2023DASCTF&0X401题目复现 by crumbling

目录

7月赛 ezAlgebra

2023DASCTF&0X401 WriteUp (qq.com)涉及到groebner_basis 一个没怎么见过的东西，打算来复现一下

(18条消息) Gröbner基的简单介绍与一些参考文献*grobner基*RayLee23333的博客-CSDN博客里面还有提到一篇论文——**Gr¨obner Bases: a Tutorial**。

[DASCTF 2023 & 0X401七月暑期挑战赛] crypto 石氏是时试的博客-CSDN博客还参考了这个

源码：

```
from Crypto.Util.number import getPrime, bytes_to_long

def YiJiuJiuQiNian(Wo, Xue, Hui, Le, Kai):
    Qi = 1997
    Che = Wo+Hui if Le==1 else Wo*Hui
    while(Xue):
        Qi += (pow(Che, Xue, Kai)) % Kai
        Xue -= 1
    return Qi

l = 512
m = bytes_to_long(flag)
p = getPrime(l)
q = getPrime(l//2)
r = getPrime(l//2)
n = p * q * r
t = getrandbits(32)
c1 = YiJiuJiuQiNian(t, 4, p, 1, n)
c2 = YiJiuJiuQiNian(m, 19, t, 0, q)
c3 = YiJiuJiuQiNian(m, 19, t, 1, q)
print(f"n = {n}")
print(f"c1 = {c1}")
print(f"c2 = {c2}")
print(f"c3 = {c3}")
"""
n =
11915614484595600476950747808532507941419024878065406084025786947796514030472708
86853165794450172145761820103735482734741217277789235825448532935349968053407953
55149795694121455249972628980952137874014208209750135683003125079012121116063371
90298570690748298868789581378898027589680446128540377903650889759210 3
```

```
c1 =
185012145382155564763088060801282407144264652101028110644849089283749320447842262397065972319766119386744305208284231153853897076876529326779092899879401876069911627013491974285327376378421323298147156687497709488102574369005495618201253946225697404932436143348932178069698091761601958275626264379615139864425
c2 =
722022978284031841958768129010024257235769706227005483829360633993196299360813
c3 =
999691052172645326792013409327337026208773437618455136594949462410165608463231
"""
```

第一部分：

c1 = 1997 + (p+t)^4 + (p+t)^3 +(p+t)^2+(p+t) mod n
c1 = 1997 + (p+t)^4 + (p+t)^3 +(p+t)^2+(p+t) mod p
c1 = 1997 + t^4 + t^3 + t^2 + t mod p 该式显然对模n成立
c1 = 1997 + t^4 + t^3 + t^2 + t mod n

故在模n下求一元coppersmith可以解得t

将t代入f，由上方第三式可知，此时f=k*p，所以可以获得p

```
n =
119156144845956004769507478085325079414190248780654060840257869477965140304727088685316579445017214576182010373548273474121727778923582544853293534996805340795355149795694121455249972628980952137874014208209750135683003125079012121116063371902985706907482988687895813788980275896804461285403779036508897592103
c1 =
185012145382155564763088060801282407144264652101028110644849089283749320447842262397065972319766119386744305208284231153853897076876529326779092899879401876069911627013491974285327376378421323298147156687497709488102574369005495618201253946225697404932436143348932178069698091761601958275626264379615139864425
PR.<x>=PolynomialRing(Zmod(n))
f=x^4+x^3+x^2+x+1997-c1
t=f.small_roots(X=2^32,beta=0.4)[0]
#t=2915836867
f= t^4+t^3+t^2+t-c1+1997
p=gcd(f,n)
print(p)
#p=126740450653809639363690066409137072060921652543723275475752875891165333430054566156353880486838286850308601535313907069457090865185109269806442759157264 13
```

第二部分：

论文对于求解环上方程组，先举了多元变量高斯消去法的例子和单变量欧几里得算法的例子，然后这个什么**Gr¨obner bases**是基于这俩者的关键特征提取出来的计算全局基算法。

[Gröbner basis与多元多项式方程组 - 知乎 (zhihu.com)]

多元环上取余运算的化简会用到Gr¨obner bases

简单看了一圈，大概意思就是对于一个多元多项式方程组，可以通过求Gr¨obner bases的方式，找到一个类似于线性代数中基础解系（和高斯消去法也是类似的原理，高斯消去法应该就是线性代数中将（A|b）化为行阶梯，只是该方法应用范围更广）的东西，然后求出来的这个基各个值为0，以此求得变量的值。

che=mt

c2-1997=che^{19+che}18 +....+che mod q

c2-1997=che^{19+che}18 +....+che mod (n//p)

che=(m+t)

c3-1997=che^{19+che}18+....+che mod q

c3-1997=che^{19+che}18+....+che mod (n//p)

t=2915836867

所以这里就是求解一个模n//p下3个方程的二元多项式方程组(有wp说直接模n不行，但是还是出来了)

```
c2 =
722022978284031841958768129010024257235769706227005483829360633993196299360813
c3 =
999691052172645326792013409327337026208773437618455136594949462410165608463231
P.<m,m0> = PolynomialRing(Zmod(n))#m0没用，只是1个变量后面会报错
f1=t-2915836867
f2=1997-c2
f3=1997-c3
for i in range(1,20):
    f2+=(m*t)^i
    f3+=(m+t)^i
F=[f1,f2,f3]
B=Ideal(F).groebner_basis()
#B=[m +
211587317163762260903924988419156601191512491515782936340827499896593072250470654545625564907947202412518312942692482529928287824283168341668284048761814918748717871446648499846061142498203381902520504912230662739537775067055364808444751419338486181133434153758670666175128055758690136945230345731112591142 74,
870380690328400520055209082722377889081690435802210407111494940839757434789 69]
q=B[1]
r=n//int(q)//int(p)
```

然后在模q下f2，f3=0，gcd可以出q

```
m=B[0].constant_coefficient()#wp看到的，这函数干嘛的,应该是求解这个全局基用的，但是按我的理解这里的m不应该是-211...吗为什么这里出来是正的，看wp那边也是手动加了负号，emmm迟点再查查
m=(-m)%q
#m直接出来东西不对。因为是在模q下，真m值可能大于q,看都是爆破的，前面在模q下解copper试了一下也没出来东西,那我也来爆一个
for i in range(10000000):
    flag=long_to_bytes(int(m)+i*int(q))
    if b"dasctf{" in flag :
        print(flag)
        break
```

dasctf{ShangPoXiaPoYaSiLeYiQianDuo}

出来了。另外有个求出q后不继续求解groebner_basis，用其他方式求m的。

想明白了，直接贴一个吧。

```
P.<x> = PolynomialRing(Zmod(q))
f1=1997-c2
f2=1997-c3
for i in range(1,20):
    f1+=(x*t)^i
    f2+=(x+t)^i
print(-gcd(f1,f2)[0])
```

简单来说就是gcd的结果确实是q，但是是x+int的形式，所以有x+int=q这就解出来了x也就是上面那个m

小结：1.老方法，利用模数的“变化”（指p变n n变q这种）来调整式子，以方便某些求解。2.学到了个新东西Gr¨obner Bases，不过学的很粗浅大概意思就是对于一个多元多项式方程组，可以通过求Gr¨obner bases的方式来求解变量，不太明白更具体的使用范围和条件。

7月赛ezRSA

DASCTF 2023 & 0X401七月暑期挑战赛 Writeup - 星盟安全团队 (xmcve.com)深度搜索 来再玩一下

源码：

```
from Crypto.Util.number import *
from secret import secret, flag
def encrypt(m):
    return pow(m, e, n)
assert flag == b"dasctf{" + secret + b"}"
e = 11
p = getPrime(512)
q = getPrime(512)
n = p * q
P = getPrime(512)
Q = getPrime(512)
N = P * Q
gift = P ^ (Q >> 16)
print(N, gift, pow(n, e, N))
print(encrypt(bytes_to_long(secret)),
    encrypt(bytes_to_long(flag)))
```

```
#
75000029602085996700582008490482326525611947919932949726582734167668021800854674
61607429710996207804843571467208845293930077626878888801612563208452941923003843
67387615509069066710103129308017510000222003608570893382310020887304712772773192
53053479367509575754258003761447489654232217266317081318035524086377
80067306155754013504701756014635184816853961140032902991314690012426363697478558
17476589805833427855228149768949773065563676033514362512835553274555294034
14183763184495367653522884147951054630177015952745593358354098952173965560488104
21351756309867602851654191585575406671947548750334891418167492907247223844985308
21180648238353223136807058894323134199767386943175948430460014488555759864133381
42129464525633835911168202553914150009081557835620953018542067857943
#
69307306970629523181683439240748426263979206546157895088924929426911355406769672
38598482978480467382164397678092802420909236009267045797815430940259114568982557
12095158684356087539238700436478928165746846639934157964650740273694077990099293
34083395577490711236614662941070610575313972839165233651342137645009
46997465834324781573963709865566777091686340553483507705539161842460528999282057
88036225941665401285423773952727744859975580561462253182725713695966403509820920
61102908794827260831910051649612001252969994495987662014350570916242252183515372
78712880859703730566080874333989361396420522357001928540408351500991
```

求解q,p部分。

用的是星盟wp里深搜的方法。

几个复现时候注意到的小点：

1. 从高位搜会比低位搜代码难看一点，但应该也是可行的
2. 整个代码比较整齐，主要通过两个限制条件快速筛去大部分不可能情况（仅保证gift这一个条件会因为范围小出不来，当时做题的时候有人说只保证了pq乘积的高位与n高位相等，应该也是条件给少了一个）
3. 因为gift是p异或q右移16位的结果，所以p的最后一位1相当于异或了q的第十七位。这也就是为什么只搜p而不是同时搜p，q，传入的也不是q的末位1而是q的末17位，在调用函数的时候才会有爆破了q后17位的操作。

```
N=75000029602085996700582008490482326525611947919932949726582734167668021800854674
74616074297109962078048435714672088452939300776268788888016125632084529419230038
43673876155090690667101031293080175100002220036085708933823100208873047127727731
92530534793675095757542580037614474896542322172663170813180355240863 77
gift=80067306155754013504701756014635184816853961140032902991314690012426363697478
78558174765898058334278552281497689497730655636760335143625128355532745552940 34
c1=14183763184495367653522884147951054630177015952745593358354098952173965560488
10421351756309867602851654191585575406671947548750334891418167492907247223844985
30821180648238353223136807058894323134199767386943175948430460014488555759864133
38142129464525633835911168202553914150009081557835620953018542067857943

def findp(p,q):
    if len(p)==512:
        p1=int(p,2)
        if N % p1 ==0:
            print(p1,N//p1)
    else:
        bit=len(p)
        p1=int(p,2)
```

```
        q1=int(q,2)
        if (p1^(q1>>16))%(2**bit)==gift%(2**bit) and p1*q1%(2**bit)==N%
(2**bit):#当目前深搜出来的位数符合实际，继续搜索。
            findp('1'+p,'1'+q)
            findp('0'+p,'1'+q)
            findp('0'+p,'0'+q)
            findp('1'+p,'0'+q)


for i in range(2**17):
    findp('1',bin(i)[2:])#else会返回none 所以没办法赋值。
```

下一步，有叫关联信息攻击的，也有叫明文线性攻击的

找了一下，叫Franklin-Reiter相关消息攻击

Franklin-Reiter相关消息攻击_Emmaaaaaaaaaa的博客-CSDN博客~~别吐槽为什么又是csdn了，我的bing就是出来了这些，我又懒得多走一步geogle~~

~~简单概括一下~~

说的挺清晰的，不概括了：

Related Message Attack ¶

攻击条件

当 Alice 使用同一公钥对两个具有某种线性关系的消息 M1 与 M2 进行加密，并将加密后的消息 C1，C2 发送给了 Bob 时，我们就可能可以获得对应的消息 M1 与 M2。这里我们假设模数为 N，两者之间的线性关系如下

$$M_1 \equiv f(M_2) \bmod N$$

其中 f 为一个线性函数，比如说 $f = ax + b$。

在具有较小错误概率下的情况下，其复杂度为 $O(elog^2N)$。

这一攻击由 Franklin，Reiter 提出。

攻击原理

首先，我们知道 $C_1 \equiv M_1^e \bmod N$，并且 $M_1 \equiv f(M_2) \bmod N$，那么我们可以知道 M_2 是 $f(x)^e \equiv C_1 \bmod N$ 的一个解，即它是方程 $f(x)^e - C_1$ 在模 N 意义下的一个根。同样的，M_2 是 $x^e - C_2$ 在模 N 意义下的一个根。所以说 $x - M_2$ 同时整除以上两个多项式。因此，我们可以求得两个多项式的最大公因子，如果最大公因子恰好是线性的话，那么我们就求得了 M_2。需要注意的是，在 $e = 3$ 的情况下，最大公因子一定是线性的。

```
def GCD(a,b):
    if b==0:
        return a.monic()#a.monic()是一个多项式a的方法，用于将多项式转化为首项系数为1的首一多项式（monic polynomial）
    else:
        return GCD(b,a%b)
PR.<x>=PolynomialRing(Zmod(n))
f1=x^e-c2
for i in range(50):
    f2=(bytes_to_long(b"dasctf{"+b'\x00'*i+b"}")+256*x)^e-c3
    res=GCD(f1,f2)
    if res[0]!=1:
        print(long_to_bytes(int(-res[0]%n)))
```

另外，根据上一题的复现经验，也来试试groebner_basis的求解

其实之前试过但没有出来，原因在于爆破flag长度的时候使用了*256*i*，这里应该使用*b'\x00'*i*转long

另外当时的n没有进行+N处理，如果仔细检查可以发现n可分解而且最后一个数是2，加了N后就不可分解了，这手真是坑啊，如果临场做到爆破结束没出结果肯定会很懵逼吧草。
~~到时候是不是也可以坑一手新人~~（坏笑

~~顺带吐槽这个sage在我没有给n套int的时候n+N没有成功，害我浪费了不少时间，原理是用pow函数的时候出来的n是在环N上的，所以加了等于没加，超~~

总之以下是exp，flag需要肉眼识别一下：

```
from gmpy2 import *
from sage.all import *
from Crypto.Util.number import*
N=75000029602085996700582008490482326525611947919932949726582734167668021800854674616074297109962078048435714672088452939300776268788888016125632084529419230038436738761550906906671010312930801751000022200360857089338231002088730471277277319253053479367509575754258003761447489654232217266317081318035524086377
c1=14183763184495367653522884147951054630177015952745593358354098952173965560488104213517563098676028516541915855754066719475487503348914181674929072472238449853082118064823835322313680705889432313419976738694317594843046001448855575986413338142129464525633835911168202553914150009081557835620953018542067857943
c2=69307306970629523181683439240748426263979206546157895088924929426911355406769672385984829784804673821643976780928024209092360092670457978154309402591145689825571209515868435608753923870043647892816574684663993415796465074027369407799009929334083395577490711236614662941070610575313972839165233651342137645009
c3=46997465834324781573963709865566777091686340553483507705539161842460528999282057880362259416654012854237739527277448599755805614622531827257136959664035098209206110290879482726083191005164961200125296999449598766201435057091624225218351537278712880859703730566080874333989361396420522357001928540408351500991
e=11
P=80068471719125770690851668777586269543048247561387582665577063916629878060651324485441178400314997079382279550941097797326090353102527230664703308626226412
Q=93669865293770697833946258489201069512201341115483432653116771639921695554364215697307032911287714728858652887983440380009849119218430882009977253246822972
phi=(Q-1)*(P-1)
d=gmpy2.invert(e,phi)
n=pow(c1,d,N)
n=int(n)+N
```

```
def getflag(a,e,n,c2,c3):
    P.< m,m0 >= PolynomialRing(Zmod(n))
    f1 = m ^ e - c2
    f2 = (a+256*m) ^ e - c3
    F=[f1,f2]
    B=Ideal(F).groebner_basis()
    res=[i.constant_coefficient() for i in B]
    #print(B)
    print(long_to_bytes(int(-res[0]%n)))

for i in range(50):
    getflag(bytes_to_long(b"dasctf{"+b'\x00'*i+b"}"),e,n,c2,c3)
```

最后，关于第一部分，再来复现一下当时有人提到的另一个思路的做法（也在wp中有看到）