2023WMCTF 题目复现 部分 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。

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，感觉很久没有接触过了，而且是改过一点的hnp

硬看exp，感觉暂时还复现不出来他的格和原理，~~前方的复现等以后再来吧~~等我再玩会儿别的格密码再回来复现。

*inverse_mod(2**16,p)*的操作应该是将数据从模p上，左移16应该是为了保证最短向量在格上吧。

这是官方wp中构建的矩阵：

$$[l_0 \quad l_1 \quad \dots l_{37} \quad secret \quad -1] * \begin{bmatrix} p*2^{16} & & & & \\ & p*2^{16} & & & \\ & & \dots & & \\ tmpb0 & tmpb1 & \dots & 1 & 0 \\ tmpc0 & tmpc1 & \dots & 0 & 2^{512} \end{bmatrix} = [\quad secret \quad 2^{512}]$$

其中

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

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

```
p
=109697642303329928812611624447181440627566477276121608129718563440492967390445402474980831600299879673191169169353428588713298496367204213329364679667758 67
```

```
bs =
[466479397234195318966655511533515567605473942236413501241187257199593018656410242669723164108952275034775363126757729811478211698786151371762098653705238285070660703301667079387625880734011912630343487559215508027023387081424940873925939506449834218123086788543426025852607221365590184377651133791637885064669395044983421812308678854342602585260722136559018437765113379163788506466,28300703995330512312418624261080947942289966102987995190144910902014187102901474079864508319387055540427399722992150146859528587681120945277105384035152731019472718931365397643166463952635100145886492082451048371018929638491018100156236229015358104949578406959304393492243075814133461928974713846806252456171180021300302579123737431444119515922647288269239385977007476609140589657731518846449702342402815477651223076713091123513394700434515165982140923425583848078992378190732605199337414271971061665013925087942269438591896597575610893291671195449984745067740479119096438873783304535686779740112967166240956658651830,219441723479845502737210510029565916805094786884462587128280961857833401639332897542126126236378617048764714610295838599905818629703666810284483625404293557389884863967391260146277674591022928486866404538488288575019736698857557997778014063008476773913962562050644302349190677684403825361742825597437864986433,529219322672869895500383902029828479088034784281683711243108019719041436883064946019177611785752115857048196479590741335588096632178120832015737453265865732362911077346664074870218459728403168333136177794566098857751561979881924095744245711390550227194626188940179976725099365400621385309105352145786623574861791279679876950406014873413755698272590003825746211539386722962275030323568985409399878270592069459030965256384719222528703580051044483771487910104714101023577692746670269590158012828373238096018525440022558445887700576685844568831578281023419638089675755565506578355924897222011592861425293647962328226884287075296283820197755800716200075064027479608647790939509666611647949784825089183786463297946121620221227282739788414188304450422136552805629818171807781989435101016155388320981686069320173122445065407658884121284250080193476126715916306363926890266521441209072392125152387026448128899855816078566448516652,100199634615109735686226511639075734476809483606419722756890604048211081454512021412403535771895420459194644844357302156659072589224248286985423455117420573704289763983510476816869859899336074151881695530276446910838716886376076917118273772330493711481512703599255459570222793869234767208753520021183441632401366016546135886969621987828562360884651859443016817174689540207831252024876805875502572479737347927230600821119715475408523076262208330090394633364213064,3018686464860030850959320579594895527678223791378490625855331881041828221928114456953089092669455959824003716530545508478387537003475350584348888269292369,364453993302904289896756496810944674930383845239978266100684672810479205724380134277408222980629029131484502808668454547446233449410785366503379893944077126195229708695923641988438207879876366524141164601546483295877053360655694760145305642789647746119139219128673595630137874358724718498685034753555738459335,261362314389476873976897558951337538378092134286347632234681992176562625438223548162096922286625463416020262461099400205549250922037779285889586320711811490515356321716440602193657426678870041151190163330221438626858833378204623116821050556166007466240546417178886627212440990375892435283361328587699242930176322215694815017710290130308304143067518883521417429269607649199684266970302719242998155799931358383625882532121756748894334198809061646979762067511472215,860101530244518023270928405689949860658313309915924443816915399533927072454769756717304953902087630866539900522094506431195461435822814711360863378493808316874658440024011728239600286438978619677030258659310252184717056189217255217534608083539111979295024060439740903737763326662957118983905569420273519615887030054198985918505215104013792879492290315721793748371422498525384391611872908723072277255873654142484830006969158725571070189294778192659597407886384386,438626233062860631396523386394507588723865798421924073678407577877627433159032251541094618091174355855386323589925517929135034734070543349490232574949313 2,
```

```
365376577820873710494780684428742243266494811649412320457459522942383085144903386604070441361876588812101416607501823124949603340852930845009715869767432877482376393583082496250220531096430172364928324823175327586492263996969749792537707490670682655093720156047019481749703800475679413455318150445298635274455059530711360442307499155801023000099897309611286878255756074194619615955048552735990659714305010701125554396886322209261008087604681515049107555729814797,41695216175637370346412712056628180884540283608383904492803311566295994334139872628057518030985220580698179788261375927325589729557360365631211881103222272679851325283585421172011583732454728086461602611900834744418413922900914192507355252821758458704927814607793565756967978874045645146257076017218453114251022498296101289605683679710305494303148483272469471317063526382258459200956159753952584275872493055946897112115489770408205515479017905107085781303456465710835217921679675137610225789921492245946084819290173650012084974942027217984717415920266806435543455936192663961817306783907655536400218855307984549922331,786372033673891941535662491107917857484755405324310617500720950802177111919790610188738706333929321254883123477282244417141631445443299902819038472761905478637203367389194153566249110791785748475540532431061750072095080217711191979061018873870633392932125488312347728224441714163144544329990281903847276190547963571593887612645315364859989332125003912261360022752028961094802906806642026065761338966772207051696192941136659677997491454214318463524532207232130187,74576486618741258913984524175113541834553629298721871691762223582883810403592577715066251301270248397660594215162069485734473240168314480156290791857108980552223815763689572351359880043337487833125217831283315257566985932571862594383003112520360481258894183827848713832267150793744284381677539439632823567868]
rs=[7892, 27695, 17262, 9427, 35390, 55927, 53600, 54840, 26665, 59149, 44463, 59305, 31986, 34927, 898, 38303, 39933, 8932, 47956, 47060, 7248, 49771, 59443, 43817, 12310, 21118, 33391, 49267, 44481, 63643, 26526, 17947, 19244, 15097, 28377, 49845, 50966, 42773]
secret =84154791220417185320974615154095790122534078843129423551804196025792232714148881088642784055378354786681040713044728454406673137057817542324039512026541134
cols = len(rs)
M = Matrix(ZZ,cols+2,cols+2)
tmpb = [int(bi * inverse_mod(2**16,p))<<16 for bi in bs]
tmpc = [int(ri * inverse_mod(2**16,p))<<16  for ri in rs]

for i in range(cols):
    M[i,i] = p<<16

M[-2] = tmpb + [1,0]
M[-1] = tmpc + [0,2^512]
ML = M.LLL()

for mi in ML:
    if abs(mi[-1]) == 2^512:
        if mi[-2] % p ==secret:
            print(mi)
        if abs(mi[-2]) % p == secret:
            print(mi)
        if abs(- mi[-2]) % p == secret:
            print(mi)
        if - mi[-2] % p == secret:
            print(mi)
```

badprime

源码：

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

M =
0x7cda79f57f60a9b65478052f383ad7dadb714b4f4ac069997c7ff23d34d075fca08fdf20f95fbc
5f0a981d65c3a3ee7ff74d769da52e948d6b027 0dd736ef61fa99a54f80fb22091b055885dc22b9f
17562778dfb2aeac87f51de339f71731d207c0af3244d35129feba028a48402247f4ba1d2b6d0755
baff6

def getMyprime(BIT):
    while True:
        p = int(pow(65537, getRandomRange(M>>1, M), M)) + getRandomInteger(BIT-
int(M).bit_length()) * M
        if isPrime(p):
            return p

p = getMyprime(1024)
q = getPrime(1024)
n = p * q
m = bytes_to_long(flag)

print("Try to crack the bad RSA")
print("Public key:", n)
print("The flag(encrypted):", pow(m, 65537, n))
print("Well well, I will give you the hint if you please me ^_^")
leak = int(input("Gift window:"))
if M % leak == 0:
    print("This is the gift for you: ", p % leak)
else:
    print("I don't like this gift!")
```

题解：

这题传M然后copper，很简单。但是官wp提到了个cve，想来看看怎么个事。

> 其实是一个cve（CVE-2017-15361），其做法是通过提交65537的阶在其上较小的M的因子，就可以让a很小

~~说实话我都没读通这段话~~

GKCTF2020有同考察点的题目。

简单来说就是rsalib（若素数生成器）的素数生成有漏洞，是用p=k*M+(65537**a %M)生成的，其中M为前n个素数乘积，n是仅取决于所需密钥大小的常数。a和k理论上也不会太大，由此就可以生成一个素数p。
通过变种的Coppersmith方法可以在2小时内破解512位RSA密钥。

官方wp大概就是实现了基于coppersmith的矩阵。

另外这里有现成解密ROCA脚本FlorianPicca/ROCA: A Sage implementation of the ROCA attack (github.com)

~~国际赛怎么这么难复现啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊~~

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

WMCTF 2023_OFFICAL_WRITE-UP_CN.pdf

GKCTF2020_Crypto*复现*roca漏洞_M3ng@L的博客-CSDN博客

ROCA vulnerability - Wikipedia

弱素数生成器 （RSALib） (asecuritysite.com)

GKCTF2020]Backdoor - 代码先锋网 (codeleading.com)

那么爱沙尼亚身份证系统和TPM的问题是什么？弱素数生成器（和 RSA！ |作者：比尔·布坎南教授 OBE |A安全网站：当鲍勃遇见爱丽丝 |中等 (medium.com)