2023WMCTF部分题目复现 by crumbling

总感觉先贴参考有点乱，这次我把参考的博客啥的都扔最后面。

badprime涉及了一个漏洞 干脆单独拿出来了。

目录

signin

源码：

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

def pr(msg):
    print(msg)

pr(br"""
                        ....''''''....
                    .`",:;;II;II;;;;:,"^'.
                  '"IlllI;;;;;;;;;;;;;Il!!l;^.
                `l><>!!!!!!!!iiiii!!!!!!!!i><!".
             ':>?]__++~~~~~<<<<<<<<<<<<<<<~~+__i".
           .:i+}{]?-__+++~~~~~~<<<<<~~~~~~+_-?[\1_!^
          .;<_}\{]-_++~<<<<<<<<<<<<<<<<<<~+-?]\|]+<^
          .!-{t|[?-}(|(){_<<<<<<<<<_}1)))1}??]{t|]_"
           !)nf}]-?/\){]]]_<<<<<<<<<_]]}}{\/?-][)vf?`
          '!tX/}]--<]{\Un[~~<<<<<~~<~-11Yz)<--?[{vv[".
         .<{xJt}]?!ibmO%&Ci><<<<<<<<!OkJW%w+:-?[{uu)},
          !1fLf}]_::xmqQj["I~<<<<<<>"(ZqOu{I^<?[{cc)[`
          `}|x\}]_+<!<+~<<__~<<<<<<+_<<_+<><++-[1j/(>
           !\j/{]-++___--_+~~<i;I>~~~__-_______?}(jf}`
            ;~(|}?_++++~~++~+]-++]?+++~~~~+++-[1/]>^
              ;\([?__+_-?]?-_-----__-]?-_+++-]{/].
               l||}?__/rjffcCQQQQQLUxffjf}+-]1\?'
                ,[\)[?}}-__[/nzXXvj)?__]{??}((>.
                 .I[|(1{]_+~~~<~~<<<~+_[}1(1+^
                    ,~{|\)}]_++++++-?}1)1?!`
                      ."!_]{11))1{}]-+i:'
                          .`^","^`'.
""".decode())

def gen_prime(bit):
    while 1:
        P = getPrime(bit)
        if len(bin(P)) - 2 == bit:
            return P
```

```
pq_bit = 512
offset = 16

P,Q = [gen_prime(pq_bit) for i in range(2)]
N = P * Q
gift = int(bin(P ^ (Q >> offset))[2+offset:],2)
pr(N)
pr(gift)

inpP = int(input())
if inpP != P:
    pr(b"you lose!")
    exit()

secret = randrange(0,P)
bs = [randrange(0,P) for _ in range(38)]

results = [(bi * secret) % P for bi in bs]
rs = [ri & (2 ** offset - 1)  for ri in results]

pr(bs)
pr(rs)
inpsecret = int(input())
if inpsecret == secret:
    pr(flag)
```

第一部分

和前两天七月赛类似的问题，比赛的时候改了深搜的写法但是没有出来结果（深搜部分+cooper；深搜+爆破gift高位，前一个2小时没出东西，后一个代码没跑完）。

解法一：来自学长tr0uble。在确定p的高位后通过N//p求出p的高x位，然后再利用gift异或求p接下来的x位，不停重复（考虑进位x选择一个小于16的数）。不过因为这次无法直接从gift获得p的高16位，所以爆破p的高16位。这种解法显然效率比深搜高。~~七月赛那题只复现了深搜简直是大伏笔啊啊啊啊啊啊~~

以下是核心部分：

```
xbits=8
for p_high in range(1,2**16):
    p_highbit=bin(p_high)[2:].zfill(16)
    for i in range((512-16)//xbits):
        p0=int(p_highbit.ljust(512,'0'),2)
        qbar=N//p0
        q_highbit=bin(qbar)[2:]
        left=i*xbits
        right=left+xbits
        p_xbits=bin(int(q_highbit[left:right],2)^int(gift[left:right], 2))
[2:].zfill(xbits)
        p_highbit+=p_xbits
    p=int(p_highbit,2)
    q=N//p
    if p*q==N:
        print(p)
```

```
        break
```

x选择8还挺巧妙的，七月赛wp中有见过选择10还是5的，会导致留下几位需要爆破，8是刚刚好求出p，同理x=4也一样.

ljust函数用于右填充0直到512位，并且左对齐，和zfill刚好相反

解法二：来自石氏是时试。思路是深搜，但搜法是爆破phigh然后从高位进行深搜。

gift的条件用在了 `if v=='0'`,一次淘汰一半；

N的条件用在了`if tp*tq>N` `if (tp+(1<<(l+1)))*(tq+(1<<(l+17)))<N`

和低位搜比可以减少一个2**15的爆破，从O(n^2)到了O（n）。所以在较短时间内出的来结果，不过代价是fac函数里面打了不少“补丁”。

同样重新整合了一份核心代码：

```
def fac(allgift,tp,tq):
    global p
    if p != 0:
        return
    if len(allgift) == 0:
        return
    if tp*tq>N:
        return
    if N%(tp+1)==0:
        print(tp+1)
        p=tp+1
        return

    prebit = allgift[0]
    allgift0 = allgift[1:]
    l = len(allgift0)

    if (tp+(1<<(l+1)))*(tq+(1<<(l+17)))<N:
        return

    if prebit == '0':
       fac(allgift0, tp, tq)
       fac(allgift0, tp+(1<<l), tq+(1<<(l+16)))
    else:
       fac(allgift0, tp+(1<<l), tq)
       fac(allgift0, tp, tq+(1<<(l+16)))


N =
98979914203938129582559002435937496221944524872097792710858591458834482668375238
15337596773223651982800905263471215515435069079329056893475756763224458203730285
34876547844277034896855935249356437309622659779180607498859613771088338706860123
25836049467328687008560530162352822066119244817994253171179069223963
gift =
15569459691529899157271283568011948613756283529934606272457685819646703573131042
62909304250574318310707440192203879575761983592840210462045713094747 34
```

```
offset = 16
allgift = bin(gift)[2:].zfill(512-offset)
p = 0
for phigh in range(2**15):
    tp = (1<<511)+ (phigh<<512-offset-1)
    tq = 1<<(511)
    fac(allgift,tp,tq)
    if p != 0:
        break
```

深搜还真是深奥orz

解法三：官方wp。

第二部分

hnp，但是改过一点的hnp

看到了新的wp，可以复现构造的矩阵了。

$r_i=(b_i*s)\%p \ \%2^{16}$

$r_i=b_i*s+k*p+l*2^{16}$

$l*2^{16}=b_i*s-r_i \pmod p$

inv=inverse(2^{16},p)

$l=b_i*s*inv-r_i*inv+k*p$

大小:r_i——16bits；b_i,s,p——512bits;l——（512-16）bits

构造：

$$[k_0 \quad k_1 \quad \dots k_n \quad secret \quad -1] * \begin{bmatrix} p & & & & \\ & p & & & \\ & & \dots & & \\ tmpb0 & tmpb1 & \dots & K/p & 0 \\ tmpc0 & tmpc1 & \dots & 0 & K \end{bmatrix} = [l_0 \quad l_1 \quad \dots l_n \quad K*secret/p \quad -K]$$

（为了让博客正常显示不小心把原版的给删掉了，现在只有图片了，凑合用吧，只可惜下次得重写一遍）

其中

n=37,K=2^{496}(因为l为496位，对于由512位的p决定大小的格而言较小，若K过大会导致不是最短向量)

`tmpb = [int(bi * inverse_mod(2**16,p)) for bi in bs]`

`tmpc = [int(ri * inverse_mod(2**16,p))  for ri in rs]`

```
from gmpy2 import *
```

```
p =
9463395021022080495725625579099709864198202996192818493676075430361086175809577174253865589866353281287908307347544682931439681148579311956298173287376473
```

```
b =
[6099745272052586004179912608738971034534930536137743613897081917185107394368705
591323971750395506839750452649288267772188419489756675205679949408086451232,
39511917478127290454402428208951246071894802510951632952426284509429987523649736
01131023646206377502005591988554681151953526704565202075013281769088815523,
14044205975544040301072727709222539966781623336873521956182512188639998502488246
92838151822875031075053556888677319712477280556217652901167451648905364386,
44882945723336567082594203775397375050039961596776564680260971146017116079855670
15550450770914323371829296766770532559711193361180597308950367687185966302,
54811023221874794194365058290748656466840953273651951502224674184428734653574874
02747218531517017371054667814520443813042860956594726076176855372054132653,
27138027881336982694092499995362004191403141211304732586562060524290021709517416
96862581935955915442467962543363756219468741646383480138223283730677285687,
43884714189378788737602443112269311023119677611395973012275950954540374950669608
91301104963760391091058605030114648033892461656189445282496553583505973028,
73677646473164157578183929240412485128532450051359367552887242371151478799600124
1149637426869001948983773230073266488914216375314221965655672656410584443,
87085909892373253418649696422667210929081503228628078835013070224470921274655072
99112990850265948137001397701186114982614486130822490540038423320215334626,
92678023044245483979606177365977236359368112516098462907617629036548046786289238
62108480264307805107896646269881938313148864202456071920121260093838052525,
32471081838603259873430600733251547800631210724125461764640759751525034930188893
36496636379292425449406827070404175667145192082945885628262842725864496476,
45572443563947317323057525062091931373771497892674474087116799256714051084765969
6368128186714204899204016766561731806278682654146614456839201295265351084,
90200400642394389573256520107325627034961533797762913864792493773360021299774989
01663356523568148111515751758815532962162768918028366620424504879498916260,
76884165800275827699151166620187013307315428536107280836384756810903888905857996
79692871117954618858316092856071130163834051800086038254809868956867017534,
29140818030714752107656077070045261896278799123433054361653468307331801117129276
83631299251265551199278425089831815911602268284636090898745079700939295508,
16824476244440591929447510833275579273455920865074206275670503130411031920414636
42408780131750529259046595170811376763889856062916108841799386014250209204,
53410346192474761237382046668316363787566032827095418575955278121390225100354770
00927339770989486054395218479620330803691178416464134942884723827374332572,
83763297021071338484581224421449460893409524128702835759888716944916092155839353
92751355281411100977914041577559011007450313560473364023276862308392837927,
94162637888451048432542956337550807170271807986619465503432730525738616929937567
45844265654941124801439244186152547374828735493445699134588163894749640836,
29322167387705378178815150939097084151257548156042999990681338487284256712417568
19969645781862996905460305910366082553247028095515273709817106865465122590,
80977176699265372507313056098738699634429896654047213031194922309212595874480451
70648745406003491170455200904721392690716080842205006420218957357208236777,
23200953724694123811230812418139691830592171830550925641656160400301264667416918
23966421813308525807455783827406201671916779545841711101790509143391460558,
23339721642693034804689822314309448442610588554278001720279329231318010327392738
32904738225066210544462847760672864166563796956687623202151756145595323299,
94375067110465801319627271296790573678421761590584081536727137038011234113054478
77847753662475828865148714651927615052959365575959980181945973888298104933,
58029617959456022939299592529892050609071829502091847920160065646851648290795223
33038011701596715377738492900250485584441351844045455427769773524087524156,
52959942793398423823147247617500489661242016920092656337110583575711504189061022
9232923121353193340603425988395343027602415343623433336040543795697317090,
64021963720346688630558773480659739219624225905165191369778666526009024863230810
42430853494022971845631884452544526687998575817840711058028440421779395606,
```

```
123062430787540524153459070558634603443360038074517864434186499728391823799833993391992594052371329938283840904699810099504995128038252625570702202421485 3,
493939975056347483169075135120862100653453849752574405673103339066149892344140719538630864738124645424110528677664557720243499961149500030240209878315114 2,
399185999804054213325904303634359258443636279023592376183396220998902445881922546029442233672172604882678804684982986406020798975004664462183558969900936 5,
240857736341741610087615111623321249370900668053282004036464835627792781358520219123448643072918609607097113721094276603510571775439377992094100498576888,
361608350239820289260188203816562800128999210345798935193269076922862748693402913242677453467965714413898926556464611762151354078101032441014851767482553 1,
540461289195287926449611210340581148462642410841104173704311066712226688363866076643281241454284177355938951023487311900597936468768971724167867687897257 2,
203445156489499245334287469788992492964086449721386681289752859490264669010468164478534651163056896079840540046650545193016061796990330817850453299774186 8,
615749030450526546591323157155541260690574804761810366242717489151000972945947582964001554608584576422627237718093979393216411169458045467203231658878822 6,
497596431709902418360747615505300559556361553406426297413183794991871160689169474051596524255673528429571754430802216945936594719560142694909420755784822,
542847688370651421977716714506584704207773652868372716444931217200530280533107386756510704275373246757362566935922531866345842741118931942430237903807105 1,
167191420550055367364797041014390951967159063695278735167235620744159356575436434360763569041839147336092609763256831779698473331704268584943023455481585 8]
r = [48997, 62415, 23955, 36908, 52443, 4523, 22645, 22555, 31815, 15691, 47858,
27532, 21464, 23465, 45849, 59181, 27490, 6614, 16702, 57463, 52700, 28969,
31173, 41233, 61893, 36368, 17734, 53549, 17913, 33308, 63024, 61345, 33511,
53005, 26113, 59084, 35720, 44204]

M = matrix(QQ,40,40)
inv = invert(2 ** 16,p)

for i in range(38):
    M[i,i] = p
    M[-2,i] = b[i] * inv
    M[-1,i] = -r[i] * inv

M[-2,-2] = 2 ** 496 / p
M[-1,-1] = 2 ** 496

L = M.LLL()

res = L[1][-2].numerator() / 2 ** 496
# 或 res = L[1][-2] / (2 ** 496 / p) % p
print(res)
#
1005444529226476196286726437221411001182466035947403146822894574200213482908472882296123424897230218596631139138335919912390102402492391521467426075919696
# wmctf{we1cOme_brOoOOo!hope_yOu_h4v3_fun_iN_the_fTcmWWmcTf/}
```

和官方wp稍有不同，tmpb与tmpc没有进行左移16操作，格的右下两个数据也稍有不同，但是对于hnp的介绍以及本题推导更清晰，所以还是用了（见最后一个参考博客）

[WMCTF 2023] crypto 石氏是时试的博客-CSDN博客

WMCTF2023 Crypto_无趣的浅的博客-CSDN博客

https://blog.wm-team.cn/index.php/archives/39/

WMCTF-2023-Crypto_Emmaaaaaaaaaa的博客-CSDN博客