

lesson-2

RSA

RSA是1977年由罗纳德·李维斯特 (Ron Rivest) , 阿迪·萨莫尔 (Adi Shamir) 和 伦纳德·阿德曼 (Leonard Adleman) 一起提出的. 当时他们三人都在麻省理工学院工作. RSA就是他们三人姓氏开头字母拼在一起组成的

加密流程

1. 选择一对不相等且足够大的质数 p , q
2. 计算 p 、 q 的乘积, 记为 n

$$n = pq$$

3. 计算 n 的欧拉函数

$$\phi(n) = (p - 1)(q - 1)$$

4. 选择一个与 $\phi(n)$ 互素的数 e

$$\gcd(e, \phi(n)) = 1, 1 < e < \phi(n), \text{ 一般常见的 } e \text{ 为 } 65537$$

5. 计算 e 关于 $\phi(n)$ 的逆元 d

$$ed = 1 \pmod{\phi(n)}$$

6. 将 (e, n) 作为公钥公开, 用于加密; 将 (d, n) 作为私钥保存, 用于解密
7. 具体加密过程

$$c = m^e \pmod{n}$$

8. 具体解密过程

$$m = c^d \pmod{n}$$

$$m = c^d = m^{ed} = m^{k\phi(n)+1} = m \pmod{n}$$

分解 n

```
e = 65537
n =
879243482641324068752761405144999371450508936656025929924181716470424916
58461
c =
876776523868977493006385913653410163901286927839492773059878281770459325
76708
```

分解n

<http://www.factordb.com/>

yafu

```
from Crypto.Util.number import *
from gmpy2 import *

p = 275127860351348928173285174381581152299
q = 319576316814478949870590164193048041239

e = 65537
c =
876776523868977493006385913653410163901286927839492773059878281770459325
76708

n = p*q
phi = (p - 1) * (q - 1)
d = invert(e, phi)
m = pow(c, d, n)

print(long_to_bytes(m))
# bytes_to_long()
# LitCTF{factordb!!!}
```

共模攻击

```
from Crypto.Util.number import *

p = getPrime(512)
q = getPrime(512)
```

```

n =
120294155186626082670474649118722298040433501930335450479777638508444129
059776534554344361441717048531505985491664356283524886091709370969857047
470362547600390987665105196367975719516115980157839088766927450099353377
496192206005171597109864609567336679138620134544004766539483664270351472
198486955623315909571

e1 = 38317
e2 = 63409

c1 =
427031386961873950303372058605032702143531515881495061107312649525951937
572352292150676388584314935870936123971654072213941746902636910953242980
121347797030417528100289357112140388355848233851087719012164417846731998
460411090744671778916809235932063267885231581806376658136426888245937881
92044139055552031622

c2 =
504600927861114704089453162700868128072302532348093036940079026289240577
139843970411416651256157357526001149648521576849044299287715316398994969
879050673664158067710031219548524657311106294597259944549041592772285143
372781052077210115797946047612555223914465344588153899835628906319947266
87526070228315925638

```

$$c_1 = m^{e_1} \mod n$$

$$c_2 = m^{e_2} \mod n$$

其中 $\gcd(e_1, e_2) = 1$

求出一组 x_i, y_i , 使其满足 $a_i \times x_i + b_i \times y_i = \gcd(a_i, b_i)$ 。

通过扩展欧几里得算法我们可以知道，一定存在一组数 (s_1, s_2) ，使得 $s_1 e_1 + s_2 e_2 = 1$

$$c_1^{s_1} * c_2^{s_2} = m^{s_1 e_1} * m^{s_2 e_2} = m \mod n$$

```

from Crypto.Util.number import *
from gmpy2 import *

```

```

n =
120294155186626082670474649118722298040433501930335450479777638508444129
059776534554344361441717048531505985491664356283524886091709370969857047

```

```
470362547600390987665105196367975719516115980157839088766927450099353377
496192206005171597109864609567336679138620134544004766539483664270351472
198486955623315909571
e1 = 38317
e2 = 63409

c1 =
427031386961873950303372058605032702143531515881495061107312649525951937
572352292150676388584314935870936123971654072213941746902636910953242980
121347797030417528100289357112140388355848233851087719012164417846731998
460411090744671778916809235932063267885231581806376658136426888245937881
92044139055552031622
c2 =
504600927861114704089453162700868128072302532348093036940079026289240577
139843970411416651256157357526001149648521576849044299287715316398994969
879050673664158067710031219548524657311106294597259944549041592772285143
372781052077210115797946047612555223914465344588153899835628906319947266
87526070228315925638

_, s1, s2 = gcdext(e1, e2)
m = pow(c1, s1, n) * pow(c2, s2, n) % n
print(long_to_bytes(m))

# NSSCTF{same_module_attack!}
```

更多攻击方法

<https://lazzaro.github.io/2020/05/06/crypto-RSA/>

[重生之我是密码学高手|RSA的12种常见的攻击方式_广播攻击rsa-CSDN博客](#)

[RSA攻击方法总结笔记_rsa低指数攻击-CSDN博客](#)

格

前置知识

向量和向量空间

以二维向量空间举例，我们可以任意取一个点A，连接原点O和A点，这就构成成了一个向量。我们也可以用A点的坐标来表示这个向量,就像这样 $\mathbf{v}=(x,y)$

向量满足加减、数乘和点积运算，下面举例说明（加粗表示向量）

$$\mathbf{v}=(a, b), \mathbf{w}=(c, d)$$

$$\mathbf{v}+\mathbf{w}=(a+c, b+d) \quad (\text{加法同理})$$

$$c \mathbf{w}=(c a, c * b) \quad (\text{数乘})$$

$$\mathbf{v} \cdot \mathbf{w}=a c+b * d \quad (\text{点积})$$

多维是同理的

然后向量还分行向量和列向量，可以简单理解为行向量就是在纸上横着写的向量（确信），列向量就是在纸上竖着写的向量（确信）

向量大小和基

$\|\mathbf{v}\|$ 用来表示向量的长度，其大小是 $\sqrt{\mathbf{v} * \mathbf{v}}$ (高中应该就有这个知识点的)

这里还有一些关于线性独立的知识点，比如向量空间里有一组向量 \mathbf{v}_i

$$\text{满足方程 } a_1 \cdot \mathbf{v}_1 + a_2 \cdot \mathbf{v}_2 + \dots + a_k \cdot \mathbf{v}_k = 0$$

当 a_i 全为零时等式才成立，那么这 k 个向量就是**线性无关**的，反之线性相关

而向量空间的基底会要求是一组线性无关的向量，当我们有了这一组向量基，我们就可以**用这个基底来表示这个向量空间中的任意向量**

拿二维空间举例

$\mathbf{i}=(0,1) \quad \mathbf{j}=(1,0)$ 就是一组线性无关的基底，这两个向量就可以表示整个二维平面

其实了解一下矩阵会更有帮助，但是我就不介绍了（有试着想写一下，但是发现讲不清楚，还是留给大家的线代老师吧）

[线代课程](#)（放一个宋浩老师线代课的连接吧）

格定义

推荐文章

[格攻击之小未知数方程求解入门——原理与例子 | Tover's Blog](#)

这里先放一个课本上的定义

设 b_1, b_2, \dots, b_n 是 \mathbb{R}^m 中 n 个线性无关的向量 ($m \geq n$), \mathbb{Z} 为整数集, 称

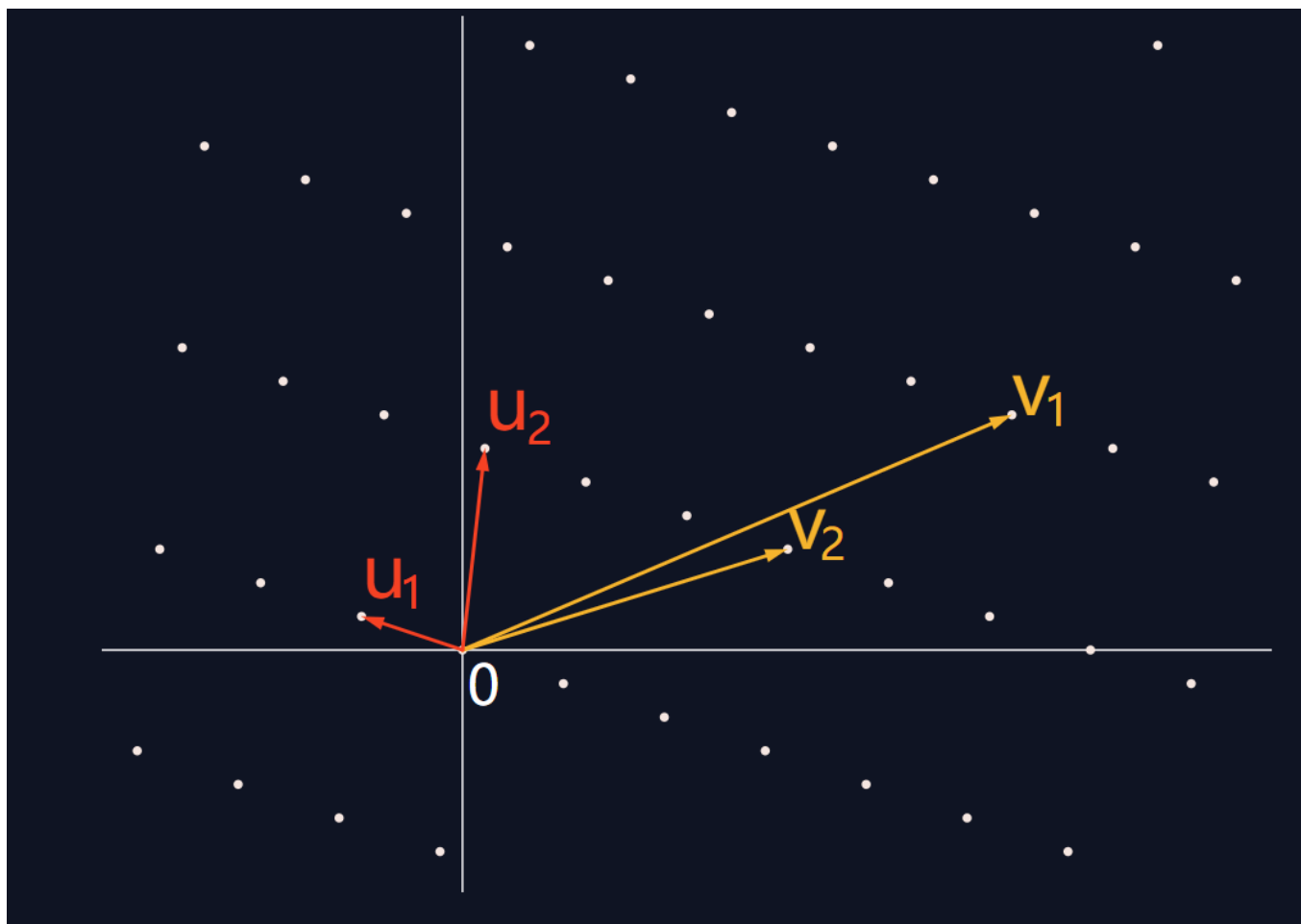
$$\mathcal{L}(b_1, b_2, \dots, b_n) = \left\{ \sum_{i=1}^n x_i b_i : x_i \in \mathbb{Z} \right\}$$

为 \mathbb{R}^m 中的一个格, 简记为 L , 称 b_1, b_2, \dots, b_n 为格 L 的一组基, m 为格 L 的维数, n 为格 L 的秩. 格 L 的基也常写成矩阵的形式, 即以 b_1, b_2, \dots, b_n 为列向量构成矩阵 $B = [b_1, b_2, \dots, b_n] \in \mathbb{R}^{m \times n}$, 那么格 L 可以写成

$$\mathcal{L}(B) = \{Bx : x \in \mathbb{Z}^n\}.$$

可能看起来比较难懂, 但是我们还是从上面那个二维平面出发, 我们已经知道了我们可以用一组基底来表示整个二维平面 (基底不唯一), 我们是通过把这两个向量乘以不同的系数后相加得到的, 那么想象一下, 我们如果令系数只为整数会怎么样呢?

这个时候, 我们通过这个基底, 就只能得到空间里的一些点, 那么这个就是一个整数格



$au_1 + bu_2$, 其中 a, b 取任意实数

靠 a, b 取整数, 则可以得到格

$$u_1 = (x_1, y_1) u_2 = (x_2, y_2)$$

$$(a, b) \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \end{pmatrix}$$

把矩阵当作B，向量设为x，就可以写成 xB

我们把基底写成一个矩阵，那么就可以得到上面定义里的写法，同时这个矩阵的**行列式**的值，叫做这个**基本域的容积**

同时如果我们扩展到n维向量空间，就是定义里的表示

格常见问题

通过上面我们简单了解了一下格是什么，长什么样，那么就要考虑能拿它干什么

这就得提一下格上的两个常见问题，一个是最短向量问题（SVP），另一个是最近向量问题（CVP）

SVP是指我们要在格L上找到一个最短的非零向量 v ，使得 $\|v\|$ 最小

LLL格基规约

这里的运用就是根据关系找到一个基底的矩阵，矩阵中所有数是已知的，然后我们要求解的向量（要求解的未知数构成的向量）是一些数作用在这个矩阵后得到，这个向量就在格上，然后我们期望这个向量是最短向量，以此来求解未知数，下面放一个代码

```
# sage
from gmpy2 import *
from Crypto.Util.number import *
#from tqdm import *

a =
134812492661960841508904741709490501744478747431860442812349873283670029
478557996515894514952323891966807395438595833662645026902457124893765483
848187664404810892289353889878515048084718565523356944401254704006179297
186883488636493997227870769852726117603572452948662628907410024781493099
700499334357552050587
b =
152286591565688386740348231717146038132479822729836552365085118456780249
624001176807859393885859529672439389139778265881664724348978066199941181
190043931982178426611753918849840764839719484963194107473739185239931895
166959388190793522098628263838865650309096315396825424413192888702580008
```

```
860934171497410392121920297269132166119813555392841100218478013957114977
203728374908650420175843858941737833694073292635280625609386525582480320
2598635567105242590697162972609
```

```
mat = [[b,0],[a,1]]
M = Matrix(ZZ,mat)
qq,pp= M.LLL()[0]    # 最短向量

print(pp)
print(qq)
```

要运用LLL算法是有条件的，就是我们要通过这个算法来求最短向量，那么我们要求的向量起码得是最短向量，这就得看一下上面的最短向量存在定理，我们要求解的向量长度满足条件后才是最短向量，才可以用LLL算法，然后格的维数也不能太高，不然准确度不高，耗时也会长。

题目(NTRU)

```
from secret import flag
from Crypto.Util.number import *

q = getPrime(2048)

f = getPrime(1024)
g = getPrime(768)

h = (inverse(f, q) * g) % q

m = bytes_to_long(flag)

e = (getPrime(32) * h + m) % q

print((h, q))
print(e)

#
(89164527228214184632487268257212570217441942868747069158324446317715966
161164917750914731427988672785985864826783876689867644612651311191645004
737199398943431634963255563401814296759376414959813538577246270818473042
469870743037226421729888641389674040242012460503871528540017467631044177
732144089068793669587297442596127772575423515015920194837456218248947900
```

```

966392057714215602951756331528776677200383961545716978613268214831708352
380928797472975066069833228907062208242615815333248248585990826118860266
687885777579709848922926092710821763114335079319936729459258839856293115
14143607457603297458439759594085898425992,
319858426364986859453309057265394989014436949557363320736397444663890393
731436189205111222888442828494072902058049916341678164174687034592291388
913481151919213952783366956842104371306813379716860080480543404996547213
177212412399907010996852072534766429315865633636381416360119412689629996
411302638281515384891392546250993301995575031536800893875388635744801348
982113112522274638708389477774793099281957912410051274458216716846072377
068493083729233727955737320003650728151121195337026146203252381838992661
476821938928663306780769251996745545690181031642282787421517788323194061
35513140669049734660019551179692615505961)
#
200417136138763820079692840566981490071542488574207525204968292463245121
971882110296659907135996679840197155034865071262245580921763922824866893
479530698151232127790907839095452441603189383575293074820256977693941149
670285645463553108836704621975280111817685888784478568751732638008850486
761909782068512688874455277853875321673709437451805381689654616120970370
415709123656481254498041092996309588403983977219168608766878084740043918
438698133968584687308776277332348327443287684438306694693459267668824463
787658473344215950344706391713975873953419774535368599464104312522872033
12913117023084978959318406160721042580688

```

$$h = f^{-1}g \pmod{q}$$

$$e = rh + m \pmod{q}$$

$$e = rf^{-1}g + m \pmod{q}$$

$$ef = rg + mf \pmod{q}$$

$$mf = ef \pmod{q} \pmod{g}$$

$$m = eff^{-1} \pmod{g}$$

求解思路，所以要求f和g

$$h = f^{-1}g \pmod{q}$$

$$fh = g \pmod{q}$$

$$fh = g + kq$$

$$fh - kq = g$$

$$f * 1 = f$$

$$(f, -k) \begin{pmatrix} 1 & h \\ 0 & q \end{pmatrix} = (f, g)$$

Hermite's Theorem

每一个 n 维的 lattice L , 都包含一个非零向量 $\mathbf{v} \in L$ 满足

$$\|\mathbf{v}\| \leq \sqrt{n} \det(L)^{1/n}$$

```

from random import randint
from gmpy2 import *
from Crypto.Util.number import *
import math

h,q =
(89164527228214184632487268257212570217441942868747069158324446317715966
161164917750914731427988672785985864826783876689867644612651311191645004
737199398943431634963255563401814296759376414959813538577246270818473042
469870743037226421729888641389674040242012460503871528540017467631044177
732144089068793669587297442596127772575423515015920194837456218248947900
966392057714215602951756331528776677200383961545716978613268214831708352
380928797472975066069833228907062208242615815333248248585990826118860266
687885777579709848922926092710821763114335079319936729459258839856293115
14143607457603297458439759594085898425992,
319858426364986859453309057265394989014436949557363320736397444663890393
731436189205111222888442828494072902058049916341678164174687034592291388
913481151919213952783366956842104371306813379716860080480543404996547213
177212412399907010996852072534766429315865633636381416360119412689629996
411302638281515384891392546250993301995575031536800893875388635744801348
982113112522274638708389477774793099281957912410051274458216716846072377
068493083729233727955737320003650728151121195337026146203252381838992661
476821938928663306780769251996745545690181031642282787421517788323194061
35513140669049734660019551179692615505961)
f = getPrime(1024)
g = getPrime(768)

K=2**0

e=math.e
pi=math.pi

```

```

n = 2
det = q

# 行列式的值(赫米特)
temp1=gmpy2.iroot(n,2)[0] * gmpy2.iroot(det,n)[0]
print(temp1.bit_length())

# 高斯
# temp = (n/(2*math.e*math.pi))**(0.5)
# a= gmpy2.iroot(det,n)[0]
# temp1=int(temp*a)
# print(temp1.bit_length())

# 向量大小
temp2 = gmpy2.iroot(f**2+g**2,2)[0]
temp2=int(temp2)
print(temp2.bit_length())
if temp1.bit_length() >= temp2.bit_length():
    print("true")
else:
    print("false")

```

可以得到解题代码为

```

from sage.all import *
from Crypto.Util.number import *
from gmpy2 import *

h,p=
(89164527228214184632487268257212570217441942868747069158324446317715966
161164917750914731427988672785985864826783876689867644612651311191645004
737199398943431634963255563401814296759376414959813538577246270818473042
469870743037226421729888641389674040242012460503871528540017467631044177
732144089068793669587297442596127772575423515015920194837456218248947900
966392057714215602951756331528776677200383961545716978613268214831708352
380928797472975066069833228907062208242615815333248248585990826118860266
687885777579709848922926092710821763114335079319936729459258839856293115
14143607457603297458439759594085898425992,
319858426364986859453309057265394989014436949557363320736397444663890393

```

```
731436189205111222888442828494072902058049916341678164174687034592291388
913481151919213952783366956842104371306813379716860080480543404996547213
177212412399907010996852072534766429315865633636381416360119412689629996
411302638281515384891392546250993301995575031536800893875388635744801348
982113112522274638708389477774793099281957912410051274458216716846072377
068493083729233727955737320003650728151121195337026146203252381838992661
476821938928663306780769251996745545690181031642282787421517788323194061
35513140669049734660019551179692615505961)
```

c

```
=20041713613876382007969284056698149007154248857420752520496829246324512
197188211029665990713599667984019715503486507126224558092176392282486689
347953069815123212779090783909545244160318938357529307482025697769394114
967028564546355310883670462197528011181768588878447856875173263800885048
676190978206851268887445527785387532167370943745180538168965461612097037
041570912365648125449804109299630958840398397721916860876687808474004391
843869813396858468730877627733234832744328768443830669469345926766882446
378765847334421595034470639171397587395341977453536859946410431252287203
312913117023084978959318406160721042580688
```

```
def decrypt(f, g, c):
    a = c * f % p
    return a * pow(f, -1, g) % g

M = matrix([[1, h], [0, p]])
L = M.LLL()
shortest_vector = L[0]
if shortest_vector < 0:
    shortest_vector = -shortest_vector
f, g = shortest_vector
m = decrypt(f, g, c)
print(m)
print(long_to_bytes(int(m)))
```

#

```
240545625414656445795697416299836828697587638044418742943136404284040669
983557024929358783705357829768985339005
# flag{Lattice_reduction_magic_on_NTRU#82b08b2d}
```

coppersmith

定理

设 N 是一个大整数, $p \geq N^\beta$ 是 N 的一个因子, 设 $f(x)$ 是一个 d 阶(次数为 d)的多项式, 令 $X = \frac{1}{2}N^{\frac{1}{d}-\epsilon}$, 其中 $0 \leq \epsilon \leq \min\{0.18, \frac{1}{d}\}$, 那么对于给定的 $(N, f(x))$, 有

- 在模 N 意义下, 可以快速求出 $f(x) = 0$ 满足 $|x| \leq X$ 的全体正整数解 (就是求出 X 以内的根)
- 给定 β , 可以快速求出模 p 意义下较小的根 (就是求模因子意义下的根, 可以解决低位丢失类题目)

原理

有了这个定理, 相信大家一定好奇为什么, 为什么可以这么做, 但是这个原理挺复杂的, 下面会简单讲一下我的理解, 不一定对, 建议直接看截图

就是把一个多项式看成一个多维向量, 再构造成一个格, 之后再运用LLL格基规约找到一个范数最小的向量, 然后根据一个引理, 就可以找到这个多项式的一个根。

定理 1 (Coppersmith). N 是一个正整数, $F \in \mathbb{Z}[x]$ 是一个次数为 d 的首一多项式. 令 $X = \frac{1}{2}N^{1/d-\epsilon}$, 其中 $0 < \epsilon < \min\{0.18, 1/d\}$. 然后对于给定的 (N, F) , 我们可以高效地找出同余方程 $F(x) = 0 \pmod{N}$ 满足 $|x| < X$ 的全体正整数解. 求解的过程需要用到 LLL 格基归约算法.

引理 2 (Howgrave-Graham). 设 $h(x) \in \mathbb{Z}[x]$ 为次数为 d 的多项式, X 为正实数, 假设 $\|h(xX)\| < M(d+1)^{-1/2}$. 那么 $|x_0| < X, h(x_0) = 0 \pmod{M}$ 蕴含 $h(x_0) = 0$.

证明. 在题设的条件下:

$$\begin{aligned}
 |h(x_0)| &= \left| \sum_{i=0}^d a_i x_0^i \right| \\
 &\leq \sum_{i=0}^d |a_i| |x_0|^i = \sum_{i=0}^d |a_i X^i| (x_0/X)^i \\
 &\leq \sum_{i=0}^d |a_i X^i| \leq \left(\sum_{i=0}^d (d+1) |a_i X^i|^2 \right)^{1/2} \quad (\text{Cauchy 不等式}) \\
 &= (d+1)^{1/2} \|h(xX)\| < M
 \end{aligned} \tag{5}$$

故 $h(x_0) = 0 \pmod{M}$ 蕴含 $h(x_0) = 0$. □

small_roots

知道这么一个定理后我们期望在代码层面实现它, 这就要用到Sagemath里的small_roots这个函数, 下面会具体介绍这个函数, 首先放一个具体实现

```
R.<x> = PolynomialRing(Zmod(n), implementation='NTL')
P = p + x
low_p = P.monic().small_roots(X = 2**340, beta = 0.4)
print(low_p)
```

PolynomialRing：构造多项式环

Zmod(n)：模运算

implementation='NTL'：执行 NTL

small_roots(X=? , beta=? , epsilon=?)：计算多项式的小整数根，返回结果是一个列表

X：根的绝对上界，比如说下面那道题目，上界就是 2^{340}

beta：coopersmith里的一个参数，给定 β ，以快速求出模某个 p 意义下较小的根，其中 $p \geq n^\beta$ ，是 n 的因数， β 一般取0.4

epsilon：也是coppersmith里的一个参数，程序默认好像是 $\beta/8$ ，平时不太会用到这个参数，需要时加在beta后面就行

monic()：用于将多项式的首项系数归一化为1。它接受一个多项式作为参数，然后返回一个新的多项式，其中首项系数已经被归一化为1。这个过程可以简化多项式的表达式，使其更易于计算和分析。

这样简单的使用方法就了解了，可以看看下面的题目，自己运用一下

题目

m低位丢失

```
n=1311206182068564323966383116692832711957942583063245856880154440650676
946127959096277234024918356943755939420063552618369860458238576938115956
371082368941727447954962759609539862118299589145451695372202506892629351
250538312522757916977894663136996175358785634458225768367231323037860332
400533778891390243402343188706145436856610074761858259027038591820465615
608905351970953600190696400863570851067255021954689400609148352035543609
105386631271843131849878363771277387842377746731660586551624817624878063
713261580788627202984377018683342579204910818748733823785080620372821737
4848799250419859646871057096297020670904211
e=3
```

```
m=random.getrandbits(512)
```

```
c=pow(m,e,n)=15987554724003100295326076036413163634398600947695096857803
937998969441763014731720375196104010794555868069024393647966040593258267
888463732184495020709457560043050577198988363754703741636088089472488971
050324654162166657678376557110492703712286306868843728466224887550827162
442026262163340935333721705267432790268517
```

```
((m>>72)
<<72)=251918859427175920575786448609760554013540750157107862723884944356
1219057751843170540261842677239681908736
```

```
long_to_bytes(m).encode('hex')=
```

$$c = m^3 \mod n$$

11101表示29，右移3位得到11，再左移三位得到11000，表示24，缺少了101

$$m = m_{high} + x$$

$$c = (m_{high} + x)^3 \mod n$$

$$(m_{high} + x)^3 - c = 0 \mod n$$

```
from gmpy2 import *
from Crypto.Util.number import *
```

```
n=1311206182068564323966383116692832711957942583063245856880154440650676
946127959096277234024918356943755939420063552618369860458238576938115956
371082368941727447954962759609539862118299589145451695372202506892629351
250538312522757916977894663136996175358785634458225768367231323037860332
400533778891390243402343188706145436856610074761858259027038591820465615
608905351970953600190696400863570851067255021954689400609148352035543609
105386631271843131849878363771277387842377746731660586551624817624878063
713261580788627202984377018683342579204910818748733823785080620372821737
4848799250419859646871057096297020670904211
e=3
```

```
c=1598755472400310029532607603641316363439860094769509685780393799896944
176301473172037519610401079455586806902439364796604059325826788846373218
449502070945756004305057719898836375470374163608808947248897105032465416
216665767837655711049270371228630686884372846622488755082716244202626216
```

```
3340935333721705267432790268517
```

```
m1 =  
251918859427175920575786448609760554013540750157107862723884944356121905  
7751843170540261842677239681908736
```

```
R.<x> = PolynomialRing(Zmod(n), implementation='NTL')  
f = (m1+x)^3-c
```

```
m_low = f.monic().small_roots(X = 2**72, beta = 0.5, epsilon=0.02)[0]
```

```
print(m_low)  
print(long_to_bytes(int(m1+int(m_low))))
```

p低位丢失

```
n=1278462572903278959276662520307401810135491775149295268508380882550422  
181684731091044753213361695426227120587765125559899530563919432960749304  
794121275452387940274406507618377845264060262524285118409554610020056511  
301669016105380895038445899688157426657399252635795450749139797827860410  
252473139305930347635016773823782264724642583648253315002592305154443133  
050252204383387258048314259457180218932159901672574126025417079339377729  
314501052568656190442761364818484361930124141426434305736819241655113440  
410038615575129742461625469704104385185208107130621946299196984912366824  
8321130382231769250865190227630009181759219
```

```
e=65537
```

```
m=random.getrandbits(512)
```

```
c=pow(m,e,n)=62782408615711924505647887580059895955377425016167078750608  
325396078823073758876178738568612582876566561756788790422803083953531798  
758960876153450000312824716423377479478423151821280427005640456571042661  
393826430299801542115339387972926355129202454375642270295647002295953722  
126917208461908136849869393055045615354362817030632420626621634838670700  
866112871743142623748651130976728617551823862023050720195286726128388098  
686875267654961395878528891498942922458284921839547167229541003685888183  
636336488516427698323731223583159185804490836937685548412761493354595554  
4787160352042318378588039587911741028067576722790778
```

```
((p>>128)  
<<128)=97522826022187678545924975588711975512906538181361325096919121233
```

```
043973599759518562689050415761485716705615149641768982838255403594331293
651224395590747133152128042950062103156564440155088882592644046069208405
360324372057140890317518802130081198060093576841538008960560391380395697
098964411821716664506908672
```

$$p = p_{high} + x$$

$$p = 0 \pmod{p}$$

$$p_{high} + x = 0 \pmod{p}$$

```
from gmpy2 import *
from Crypto.Util.number import *
```

```
n=1278462572903278959276662520307401810135491775149295268508380882550422
181684731091044753213361695426227120587765125559899530563919432960749304
794121275452387940274406507618377845264060262524285118409554610020056511
301669016105380895038445899688157426657399252635795450749139797827860410
252473139305930347635016773823782264724642583648253315002592305154443133
050252204383387258048314259457180218932159901672574126025417079339377729
314501052568656190442761364818484361930124141426434305736819241655113440
410038615575129742461625469704104385185208107130621946299196984912366824
8321130382231769250865190227630009181759219
```

```
e=65537
```

```
c=6278240861571192450564788758005989595537742501616707875060832539607882
307375887617873856861258287656656175678879042280308395353179875896087615
345000031282471642337747947842315182128042700564045657104266139382643029
980154211533938797292635512920245437564227029564700229595372212691720846
190813684986939305504561535436281703063242062662163483867070086611287174
314262374865113097672861755182386202305072019528672612838809868687526765
496139587852889149894292245828492183954716722954100368588818363633648851
642769832373122358315918580449083693768554841276149335459555447871603520
42318378588039587911741028067576722790778
```

```
p1=975228260221876785459249755887119755129065381813613250969191212330439
735997595185626890504157614857167056151496417689828382554035943312936512
243955907471331521280429500621031565644401550888825926440460692084053603
243720571408903175188021300811980600935768415380089605603913803956970989
64411821716664506908672
```

```
R.<x> = PolynomialRing(Zmod(n), implementation='NTL')
```

```
f = p1+x

p_d = f.monic().small_roots(X = 2**128, beta = 0.5,epsilon=0.02)[0]
print(p1+p_d)
p = int(p1+p_d)
q = n//p

phi_n = (p - 1) * (q - 1)
d = inverse(e, phi_n)
m = pow(c, d, n)

print(long_to_bytes(int(m)))
```