



中南大學
CENTRAL SOUTH UNIVERSITY

密码学与应用

实验报告(现代密码及协议)

学生姓名	杨凯楠
学 号	8208201004
专业班级	信息安全 2002 班
指导教师	段桂华
学 院	计算机学院
完成时间	2021 年 12 月 5 日

一、 因式分解攻击

分析：这次实验的因式分解攻击的质数并不是太大，用 python 标准库中的 `libnum.factorize(n)` 即可将其在很短的时间内分解，然后利用 RSA 算法原理就能很轻松将其破解。如果公钥 n 更大一点，则可能需要动用 yafu 等工具。

```
C:\Users\笑靥如花\AppData\Local\Programs\Python\Python39\python.exe
{4054599503: 1, 3564511933: 1}
4054599503
13204463134944167533
Press any key to continue . . .
```

```
n = 14452668311979369299

e = 65537

c = 10591060923058788553

输入解密后的消息：

13204463134944167533

答案正确！
Please input your student id: 8208201004
```

代码：

```
1. import gmpy2
2. import libnum
3.
4. n = 14452668311979369299
5.
6. e = 65537
7.
8. c = 10591060923058788553
```

```

9. print(libnum.factorize(n))
10. q=int(input())
11.
12. p=n//q
13. phi=(p-1)*(q-1)
14. d=gmpy2.invert(e,phi)
15. m=pow(c,d,n)
16. print(int(m))

```

二、 选择密文攻击

分析:

攻击思路:

令 $m2=2$

$c2=2^e$

计算 $X = (C \times 2^e) \bmod n$

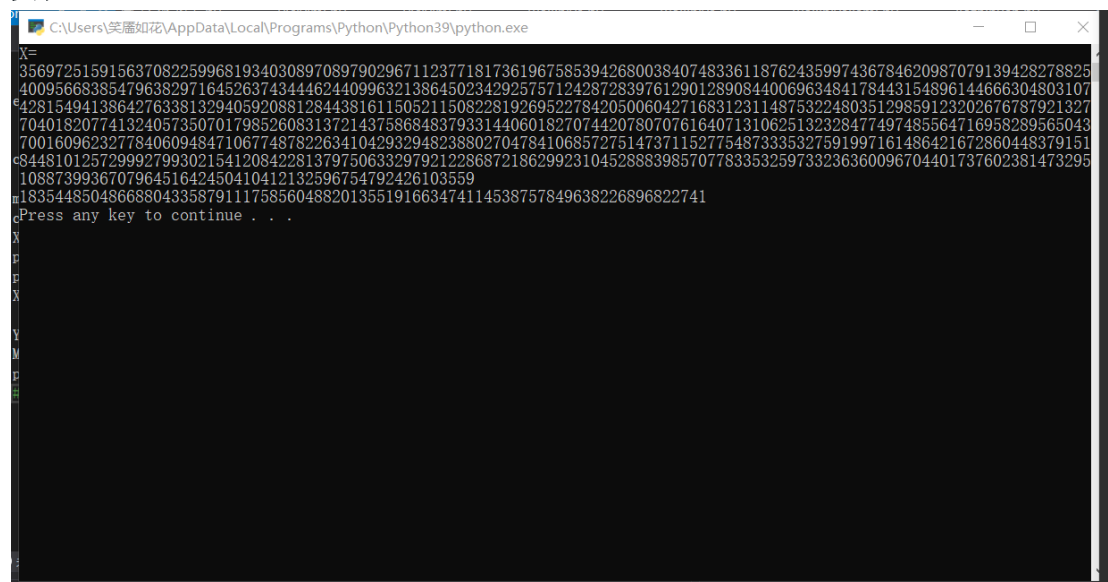
将X作为选择明文提交，并收到 $Y = X^d \bmod n$ 。

因此: $X = (C \bmod n) \times (2^e \bmod n) = (M^e \bmod n) \times (2^e \bmod n) = (2M)^e \bmod n$

因此: $Y = (2M) \bmod n$

然后计算 $M = (Y * gmpy2.invert(2,n)) \% n$ ，得到 M

实现:



The screenshot shows a Python terminal window with the following content:

```

C:\Users\笑靥如花\AppData\Local\Programs\Python\Python39\python.exe
X=
356972515915637082259968193403089708979029671123771817361967585394268003840748336118762435997436784620987079139428278825
400956683854796382971645263743444624409963213864502342925757124287283976129012890844006963484178443154896144666304803107
428154941386427633813294059208812844381611505211508228192695227842050060427168312311487532248035129859123202676787921327
704018207741324057350701798526083137214375868483793314406018270744207807076164071310625132328477497485564716958289565043
700160962327784060948471067748782263410429329482388027047841068572751473711527754873335327591997161486421672860448379151
844810125729992799302154120842281379750633297921228687218629923104528883985707783353259733236360096704401737602381473295
108873993670796451642450410412132596754792426103559
18354485048668804335879111758560488201355191663474114538757849638226896822741
Press any key to continue . . .
X
C
E
X
Y
M
C
E

```

输入 '-h' 获取帮助...

```
-d 35697251591563708225996819340308970897902967112377181736196758539426800384074833611876243599743678
46209870791394282788254009566838547963829716452637434446244099632138645023429257571242872839761290128
90844006963484178443154896144666304803107428154941386427633813294059208812844381611505211508228192695
22784205006042716831231148753224803512985912320267678792132770401820774132405735070179852608313721437
58684837933144060182707442078070761640713106251323284774974855647169582895650437001609623277840609484
71067748782263410429329482388027047841068572751473711527754873335327591997161486421672860448379151844
81012572999279930215412084228137975063329792122868721862992310452888398570778335325973323636009670440
1737602381473295108873993670796451642450410412132596754792426103559
```

签名结果：

```
36708970097337608671758223517120976402710383326948229077515699276453793645482
```

```
-s 18354485048668804335879111758560488201355191663474114538757849638226896822741
```

答案正确、(*▽*)ノ

Please input your student id: 8208201004

代码：由于代码中数字位数占比甚至比实际代码量还高，故而不贴全部代码了，只贴去掉数字的部分

```
1. import libnum
2. import gmpy2
3. n =
4.
5. e = 65537
6.
7. c =
8.
9. m2=2
10. c2=
11.
12. X=(c*(2**e))%n
13. print("X=")
14. print(X)
15. X=
16.
17. Y=3670897009733760867175822351712097640271038332694822907751569927645
    3793645482
18. M=(Y*gmpy2.invert(2,n))%n
19. print(M)
20. #M=183544850486688043358791117585604882013551916634741145387578496382
    26896822741
```

三、 共模攻击

分析：生成密钥的过程中使用了相同的模数 n ，此时用不同的密钥 e 加密同一信息 m 即：

$$c1 = m^{e1} \% n$$

$$c2 = m^{e2} \% n$$

若两个密钥 e 互素根据扩展的欧几里得算法则存在 $s1, s2$ 有：

$$e1 * s1 + e2 * s2 = \gcd(e1, e2) = 1$$

经过数学计算：

$$\begin{aligned} & (c1^{s1} * c2^{s2}) \% n \\ = & (m^{e1} \% n)^{s1} * (m^{e2} \% n)^{s2} \% n \\ = & m^{(e1 * s1 + e2 * s2)} \% n \\ = & m \% n \\ = & m \end{aligned}$$

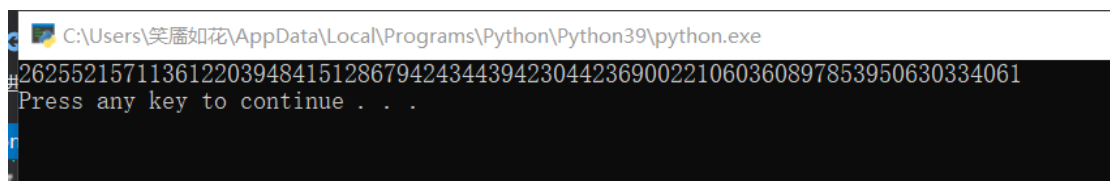
也就是在不知道私钥的情况下，得到了明文 m

在此攻击过程中，若 $s2$ 或 $s1$ 有一个为负数，则需要将其变换为

$$s1 = -s1$$

$$c1 = \text{gmpy2.invert}(c1, n)$$

实现：



```
C:\Users\笑靥如花\AppData\Local\Programs\Python\Python39\python.exe
26255215711361220394841512867942434439423044236900221060360897853950630334061
Press any key to continue . . .
```

输入 '-h' 获取帮助...

```
-d 35697251591563708225996819340308970897902967112377181736196758539426800384074833611876243599743678
46209870791394282788254009566838547963829716452637434446244099632138645023429257571242872839761290128
90844006963484178443154896144666304803107428154941386427633813294059208812844381611505211508228192695
22784205006042716831231148753224803512985912320267678792132770401820774132405735070179852608313721437
58684837933144060182707442078070761640713106251323284774974855647169582895650437001609623277840609484
71067748782263410429329482388027047841068572751473711527754873335327591997161486421672860448379151844
81012572999279930215412084228137975063329792122868721862992310452888398570778335325973323636009670440
1737602381473295108873993670796451642450410412132596754792426103559
```

签名结果：

```
36708970097337608671758223517120976402710383326948229077515699276453793645482
```

```
-s 18354485048668804335879111758560488201355191663474114538757849638226896822741
```

答案正确、(*▽*)ノ

Please input your student id: 8208201004

代码：

```
1. n =
2. e1 = 42131
3. c1 =
4. e2 = 37561
5. c2 =
6. import libnum
7. import gmpy2
8. s=gmpy2.gcdext(e1,e2)#gcdext(a, b) 返回一个 3 元素元组 ( g , s , t ) 使得 g == gcd( a , b ) 和 g == a * s + b * t
9. s1=s[1]
10. s2=s[2]
11. if s1<0:
12.     s1=-s1
13.     c1=gmpy2.invert(c1,n)
14. if s2<0:
15.     s2=-s2
16.     c2=gmpy2.invert(c2,n)
17. m=int(pow(c1,s1,n)*pow(c2,s2,n)%n)
18. print(m)
19.
20. m=5875399524593940334863216137173017183468825719172085194750838045412
    8181010988
```

四、 针对 RSA 的低加密指数广播攻击

分析：本道题解题方法在题干里已经说过了，简明的说来，就是

当 $e=3$ 时

$$c_i = m^3 \bmod N_i$$

可以设 c 且 c 满足 $0 < c < N_1 * N_2 * N_3$

$$\text{则 } c = c_1 \bmod N_1, c = c_2 \bmod N_2, c = c_3 \bmod N_3$$

$$c = m^3 \bmod N_1 * N_2 * N_3$$

由于 $m < N_i$ 对于所有的 i 都成立，所以有 $m^3 < N_1 * N_2 * N_3$

从而，就可以得 $c = m^3$ ，对 c 做立方根运算即可得到消息 m 。

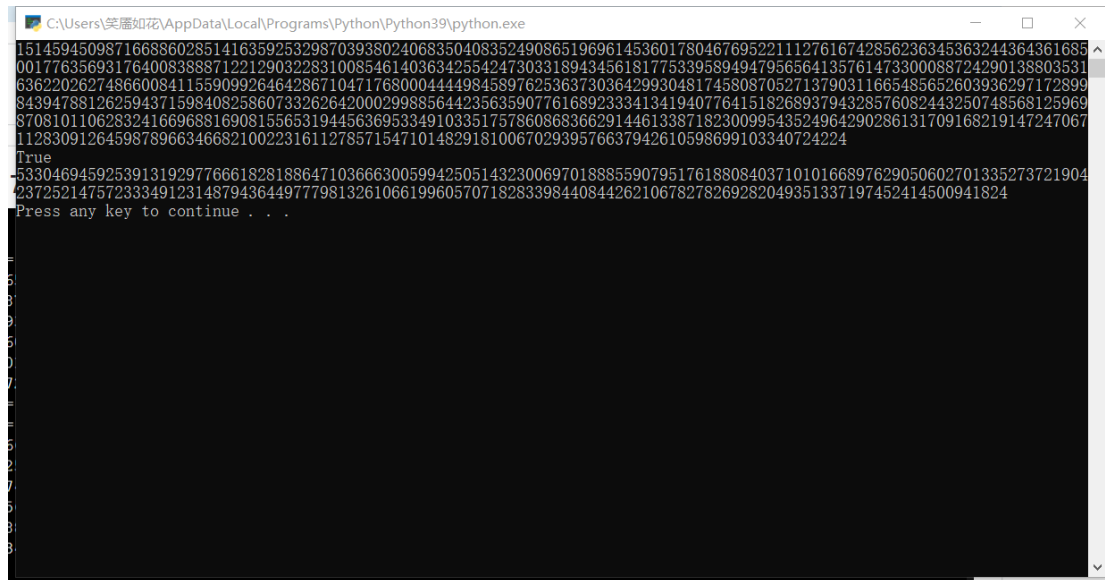
本题的一大难点是对一个大数进行开立方根的运算普通的 python 开根方法 $c^{**}(1/3)$ 和 $\text{pow}(c, (1/3))$ 均会带来精度损失得不到正确的结果。因此我查阅了各种方法，终于发现了 gmpy2 库中的函数 $\text{iroot}(m, a)$ 表示对 m 开 a 次方，返回只有两个，第一个为结果，第二个为 bool 类型，True 表示能被开方成为整数。

代码：

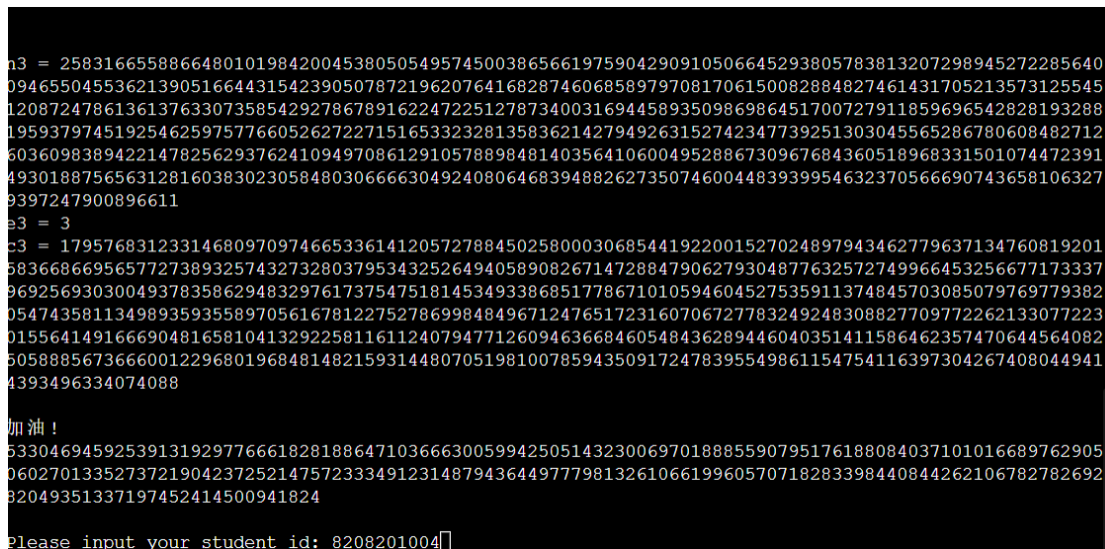
```
1. from gmpy2 import*
2.
3. n1 =
4. e1 = 3
5. c1 =
6.
7. n2 =
8. e2 = 3
9. c2 =
10.
11. n3 =
12. e3 = 3
13. c3 =
14. N=n1*n2*n3
15. N1=N//n1
16. N2=N//n2
17. N3=N//n3
18. niN1=invert(N1,n1)
19. niN2=invert(N2,n2)
20. niN3=invert(N3,n3)
21. m3=(c1*N1*niN1+c2*N2*niN2+c3*N3*niN3)%N
22. print(int(m3))
23. status = iroot(m3, 3)[1] #status 为 1 表示改数能被开立方成整数
24. print(status)
```

```
25. m = iroot(m3, 3)[0] #开立方后的结果
26. print(int(m))
```

结果:



```
C:\Users\笑塵如花\AppData\Local\Programs\Python\Python39\python.exe
151459450987166886028514163592532987039380240683504083524908651969614536017804676952211127616742856236345363244364361685
001776356931764008388871221290322831008546140363425542473033189434561817753395894947956564135761473300088724290138803531
636220262748660084115590992646428671047176800044449845897625363730364299304817458087052713790311665485652603936297172899
843947881262594371598408258607332626420002998856442356359077616892333413419407764151826893794328576082443250748568125969
870810110628324166968816908155653194456369533491033517578608683662914461338718230099543524964290286131709168219147247067
112830912645987896634668210022316112785715471014829181006702939576637942610598699103340724224
True
533046945925391319297766618281886471036663005994250514323006970188855907951761880840371010166897629050602701335273721904
237252147572333491231487943644977798132610661996057071828339844084426210678278269282049351337197452414500941824
Press any key to continue . . .
```



```
n3 = 258316655886648010198420045380505495745003865661975904290910506645293805783813207298945272285640
09465504553621390516644315423905078721962076416828746068589797081706150082884827461431705213573125545
12087247861361376330735854292786789162247225127873400316944589350986986451700727911859696542828193288
19593797451925462597577660526272271516533232813583621427949263152742347739251303045565286780608482712
60360983894221478256293762410949708612910578898481403564106004952886730967684360518968331501074472391
49301887565631281603830230584803066663049240806468394882627350746004483939954632370566690743658106327
9397247900896611
e3 = 3
c3 = 179576831233146809709746653361412057278845025800030685441922001527024897943462779637134760819201
58366866956577273893257432732803795343252649405890826714728847906279304877632572749966453256677173337
96925693030049378358629483297617375475181453493386851778671010594604527535911374845703085079769779382
05474358113498935935589705616781227527869984849671247651723160706727783249248308827709772262133077223
01556414916669048165810413292258116112407947712609463668460548436289446040351411586462357470644564082
50588856736660012296801968481482159314480705198100785943509172478395549861154754116397304267408044941
4393496334074088
加油!
53304694592539131929776661828188647103666300599425051432300697018885590795176188084037101016689762905
06027013352737219042372521475723334912314879436449777981326106619960570718283398440844262106782782692
82049351337197452414500941824
Please input your student id: 8208201004
```


五、 bloom 门限方案协议

门限方案作为一个大题目,实验要求比前面的实验高不少,但是由于这学期的特殊性,本专业学生没有时间写一个完整的,具有多种方案的,有 GUI 的门限方案协议,故而只将其中最核心的,最根本的算法实现后提交。

分析:

1.1 (k, n) —门限方案的引入

定义 设 n 个参与者 P_i 各自拥有关于 S 的子秘密为 $s_i, s_i \in S, 1 \leq i \leq n$, 组成集合 $A \subseteq P(\{1, 2, \dots, n\})$ 。

一个 A ——秘密共享方案是由 $(S, (I_1, I_2, \dots, I_n))$ 按照如下条件生成的:

①对于任意 $A \in A$, 对于集合 $\{I_i | i \in A\}$ 而言求解 S 是容易的;

②对于任意 $A \in P(\{1, 2, \dots, n\}) \setminus A$, 对于集合 $\{I_i | i \in A\}$ 而言得不到有关 S 的任何信息。

集合 A 被称作授权接入结构或简称接入结构。其中一类重要的秘密共享方案就是门限类秘密共享方案,在这种方案中,如果需要门限为 k , 且 $2 \leq k \leq n$, 那么最小的接入结构就是

$$A_{\min} = \{A \in P(\{1, 2, \dots, n\}) \mid |A| = k\}。$$

在这种情况下,一个 A ——秘密共享方案就称作为一个 (k, n) —门限(秘密共享)方案。

1.2 Asmuth-Bloom 门限方案

1.2.1 参数选取

设秘密分发者为 Dealer, 它选取一大素数 q , 正整数 S , 以及 n 个严格递增的数 m_1, m_2, \dots, m_n 。

满足以下条件:

- ① $s < q$;
- ② $(m_i, m_j) = 1, (\forall 1 \leq i, j \leq n, i \neq j)$;
- ③ $(q, m_i) = 1, (\forall 1 \leq i \leq n)$;
- ④ $N = \prod_{i=1}^k m_i > q \prod_{i=1}^{k-1} m_{n-i+1}$ 。

以 S 作为秘密, 再随机选取 A , 满足: $0 \leq A \leq [N/q] - 1$, 并公布 q 。

1.2.2 具体算法

秘密分发者 Dealer 计算 $y = s + Aq$ (由上知 $y < N$) 和 $y_i = y \pmod{m_i}$, 并将 (m_i, y_i) 分发给秘密保管者 P_i 作为 P_i 保管的子秘密, 集合 $\{(m_i, y_i)\}_{i=1}^n$ 即

构成一个 (k, n) 门限方案。因为当 k 个秘密保管者(记为 i_1, i_2, \dots, i_k) 提供出各自保管的子秘密时, 由 $\{(m_{i_j}, y_{i_j})\}_{j=1}^k$ 建立方程组

$$\begin{cases} y = y_{i_1} \pmod{m_{i_1}} \\ y = y_{i_2} \pmod{m_{i_2}} \\ \vdots \\ y = y_{i_k} \pmod{m_{i_k}} \end{cases}$$

根据中国剩余定理可求得: $y = y' \pmod{N'}$, 式中 $N' = \prod_{j=1}^k m_{i_j} \geq N$ 。因为 $y < N \leq N'$, 所以 $y = y'$ 是唯一的, 再由 $s = y - Aq$ 即得到秘密 s 。这里由于 q 是公开的, 所以得到 $[s]_q = [y]_q$ (即 $s = y \pmod{q}$), 再由 $s < q$ 就得到了秘密 $s = [y]_q$ 。

代码实现:

```
1. import gmpy2
2.
3. p = 75297982672350428988551784994988486001210156146516974173533177598
   866562086172121532593369921196628861659049139917441621442463309622499
   634743241964588561467966444355766623319225394364567716222658670725978
```

```
321207057158262516613844538997580248744278567699989466427677982208040
0611576514387245013421805401
```

```
4.
```

```
5.
```

```
6. x1 = 5422197891194869945108091920780487037119710907425950992202248813
699671885850292413002994260566515773190907102631754099765447641994104
842125931973264989435508691754375331952180052267295341542395421023439
357473313832654914242596327883518321327000503903366522099501204763421
2734961880592507135746380463260174818
```

```
7. m1 = 1529079146490237154060159972787175844397267115895767133248279554
199289222420428276103988827829460824700437738918503467288359945114793
028567960323576734753747874178256439564958339202166899105904767613955
928099839863987862264758067559604954061219942434650817675316705877750
68735762154790201776599042843130603693
```

```
8.
```

```
9.
```

```
10. x2 = 1541939301530551721507680789932808230054048690578283345022111207
690776513904988773249929045149871382966906126045052785110282164093557
707773731974540000312012529483713866086651678017248660253545115270748
903673820526429601544860437444236031536896693086740101794161730732861
29376243765931435178227789514359022981
```

```
11. m2 = 3110836172798679684588377974099942207287492002220569475342260314
758889886091369337351791014242898322492512055371379505107133320977511
653620455552773144993195983845091749821784766425819062023745877484180
975096846609393616449893045143585022808616247494825662783636688770738
19192523956174250049481732836412437403
```

```
12.
```

```
13.
```

```
14. x3 = 8600371443572051585842463400253340575938881097926158220672131919
244890582906792291470394039761972502136127005899688501819919989372693
053166409145727446557880943870611180351582090702471046627675276849736
392006934432156943442746925264673051664806582008622787288898520133565
2137316440212569709116659496043385570
```

```
15. m3 = 4936923996650994686321263958104153352234913951651490774816824326
031035410207038767684327060929582121730415135133867534052139573834664
354307773015691078671465465307624815112738565708953936264162748845772
803967944897196860587050336343984976555196745291773647142366621134033
68054318288511717707726922373804210937
```

```
16.
```

```
17.
```

```
18. Mlist=[m1,m2,m3]
```

```
19. Xlist=[x1,x2,x3]
```

```
20. M=1
```

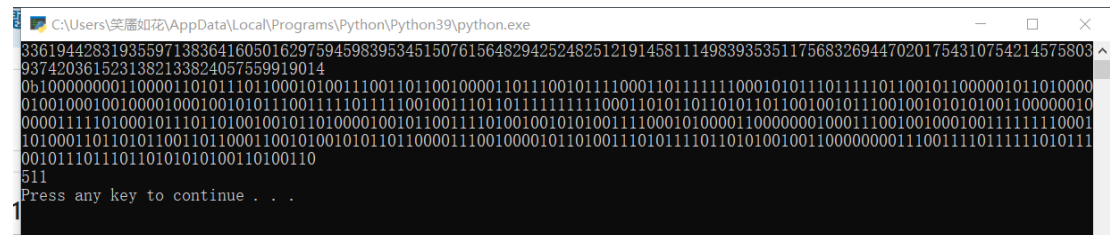
```
21. for i in Mlist:
```

```

22.     M*=i
23.
24. A=[]
25. W=[]
26. T=[]
27.
28. for i in range(3):
29.     A.append(Xlist[i]%Mlist[i])
30.     W.append(M//Mlist[i])
31.     T.append(gmpy2.invert(W[i],Mlist[i]))
32. x=0
33. for i in range(3):
34.     x+=A[i]*T[i]*W[i]
35. x=x%M
36. s=x%p
37. print(s)
38. print(bin(s))
39. #print(len(bin(s))-2)

```

实现:



```

C:\Users\笑靥如花\AppData\Local\Programs\Python\Python39\python.exe
336194428319355971383641605016297594598395345150761564829425248251219145811149839353511756832694470201754310754214575803
9374203615231382133824057559919014
0b10000000011000011010111011000101001110011011001000011011100101111000110111111000101011101111011001011000001011010000
01001000100100001000100101011100111101111001001110110111111100011010110110110110010010110010010101001100000010
000011111010001011101101001001011010000100101100111101001001010011110001010000110000000100011100100100010011111110001
101000110110101100110110001100101001010110110000111001000010110100111010111011010100100110000000011001111011111010111
0010111011010101010100110100110
511
Press any key to continue . . .

```

```

Secret 3: x = 860037144357205158584246340025334057593888109792615822067213191924489058290679229147039
40397619725021361270058996885018199199893726930531664091457274465578809438706111803515820907024710466
27675276849736392006934432156943442746925264673051664806582008622787288898520133565213731644021256970
9116659496043385570, m = 4936923996650994686321263958104153352234913951651490774816824326031035410207
03876768432706092958212173041513513386753405213957383466435430777301569107867146546530762481511273856
57089539362641627488457728039679448971968605870503363439849765551967452917736471423666211340336805431
8288511717707726922373804210937

Please input real secret(which is a 512-bit-number): 100000000110000110101110110001010011100110110010
0001101110010111100011011111100010101110111101100101100000101101000001001000100100010010010101110
011110111110010011101011111111000110101101101101100100101110010010101001100000010000011111010
0010111011010010010110100001001011001111010010010101001111000101000011000000010001110010010001001111
111000110100011011010110011011000110010010101101100001110010000101101001110101111010100100110000
0000111001111011111010111001011101101101010100110100110

FAILED...
Please input real secret(which is a 512-bit-number): 336194428319355971383641605016297594598395345150
76156482942524825121914581114983935351175683269447020175431075421457580393742036152313821338240575599
19014

Congratulation! Please leave your student number!
Please input your student id: 8208201004

```