于右移k位，避免除法运算；2:$gcd(R, N) = 1$，这使得能够求出下面的m。
- 蒙哥马利形式有什么用？往下看

前面只是铺垫，下面才真正开始。为了计算一开始的$ab \pmod N$，需要用到上面的蒙哥马利形式。令$X = a'b'$，我们可以设计一个函数来计算$XR^{-1} \pmod N$，简单计算发现这个函数的计算结果为$X_1 = XR^{-1} = a'b'R^{-1} = abR \pmod N$，这样在调用一遍函数计算$X_1R^{-1} \pmod N$就得到我们最终需要的结果$ab \pmod N$了。我们称这个算法叫蒙哥马利约简算法。所以说，蒙哥马利约简的产生是为了蒙哥马利模乘计算服务的。

2、蒙哥马利约简

根据上述分析可知蒙哥马利算法的核心在于蒙哥马利约简。而且前面提到，蒙哥马利算法的主要思想是把取模运算变得简单，到底是怎么做到的呢？

主要思想：蒙哥马利约简是计算$XR^{-1} \pmod N$，这相当于$\frac{X}{R} \pmod N$，如何才能避免除法？从前面的R的定义中我们知道，$R = 2^k$，所以$\frac{X}{R} = X >> k$ （X右移k位）。但是新的问题出现了，右移k位可能会抹掉X的低位中的一些1，如$7 \div 4 = 0b111 >> 2 = 0b1 = 1$，这个不是精确计算，而是向下取整的除法。当且仅当X是R的整数倍时，$X \div R$严格等于$X >> k$。所以我们实际上是找一个m，使得$X + mN$是R的倍数，这样计算$\frac{X+mN}{R} \pmod N$就可以了。

m如何找

根据R的定义，$gcd(R, N) = 1$，根据扩展的欧几里得算法，有$RR' - NN' = 1$并且有$0 < N' < R, 0 < R' < N < R$。

$$
\begin{gathered}
X + mN \equiv 0 \pmod R \\
XN' + mNN' = 0 \pmod R \\
XN' + m(RR' - 1) \equiv 0 \pmod R \\
XN' \equiv m \pmod R
\end{gathered}
$$

这样就求出了m。

约简算法总流程

1. 计算$N' = N^{-1} \pmod{R}$，计算$m = XN' \pmod{R}$；
2. 计算$y = \frac{X+mN}{R}$:将$X + mN$右移k位；
3. 若$y > N$，则$y = y - N$；

   这时的y满足：$0 < y < 2N$。因为

$$X < N^2, m < R, N < R$$

   所以

$$\frac{X+mN}{R} < \frac{N^2 + R \cdot N}{R} < \frac{RN + R \cdot N}{R} = 2N$$

4. 返回y。

3、蒙哥马利模乘流程

Montgomery Multiply(a, b, N):

1. 计算$a' = aR \pmod{N}, b' \equiv bR \pmod{N}$，$X = a'b'$；
2. 调用蒙哥马利约简算法：X_1=Montgomery reduction$(X, R, N) = X/R = abR \pmod{N}$；
3. 再调用蒙哥马利约简算法：y=Montgomery reduction$(X_1, R, N) = X_1/R = ab \pmod{N}$；
4. 返回y。

4、复杂度分析

蒙哥马利算法是为了简化模N的复杂度，当N是比较大的数时，模N需要多次的加、减、乘运算。从上述过程看来，蒙哥马利约简的复杂度确实降低了，因为只有模R的移位运算。但是看蒙哥马利模乘的流程，在第一步中进行蒙哥马利表示时就计算了两次模N，（$a' = aR \pmod{N}, b' \equiv bR \pmod{N}$，$X = a'b'$），总的来看复杂度也没有降低啊？实际上，第一步可以看成是蒙哥马利的预先计算，在硬件实现中，先把预先计算的算好，在后面运行就会快很多。尤其是当出现大量的模乘运算时，可以通过并行运算进行预计算，这会大量节省运行时间。

5、蒙哥马利模幂

当进行模幂运算时，也可以利用蒙哥马利算法。如计算$a^e \pmod{N}$。根据，模幂运算中就有很多步的模乘运算，这恰恰可以发挥蒙哥马利模乘的优势。具体的，在重复平方法的每一次模乘中都利用蒙哥马利模乘进行计算，把需要的参数提前计算好就行。

算法的本质在于预处理计算，进而简化过程，加快运算速度。

2. **分析：**

   在模拟实现后发现并没有速度上的明显提升，原因就在于gmp大数库中的模乘运算已经采用了这样的优化方式。

2. **测试样例及结果截图：**

本地测试样例的运行结果如下所示：

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_Fast_enc_dec.exe"
115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981827955639221458448578012075254857346196103069175443
22963146547237050559479531362550074578802567295341616970375194840604139615431
 85132369209828568825618990617112496413088388631904505083283536607588877201568
1
60652694590882909356250788599098839028553140764880503559070946642968945892102
 97248566623691456333947388494196120167795904532823719329772252328742149457874
7562804387615800700837018529635091671231728396437156677937438803617099
96513597923123126420711825319105397644998306957439694978932196439833315475596
 10720292836202791501880298868992459922703519492548574460874619311130191933901
99
28361968634019156025990227532137859338916660393511338846630983493269574907 24 274266622296338992470167655289918023337000 8
00381053695618470654033158073313 46
84654655179098796912662543007901138302884985227327354714303500052960379059857 90687851011410250211197538806540650787468 6
46750805872850704138724948523318145
Process finished with exit code 0
```

```
"E:\E_drive\clion\Project List\cryptography\Crypto_Experiment\cmake-build-debug\Exp8--ECC_Fast_enc_dec.exe"
115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981827955639221458448578012075254857346196103069175443
22963146547237050559479531362550074578802567295341616970375194840604139615431
 85132369209828568825618990617112496413088388631904505083283536607588877201568
0
2836196863401915602599022753213785933891666039351133884663098349326957490724
 27426662229633899247016765528991802333700080038105369561847065403315807331346
84654655179098796912662543007901138302884985227327354714303500052960379059857
 90687851011410250211197538806540650787468646750805872850704138724948523318145
75679514760352709663961092596921556647349330721274547052923221116904421
99
60652694590882909356250788599098839028553140764880503559070946642968945892102 972485666236914563339473884941961201677959
04532823719329772252328742149457874
Process finished with exit code 0
```

七：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。