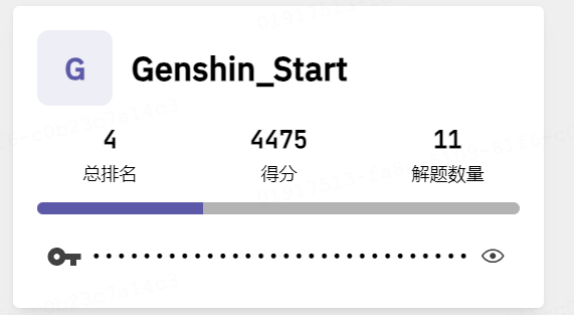
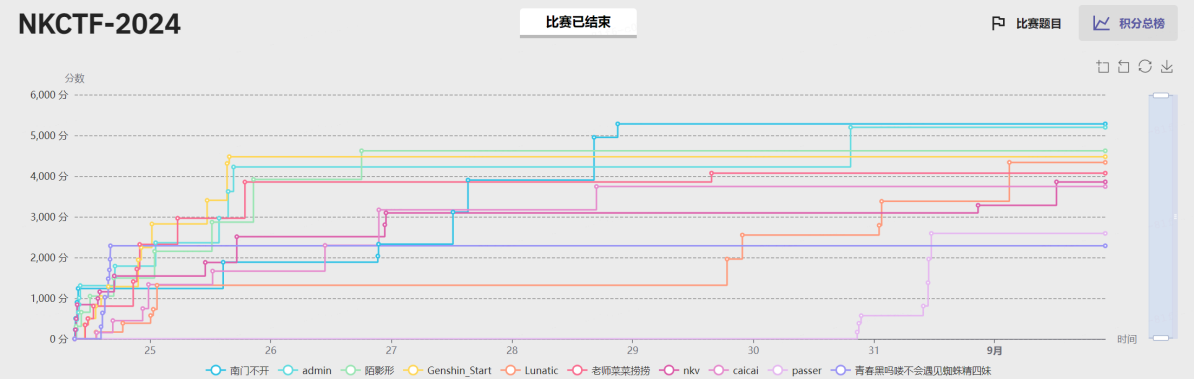
NKCTF_WriteUp_By_Luhaozhhhe

Result





Crypto

1_do_u_know_rot13(14 solves,154 points)

```
ATHtATVtAQZtAGDtAQLtA2VtAGZtAwxtAwptAzHtAJLtAQxtAzHtAJLtAwVtZmxtZmHtZmZtZmttZmNtZmRtAwLtA2D=
```

根据题目提示，是经典的rot13加密

解密得

```
NGUgNGIgNDMgNTQgNDYgN2IgNTMgNjkgNjcgNmUgNWYgNDkgNmUgNWYgNjIgMzkgMzUgMzMgMzggMzAgMzEgNjYgN2Q=
```

感觉像base64

脱一层，得到`4e 4b 43 54 46 7b 53 69 67 6e 5f 49 6e 5f 62 39 35 33 38 30 31 66 7d`

一眼ascii

ascii解密得到flag

```
NKCTF{Sign_In_b953801f}
```

2_ez_mod(5 solves,302 points)

文件内容为`daa66f47a3f5668484b8bb01b250b88a530159b232beb21f81be2bc4`

```
import sys
key = '------unkonw----'
flag = 'NKCTF{----unkown-----}'

if len(key) % 2 != 0:
    print("Error,illegal key length")
    sys.exit(1)

ciphertext = ''
for each in flag:
    for i in range(0,int(len(key)/2)):
        temp = (ord(key[i*2]) * ord(each) + ord(key[i*2+1])) % 251
    ciphertext += '%02x' % temp

print(ciphertext)
```

发现关键加密步骤是

```
for each in flag:
    for i in range(0,int(len(key)/2)):
        temp = (ord(key[i*2]) * ord(each) + ord(key[i*2+1])) % 251
    ciphertext += '%02x' % temp
```

我们知道取模是不可逆的，所以我们没法倒着解密

采用爆破的形式，我们猜测flag的前两位是NK

那么我们的密文中的da和a6分别就对应了我们的N和K

无脑爆破即可

```
temp1 = int('da', 16)
for x in range(32,126):
    for y in range(32,126):
        if (x*78 + y) % 251 == temp1:
            with open('1.txt', 'a') as f1:
                f1.write(f"{x}\t{y}\n")

temp2 = int('a6', 16)
for x in range(32,126):
    for y in range(32,126):
        if (x*75 + y) % 251 == temp2:
            with open('2.txt', 'a') as f2:
                f2.write(f"{x}\t{y}\n")
```

将两个输出的文件相比较，发现两者相同的部分为101,121

```
ciphertext = "daa66f47a3f5668484b8bb01b250b88a530159b232beb21f81be2bc4"
for i in range(0, len(ciphertext), 2):
    fl = ciphertext[i: i+2]
    fl = int(fl, 16)
    for flag in range(32, 126):
        if fl == (101 * flag + 121) % 251:
            print(chr(flag), end='')
```

解密即可，得到flag（仿射密码确实easy）

```
NKCTF{4ffin3_ciph3r_1s_easy}
```

3_ez_shamir(5 solves,302 points)

```
p = 0xe4fa76fb77c30f889bbab54d7d7a3e7edbd7ae6c42a1a443f657e95c5708fa15

(1, 0x97b7266eb7157a8328c2aefe29ac42786bebff6fa49dc9385aaaf0d04eb71fdb)
(3, 0x74a04c8d99b4213fbb346f736ce14c6dd450d70565eed5db2bbecc2bda38ff2e)
(5, 0xf1ee8c8818da5ce617da5065c921ada64f492cd823f9495f207e8af1963b268)
(7, 0xa2b1bdc65ba2b08ea56506d4def8dce068ab95ff461322d6486b1f57f8ceceba)
(9, 0xa598ea9e2e5a391e57eb4619af2bd20dc90f1104c01ee05cdfce24a4296088ef8)
```

一看感觉没啥头绪，然后看了一眼题干，发现是shamir密钥体系，之前还没有接触过

上网搜一搜对应的知识点，结果直接出脚本了

```
from decimal import Decimal
import gmpy2
import libnum
p =103569862428807273390153924605612763892112324827076587783912777922937702906389
m = ((1,
6862283840016396554869306412253317423324600930381655539048732401817578996527 5),
     (3,
527515143162851604544988089518970120072824754795915941146953107157856284178 38),
```

```
    (5,
68393047514776922975541823886707806840326791370871853557470182725238917494 80),

(7,735887231787342576481071756247492218154143336577193865390970383605743421231 94)
,

(9,749018000432834719566531960386415207346890464454458179344566116858528819156 40)
)
r=0
for i in range(5):
    ans=m[i][1]
    rev=1
    for j in range(5):
        if(i!=j):
            ans*=(0-m[j][0])
            rev*=(m[i][0]-m[j][0])
    temp=gmpy2.invert(rev,p)*ans
    r+=int(temp)
    r=r%p
r=r%p

print(libnum.n2s(int(r)))
```

把数字直接一换就出结果了(bushi

```
NKCTF{shamir_is_funny_aeb7b49d}
```

4_好大的E呀(4 solves,346 points)

```
from secret import flag
from Crypto.Util.number import *
m = bytes_to_long(flag)

p = getPrime(512)
q = getPrime(512)
N = p * q
phi = (p-1) * (q-1)
while True:
    d = getRandomNBitInteger(200)
    if GCD(d, phi) == 1:
        e = inverse(d, phi)
        break

c = pow(m, e, N)

print(c, e, N, sep='\n')

#
50731930772069400686916575246743443452690899444379694434142743274443218674216786100592969492035838858975790362301326716271359037068440695787843304326121973378828669353495488413292658283250319806410452561159285186453240700403006071553564128311178986134537345710296723446835100780095050029782699880797001290585
```

```
#
125619605854511237269883683031391390653132922696055190790276366165477690100535077
869013396660811953029595192335898397954255200805873910433163903845670979504404026
030359433123685368532858699572759217166389863783629130327509871695226910573050313
927188416593083574670460643869373616920128694339874731859790544747
#
128216550721871953992837401168487712446571832164773495937413965460678888368402993
356042919515750740340152446678834773547343464951187081474401459704326412662982617
641938704869418542453910363454470595033146721287218097966251455246701380660582645
862052493991746737373581632846045108023823704823323628291586628263
```

看到e很大，一眼wiener attack

板子题，一秒出

```
def wienerAttack(N, e):
    """
    维纳攻击

    `Parameters`:
        N - p * q\\
        e - public key

    `Returns`:
        p, q, d
    """
    cf = continued_fraction(e / N)
    convers = cf.convergents()
    for pkd in convers:
        # possible k, d
        pk, pd = pkd.as_integer_ratio()
        if pk == 0:
            continue

        # verify
        if (e * pd - 1) % pk != 0:
            continue

        # possible phi
        pphi = (e * pd - 1) // pk
        p = var('p', domain=ZZ)
        roots = solve(p ** 2 + (pphi - N - 1) * p + N, p)
        if len(roots) == 2:
            # possible p, q
            pp, pq = roots
            if pp * pq == N:
                return pp, pq, pd
    raise ValueError('Could not factor N!')

N =
128216550721871953992837401168487712446571832164773495937413965460678888368402993
356042919515750740340152446678834773547343464951187081474401459704326412662982617
641938704869418542453910363454470595033146721287218097966251455246701380660582645
862052493991746737373581632846045108023823704823323628291586628263
```

```
e =
125619605854511237269883683031391390653132922696055190790276366165477690100535077
869013396660811953029595192335898397954255200805873910433163903845670979504404026
030359433123685368532858699572759217166389863783629130327509871695226910573050313
927188416593083574670460643869373616920128694339874731859790544747
c =
507319307720694006869165752467434434526908994443796944341427432744432186742167861
005929694920358388589757903623013267162713590370684406957878433043261219733788286
693534954884132926582832503198064104525611592851864532407004030060715535641283111
789861345373457102967234468351007800950500297826998807970012905 85
p, q, d = wienerAttack(N, e)
m = pow(c, d, N)
print(m)
```

得到m的值：
906582538455463826518750613438150372456331783273267790986825649060338871063154 9

long_to_bytes得到flag

```
NKCTF{rsa-wiener-attack-d3752538}
```

5_rotate(4 solves,580 points)

```
import random
from secret import flag

ror = lambda x, l, b: (x >> l) | ((x & ((1<<l)-1)) << (b-l))

N = 1
for base in [2, 3, 7]:
    N *= pow(base, random.randint(123, 456))
e = random.randint(271828, 314159)

m = int.from_bytes(flag, byteorder='big')
assert m.bit_length() < N.bit_length()
a=[]
for i in range(m.bit_length()):
    a.append(pow(ror(m, i, m.bit_length()), e, N))
with open('chall.txt', 'w') as file:
    for num in a:
        file.write(str(num) + '\n')
```

题目有点难懂，丢到gpt让他看看（

2. **Rotation Function**:

```python
ror = lambda x, l, b: (x >> l) | ((x & ((1<<l)-1)) << (b-l))
```

- `ror` is a lambda function performing a bitwise "rotate right" operation.
- It takes three arguments: (the number to rotate), (the number of bits to rotate), and (the total bit length). `x` `l` `b`

3. **Generating** `N` :

```python
N = 1
for base in [2, 3, 7]:
    N *= pow(base, random.randint(123, 456))
```

- `N` is computed as a product of powers of 2, 3, and 7.
- Each base is raised to a random exponent between 123 and 456, and the results are multiplied together to get . `N`

4. **Generating** `e` :

```python
e = random.randint(271828, 314159)
```

- `e` is a randomly generated integer between 271828 and 314159. This range is interesting because it encompasses the famous mathematical constants e and π.

我们发现我们需要的结果和我们输出的c的值是一一对应的，根据每次pow得到的结果都能确定出我们的一位flag

```python
from Crypto.Util.number import *
from Crypto.PublicKey.RSA import *


with open(r'.\chall.txt','r+') as f:
    cipher = f.readlines()

m = ""
for s in cipher:
    s = int(s,10)
    if s%2 == 1 :
        m += "1"
    else:
        m += "0"

m = int(m[::-1],2)
print(long_to_bytes(m))
```

得到flag

```
NKCTF{0h_1t_l34ks_th3_l34st_s1gn1f1c4nt_b1t}
```

6_rsa_chall(3 solves,656 points)

```python
from flag import text,flag
import md5
from Crypto.Util.number import long_to_bytes,bytes_to_long,getPrime

assert md5.new(text).hexdigest() == flag[6:-1]

msg1 = text[:xx]
msg2 = text[xx:yy]
msg3 = text[yy:]
```

```
msg1 = bytes_to_long(msg1)
msg2 = bytes_to_long(msg2)
msg3 = bytes_to_long(msg3)

p1 = getPrime(512)
q1 = getPrime(512)
N1 = p1*q1
e1 = 3
print pow(msg1,e1,N1)
print (e1,N1)

p2 = getPrime(512)
q2 = getPrime(512)
N2 = p2*q2
e2 = 17
e3 = 65537
print pow(msg2,e2,N2)
print pow(msg2,e3,N2)
print (e2,N2)
print (e3,N2)

p3 = getPrime(512)
q3 = getPrime(512)
N3 = p3*q3
print pow(msg3,e3,N3)
print (e3,N3)
print p3>>200
```

简单分析一下，分为三个部分

第一部分是一个低指数解密攻击，e=3

第二部分是一个广播攻击，n和msg相同，c和e变化

第三部分就是一个简单的coppersmith的p的高位攻击

task1：

```
#task1
from gmpy2 import *
c1=191057652855106675533138988134982202124211775276471878025499139142639689454931
446333906706051162510645503647047893588300721333491088087990750215404798151826576
677636171780441109394588346549225407041963304519793493530315785184791994544804581
379847344022480114644673127536832345433199558 93
e1=3
n1=123814470394550598363280518848914546938137731026777975885846733672494493975703
069760053867471836249473290828799962586855892685902902050630018312939010564945676
699712246249820341712155938398068732866646422826619477180434858148938235662092482
058999079105450136181685141895955574548671667320167741641072330259009
for i in range(200000000):
    if gmpy2.iroot(c1+n1*i,3)[1]==1:
        res=gmpy2.iroot(c1+n1*i,3)[0]
        print (i,res)

        break
```

通过上面的脚本解出msg1

```
msg1=267334379257781603687613466720913534310764480084016847281446486946801530200295563483353634338157
```

task2：

```
#task2
import gmpy2 as gp
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)
n = 
111381961169589927896512557754289420474877632607334685306667977794938824018345795836303161492076539375959731633270626091498843936401996648820451019811592594528673182109109991384472979198906744569181673282663323892346854520052840694924830064546269187849702880332522636682366270177489467478933966884097824069977
c1 = 
549957513872587987918954132161722846534070540797657697041707630238301309814802729433384452456892937293082005742179590184625127905236222524792584194988583078981189070767734702535333448779595087662857305090678296844273757593456237016059970671356594042966638774537587010107265618249516026155010788189144109596l0
c2 = 
912909352674583565419593273812200674661048904553911039896398228557537978053541397419599579519839431461085527627564444755452503437667982203482403775901128548904823757448760161917734718537040147359366084362101536698294542881998388276464027425541340172802137072223384962712898946813126062395129248428452683669501
e1 = 17
e2 = 65537
s = egcd(e1, e2)
s1 = s[1]
s2 = s[2]
if s1<0:
    s1 = - s1
    c1 = gp.invert(c1, n)
elif s2<0:
    s2 = - s2
    c2 = gp.invert(c2, n)

m = pow(c1,s1,n)*pow(c2,s2,n) % n
print(m)
```

得到msg2

```
msg2=419330585328454910382119580760949262409503142808521987944834210433732294500138768023601196047229681529323314430373027397990583776206765291330889843372880086477679463819805560742250306541059589467674053168036722769662235202624767645254006402032261903612538114634660365544548769557482491913l
```

task3：

```
#task3
c=592136964423737658959487026116597567798138976530220809056355456369054340383064689352839626860590374619402276187156958755890555936963525946301070827147570368158754971385237386950668119850363156249278970811531903296368640051337570969910356079181065291514518343694423136738495636352484650142894093742913814296460e=65537
n=113432930155033263769270712825121761080813952100666693606866355917116416984149165507231925180593860836255402950358327422447359200689537217528547623691586008952619063846801829802637448874451228957635707553980210685985215887107300416969549087293746310593988908287181025770739538992559714587375763131132963783147
p_high200=711728669592547291800107184697390034264010777021485892818841976562815147862023604284826579029020
#Sage
n =
p4 =  #p去0的剩余位
pbits = 1024
kbits = pbits - p4.nbits()
print(p4.nbits())
p4 = p4 << kbits
PR.<x> = PolynomialRing(Zmod(n))
f = x + p4
roots = f.small_roots(X=2^kbits, beta=0.4)
if roots:
    p = p4 + int(roots[0])
    q = n//p
    print(f'n: {n}')
    print(f'p: {p}')
    print(f'q: {q}')
```

先解出p和q，然后正常rsa解密就可以了

```
p=114370387635810102631164939837335460144033438592180037075127967069288808480352399907404283340911064439827693865177537038900024786984185497775532689064964232147821073815543685389957763965574467468668612471912489383396408763682689305894821073815543685389957763965574467468668612471912489383396408763682689305893
q=99180331989638797983623295076372567060105629624873297424009331927215493070873324821073815543685389957763965574467468668612471912489383396408763682689305893
n=p*q
from gmpy2 import *
from Crypto.Util.number import *
phi=(p-1)*(q-1)
c=592136964423737658959487026116597567798138976530220809056355456369054340383064689352839626860590374619402276187156958755890555936963525946301070827147570368158754971385237386950668119850363156249278970811531903296368640051337570969910356079181065291514518343694423136738495636352484650142894093742913814296460e=65537
d=invert(e,phi)
m=pow(c,d,n)
print(m)
```

得到m，再字符转化得msg3

```
msg3=97843087147756905198977654765902035972105683863579736247431188643611696235429285118172006000097914357119837885601239174207851058692737678379775753907823908834975864414481289815510662354365095394060654382256742313035020720789538049963800115144384199717629954869273705672442363188 2
```

然后把三部分合在一起，直接反套md5就出flag了

```
msg1=26733437925778160368761346672091353431076448008401684728144648694680153020029556348335363433815 7
msg2=41933058532845491038211958076094926240950314280852198794483421043373229450013876802360119604722968152932331443037302739799058377620676529133088984337288008647767946381980556074225030654105958946767405316803672276966223520262476764525400640203226190361253811463466036554454876955748249191377
msg3=97843087147756905198977654765902035972105683863579736247431188643611696235429285118172006000097914357119837885601239174207851058692737678379775753907823908834975864414481289815510662354365095394060654382256742313035020720789538049963800115144384199717629954869273705672442363188 2
flag = long_to_bytes(msg1)+long_to_bytes(msg2)+long_to_bytes(msg3)


from Crypto.Util.number import *

import hashlib
print(hashlib.md5(flag).hexdigest())
```

flag:

```
NKCTF{3943e8843a19149497956901e5d98639}
```

7_xor&rox(4 solves,580 points)

```
from Crypto.Util.number import *
from hashlib import md5

a = getPrime(512)
b = getPrime(512)
c = getPrime(512)
d = getPrime(512)
d1 = int(bin(d)[2:][::-1] , 2)
n1 = a*b
x1 = a^b
n2 = c*d
x2 = c^d1
flag = md5(str(a+b+c+d).encode()).hexdigest()
print("n1 =",n1)
print("x1 =",x1)
print("n2 =",n2)
print("x2 =",x2)
```

```
#n1 =
838763494437926958008581070260411839823209237328177881964030384369078520459686780
327443648205912546537901020515487329742729466722196532044686409153157035785204306
355358928700379204148275065781575309209873884712034553577762608564324840542971000
459725270977198709471700533063755983088785582047348882467797165 99
#x1 =
470074176751536775598897975923770635978979028109069024580032435083767762464518452
611002794398395269024667944527936899900883918340630147557934989195225 7846
#n2 =
652881484543771018418888718488067046944779065870107552864512166327018684577228481
396960369285618888507174426167825833099757141726264764854833612171745147474680995
678706402774410043223446717174443060553985137330530545975860900749215407943476151
535422868932724159317093962621184160628870032900700011730355873 41
#x2 =
360438668861232087414353226298838456221365979857858321089214326157690828111222335
667890008387032752724223823751317036766004395437606300416722855059211 0478
```

看到这两行代码`n1 = a*b x1 = a^b`

肯定是爆破求解，我们从低位开始爆破即可

```
import itertools

n1 =
838763494437926958008581070260411839823209237328177881964030384369078520459686780
327443648205912546537901020515487329742729466722196532044686409153157035785204306
355358928700379204148275065781575309209873884712034553577762608564324840542971000
459725270977198709471700533063755983088785582047348882467797165 99
x1 =
470074176751536775598897975923770635978979028109069024580032435083767762464518452
611002794398395269024667944527936899900883918340630147557934989195225 7846

a_list, b_list = [0], [0]

cur_mod = 1
for i in range(720):
    cur_mod *= 2
    nxt_as, nxt_bs = [], []
    for al, bl in zip(a_list, b_list):
        for ah, bh in itertools.product([0, 1], repeat=2):
            aa, bb = ah*(cur_mod // 2) + al, bh*(cur_mod // 2) + bl
            if ((aa * bb % cur_mod == n1 % cur_mod) and ((aa ^ bb) == x1 %
cur_mod)):
                nxt_as.append(aa)
                nxt_bs.append(bb)

    a_list, b_list = nxt_as, nxt_bs

for a, b in zip(a_list, b_list):
    if a * b == n1 and a*b-n1==0 and (a^b)-x1==0:
        break

print(a)
print(b)
```

```
#783614713961065522371146974720016406948487889462166870664740637786609468164555354874619497753277560280939259937394201154154977382033483373128424196987617
#107037741825710733611127913760323800963606979268403624832421058781155524370216748615287145980896034060328444187587257448794765963592252653335302358033658470
```

这样我们就得到了a和b的值

然后根据下两行我们再求解我们的c和d就可以了，也是有脚本

```
import itertools

n1 = 
652881484543771018418888718488067046944779065870107552864512166327018684577228481396960369285618888507174426167825833099757141726264764854833612171745147474680995678706402774410043223446717174443060553985137330530545975860900749215407943476151535422868932724159317093962621184160628870032900700011730355873410

x1 = 
360438668861232087414353226298838456221365979857858321089214326157690828111222335667890008387032752724223823751317036766004395437606300416722855059211047810

cl_list, dl_list, ch_list, dh_list = [1], [1], [1], [1]

x1_bits = [int(x) for x in f'{x1:0512b}'[::-1]]
print(x1_bits)
print('\n')
mask = 2
for i in range(1,256):
    mask*=2
    scl_list, sdl_list, sch_list, sdh_list = [], [], [], []
    for j in range(len(cl_list)):
        for cl in range(2):
            for dl in range(2):
                for ch in range(2):
                    for dh in range(2):
                        if (cl ^ dh == x1_bits[511-i] and ch ^ dl == x1_bits[i]):
                            temp1 = ((mask // 2 * cl + cl_list[j]) * (mask // 2 * dl + dl_list[j]))%mask
                            temp2 = n1 % mask
                            if (temp1 == temp2):
                                g1 = bin(ch_list[j])[2:] + bin(ch)[2:] + '1' * (510 - 2 * i) + bin(cl)[2:] + bin(cl_list[j])[2:].zfill(i)
                                g1 = int(g1, 2)
                                g2 = bin(dh_list[j])[2:] + bin(dh)[2:] + '1' * (510 - 2 * i) + bin(dl)[2:] + bin(dl_list[j])[2:].zfill(i)
                                g2 = int(g2, 2)
                                if(g1 * g2 < n1):
                                    continue
                                g1 = bin(ch_list[j])[2:] + bin(ch)[2:] + '0' * (510 - 2 * i) + bin(cl)[2:] + bin(cl_list[j])[2:].zfill(i)
                                g1 = int(g1, 2)
                                g2 = bin(dh_list[j])[2:] + bin(dh)[2:] + '0' * (510 - 2 * i) + bin(dl)[2:] + bin(dl_list[j])[2:].zfill(i)
                                g2 = int(g2, 2)
```

```
                                if (g1 * g2 > n1):
                                    continue
                                scl_list.append(mask // 2 * cl + cl_list[j])
                                sch_list.append(ch_list[j]*2 + ch)
                                sdl_list.append(mask // 2 * dl + dl_list[j])
                                sdh_list.append(dh_list[j] * 2 + dh)
    cl_list,dl_list,ch_list,dh_list = scl_list,sdl_list,sch_list,sdh_list

print(cl_list)
print(dl_list)
print(ch_list)
print(dh_list)




d=int(bin(ch_list[0])[2:]+bin(cl_list[0])[2:].zfill(256),2)
c=int(bin(dh_list[0])[2:]+bin(dl_list[0])[2:].zfill(256),2)

print(c)
print(d)
print(c * d - n1)
print(((c ^ int(bin(d)[2:][::-1],2)) - x1))

a=78361471396106552237114697472001640694848788946261668706647406377866094681645553548746194977532775602809392599373942011541549773820334833731284241969876176
b=10703774182571073361112791376032380096360697926840362483242105878115552437021674861528714598089603406032844418758725744879476596359225265333530235803365847
c=80469254367102041924383040558747788658954169969708438696988588656039534111703695269977842242104917691403880469609666446281544892032869402938814271880583276
d=81134277890200785266828179169439421534891877861073079587655860326107413542892805392648534697836213150493855498849031338062941836143520849883651096302506836
print(bin(c ^ int(bin(d)[2:][::-1],2)))
import hashlib
flag = hashlib.md5(str(a+b+c+d).encode()).hexdigest()
print("NKCTF{%s}"%flag)
```

得到flag

```
NKCTF{f28ed218415356b4336e2f778f2981bb}
```

8_easy_lwe(3 solves,907 points)

```
from Crypto.Util.number import *
from secrets import flag
assert len(flag) == 38
t = 30
p = getPrime(512)
x = getPrime(512)
while x > p:
    x = getPrime(512)
rs = []
```

```
cs = []
ss = []
for i in range(t):
    r = getPrime(512)
    s = getPrime(400)
    c = (r * x + s) % p
    rs.append(r)
    cs.append(c)
    ss.append(s)

enc = pow(x,flag,p)
print(f'p = {p}')
print(f'rs = {rs}')
print(f'cs = {cs}')
print(f'enc = {enc}')
```

发现很明显是拼接题，前半部分是lwe attack，后半部分就是一个光滑的离散对数问题求解

直接用脚本拼接就解决了这个问题

```
from sage.modules.free_module_integer import IntegerLattice
from Crypto.Util.number import *
import hashlib
row = 30
column = 1
p =
689710844307598174448475871608104541785422754371310640429478965518010545749904217
971744734259379018094341501404483087292516516345747620981935669424484007 9
```

```
rs =
[128446345492630532287597492644036370227402900082869874015850689527419352774155276783800212126248467222425007084227595635589959369772745803013794941957024612251634003683452916928102291170339939586644029776192301741341674585154859358419625191986830852794085541953563738986709807899575511700135958334229151930861,
70513706660775421972486380130117938244770737773222195458823678818071300661684441349645713981121518488340326549783682552186497207380409454298376928570319577730468623519529303685055932845462675545712958723773231115580712787012314729757919629792565515195337239885568705518850737424076304811981923897502893921078837764976601383087200069125827386728887523443269281538109102214535950777112843020415125294574502116027872107614611723264298805940241870254198730434358771205673513714546003658084103833210031116036884387316464960634304241689689879323324987390221868396628396972146008739012062225475802777996074092612300537757191954455449440606823224324918643367638346932281715221003756303205620290937782574077538308760564737415047709671895645422594609371069186498856310912246023449098354751049599873213988024512286270964608502444597112110163392131757813461977030270733012385926751192637938686124570227538910606279104888073013,
11308837998867241929817950595621831002334468993828126438599805989088017326675963100044309448653090403889186401929445861220402556074702741108929442867300279,
91846228874142093615165931011295565698118882146075566300949697639104269537860207558380941849723974802766661706859264251370635593949691662163929392570915411289640006951589089743008781598254567183064520102366511242977964076889909128729145240836944591946414439072620080887506638924012690981123959709289373345733911227025698697471809912850435140886785315702278826761054472525227951791647003561270585720797267604996360933395122286757099101227901032364782594523739698877,
81621234906563174903618800206679190727080910537168918706915442174901264449975034040941742460879388289936963351914885833064435772087967942740992820134272471336698988944267029146126231375797760009596205747086347551908864826730112971995336894341956214427667940096712272755476401313291850556467724397997880732304199208574559454085882039721934444375331643513092990409114692750590920317558114924605856539484815229955578017818382154076485729993584566125258120675383725797139402473546047825312503780125417567716958846513076797328672521987900978293260385267945187604725349720103672258987935569856239987227455748213833342843243,
13108142660294572752252393081421368493392921884487755391460006730258159004638343897340537616297811742032405724656497443006056456690449881719305597286675631,
13276762958403786077380090195631980415297280849950287990717193547481553124160398455403123819234755237450529090601858784999113026218918277529515287668651121,
12463094640052886550696551772104539361264529587569204472038955376345085195998921095774583176899949596998985033050547755235409943131811058035802010421860899,
11307743131694864808301935844724645695851330736969875190167422024500753079857478680029193758960169072890576310607053767920339034290416580654771095674487943,
10053742503547378455068966704402695956702795408343604912294923217443553169726438945982031485796964462946592530592946335569560364464958066521486506177193131,
97036957634517991252589617762293255108142893586792133054185593819014964498495842442118348723137678449962555567210410076546251538091289874229921022924725338148189465927721940294369879439913703690047528695196368949823197675716174991296513758196009346701553643721225250628151384047219921709201619262393792138023,
91141509109642378184183678402077245289173024068361579282238726224429286042498644868587557371496406832598342991659006965850385691886276820220027090589022911,
12273514376180781903469287345188404399033432117915094289694407562166649079228640510678711431664410226301556172582177240184695103942141430877677144285616059,
83550057216844255148829339102865841483053445805896231129595174289939684385338669062237777780580969623332032371112453284366009941201689241438496857282688116,
89578838388074714921474808166835266360196984641331852376682432686671698008116967704844871235601979884484344751123520057682864175293191821622458405236970011,
12168542584724814356632409768687396920143300559579648963851924568387314914334359
```

```
059426855512101804484196740602194963951160818667849180592371334140412278 33,
122859350079308255716721283468043136071961904656907598707582787050860347788086628
860564608279359862852591850715144909428315853131909463868786226088683455 63,
771991381785957237716497334365115593406029660790853784525675547246502520275123998
075895009486506775140788956936997401113980140158693911914777346611169991 3]
```

```
cs =
[391132590126177073106634372735309338560719688360102224442685746007433842069261001241457162351215248547424816922003058783984972275773859682519433853455847,
419855511732532587458401969141857307173316764021393374958234744251899758845221167314372217928177360245550700139598368100976984841400720626868218481616874 4,
442217366663498323489509879881396203787541756823570852433982670927138174888493617837176757406479417741661571012022391472587323983612165470520861457641353 3,
354026042255569788786962754620816471155001590937834010507765217748195957655067837972345098180755686357261075982466063041867054620373317005862675508079799 8,
645149846749893520109251486562793167709107878799709741420843099218326495057902237337225448659545811788730539331766371233769933150372512428701713480848487 4,
343962958196352435181043091073733612461631664165619064124843450462177423594351461730185791704111161710485024514874642718006974394061256071821317790342730 6,
427946819148183221249693924209348604427897693796508547556700822806118494751315601236958697048654308313056562890629660055302457409948124653487824292063721 2,
410213545551806113391902767057132527997622264798445235305139586455430952122349876182308471707710221364861282651366162959997160955523576015204954905723434 2,
329051927890365028889097463563711966673066795688728876214731188783168691555262156515161429328581094087585127929869064685419149676592073496155898360311360,
167434720989689757150235245106318883493890443032995175211192111523034994782318812197298002556387888720150762941981173691069096502092375142410152181605797 0,
477908431781137505015957499474629748659227124713782347137519962678895657699862718122048995250793776804250120309839196670229781253746321179983792168446754 1,
524033181578432279214454987387365863672623309322841548909800298222076967671868113273779499470871638917416282072164674477662441373531824059774536349042758 4,
268971689492260487545520769525366521285347030834174304095736795772715561419974356222514735961451418987798315689274966980480016325261748044656547999014802 1,
449708769594599088851244243076921016853502252396793496349534051273454215985560340288452398756880916680293627457774430655982228613348249480600180821975835,
158460397833128933535299715105966677327794345835716105127865809042006702368023141425555780541028814409265312156876613637272809530098274330969634703112142 4,
487426005315170037480933705376303248918472533419649516035827503858682402792023873388670316301845081480593736382522345927737307359102108227661013511897683 4,
352437413136290690054529729194711017729886256471845182183979496016935608204254838655336348092109745290272303385474944328868298355884705284329366681542519 6,
654412359149956923202191337029357047777670931500878353172088654578477347148676924071126256240168314593771561243521381637268018932114192879050949028262989 1,
487386116622811896709956908647854816712743141501779167881241967679175446693583203487086200065878960908416689193397001384985014671837981994373726997065486 6,
410081787443670307171665516397214503610498597316483054782592959087192082598124193463397722754793451414266078606129102665780235740402423628795530937248951 6,
343238276681348130286495167739162902430650061145485619903964358840996341335935043000395056684771452815629410388891486531126938900311458948803147120186532,
268371072435041299877039231883243488530453832503315993737948931992434668919744572073420984190261223548501686625499404596971641302019729642832383240415118 2,
590946464110570417999910431156241636309016676234164469118816971618295897127039600742258142981317293393058147577130603449522405497272523075767544473195348 0,
105593489999747649490909047130635486331667382136386336225885304397053465240127478919767755821518824907483782900333573321189021164850168965634582485850737 3,
499237936654264569137595924746588888977811815398214210095680944085574565974523557628057831618546930662001784569031255404377065105812653604017311394952439 6,
653345639824478990763677940704551556713519547428418537968951838755834599762743542158243739005323467599136180853227826407796854019740774374427910687171626 7,
516936039876727027585379024231521367163388042821260376630130885336306309260958257295756113802280688789563414089964002557075991925761553737570600815968023 9,
203310740924699994885931266978520636166917734618272844754878785050509801614513144
```

```
7391644508349361780657921127972029591063652826992455783090609052977427O6,
3143563289239398127009575193211845399079310618985464994769603542400451633289266080869317336163844517539211542909055869608349639432145332113320465388067087,
4016252180207572047405081190649590978593306403200098541033213590567723751195926093369984531729148621419589009515870336049849542537363832071754623330736088]
enc =
1315637864146686255246675143589215932218700984880749264689270214639479160648747323586062096067740047809798944996253169402675772469028914904598116394230426
```

```
ss=
[144469599714949219032550335874890718089642874450143595901244619780447123899622869316979587443724065617010763853579700490 1,
171010600217355100242544405414039417895612402337665795730323076550837708907471774646565381546676365464824952785782556902 9,
205531330043475478561406153466123476932254345776327307231110748241228890865956430917964325156079944603808167751659896345 7,
252989375264423778348379781521294649598285555693320469401183267684009191003565368541601996360816631853674934581087645377 1,
252913862811779844347093723404507794547392820490619861900528736577671152715636749638363146356167610974186223025158614337 7,
226211033581345247443069965641288523450342685260539369189988179663421161105233073531518194075882805439889445501711373322 7,
212188963745052202935055287762667980945866229722886552164255457311977208185499031215645377428077047419416250238949716032 7,
252914254818056399639463977371353283348027372730612508671993575616191718491173680661626791607573510659054478032446758562 1,
130532929553640427617044805650070046982527213702094646014396314020035315737719650555628006799144717087503812286641555010 1,
155285209808606080873713912364591563109918782832566447937340235278054673920751610801542899085148911993806192981849192661 1,
179110499980366180660395543621538539082650804974136347042406052611173035296299923260977958474385364836325767691348705775 7,
189944247956632750759528415219892381607002029928041332902624752276507232311988475394920887537710800165719778177767929187 1,
145354116759234318833882222969166004473691231270842849283172908725574826354550690946753392059833745259373439370437697880 7,
183997025564751798803377650240378183834835231421492288912161745006073521074129495538814253493682227444340402180580333370 3,
191216088143696113080585410558069192143360524068286950819260497903804388464295997473516245366473701107827618613200326753 7,
254356546280529368247962666184877286547925164170998062244215381942825143713365374582257018474992933064429321807637594029 7,
248308315665163577450284335047727841382796581475434309879142402288069570352988405336310127686139083432729517846458609016 9,
200316910154239578125610985037809934366162065652590622601277761882567988288275012110896774738004044081230550002503047270 3,
149530658296264434476363530397334488737663400266432102309943358101161338521598447393389953955275912144955661911821392695 3,
147547564050740543526355656092715114096183810729142903355082373808559845258864569745145362281130783624017848727006620273 1,
158529625875865386972977939663347776893578707764441351887411195361226249999102037879513297617183136361274056557767358994 9,
175187098127705415388947536286518427080867277781473918760638242050295324994290853614699477569186544971588110343741156269 7,
222938501985844363324756784166851222631255202811287210304956850980043213256313922360050133448845548699584778745702305754 3,
250952275783382263687602929560198086224454899615327243368761309333704290456971180180500401323058971538017522510287877242 9,
137853868654073306130757840960866289666384343165627394966755109898696843399373409053352599977267898701472082113444788327 9,
161574546481022257886811300691723841559477397745618903724514829849857863716333262829902022222165871934717312379738199859 7,
206415146429410078951675648245681874619032120356573237173102996713692120141029706982791552268762014010402146052380441635 3,
171154549619877513105663740565968313241782837652811327008820092929723396294220254
```

```
662123982993130413403169596081421580797 3,
145576187928221733647718903962236305435104899352068796256778540977502055642269209
166444057677586393333688280376324654892 3,
188089142353641197956537433458214347488827557834236750242918912551329015327193035
507196955710553967992221790588717795024 9]

prime=p
ma=rs
res = cs

W = matrix(ZZ, ma)
cc = vector(ZZ, res)
#LWE求解x
# Babai's Nearest Plane algorithm
def Babai_closest_vector(M, G, target):
    small = target
    for _ in range(5):
        for i in reversed(range(M.nrows())):
            c = ((small * G[i]) / (G[i] * G[i])).round()
            small -=  M[i] * c
    return target - small

A1 = matrix.identity(column)
Ap = matrix.identity(row) * prime
B = block_matrix([[Ap], [W]])
lattice = IntegerLattice(B, lll_reduce=True)
print("LLL done")
gram = lattice.reduced_basis.gram_schmidt()[0]
target = vector(ZZ, res)
re = Babai_closest_vector(lattice.reduced_basis, gram, target)
# print("Closest Vector: {}".format(re))

R = IntegerModRing(prime)
M = Matrix(R, ma)
M = M.transpose()

ingredients = M.solve_right(re)

#求出x后利用PH算法对flag进行还原，因为发现p-1可以分解，p是光滑阶
#
x=678989130529777955655657192281297892237507390174976421596900330986971887807626954530405584312530155310353125233487656043340545110889520696990426845678613 9
x=int(ingredients[0])
print(x)
m=x
c=enc
n=p

def r(h, g, N, p, qi):
    Zp = Zmod(p)
    h = pow(h, N//qi, p)
    g = pow(g, N//qi, p)
    ri = discrete_log(Zp(h), Zp(g))
    return int(ri)
m=x
c=enc
```

```
n=p

tmp_list=
[2,3,193,877,2663,662056037,812430763,814584769,830092927,849943517,969016409,100
0954193,1022090869,1048277339]

r_list = []
for qi in tmp_list:
    tmp = r(c,m,n-1,n,qi)
    print(tmp)
    r_list.append(tmp)
x = crt(r_list, tmp_list)

module = 1
for i in tmp_list:
    module *= i

while True:
    if int(x).bit_length()>304:
        print('fail')
        break
    if int(pow(m, x, n))==c:
        print('x =', x)
        print(long_to_bytes(x))
        break
    x += module
```

得到flag

```
NKCTF{70b1b709ce431682addb581596320007}
```

Web

sign_in(12 solves,166 points)

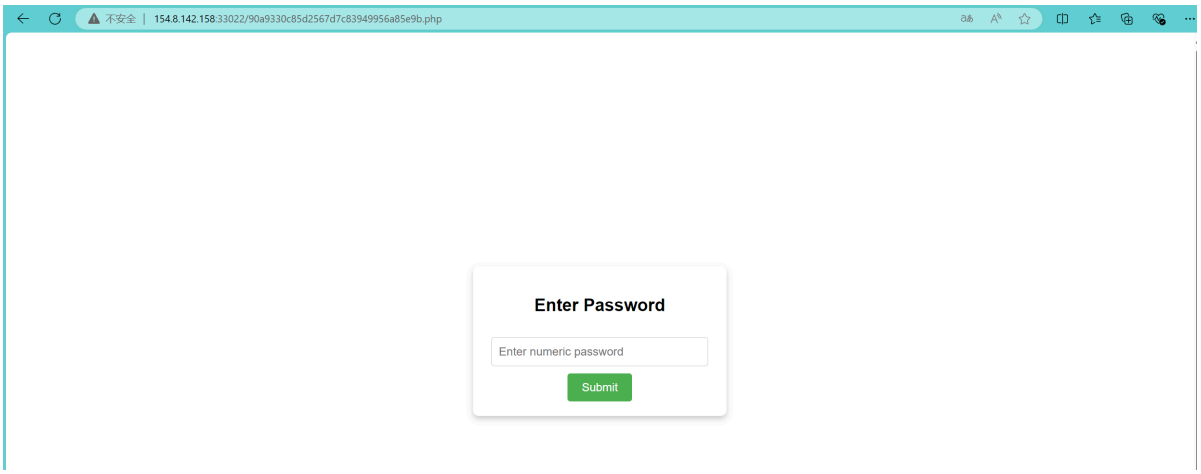
这题签到题，很简单，照着步骤来就行

先在题目的地址后面加上hint：`/www.zip`

下载对应的内容，得到hint：

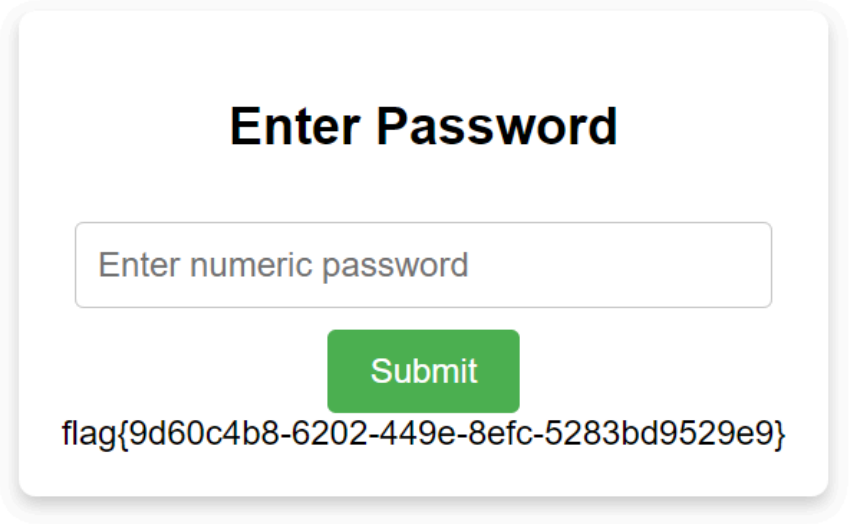
90a9330c85d2567d7c83949956a85e9b.php

将其加到网址后，得到以下界面



题目让我们输入password，发现题目提示给出了一堆密码，我们直接一个一个试一下就可以了

尝试到1919时成功出现flag




```
flag{9d60c4b8-6202-449e-8efc-5283bd9529e9}
```

Reverse

1_simple_xor(8 solves,217 points)

这题观察题目，是一个简单的脱壳加壳的re题，秒了

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  int v3; // edx
  __int128 *v4; // rax
  char v5; // cl
  int v6; // eax
  const char *v7; // rcx
  __int128 Buf1[2]; // [rsp+20h] [rbp-48h] BYREF
  __int64 v10; // [rsp+40h] [rbp-28h]
  __int16 v11; // [rsp+48h] [rbp-20h]
```

```
  char v12; // [rsp+4Ah] [rbp-1Eh]

  v10 = 0i64;
  memset(Buf1, 0, sizeof(Buf1));
  v11 = 0;
  v12 = 0;
  puts("enter your flag:");
  sub_1400010D0("%42s");
  v3 = 0;
  v4 = Buf1;
  do
  {
    v5 = v3++ + 85;
    *(_BYTE *)v4 ^= v5;
    v4 = (__int128 *)((char *)v4 + 1);
  }
  while ( v3 < 42 );
  v6 = memcmp(Buf1, &unk_140002240, 0x2Aui64);
  v7 = "good job";
  if ( v6 )
    v7 = "try again";
  puts(v7);
  return 0;
}
```

`v5 = v3++ + 85`表示做xor运算从85开始，递增

while循环代表，加壳操作需要经过42轮操作，说明我们有42个字符

`v6 = memcmp(Buf1, &unk_140002240, 0x2Aui64)`代表了将我们的加壳结果与`&unk_140002240`处内存的值进行比较，如果相同才是加密成功，否则失败

在ida中找到对应的地址中存储的内容，进行反向解密即可

```
0000000140002230  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000140002240  33 3A 36 3F 22 3C 3A 6F  69 6C 3B 57 51 4F 56 50  3:6?"<:oil;WQOVP
0000000140002250  04 51 4A 5C 5B 59 0D 41  0C 0C 0D 15 5C 13 17 4D  .QJ\[Y.A....\..M
0000000140002260  47 4E 12 4B 1D 1B 4B 4A  1C 03 00 00 00 00 00 00  GN.K..KJ........
0000000140002270  65 6E 74 65 72 20 79 6F  75 72 20 66 6C 61 67 3A  enter·your·flag:
0000000140002280  00 00 00 00 25 34 32 73  00 00 00 00 00 00 00 00  ....%42s........
0000000140002290  67 6F 6F 64 20 6A 6F 62  00 00 00 00 00 00 00 00  good·job........
00000001400022A0  74 72 79 20 61 67 61 69  6E 00 00 00 00 00 00 00  try·again.......
```

编写一个简单的py程序来进行逆向脱壳

```
encrypted_flag = bytearray([

 0x33,0x3A,0x36,0x3F,0x22,0x3C,0x3A,0x6F,0x69,0x6C,0x3B,0x57,0x51,0x4F,0x56,0x50,

 0x04,0x51,0x4A,0x5C,0x5B,0x59,0x0D,0x41,0x0C,0x0C,0x0D,0x15,0x5C,0x13,0x17,0x4D,
    0x47,0x4E,0x12,0x4B,0x1D,0x1B,0x4B,0x4A,0x1C,0x03
])

FLAG_LENGTH = 42
XOR_BASE = 85

def decrypt_flag(encrypted):
    decrypted = bytearray(FLAG_LENGTH)
    for i in range(FLAG_LENGTH):
```

```
        decrypted[i] = encrypted[i] ^ (i + XOR_BASE)
    return decrypted.decode('utf-8')

if __name__ == "__main__":
    decrypted_flag = decrypt_flag(encrypted_flag)
    print(f"Decrypted flag: {decrypted_flag}")
```

得到flag

```
flag{fa342d70-54a7-423f-abbe-ad928e3da06a}
```

2_base64(6 solves,265 points)

这题很简单，一眼出。

F5反汇编后得到

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  void *v3; // rax
  void *v4; // rbx
  __int64 v5; // r8
  int v6; // eax
  const char *v7; // rcx
  __int128 v9[2]; // [rsp+20h] [rbp-48h] BYREF
  __int64 v10; // [rsp+40h] [rbp-28h]
  __int16 v11; // [rsp+48h] [rbp-20h]
  char v12; // [rsp+4Ah] [rbp-1Eh]

  v10 = 0i64;
  memset(v9, 0, sizeof(v9));
  v11 = 0;
  v12 = 0;
  puts("enter your flag:");
  sub_1400010E0("%42s", (const char *)v9);
  v3 = (void *)sub_1400011E0(v9, 42i64);
  v4 = v3;
  v5 = -1i64;
  do
    ++v5;
  while ( *((_BYTE *)v3 + v5) );
  if ( v5 != 56
    || (v6 = memcmp(v3,
"zMXHz3S2nMiWogzMzI1IyMi1ltrMnJuTogyZzcOYmZeYnJvLmte3nwz9", 0x38ui64), v7 = "good
job", v6) )
  {
    v7 = "try again";
  }
  puts(v7);
  free(v4);
  return 0;
```

```
}
```

前面啥的都不用看，看到字符串`zMXHz3S2nMiWogzMzI1IyMi1ltrMnJuTogyZzcOYmZeYnJvLmte3nwz9`，发现前面的ZMXH很像base64的flag开头

base64得

发现不太对

将字符串大小写互换，再base64就出来了


```
flag{66b08fff-bbb5-4f65-8f3d-231265e1175f}
```