

实验报告（椭圆曲线）

【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】

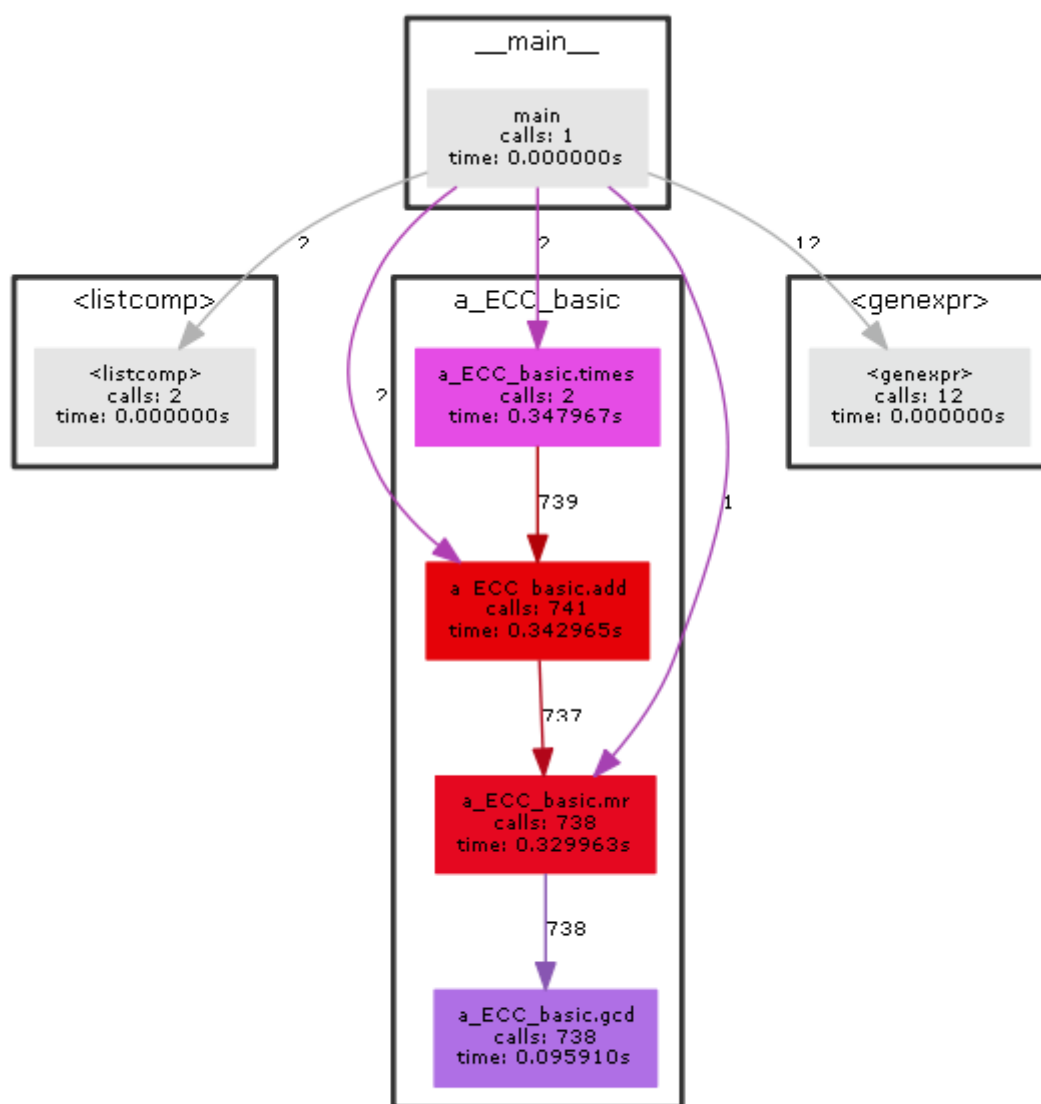
Python 3.9.7 64-bit

【实验内容】

一、ECC - 四则运算

1、算法流程

函数调用关系图：



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

函数名	函数功能
gcd()	最大公因数
mr()	有限域逆元
add()	ECC 加法
times()	ECC 乘法

2、测试样例及结果截图

```

115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981027955639221458448578012075254857346196103069175443
6490188958012896651344308408257445257157110831103742365434005633020925812192 04553412528427919723206133850964594911213526647800598970633596412071601640913
6490188955012896651344308408257445257157110831103742365434005633020925812192 04553412528427919723206133850964594911213526647800598970633596412071601640913
1090995364083643620932384207455937717363403549885589758725008319472956
91829719240076595600910287219737299259627413891073174690491219092963035830325 31474822276849859104123114646070976974921401394140157637420547181522913249875
0 0
57373321148051506091723093715148369162664271829435462559258806045759894672711 4781052393410299230721922543095592562139544443639939587029168780429993066087
65262872771489155421945763814501787279403072416537416049831455048807745490812 852554443924726005842158206461884433641214632091202947623642966545271132050758

Process finished with exit code 0

115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981027955639221458448578012075254857346196103069175443
3047540007869614443470009127706983386675937706263121223619594839405335269140 90023601239643517158939790510908951707001876773487950710252404854609212360634
144880774906741259066195719087442340025866377506648801200501236926101212902354 51288710117138148532316896987145857741690421850918580077708238138429775787679
617136796639211350227354279193125200614724602873123896002045705319700
76418670396892355653243957783665229567941867955910264199035186565505072408164 7401300615500069436533808949902133746833025504950958270819328838872254906572
7889567340706294037471595582563854754005250291461468259382893293918199494264 22043159006412747240268106565053053926535081007178340513578111610711191864756
878218229949103565407366522213245608339654612217932552302421142270846681994 78144468428648451720499658567174158897803403506995461546832258623034756259383
13358924172889012189855145261361973853304174397400758918867054805888167102912 15380847576836099072652300271027820007537913344843373231645846731885387715563

Process finished with exit code 0

115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981027955639221458448578012075254857346196103069175443
414080305663701755347267502652678064221060360046101504350874385931351189573 40001035005017200139957636277957072656151173077904702162740249716709420042809
40676005738064584945904802571952219154853024319155192479390067314569702342962 260148479531008585452265553853733198929553306608229778610666409406484312102114
1651695124807582862021267793079133691171212477804156362891432494863976849
1513938498219190848759159837883537084409003163660575171324196709357442799473 1121938196345632583614385494949414167041864937689326132933650053264392471602
112980429052981945792806805361750017649838533159287688989234058026821704764321 19896334991812450652080092926702658620188688821228463595414901333673743909614
12637770518718272361069030959675909011950070361104001676990341902418348821166 40916222875094440749705560971391188636355520322606034728049992680090883970559
72519289070001971517117073049519824288518639052252472087812461978663312943158 97782122584291753595891393418750021218406145461157584281766145451975450555990

Process finished with exit code 0

115792089210356248756420345214020892766250353991924191454421193933289684991999
115792089210356248756420345214020892766250353991924191454421193933289684991996
18505919022281880113072981027955639221458448578012075254857346196103069175443
3355216079007533303935906380612651321612502710759373585330646062780530197 27109113240765526834401014416454577203609943230515037453870169124736004012
00010812547535570130170737631320978049637371004031844919012473274996498495969 11362526459199738773134292968701001657977403298221018972570030934506087897272
7400900317700212039530259890803381877154685456201384716786430294521974383
464164349196245580330656716946399005725184054194477621641981818870586970614985 8709056416948991678621701433145646938800743046086396506250437495188833946443
104673815949190848064823792929504530617300375265159409007922482354341802926247 4972041442625078794655852217848971257495691090986485116773377813234234607845
12155434101822315229785474237673275927256428374157073820978014707268450290347 59260792094073002547749213276097350906905709657685638377740494081362239475981
49594380326687913930629211348076681153717837731434921182465624553827847405191 102275878924654171059450634351933573693661315401180203533760541918655258792286

Process finished with exit code 0

```

3、讨论与思考

- 加法算法首先处理了特殊情况：
 - ◆ P 或 Q 为 O ，结果输出另一个点；
 - ◆ $P = -Q$ ，结果输出 O 点。
- 乘法用了快速模幂的思想，ECC 的加法相当于有限域的乘法。
- 减法即加相反点（纵坐标取反）；除法即为乘逆元。
- 为了方便做减法和除法，本题代码中函数的输入为坐标值，后面题目中将包装成点（二元列表）
- `print(" ".join(str(i) for i in List))`以空格为分隔输出列表元素

二、ECC - 公钥加密

1、算法流程

伪代码：

Algorithm 1: ECC公钥加密

Input: 椭圆曲线参数 p, a, b , 基点 P , 明文点 M , 加密随机数 r , 公钥点 Q

Output: 密文点 C_1, C_2

1. **function** "pk_enc(r, P, M, Q, a, b, p)"
 2. **set** C_1, C_2 **in** $E_p(a, b)$
 3. $C_1 \leftarrow r \otimes P$ // ECC上乘法
 4. $C_2 \leftarrow M \oplus r \otimes Q$ // ECC上加法、乘法
 5. **return** C_1, C_2
 6. **end function**
-

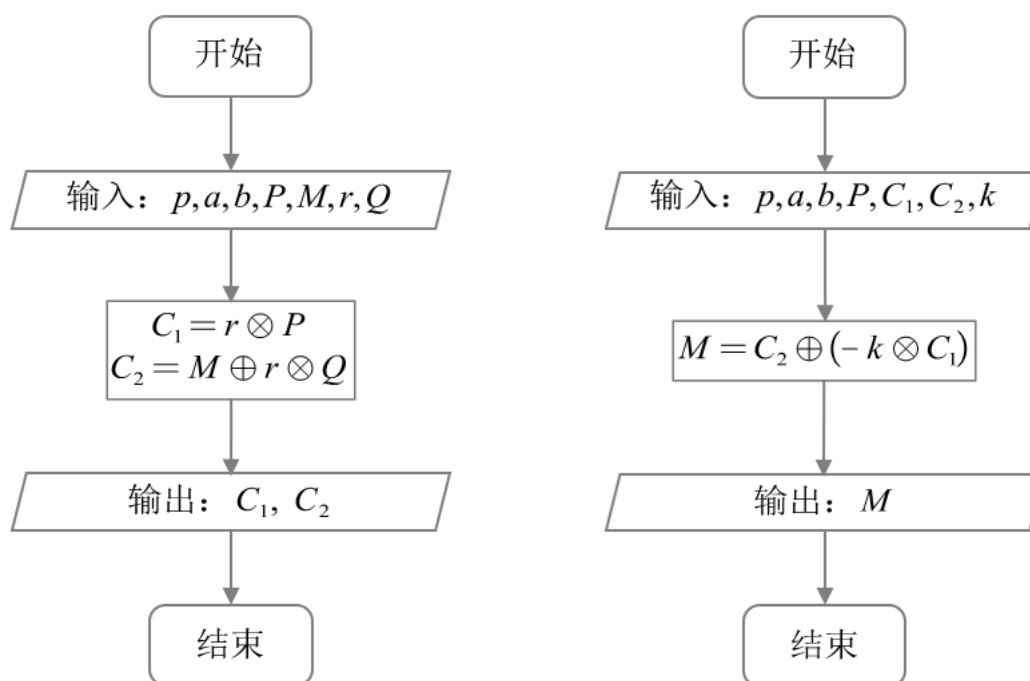
Algorithm 2: ECC公钥解密

Input: 椭圆曲线参数 p, a, b , 基点 P , 密文点 C_1, C_2 , 私钥 k

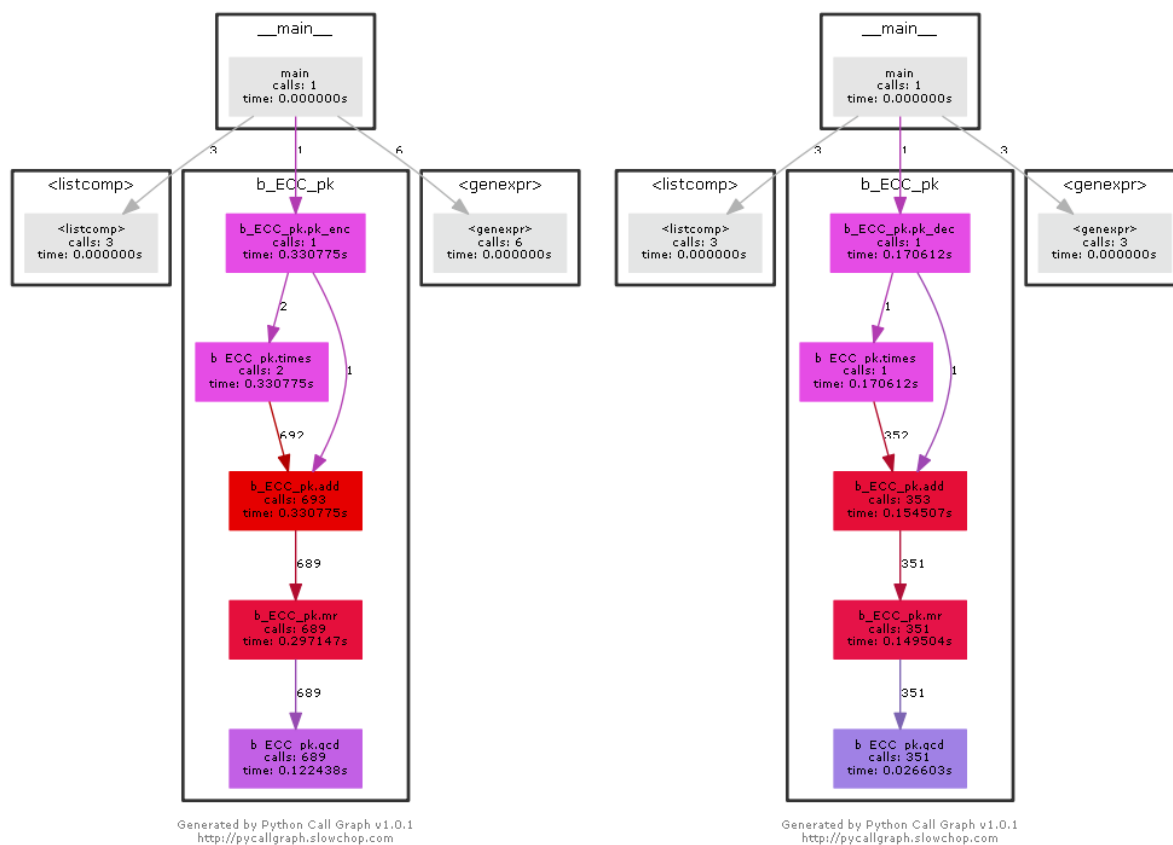
Output: 明文点 M

1. **function** "pk_dec(k, C_1, C_2, a, b, p)"
 2. **set** M **in** $E_p(a, b)$
 3. $M \leftarrow C_2 \oplus (-k \otimes C_1)$ // ECC上加法、乘法
 4. **return** M
 5. **end function**
-

流程图：



函数调用关系图（加密&解密）：



2、测试样例及结果截图

```

115792089210356248756420345214020892766250353991924191454421193933209684991996
115792089210356248756420345214020892766250353991924191454421193933209684991996
18505919022281800113072901627955639221458448578012075254857346196103069175443
22963146547237050559479531362550074578002567295341616970375194840604139615431 85132369209828560825610990617112496413080308631904505083203536607588877201568
J
62220303907324995556943859594383043316990201421103076004209933343560009233015 9063699494246492215875763452203052636231930003241027298178711426179690419737
6143919729490537225477283516893040093167174596736219377966947296905500
00690242911255079819830709740101000251691468959465004540024100127387463566707 46820188532500665427514921969913203667116884579000112361892669654798897418874
89583431844721923129778721014009509006972090847660911645111898104009749186163 26930711489030210114110241170453006487897395422517206355196890099778264510599
1034400958224313118155465677018818550669543322647226082461390490749218093950 20457019858049175615069165493796980004756588511726052018196771904984755957123
Process finished with exit code 0

115792089210356248756420345214020892766250353991924191454421193933209684991996
115792089210356248756420345214020892766250353991924191454421193933209684991996
18505919022281800113072901627955639221458448578012075254857346196103069175443
22963146547237050559479531362550074578002567295341616970375194840604139615431 85132369209828560825610990617112496413080308631904505083203536607588877201568
B
6248961704231464103192743400187174760735397105250750735010263305327672540400 91730001692067197137950635646545231615668996567283180053935793748253810095161
8742690895516700097912730019708920485146489320160016377967553137219523046423 1811291856501528150830063949435014123120972625131223552874941802645266398570
62850842480285900642567364045064088000861641279362042129880709558414313
9351157407290143374074872232898911884008298332267475567954737644846874112389 85560049020397025910646115624645298342544712923074463923232161292330016598209
Process finished with exit code 0

```

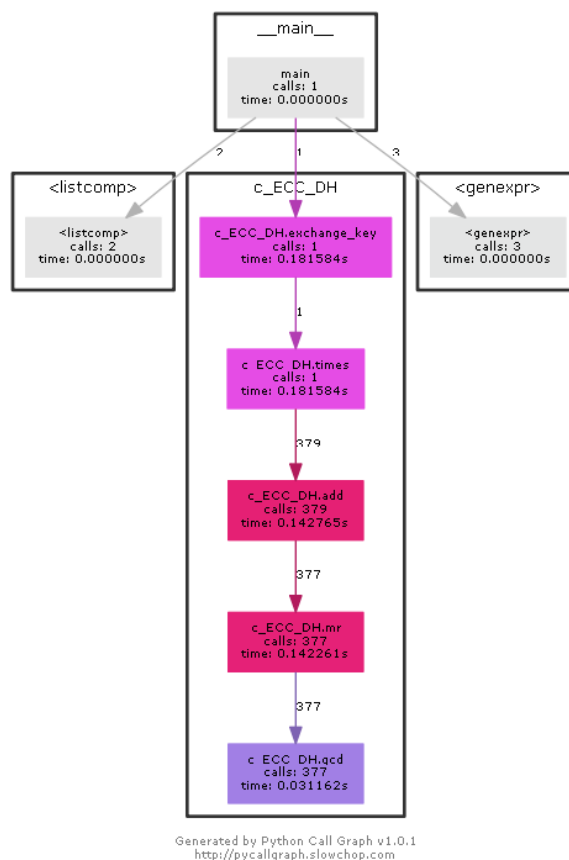
3、讨论与思考

- 直接调用 ECC 加法乘法即可。

三、ECC - DH 密钥交换协议

1、算法流程

函数调用关系图：



2、测试样例及结果截图

```
115792089210356248756420345214020892766250333991924191454421193933289684991996
115792089210356248756420345214020892766250333991924191454421193933289684991996
18505919022281880113072981827955639221458448578012075254837346196103069175443
22963146547237050559479531362550074578802567295341616970375194840604138615431 85132369209828568825618990617112496413088388631904505083283536607588877201568
81190104610913722578084428395370616100415192131930555045711089030425859347
10998809526609473383094609560610083520972252529241660151830323414352837844396 2961032009306134123953130364215041647064192751269272694162558865434604833729
17563276920628041528213370371379123587681909252989227496921943682543883264878 16290730069568713941074904653996991088981685386084447436564101776231176534603

Process finished with exit code 0

115792089210356248756420345214020892766250333991924191454421193933289684991996
115792089210356248756420345214020892766250333991924191454421193933289684991996
18505919022281880113072981827955639221458448578012075254837346196103069175443
22963146547237050559479531362550074578802567295341616970375194840604138615431 85132369209828568825618990617112496413088388631904505083283536607588877201568
281657709518093704614098804420073473505692674081264014986605448041696384829
32440091193999261301314400034434061837285384740839122516928973563270356825367 4430485060070131319799838339920791560465533008917091090716171027628252540944
125123468062732865977939974448178148724693653076366061969314356007739153319 96522631307920229937302226944281235271224804951260748197377933029315574510741

Process finished with exit code 0
```

3、讨论与思考

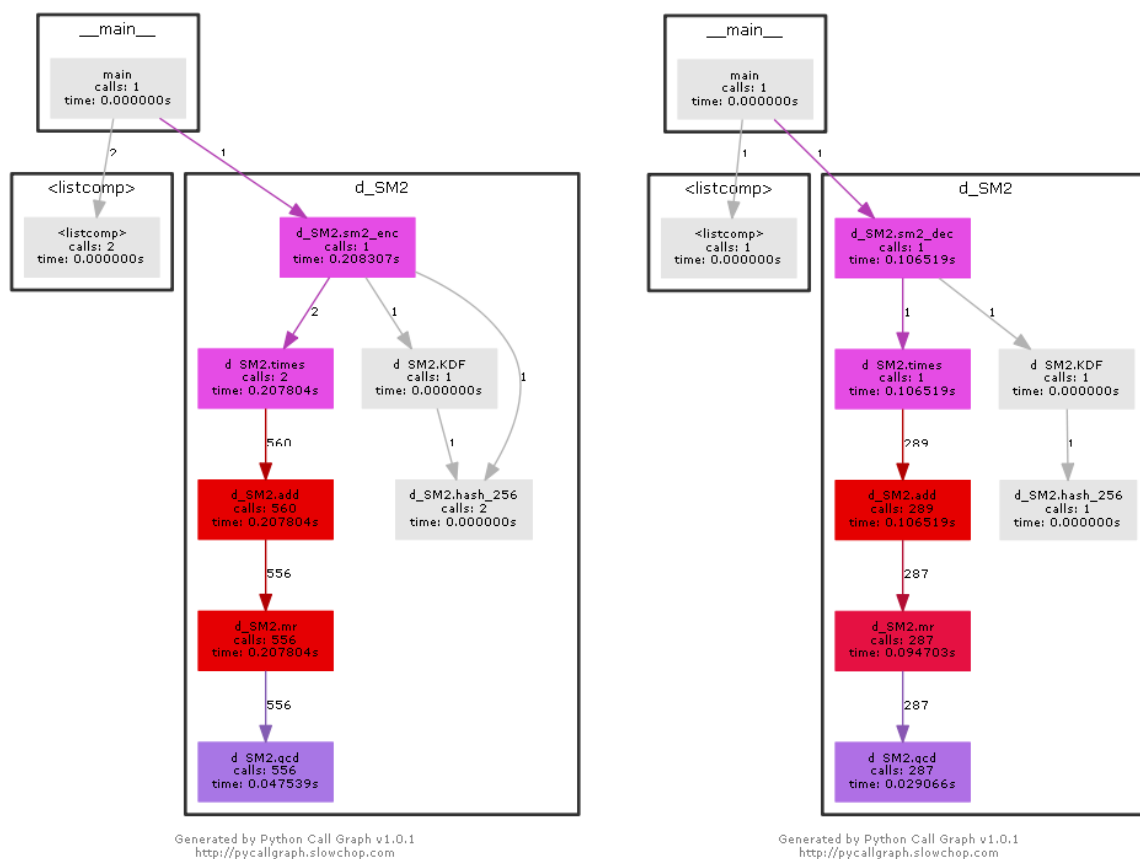
- 直接将一方的公开点乘另一方产生的随机数即可（连基点都不需要）。

四、SM2 - 公钥加密

1、算法流程

- 加密：
 - 输入：加密随机数 k ，基点 G ，公钥 P_B ，明文字符串 m ，ECC 参数 p,a,b ，安全参数 $param$ （点椭圆曲线上点坐标的比特长度）
 - $C_1 = kG$ （结果的横纵坐标填充到 $param$ 位后拼接）
 - $(x_2, y_2) = kP_B$
 - $t = KDF(x_2 || y_2, klen)$ ，（函数通过 $x_2 || y_2$ 与计数器 ct 拼接后做 256 位的哈希，输出与明文串长度相同的 t ）
 - $C_2 = m \oplus t$
 - $C_3 = Hash(x_2 || m || y_2)$ ，（16 进制字符串的拼接作为输入）
 - 输出： $C = C_1 || C_2 || C_3$
- 解密：
 - 输入私钥 d ，密文字符串 C ，ECC 参数 p,a,b ，安全参数 $param$ （点椭圆曲线上点坐标的比特长度）
 - $C_1 = C$ 前 $2 \times param$ 位（前 $param$ 位和后 $param$ 位分别作为 C_1 横纵坐标）
 - $C_3 = C$ 后 64 位， C_2 为中间位
 - $(x_2, y_2) = dC_1$
 - $t = KDF(x_2 || y_2, klen)$
 - $m = C_2 \oplus t$

函数调用关系图（加密&解密）：



2、测试样例及结果截图

```
403179031517294732442142748078710340201719307042402920039
4590062935039736669009009005240207100090023049570006117
590595294059354726415294216491069121200370220009917308769
1834960407600647824514410140269064362459190060214757221952 66972003455005182691778320537044117786519722001186001730
192
I
0x650e6372797074696f6e207374616e64617264
2009962154103254810004045704859775055103279169487015094090 4271922179579769876189437850040448302473410107141046597906
130070071001717942040792545563070218395100139972315340149
0x0423fc680b124294dfdf34dbe76e0c38d883de4d41fa0d4cf570cf14f20daf0c4d777f738d16b16824d31eeffb9de31ee1fc83b3aca55b86298
651e64547d69139b39d58f03a0017c01434a4d083ca01fd5736821047292b1615bc0f71e0e6270edcee7b3

Process finished with exit code 0

403179031517294732442142748078710340201719307042402920039
4590062935039736669009009005240207100090023049570006117
590595294059354726415294216491069121200370220009917308769
1834960407600647824514410140269064362459190060214757221952 66972003455005182691778320537044117786519722001186001730
192
0
0x0423fc680b124294dfdf34dbe76e0c38d883de4d41fa0d4cf570cf14f20daf0c4d777f738d16b16824d31eeffb9de31ee1fbcb04da135491c40
9010c55b0d4ff691a67b51a08462604fe2e1ed74514ee1f75b422d1a0ade2a367a090b5b6d91c432ee40f5c
2170091900532004219030703229170405512351379721740052731133
0x1145141919810ab19260817cd947866efedcba

Process finished with exit code 0

6087570200924500030500017151521900041029712149940220095937007003000341941307
54492052985589574000443605629857027401671041726313362585597978545915325572240
45183185393600134601425506985501001231076135519103376096391053873370470090074
2990551425407836123641046900047700234343499662916671209092036329000100225085 2940593737975541915790390447892157254200677003040126061230051964063234001314
256
0
0x650e6372797074696f6e207374616e64617264
30466142855137288460700190552050120832457161821909553502390316003960243039754 53312363470992020232197984648003141280071410796025192400967103513769615518274
34550570952043389977037539438321907097625575044301827052096699664011526290255
0x04245c26fb68b1dddbb12c4b6bf9f2b6d5fe60a383b0d18d1c4144abf17f6252e776cb9264c2a7e88e52b19903f9dc47378f605e36811f5c07423a24b8440f01b856b51bf18edac7f4506a3eed
6d13009d1a74ed74a0e5de6041875ee6e77615c7d790be82906ee44e546932e66a7970bd304540
1000104592027216167108566737329227000270175020333427927755954352719521029440
0x1145141919810ab19260817cd947866efedcba

Process finished with exit code 0
```

3、讨论与思考

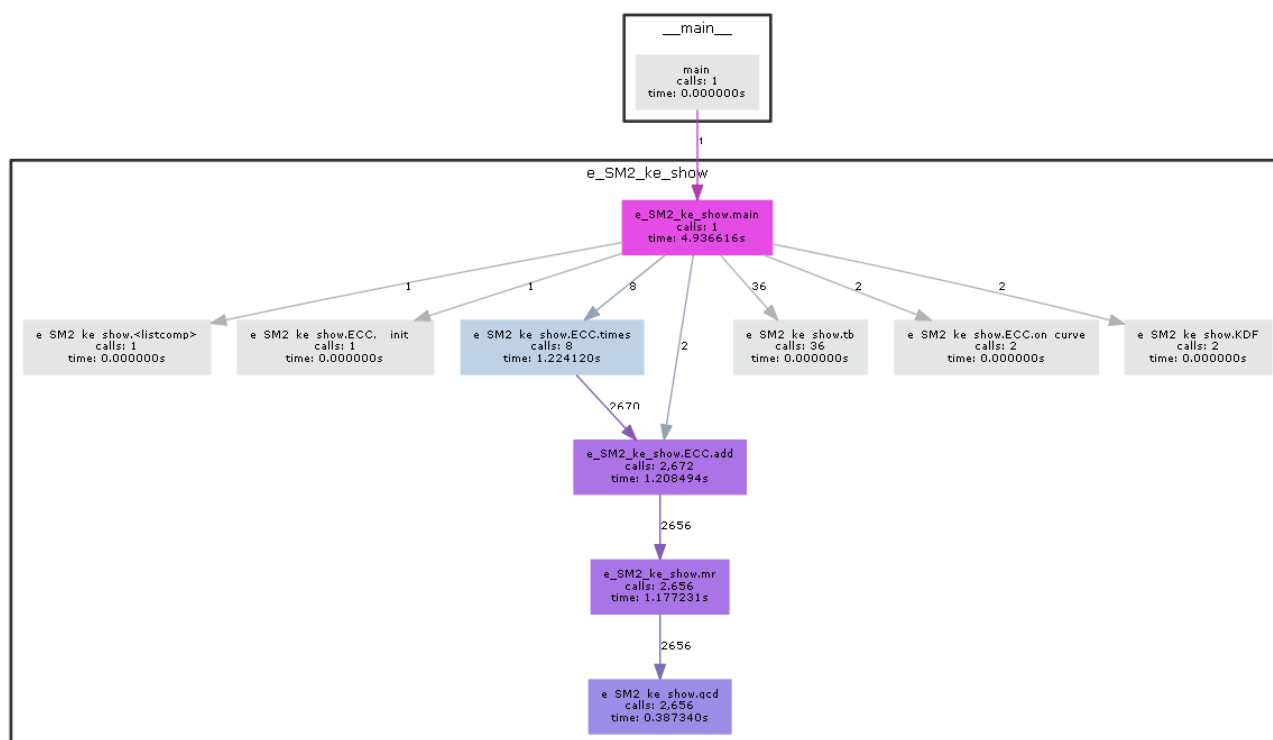
- 本题 Hash 函数用 SHA256，因此 C_3 恒为 256 位（64 位 hex）。
- 调用 Python 中 `hashlib.sha256()` 前先用 `bytes.fromhex()` 将十六进制字符串转化为字节串。
- 对于普通文本求 SHA256，可调用 `encode()` 进行编码，默认为 utf-8 格式。

五、【选做】SM2 - 密钥交换协议

1、算法流程

- 交 OJ:
 - 输入一堆参数
 - 算 w, Z_A, Z_B
 - 用户为 A、B 的情况下分别计算 S_1, S_A, S_B, S_2
- 本地实现:
 - 输入 ECC 参数 p, a, b, G, n
 - 输入用户 ID: ID_A, ID_B
 - 按协议流程进行（流程图附下页）协议
 - 若协商成功，返回协商密钥

函数调用关系图:




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

函数名	函数功能
gcd()	最大公因数
mr()	有限域逆元
ECC.on_curve()	判断点是否在 ECC 上
ECC.add()	ECC 加法
ECC.times()	ECC 乘法
KDF()	密钥分配函数
bt()	整型或十六进制字符串转为字节

2、测试样例及结果截图

OJ 平台截图：

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	详细信息
23840	2022-05-25 11:54:24	Accepted	Python	134ms	11068KB	

本地协商实例（协商成功版）：

- 见附件

本地协商实例（协商失败版）：

```
A进行第1步到第5步的计算: r_A,R_A,x_1_,t_A
A将R_A发送给B

B进行第1步到第4步的计算: r_B,R_B,x2_,t_B
B收到R_A, 对其进行检验:
B协商失败!!!

Process finished with exit code 1
```

3、讨论与思考

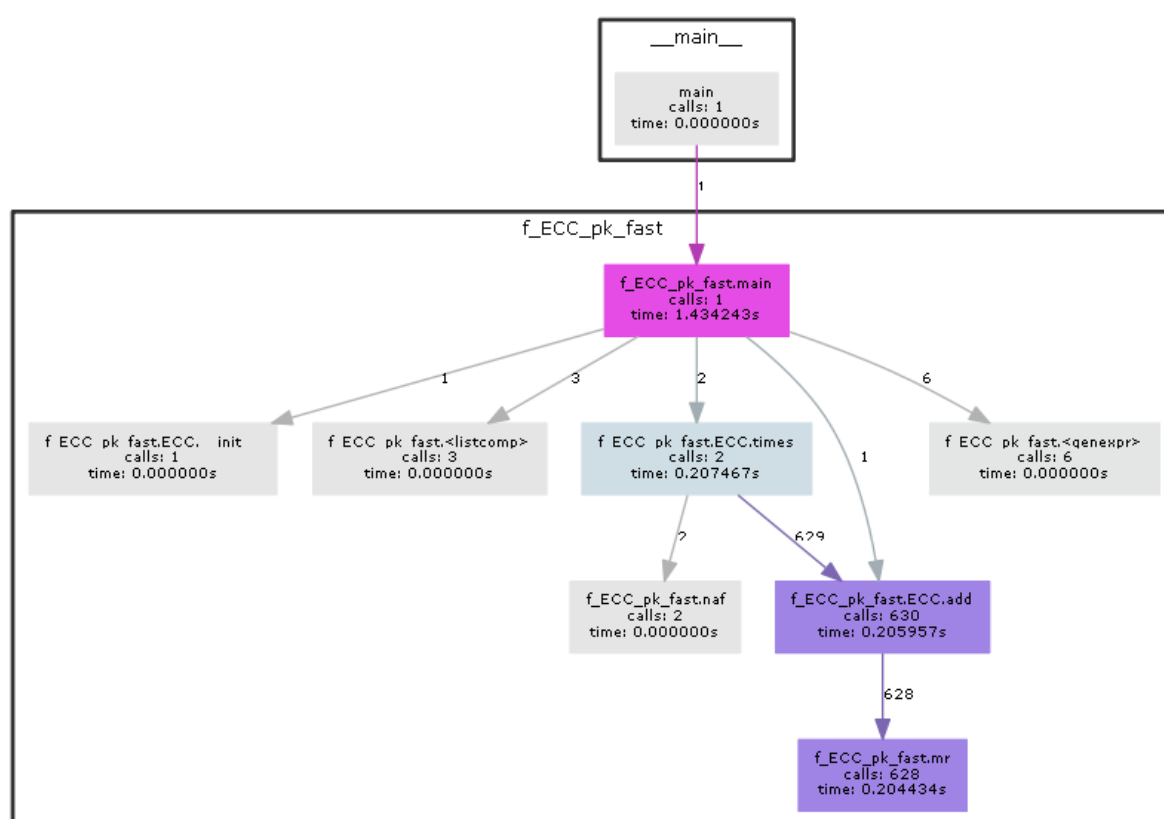
- 为了方便操作，将 ECC 上的加法、乘法运算封装成了类。
- 计算 w 调用了 `math` 库的 `ceil()` 和 `log2()`。
- 将 `int` 型和 16 进制 `str` 型转 `bytes` 型写入了一个函数，通过 `isinstance()` 判断类型后通过 `to_bytes()` 转换。
- 将余因子 h ，安全参数（椭圆曲线上点坐标的比特长度） $param$ ， $klen$ 定义为全局变量。
- 本地实现了完整的协议交互流程，程序仅需输入 ECC 基本参数 p, a, b, G, n ，用户 A、B 的标识身份 ID_A, ID_B 。
- 对于协议进行的每个阶段，运行窗口都有相应的文字说明。
- 程序对于每个协商失败的流程点都做了处理，输出相应文字并 `exit(1)`。

六、【选做】基于 ECC 的公钥加密快速实现

1、算法流程

- ECC 点乘 ($[k]P$):
 - 计算 k 的 NAF 表示
 - 采用快速模幂的思路: 若该位是 1, 加 P ; 若该位是 -1, 加 $(-P)$; $P = P + P$
- ECC 公钥 n 轮加密:
 - $C_1 = [r]P$
 - $C_2 = M + [r(n+1)]Q$
- ECC 公钥 n 轮解密:
 - $M = C_2 - [n+1]([k]C_1)$

函数调用关系图:



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

2、测试样例及结果截图

```
115792089210356248756420345214020892766250353991924191454421103933289684991999
115792089210356248756420345214020892766250353991924191454421103933289684991996
18509919021281880113072981827955639221498448578012875254857346196103069175443
22963146547237050559479511362550074578802567295341616970375194840604136615431 8513236920982856882501899001711249641368838863190450500320353660758877201568
1
6065269455088290935625078859908839028553140764880503559878946642968945892182 97248566623691456333947308494196120167795904532823719329772252328742249457874
75628043876158007008370185296350916712317283944371566779374308036170990
96513597923123126420711825319105397644998306957459694978932196439833315475596 10720292836202791501880298868992459922703519492548574460874619311130191933901
99
2836196863401915602599022753213785933891666039351133884663098349326957490724 27426662229633899247016765528991802333700080038105369561847065403315807331346
8465465517909796912662543007901138302884985227327354714303500052960379059857 90687851011410250211197538806540650787468646750805872850704138724948523318145

Process finished with exit code 0
```

```

11579208892103562487564203452140208892766250353991924191454421193933289604991999
11579208892103562487564203452140208892766250353991924191454421193933289604991996
1850591902201880113072901827955639221458448578012075254857346196103069175443
22963146547237050559479511362590074578002567295341616970375194840604130615431 85132309209820568825618990617112496413088388631904505083283536607588877201568
0
28361968634019156025990227532115785933891666039351133884663098349326957490724 27426682229633899147016765528991802333708880838105369561847065403315807331346
8465465517909879091266254300700113030284985227327354714303500052960379089857 90687891011410150211197938806540650787468646750805872850704130724048523310145
78679514760352709663961092596921956647349330721274547052923221116904421
99
60652694590882909356250788599098839028553140764880503559070946642968945892102 97248566623691456333947388494196120167795904532823719329772252328742149457874
Process finished with exit code 0

```

3、讨论与思考

- 为了方便，仍将 ECC 包装成了 class。
- 由于之前本身就用的快速模幂算法（蒙哥马利法），程序效率不低。
- NAF（Non-adjacent Form），非相邻表示，原理在《信息安全数学基础》讲过。
- NAF 表示可保证 Hamming weight 值最小，且相比于普通的二进制表示，其非零值比率可控制在 1/3 以内。
- 因此，在加入 NAF 表示后，模幂的次数再次减少，程序又有了肉眼可见的提升。

23881	2022-05-25 14:20:56	Accepted	Python	87ms	8508KB	
23880	2022-05-25 14:20:36	Accepted	Python	101ms	8596KB	

- 排行榜上，速度也从 0.5 几 Mb/s 提升到了 0.7 几 Mb/s。

3	20231025	彭逢秋	0.7093659	Python
---	----------	-----	-----------	--------

【收获与建议】

1、收获

- ~~加深了对 ECC 密码体制的理解。~~
- ~~提高了对 Python 的熟练度。~~
- ~~巩固了排版能力。~~
- 收获了劳累与繁忙。

2、建议

- 无。

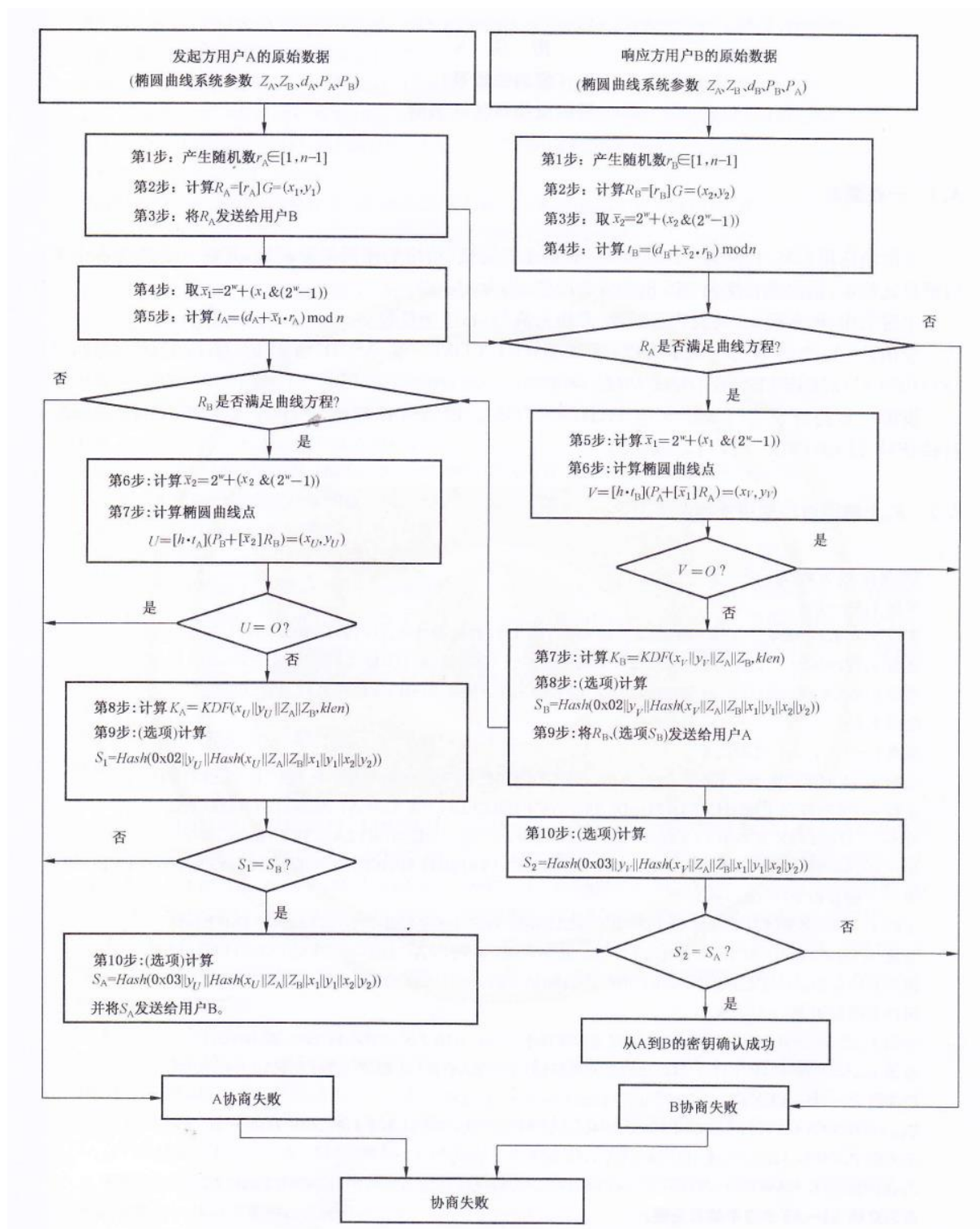
【思考题】

1、ECC 与 RSA 各自的优劣在哪里？

- RSA:
 - **原理：**基于整数分解问题。
 - **优势：**两个大素数的乘积，反向求解问题，较为简单。
 - **劣势：**性能差。需要设置很长密钥才能保证算法安全，密钥越长运算效率越低。
 - **矛盾：**因计算机算力提升，需更长的密钥来防止被攻击。但移动设备加解密需更短的密钥来保证通信效率，存在矛盾问题。
- ECC:
 - **原理：**椭圆曲线、离散对数，比 RSA 复杂。
 - **优势：**安全性高，密钥量小，灵活性好，在密钥长度相等的情况下，RSA 和 ECC 的速度相当；但是在相同的安全强度要求下，ECC 可以使用较少的位数就可以。
 - **劣势：**算法较 RSA 相比复杂，基于 ECC 的各种使用方式都被申请了专利，若新开发 ECC 标准，很容易涉及专利问题。

附件

1、SM2 - 密钥交换协议流程：



2、SM2 - 密钥交换协议本地运行实例

A进行第1步到第5步的计算: r_A, R_A, x_1, t_A

A将 R_A 发送给B

B进行第1步到第4步的计算: r_B, R_B, x_2, t_B

B收到 R_A , 对其进行检验:

检验通过!!!

B进行第5步到第6步计算: x_1, V

B对V进行检验

检验通过!!!

B进行第7步到第10步计算

K_B : 190195029633876757279476154745943734983

S_B : 40716530060286025175564024607564591315105425553938561206284353071961982670534

B将 R_B 与 S_B 发送给A

S_2 : 67406832294685844076757189330270361740536459718377945934695235390929994086387

A收到 R_B , 对其进行检验

检验通过!!!

A进行第6步到第7步计算: x_2, U

A对U进行检验

检验通过!!!

A进行第8步到第9步计算: K_A, S_1

K_A : 190195029633876757279476154745943734983

S_1 : 40716530060286025175564024607564591315105425553938561206284353071961982670534

A检验 S_1 与 S_B 是否相等

$S_1 = S_B$!!!

A进行第10步计算

S_A : 67406832294685844076757189330270361740536459718377945934695235390929994086387

B检验 S_2 与 S_A 是否相等

$S_2 = S_A$!!!

协商成功!!!

协商密钥为: 190195029633876757279476154745943734983

Process finished with exit code 0