

XYCTF Writeup By Beckoning

Crypto

Sign1n_Revenge

```
from Crypto.Util.number import *
from tqdm import *
import gmpy2
flag=b'XYCTF{uuid}'
flag=bytes_to_long(flag)
leak=bin(int(flag))
while 1:
    leak += "0"
    if len(leak) == 514:
        break

def swap_bits(input_str):
    input_list = list(input_str[2:])
    length = len(input_list)

    for i in range(length // 2):
        temp = input_list[i]
        input_list[i] = input_list[length - 1 - i]
        input_list[length - 1 - i] = temp

    return ''.join(input_list)

input_str = leak
result = swap_bits(input_str)
a=result

def custom_add(input_str):
    input_list = list(input_str)
    length = len(input_list)

    for i in range(length):
        input_list[i] = str((int(input_list[i]) + i + 1) % 10)

    result = ''.join(input_list)
    return result

input_str = a
result = custom_add(input_str)
b=result
print(b)
```

crypto部分的签到题啦，简单分析一下题目的意思，首先是对几个函数进行分析，先把输出的结果加到514位，然后将每一位作一个加1加1再模10的操作，再将前两位去掉，将剩下的512位作倒置，然后输出

```
#include <iostream>
using namespace std;
int len = 512;
char c[512];
char ans[512];

int main() {
    for (int i = 0; i < 512; i++)
        cin >> c[i];
    for (int i = 0; i < 512; i++) {
        c[i] = char(((int)(c[i]) - '0' - i - 1 + 1000) % 10 + '0');
        ans[511 - i] = c[i];
    }
    for (int i = 0; i < 512; i++) {
        cout << ans[i];
    }
    return 0;
}
```

110011001101100011000010110011101111011001110000011100000110101001101110110000101
100110011001000011100000101101011000010011100101100101011001010010110100110100011
001010011011101100101001011010011100000110001001101100011000000101101001101110110
011000110001011000100011100001100110011001100011001100110100001101010011001001100
10001111101

经过尝试，我们补充两个0，最后获得了flag

flag{8857afd8-a9ee-4e7e-8160-7f1b8ff3452d}

Sign1n[签到]

跟上面一题的思路是一样的，得到flag即可

```
flag{8857afd8-a9ee-4e7e-8160-7f1b8ff3452d}
```

happy_to_solve1

```
from Crypto.Util.number import *
import sympy
from secrets import flag

def get_happy_prime():
    p = getPrime(512)
    q = sympy.nextprime(p ^ ((1 << 512) - 1))
    return p, q

m = bytes_to_long(flag)
p, q = get_happy_prime()
n = p * q
e = 65537
print(n)
print(pow(m, e, n))
#
248522066477505450406408680939212522828052298648624138630258732032910427990967877
892884614265557167852882864925301949011300429402791095980719580123031798236451516
377591035587371262714356366577672727039083848025283660908716530241923213987850170
73393201385586868836278447340624427705360349350604325533927890879
#
147679853994731119325441768527180611861007431174071414359943742618863967810409346
321106082194821404656712699581808498860974916531059393683957165964133525630050278
675465851911032146507908847207296011715176156202021835340219876181468622605586244
58833387692782722514796407503120297235224234298891794056695442287
```

这题主要是一个happyprime函数，生成两个512位的素数，有一定的联系，这题与之前做的一题几乎一模一样，直接套用脚本即可

```
import gmpy2
import sympy
import libnum

n =
248522066477505450406408680939212522828052298648624138630258732032910427990967877
892884614265557167852882864925301949011300429402791095980719580123031798236451516
377591035587371262714356366577672727039083848025283660908716530241923213987850170
73393201385586868836278447340624427705360349350604325533927890879
```

```

c =
147679853994731119325441768527180611861007431174071414359943742618863967810409346
321106082194821404656712699581808498860974916531059393683957165964133525630050278
675465851911032146507908847207296011715176156202021835340219876181468622605586244
58833387692782722514796407503120297235224234298891794056695442287
e = 65537
p=(gmpy2.iroot(pow(2,1024)-4*n,2)[0]+pow(2,512))//2
# p=(2**1024+gmpy2.iroot((2**1024)**2-4*n,2)[0])//2
p=int(p)
while(1):
    p=sympy.nextprime(p)
    if(n%p==0):
        print(p)
        break
q=n//p
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
flag=libnum.n2s(int(m))
print(flag)

```

然后我们就可以得到flag

```
XYCTF{3f22f4efe3bbbc71bbcc999a0a622a1a23303cdc}
```

happy_to_solve2

```

import random
from Crypto.Util.number import *
from secrets import flag

def get_happy_prime():
    while 1:
        p = int("".join([random.choice("123") for _ in range(512)]))
        q = int("".join([random.choice("567") for _ in range(512)]))
        if isPrime(p) and isPrime(q):
            return (p,q)

m = bytes_to_long(flag)
p ,q= get_happy_prime()
n = p * q
e = 65537
print(n)
print(pow(m, e, n))

```

```
#
697906506747097082736076931509594586899561519277373830451275402914416296858960649
459482027106166486723487162428522597262774248272216088755005277069446993003521270
487750989061229071167729138628583207229945902389632065500739730301375338674342457
656803764567184544685006193130563116558641331897204457729877920989968662546183628
637193220770495938729301979912328865798266631957128761871326655572836258178871966
196973138373358029531478246243442559418904559585334351259080578222274926069834941
166567112522869638854253933559832822899069320370733424453856240903784235604251466
010104012061821038897933884352804297256364409157501116832788696434711523621632436
970698827611375698724661553712549209133526623456888111161142213830821361143023186
927163314212097199831985368310770663850851571934739809387798422381702174820982531
508641022827776262236373967579266271031713520262606203067411268482553539580686495
739014567368858613520107678565628269250835478345171330669316220473129104495659093
134763261751546990704365966783697780787341963138501
```

```
#
153383826085102296581238539677668696644156148059026868813759015106139131297135097
831661048493079405226972222492151356105759235749502324303047037349410709021152255
315429280760639113724345836532087970918453353723090554450581657930847674930226113
840172368662838756446364482977092478979838209396761279326533419699056209983721842
484996150025403009644653678928025861445324715419893797015875541525590135843027312
322236085581571452084477262582966972702577136904385741443870527205640874446616413
917231260133364227248928492574610248881137364204914001412269740461851747883355414
96849927294459007162322360350169800422775335552646715567802825755799597955409228
004284739743749531270833084850113574712041224896044525292591264637452797151098802
604186311724597450780520140413704697374209653369969451501627583467893160412780732
575085846467289134920886789952338174193202234175299652687560232593212131693456966
318670843605238958724126368185289703563591477049105538528244632434869965333722691
837462591128379816582723367039674028619947057144546
```

这题的两个素数较大，且不太能进行分解，于是我们观察生成两个素数的函数，发现第一个素数由123组成，第二个素数由567组成，于是想到根据乘积末尾部分来反推两个素数，由于存在多解性，我们利用dfs来求解

```
m="697906506747097082736076931509594586899561519277373830451275402914416296858960
649459482027106166486723487162428522597262774248272216088755005277069446993003521
270487750989061229071167729138628583207229945902389632065500739730301375338674342
457656803764567184544685006193130563116558641331897204457729877920989968662546183
628637193220770495938729301979912328865798266631957128761871326655572836258178871
966196973138373358029531478246243442559418904559585334351259080578222274926069834
941166567112522869638854253933559832822899069320370733424453856240903784235604251
466010104012061821038897933884352804297256364409157501116832788696434711523621632
436970698827611375698724661553712549209133526623456888111161142213830821361143023
186927163314212097199831985368310770663850851571934739809387798422381702174820982
531508641022827776262236373967579266271031713520262606203067411268482553539580686
495739014567368858613520107678565628269250835478345171330669316220473129104495659
093134763261751546990704365966783697780787341963138501"
#temp1=3
#temp2=7
def dfs(i,temp1,temp2):
    flag=False
    if(str((int('1'+str(temp1)))*(int('5'+str(temp2)))))[-i-1:]==(m[-i-1:])):
        t1=int('1'+str(temp1))
        t2=int('5'+str(temp2))
        i=i+1
```

```

        flag=True
        dfs(i,t1,t2)
        i=i-1

    if(str((int('1'+str(temp1)))*(int('6'+str(temp2))))[-i-1:]==(m[-i-1:])):
        t1=int('1'+str(temp1))
        t2=int('6'+str(temp2))
        i=i+1
        flag=True
        dfs(i,t1,t2)
        i=i-1

    if(str((int('1'+str(temp1)))*(int('7'+str(temp2))))[-i-1:]==(m[-i-1:])):
        t1=int('1'+str(temp1))
        t2=int('7'+str(temp2))
        i=i+1
        flag=True
        dfs(i,t1,t2)
        i=i-1

    if(str((int('2'+str(temp1)))*(int('5'+str(temp2))))[-i-1:]==(m[-i-1:])):
        t1=int('2'+str(temp1))
        t2=int('5'+str(temp2))
        i=i+1
        flag=True
        dfs(i,t1,t2)
        i=i-1

    if(str((int('2'+str(temp1)))*(int('6'+str(temp2))))[-i-1:]==(m[-i-1:])):
        t1=int('2'+str(temp1))
        t2=int('6'+str(temp2))
        i=i+1
        flag=True
        dfs(i,t1,t2)
        i=i-1

    if(str((int('2'+str(temp1)))*(int('7'+str(temp2))))[-i-1:]==(m[-i-1:])):
        t1=int('2'+str(temp1))
        t2=int('7'+str(temp2))
        i=i+1
        flag=True
        dfs(i,t1,t2)
        i=i-1

    if(str((int('3'+str(temp1)))*(int('5'+str(temp2))))[-i-1:]==(m[-i-1:])):
        t1=int('3'+str(temp1))
        t2=int('5'+str(temp2))
        i=i+1
        flag=True
        dfs(i,t1,t2)
        i=i-1

    if(str((int('3'+str(temp1)))*(int('6'+str(temp2))))[-i-1:]==(m[-i-1:])):
        t1=int('3'+str(temp1))
        t2=int('6'+str(temp2))
        i=i+1

```

```

        flag=True
        dfs(i,t1,t2)
        i=i-1

    if(str((int('3'+str(temp1)))*(int('7'+str(temp2))))[-i-1:]==(m[-i-1:])):
        t1=int('3'+str(temp1))
        t2=int('7'+str(temp2))
        i=i+1
        flag=True
        dfs(i,t1,t2)
        i=i-1
    if(i==512):
        print(temp1,temp2)
        return
    if(flag==False):
        return

dfs(1,3,7)

```

求解出来由很多组解，我们再去对照前面的位数，最后获得了两个素数

然后再使用RSA解密即可，求出逆元，求出d，在正向求解即可

```

from gmpy2 import *
import libnum
import gmpy2
import random
from Crypto.Util.number import *

p=1212111131211122322312213131133323312213231111333311213113222322232211312111221
1311111122233111221112223111221331112322223333233123112232212332112312313332321
33313211133323333223111322123121332223132213231133313211122112311132311232213112
33223233311212333321231312223211233122211223233111221311211323323222232121322313
121132212231111333133122212121313112121232211212221232332131123111321323331222
311113213332112321112222213321223323221231313332212122323312231122221131113333
1123122122331232313131221113

q=575777656665655656556576565767667767566667565755756555575777656666576667565655
656757767766555755665576576776765555667775657655565766756676666767656665565677677
575575677575566765767566567757566765666677665667765657566576655676775766755665575
556755677655667565666675776766775765766575756666777655577766766767575676766676766
655755575756657666665667667757557557756776556657755775656676655776567675675655766
765575657567767656776677666577765677657767657757675757666577755555557557565655565
5657776566765775556575765677

phi=(p-1)*(q-1)
e=65537

```

```

c=1533838260851022965812385396776686966441561480590268688137590151061391312971350
978316610484930794052269722224921513561057592357495023243030470373494107090211522
553154292807606391137243458365320879709184533537230905544505816579308476749302261
138401723686628387564463644829770924789798382093967612793265334196990562099837218
424849961500254030096446536789280258614453247154198937970158755415255901358430273
123222360855815714520844772625829669727025771369043857414438705272056408744466164
139172312601333642272489284925746102488811373642049140014122697404618517478833554
149684992729445900716232236035016980042277533355526467155678028257557995979554092
280042847397437495312708330848501135747120412248960445252925912646374527971510988
026041863117245974507805201404137046973742096533699694515016275834678931604127807
325750858464672891349208867899523381741932022341752996526875602325932121316934569
663186708436052389587241263681852897035635914770491055385282446324348699653337226
91837462591128379816582723367039674028619947057144546
d=invert(e,phi)
n=6979065067470970827360769315095945868995615192773738304512754029144162968589606
494594820271061664867234871624285225972627742482722160887550052770694469930035212
704877509890612290711677291386285832072299459023896320655007397303013753386743424
576568037645671845446850061931305631165586413318972044577298779209899686625461836
286371932207704959387293019799123288657982666319571287618713266555728362581788719
661969731383733580295314782462434425594189045595853343512590805782222749260698349
411665671125228696388542539335598328228990693203707334244538562409037842356042514
660101040120618210388979338843528042972563644091575011168327886964347115236216324
369706988276113756987246615537125492091335266234568881111611422138308213611430231
869271633142120971998319853683107706638508515719347398093877984223817021748209825
315086410228277762622363739675792662710317135202626062030674112684825535395806864
957390145673688586135201076785656282692508354783451713306693162204731291044956590
93134763261751546990704365966783697780787341963138501
m=pow(c,d,n)
print(m)
flag=long_to_bytes(m)
print(flag)

```

解出flag

```
XYCTF{7f4b2241951976ce5ef6df44503209059997e5085d1bc21f6bef4d9effb29fd0}
```

happy_to_solve3

```

import gmpy2
from Crypto.Util.number import *
import random
from secrets import flag

n = 1024
rho = 2
gamma = 0.352
delta = (1 - gamma) / 2

def get_happy_prime():
    p = getPrime(n // 2)
    q = getPrime(n // 2)

```



```

N = p * q
if 2 * q - p > gmpy2.iroot(N, 3)[0] and 2 * q - p < int(N**gamma / 16):
    d = random.randint(int(N ** (delta - 0.001)), int(N**delta))
    e = gmpy2.invert(d, (p - 1) * (q - 1))
    return N, e

N, e = get_happy_prime()
m = bytes_to_long(flag)
print(N)
print(e)
print(pow(m, e, N))
#
262520792590557046276663232446026434827811478251833823563978322470880473096166170
973396967071440576532100045206129176936699321058518888771220060450723297988984995
658582283884954240129867538637146924315603797818982078845855992582229939200744016
516540207525916858674450616913948112117516831530578235916528257187
#
202935305174706906986376186864051444100197589482194720650385604617995167023220940
138899902073948844285283834058445151666398192104922822110762965750590312021079147
170573038600118139119881722125346545331027913695559130179278058419339157557267195
396326664433859683010193688491694572425231599139974726246205888139
#
661734062046475831411748478142835403893264010564909705920175778107750870171973924
566341676096800609315530177120585280567696181119258169212725713062466675275463663
3105778190535353621522097136372533808592352355902867795175526359353842316942554
50099095196595516174794966487058693308544109311774074970769293357

```

论文题

Input: N, e, δ .

1. Compute the CF expression of $\frac{e}{N - \frac{3}{\sqrt{2}}\sqrt{N+1}}$.
2. For every convergent $\frac{t_1}{d_1}$ of the expression above
if the roots of $x^2 - (N + 1 - \frac{ed_1 - 1}{t_1})x + N = 0$
are positive integers less than N
then return the roots as p, q ;
3. Return ("failure");

我们还是使用我们的连分数攻击去完成，将下面的底换一下就可以了

```

from Crypto.Util.number import *
N =
262520792590557046276663232446026434827811478251833823563978322470880473096166170
973396967071440576532100045206129176936699321058518888771220060450723297988984995
658582283884954240129867538637146924315603797818982078845855992582229939200744016
516540207525916858674450616913948112117516831530578235916528257187

```

```

e =
202935305174706906986376186864051444100197589482194720650385604617995167023220940
138899902073948844285283834058445151666398192104922822110762965750590312021079147
170573038600118139119881722125346545331027913695559130179278058419339157557267195
396326664433859683010193688491694572425231599139974726246205888139
c =
661734062046475831411748478142835403893264010564909705920175778107750870171973924
566341676096800609315530177120585280567696181119258169212725713062466675275463663
310577819053535536215222097136372533808592352355902867795175526359353842316942554
50099095196595516174794966487058693308544109311774074970769293357
import mpmath

# numerator(n):分子, denominator(d): 分母
def t_cf(n, d): # 将分数 x/y 转为连分数的形式
    res = []
    while d:
        res.append(n // d)
        n, d = d, n % d
    return res

def cf(sub_res): # 得到渐进分数的分母和分子
    n, d = 1, 0
    for i in sub_res[::-1]: # 从后面往前循环
        d, n = n, i * n + d
    return d, n

def list_fraction(x, y): # 列出每个渐进分数
    res = t_cf(x, y)
    res = list(map(cf, (res[0:i] for i in range(1, len(res))))) # 将连分数的结果逐一截取以求渐进分数
    return res

def wienerAttack(e, n):
    for (d, k) in list_fraction(e, n): # 用一个for循环来注意试探e/n的连续函数的渐进分数, 直到找到一个满足条件的渐进分数
        if k == 0: # 可能会出现连分数的第一个为0的情况, 排除
            continue
        if (e * d - 1) % k != 0: # ed=1 (mod φ(n)) 因此如果找到了d的话, (ed-1)会整除φ(n), 也就是存在k使得(e*d-1)//k=φ(n)
            continue

        phi = (e * d - 1) // k # 这个结果就是 φ(n)

        print(phi)

mpmath.mp.dps = 100
n1 = N - int(mpmath.ceil(3 / mpmath.sqrt(2) * mpmath.sqrt(N))) + 1
cc = wienerAttack(e, n1)

```

得到结果:

```
202935305174706906986376186864051444100197589482194720650385604617995167023220940
138899902073948844285283834058445151666398192104922822110762965750590312021079147
170573038600118139119881722125346545331027913695559130179278058419339157557267195
396326664433859683010193688491694572425231599139974726246205888138
262520564991961275420673875773070804282808796606754021607307335335619279893868790
647768383959491270905218236399009898325893831488921437964561453566721084486842811
658954058444408145925293802238746169279287173461489257763832041529698314457060542
129716195693163164404676090644575106456469813355541773527006765933
262520792590557046276663232446026434827811478251833823563978322470880473096166170
973396967071440576532100045206129176936699321058518888771220060450723297954614316
778794257384313010822227390070404000045908427685570416864970143425964515572245710
744222130111609693526759178286535258777660072393577806415536881976
262520792590557046276663232446026434827811478251833823563978322470880473096166170
973396967071440576532100045206129176936699321058518888771220060450723297954614316
778794257384313010822227390070404000045894108217119259056887931400362882611068288
718174599033635822637538598755282646480225909171284360275954895012
```

然后我们再——检验就可以了，找到第三个是符合条件的

```
N =
262520792590557046276663232446026434827811478251833823563978322470880473096166170
973396967071440576532100045206129176936699321058518888771220060450723297988984995
658582283884954240129867538637146924315603797818982078845855992582229939200744016
516540207525916858674450616913948112117516831530578235916528257187
e =
202935305174706906986376186864051444100197589482194720650385604617995167023220940
138899902073948844285283834058445151666398192104922822110762965750590312021079147
170573038600118139119881722125346545331027913695559130179278058419339157557267195
396326664433859683010193688491694572425231599139974726246205888139
c =
661734062046475831411748478142835403893264010564909705920175778107750870171973924
566341676096800609315530177120585280567696181119258169212725713062466675275463663
310577819053535536215222097136372533808592352355902867795175526359353842316942554
50099095196595516174794966487058693308544109311774074970769293357
from gmpy2 import *
phi=26252079259055704627666323244602643482781147825183382356397832247088047309616
617097339696707144057653210004520612917693669932105851888877122006045072329795461
431677879425738431301082222739007040400004590842768557041686497014342596451557224
5710744222130111609693526759178286535258777660072393577806415536881976
d=invert(e,phi)
m=pow(c,d,N)
print(long_to_bytes(m))
```

输出我们的flag

```
XYCTF{68f1880cdfd99fbf9a156946cb39dd86477886f1d115636e149e12c16f99af0}
```

factor1

```
import gmpy2
import hashlib
```

```

from Crypto.Util.number import *

p = getPrime(512)
q = getPrime(512)
d = getPrime(512)
e = gmpy2.invert(d, (p**3 - 1) * (q**3 - 1))
flag = "XYCTF{" + hashlib.md5(str(p + q).encode()).hexdigest() + "}"
print(e)
print(p * q)
#
172005065945326769176157335849432320425605083524943730546805772515111751580759726
759492349719668775270727323745284785341119685198468883978645793770975366048506237
371435027612758232099414404389043740306443065413069994232238075194102578269859784
981454218948784071599231415554297361219709787507633404217550013282713899284609273
532223781487419770338416653260109238572639243087280632577902857385265070736208291
583497988891353312351322545840742380550393294960815728021248513046077985900158814
037534487146730483099151396746751774427787635287611736111679074330407715700153025
952858666841328055071403960165321273972935204988906850585454805923440635864200149
69439876776539993952528995717480620593326867245714074205285828967234591508039849
777840636255379730281105670496110061909219669860172557450779495125345533232776767
292561378244884362014224844319802810586344516400297830227894063759083198761120293
919537342405893653545157892446163
#
990751853894430780083272143283287477923851538838365997530969714123773668658262540
335342938860348288042190374662461755263470140458118525319945375203030631139854860
630224449727612765314225386949150301594209894012800120252491291118716498311850478
20236417385693285461420040134313833571949090757635806658958193793

```

这道题刚开始一直在考虑求e的逆元那一步，感觉有点问题，后来我用了**低指数加密破解**

简单来说就是将逆元后面的那一串式子看成n的3次，然后对e和n的三次做连分数攻击就可以了

```

import gmpy2
n=9907518538944307800832721432832874779238515388383659975309697141237736686582625
403353429388603482880421903746624617552634701404581185253199453752030306311398548
606302244497276127653142253869491503015942098940128001202524912911187164983118504
7820236417385693285461420040134313833571949090757635806658958193793
e=1720050659453267691761573358494323204256050835249437305468057725151117515807597
267594923497196687752707273237452847853411196851984688839786457937709753660485062
373714350276127582320994144043890437403064430654130699942322380751941025782698597
849814542189487840715992314155542973612197097875076334042175500132827138992846092
735322237814874197703384166532601092385726392430872806325779028573852650707362082
915834979888913533123513225458407423805503932949608157280212485130460779859001588
140375344871467304830991513967467517744277876352876117361116790743304077157001530
259528586668413280550714039601653212739729352049889068505854548059234406358642001
49694398767765399939525289957174806205933268672457140742052858289672345915080398
497778406362553797302811056704961100619092196698601725574507794951253455332327767
672925613782448843620142248443198028105863445164002978302278940637590831987611202
93919537342405893653545157892446163
n_3=n*n*n

# numerator(n):分子, denominator(d):分母

```

```

def t_cf(n, d): # 将分数 x/y 转为连分数的形式
    res = []
    while d:
        res.append(n // d)
        n, d = d, n % d
    return res

def cf(sub_res): # 得到渐进分数的分母和分子
    n, d = 1, 0
    for i in sub_res[::-1]: # 从后面往前循环
        d, n = n, i * n + d
    return d, n

def list_fraction(x, y): # 列出每个渐进分数
    res = t_cf(x, y)
    res = list(map(cf, (res[0:i] for i in range(1, len(res))))) # 将连分数的结果逐一截取以求渐进分数
    return res

def wienerAttack(e, n):
    for (d, k) in list_fraction(e, n): # 用一个for循环来注意试探e/n的连续函数的渐进分数，直到找到一个满足条件的渐进分数
        if k == 0: # 可能会出现连分数的第一个为0的情况，排除
            continue
        if (e * d - 1) % k != 0: # ed=1 (mod φ(n)) 因此如果找到了d的话，(ed-1)会整除φ(n)，也就是存在k使得(e*d-1)//k=φ(n)
            continue

        phi = (e * d - 1) // k # 这个结果就是 φ(n)

        print(phi)

wienerAttack(e, n_3)

```

连分数破解完之后，我们得到了很多组可能的结果，我们一组一组进行验证，最后求出p1, p2, q1, q2，然后就可以解出答案了

```

import gmpy2
import hashlib
from Crypto.Util.number import *

n=9907518538944307800832721432832874779238515388383659975309697141237736686582625
403353429388603482880421903746624617552634701404581185253199453752030306311398548
606302244497276127653142253869491503015942098940128001202524912911187164983118504
7820236417385693285461420040134313833571949090757635806658958193793
p1=107549594935735464395102768293002467693731244361281709550503790419865048692217
50743052397622171703140881050431144683659643071578143360949942206693325622779
p2=921204635393037659499689008949471873689437899199138124824253231962862744968166
4076081705664941905594411935750003102856235503684466394327681725704255564467
q1=921204635393037659499689008949471873689437899199138124824253231962862744968166
4076081705664941905594411935750003102856235503684466394327681725704255564467
q2=107549594935735464395102768293002467693731244361281709550503790419865048692217
50743052397622171703140881050431144683659643071578143360949942206693325622779
flag = "XYCTF{" + hashlib.md5(str(p1 + q1).encode()).hexdigest() + "}"
print(flag)

```

运行程序, 得到flag

```
XYCTF{a83211a70e18145a59671c08ddc67ba4}
```

factor3

```
from Crypto.Util.number import *
import random

flag = b'XYCTF{*****}'
m = bytes_to_long(flag)
def gainPrime():
    while True:
        x = random.getrandbits(256)
        y = random.getrandbits(256)

        if y % 2 == 0:
            continue

        p = x ** 3 + 3 * y ** 3
        if p.bit_length() == 768 and p % 2 == 1 and isPrime(p):
            return p

p, q = gainPrime(), gainPrime()
N = p * q
phi = (p ** 2 + p + 1) * (q ** 2 + q + 1)
d = getPrime(320)
e = inverse(d, phi)
c = d**2^m

print(f"N: {N}")
print(f"e: {e}")
print(f"c: {c}")

N:
913125842482770239379848062277162627509794409924607555622246822717218133091223291
889541294440266178282194506242444509803611492259403578922020590849630191477864719
052980160940803309686069818208833547621252544423652489179493083138385424424384165
228024273745733240109761707533778691158938848158094054261174692601673435971526522
219273943464877956131040249169850420336023942653021547841666224446678539579529590
840999008107782784268926145671962239929431694391039559247
```

e:

```
494518390582436635999115147756676313570637682518235195828939117782099618734167908
630788943568232122157772909140885391963441876427590731524706959546524212914108888
799081844320513851526790475333924396837458796755678072486028072639014677580265244
176441153444956871730684233063789931539669072735599696830757690822185323538738397
827461580678488181113667710378657058297572328491762536595872579603698945272140918
157163640403488075948987156585480146162739943419183496337465468187233821931312507
662218106713861638334075899266373256620752680354704533272722692596941861606161634
082613228896420520465402725359166156632884432690715903666803067996854084671477445
131853993177110154928274312496230096270510089973592664248613332000290545537840595
645944390047611474888693558676781309912289044962293014118087259307560444929227407
113819165713213046898243995956550944640168932947118400215917515277554126694376415
569909534496134700668701465649939
```

c:

```
445093133736946148210694599254213355758596289403050506511087038911256532987550295
276218237292611703737321050951657095848360656627436984055113238112866574426616579
2377925899683228751870742727716
```

这题跟factor1是一样的，先是用wiener attack，再穷举求出d就可以了，再作异或操作即得到flag

```
import gmpy2
n=9131258424827702393798480622771626275097944099246075556222468227172181330912232
918895412944402661782821945062424445098036114922594035789220205908496301914778647
190529801609408033096860698182088335476212525444236524891794930831383854244243841
652280242737457332401097617075337786911589388481580940542611746926016734359715265
222192739434648779561310402491698504203360239426530215478416662244466785395795295
90840999008107782784268926145671962239929431694391039559247
e=4945183905824366359991151477566763135706376825182351958289391177820996187341679
086307889435682321221577729091408853919634418764275907315247069595465242129141088
887990818443205138515267904753339243968374587967556780724860280726390146775802652
441764411534449568717306842330637899315396690727355996968307576908221853235387383
978274615806784881811136677103786570582975723284917625365958725796036989452721409
181571636404034880759489871565854801461627399434191834963374654681872338219313125
076622181067138616383340758992663732566207526803547045332727226925969418616061616
340826132288964205204654027253591661566328844326907159036668030679968540846714774
451318539931771101549282743124962300962705100899735926642486133320002905455378405
956459443900476114748886935586767813099122890449622930141180872593075604449292274
071138191657132130468982439959565509446401689329471184002159175152775541266943764
15569909534496134700668701465649939
n_2=n*n

# numerator(n):分子, denominator(d): 分母
def t_cf(n, d): # 将分数 x/y 转为连分数的形式
    res = []
    while d:
        res.append(n // d)
        n, d = d, n % d
    return res

def cf(sub_res): # 得到渐进分数的分母和分子
    n, d = 1, 0
    for i in sub_res[::-1]: # 从后面往前循环
```

```

        d, n = n, i * n + d
    return d, n
def list_fraction(x, y):    # 列出每个渐进分数
    res = t_cf(x, y)
    res = list(map(cf, (res[0:i] for i in range(1, len(res))))) # 将连分数的结果逐一截取以求渐进分数
    return res

def wienerAttack(e, n):
    for (d, k) in list_fraction(e, n): # 用一个for循环来注意试探e/n的连续函数的渐进分数，直到找到一个满足条件的渐进分数
        if k == 0: # 可能会出现连分数的第一个为0的情况，排除
            continue
        if (e * d - 1) % k != 0: # ed=1 (mod  $\phi(n)$ ) 因此如果找到了d的话，(ed-1)会整除 $\phi(n)$ ，也就是存在k使得(e*d-1)//k= $\phi(n)$ 
            continue

        phi = (e * d - 1) // k # 这个结果就是  $\phi(n)$ 

    print(phi)

wienerAttack(e,n_2)

```

解出所有的可能

494518390582436635999115147756676313570637682518235195828939117782099618734167908
630788943568232122157772909140885391963441876427590731524706959546524212914108888
799081844320513851526790475333924396837458796755678072486028072639014677580265244
176441153444956871730684233063789931539669072735599696830757690822185323538738397
827461580678488181113667710378657058297572328491762536595872579603698945272140918
157163640403488075948987156585480146162739943419183496337465468187233821931312507
662218106713861638334075899266373256620752680354704533272722692596941861606161634
082613228896420520465402725359166156632884432690715903666803067996854084671477445
131853993177110154928274312496230096270510089973592664248613332000290545537840595
645944390047611474888693558676781309912289044962293014118087259307560444929227407
113819165713213046898243995956550944640168932947118400215917515277554126694376415
569909534496134700668701465649938
989036781164873271998230295513352627141275365036470391657878235564199237468335817
261577887136464244315545818281770783926883752855181463049413919093048425828217777
598163688641027703053580950667848793674917593511356144972056145278029355160530488
352882306889913743461368466127579863079338145471199393661515381644370647077476795
654923161356976362227335420757314116595144656983525073191745159207397890544281836
314327280806976151897974313170960292325479886838366992674930936374467643862625015
324436213427723276668151798532746513241505360709409066545445385193883723212323268
165226457792841040930805450718332313265768865381431807333606135993708169342954890
263707986354220309856548624992460192541020179947185328497226664000581091075681191
291888780095222949777387117353562619824578089924586028236174518615120889858454814
227638331426426093796487991913101889280337865894236800431835030555108253388752831
139819068992269401337402931299877
834499784107861823248506811839391279150451089249521892961334761257293106613908345
814456342271391706141241784175244098938308166471559359447942994234759609292558749
848450612290867124451458927125997419663211719525206747320172372578337268416697599
547744446438364721045529643295145509473191560241324488401903603262437733471621046
333841417394948805629314261263983785877153304329849280505534978081241970146737799
390213643180886128163915826737997746649623654519872150069472977565957074509089856
679993055079641514688753080012004870547520148098563899897719543757339391460397757
514409823762709628285367099043592889317992480165583087437730177244691267883118188
660003613486373386441462902337388287456485776830437620919534997750490295595106005
152531158205344363874670380267068460476987763373869461324272250081508250818071249
504569842141047016640786743176679719080285074348262300364360807030872588796760201
274222339462227307378433723284272
833798804209868926885597743428680923994376689883858923648251158413609393250010581
389051354245825579589148843110244363516749941829705615281705885230450674458287010
099343692585025388132084483695633586838705373864316786817375599573951862278045946
006150851323308670520946217691496264921675220721213447466068569831512505491584734
807189423945368087744747402041988282527302954157122933011299755788279613894864647
286144803716494106382662520738127669495201009984205649985714336107675613477358442
423855011266706320679789186508955691828275482443228137631311692451634084140175322
596818203008684232335802209167702814849777136786055309765737995819234264295466468
414424886749653691547737803992981154373858670285950028806340656537153716451114378
660962319338320775240176128965474201402381863526217391037528765336199252413877793
657795095634278978406411712164357046958884719590893726787739613415378848305826544
330558861623813812000519592914963

```
833798804209868926885597743428680923994376689883858923648251158413609393250010581
389051354245825579589148843110244363516749941829705615281705885230450674458287010
099343692585025388132084483695633586838705373864316786817375599573950104127077186
395659995392757752995882899938289080362111941272988091485639339559843502059915205
444870110902918436499616297599187963509517327010042875646498204271294817693848615
863916356218338119052317606715052301065698671485777819689151877833072309866131554
613724414897565192219044775533159747352618965075724249708246214305354069900520387
953575676413961901660414173017690214722328166929435552701525033444467609524741628
219332675647717566112424485499891287720591551300843895993070967137359177985875585
111358639075519420435411163829529375854064770748125965898317391058461098059863041
945954210826787800935108724310961342056565354878553782400682837769329116302566267
706973304162817085732088023207009
```

然后一个一个试，求出逆元d，d需要满足是320位二进制，即93位十进制

```
d=2109723047551375043305134722302342646596769444055829710618826161103186815230448
177424794300667429
```

然后就进行d的平方操作，再和c作异或就可以了，得到flag

```
XYCTF{I_love_to_read_the_crypto_paper_and_try_to_ak_them}
```

babyRSAMAX

```
from Crypto.Util.number import *
from gmpy2 import *
from random import choice

flag = b'XYCTF{*****}'
e = '?'

def getBabyPrime(nbits):
    while True:
        p = 1
        while p.bit_length() <= nbits:
            p *= choice(sieve_base)

        if isPrime(p+1):
            return p+1

p = getBabyPrime(512)
q = getBabyPrime(512)
n = p*q
gift1 = (pow(p,e,n)-pow(q,e,n)) % n
gift2 = pow(p+q,e,n)

t = 65537
x = bytes_to_long(e)
y = pow(x, t, n)

m = bytes_to_long(flag)
c = powmod(m, e, n)
```

```

print(f'n = {n}')
print(f'gift1 = {gift1}')
print(f'gift2 = {gift2}')
print(f'c = {c}')
print(f'y = {y}')

...

n =
393324238727402107832460690308559462441049823811571668439775997802339111831585609
013773599254350923266533039642615501586585515186260140487834352454715369598448740
365169315424447195499979714826449055234594077753927022110861492794737847962020202
81909706723380472571862792003687423791576530085747716706475220532321
gift1 =
454940244474633832734900723581818779395028510509172616757355241267841675969466016
695678275563144727166210856408438209856299995022870830090220157158341911629993226
447838119703440233848187293757617219720251977078245834360606054469460885284422840
0457232100904217062914047342663534138668490328400022651816597367310
gift2 =
111061215998959709920736448050860427855012026815376672067601244053580566359594802
604251992986382187891022583247997994146019970445247509119719411310760491983876636
264003942870756402328634092146799825005835867245563420135253048223898334460067523
975023732153230791136870324302259127159852763634051238811969161011462
c =
169389278252344072670260175610454902656984918408149294321528397450359461187437145
666233150338026810090176955263743973703439843609979031658425914142031971849465884
703557289849125220407446919748196301181639762592469415790636878579941933095541298
16268931672391946592680578681270693589911021465752454315629283033043
y =
181365000127096770984130649129771690896942524888851098510938188127036275503138556
492786931311254053478085396634104452685670558902029504847330576208878699244635006
002488111774104126039140596281718267442171523919721127466845094766639459412176433
3794138308442124114744892164155894256326961605137479286082964520217

...

```

先是利用生成n的漏洞，破解出p和q

```

p=2364384004775215979229504451537962651990724045771831909531148051705228759045517
80358338769440558816351105253794964040981919231484098097671084895302287425479
q=1663537893730573521952685751683977503626438222012535089410528359454206249832164
56266478176579651490080696973849607356408696043718492499993062863415424578199

```

然后我们就可以解出e了，利用求逆元的方式，求出e

```

from gmpy2 import *
import libnum
import gmpy2
import random
from Crypto.Util.number import *

n=3933242387274021078324606903085594624410498238115716684397759978023391118315856
090137735992543509232665330396426155015865855151862601404878343524547153695984487
403651693154244471954999797148264490552345940777539270221108614927947378479620202
0281909706723380472571862792003687423791576530085747716706475220532321

```

```

p=2364384004775215979229504451537962651990724045771831909531148051705228759045517
80358338769440558816351105253794964040981919231484098097671084895302287425479
q=1663537893730573521952685751683977503626438222012535089410528359454206249832164
56266478176579651490080696973849607356408696043718492499993062863415424578199
c=1813650001270967709841306491297716908969425248888510985109381881270362755031385
564927869313112540534780853966341044526856705589020295048473305762088786992446350
060024881117741041260391405962817182674421715239197211274668450947666394594121764
333794138308442124114744892164155894256326961605137479286082964520217
phi=(p-1)*(q-1)
e=65537
d=invert(e,phi)
m=powmod(c,d,n)
print(m)
print(long_to_bytes(m))
print(phi)

```

求解出

```
b'XYCTF{e==4096}'
```

说明我们已经求解出了e的值为4096

然后再进一步，但是我们发现，下一个mod似乎没有整数解，因为e和phi不互素（具体参考[e与phi不互素的情况 e和phi不互素-CSDN博客](#)）

我们使用有限域上的解的方式，用sage环境分别解出p和q的有限域，然后再用CRT把两个解合起来，查找是否存在字符XYCTF即可

```

[(2364384004775215979229504451537962651990724045771831909531148051705228759045517
80358338769404214275971858584747720119200208117068403781208566503489403215434,
1), (36344540379246669047243921781711114415694316462518391812884210045, 1)]
[(1663537893730573521952685751683977503626438222012535089410528359454206249832164
56266478176543306949701450304802363434626984929302798183530544471602540368154,
1), (36344540379246669047243921781711114415694316462518391812884210045, 1)]

```

代码：

```

res1=
[(2364384004775215979229504451537962651990724045771831909531148051705228759045517
80358338769404214275971858584747720119200208117068403781208566503489403215434,
1), (36344540379246669047243921781711114415694316462518391812884210045, 1)]
res2=
[(1663537893730573521952685751683977503626438222012535089410528359454206249832164
56266478176543306949701450304802363434626984929302798183530544471602540368154,
1), (36344540379246669047243921781711114415694316462518391812884210045, 1)]

#循环使用中国剩余定理
for i in res1:
    for j in res2:
        m = libnum.solve_crt([int(i[0]),int(j[0])],[p,q])
        flag = long_to_bytes(m)
        if b'XYCTF' in flag:
            print(flag)

```

输出结果

```
XYCTF{Rabin_is_so_bigggg!}
```

Complex_dlp

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

class Complex:
    def __init__(self, re, im):
        self.re = re
        self.im = im

    def __mul__(self, c):
        re_ = self.re * c.re - self.im * c.im
        im_ = self.re * c.im + self.im * c.re
        return Complex(re_, im_)

    def __str__(self):
        if self.im == 0:
            return str(self.re)
        elif self.re == 0:
            if abs(self.im) == 1:
                return f"{'-' if self.im < 0 else ''}i"
            else:
                return f"{self.im}i"
        else:
            return f"{self.re} {'+' if self.im > 0 else '-'} {abs(self.im)}i"

def complex_pow(c, exp, n):
    result = Complex(1, 0)
    while exp > 0:
        if exp & 1:
            result = result * c
            result.re = result.re % n
            result.im = result.im % n
        c = c * c
        c.re = c.re % n
        c.im = c.im % n
        exp >>= 1
    return result


flag = flag.strip(b"XYCTF{").strip(b"}")
p =
112723685494221574448217085928424568492250781847843931942888858489892752057957902
7
g = Complex(3, 7)
x = bytes_to_long(flag)
```

```
print(complex_pow(g, x, p))
# 5699996596230726507553778181714315375600519769517892864468100565238657988087817
+
198037503897625840198829901785272602849546728822078622977599179234202360717671908
i
```

这道题根据hint，首先考虑**将复数域转化成实数域**，方法就是**将两边同时乘上对应的共轭复数**，这样两边就变成整数了

```
p
=11272368549422157444821708592842456849225078184784393194288885848989275205795790
27
g = 3**2 + 7**2
c=5699996596230726507553778181714315375600519769517892864468100565238657988087817
**2+19803750389762584019882990178527260284954672882207862297759917923420236071767
1908**2
```

然后就很简单了，用sage求一下离散对数，在转一下long_to_bytes就搞定了

 29391b7976096860a2c172ece0fa208

求出flag是

```
XYCTF{__c0mp13x_d1p_15_3@5y_f0r_y0u__}
```

反方向的密码 相思

```
from Crypto.Util.number import *
import hashlib
from secrets import flag

def hash(x):
    return hashlib.sha256(x.encode()).digest()

def pad(message):
    return message + hash(str(len(message)))

m = bytes_to_long(pad(flag))
p = getStrongPrime(512)
q = getStrongPrime(512)
n = p * q
e = 3
print(pow(m, e, n))
print(n)
```

```
#
120440199294949712392334113337541924034371176306546446428347114627162894108760435
789068328282135879182130546564535108930827440004987170619301799710272329673259390
065147556073101312748104743572369383346039000998822862286001416166288971531241789
864076857299162050026949096919395896174243383291126202796610039053
#
143413213355903851638663645270518081058249439863120739973910994223793329606595495
141951165221740599158773181585002460087410975579141155680671886930801733174300593
785562287068287654547100320094291092508723488470015821072834947151827362715749438
612812148855627557719115676595686347541785037035334177162406305243
```

这题的主要思路是，利用coppersmith来进行高低位的攻击，因为我们知道，flag的前几位必然是XYCTF{，所以相当于前面的m的位数已经知道了，包括最后的位数，也是可以确定的，所以我们就直接猜测flag的位数，然后逐一验证，看是否有正确的m。

验算到38时终于出现了答案，说明**flag的长度是38**

对应的m的值为

58964190787951927773278389967057377362495121527440001979648729026891046689

我们将其long_to_bytes即可，得到最后的flag

```
XYCTF{!__d3ng__hu0__1@n__3h@n__Chu__!}
```

fakeRSA

```
from Crypto.Util.number import *

flag = b'XYCTF{*****}'
n = ZZ(bytes_to_long(flag))
p = getPrime(int(320))
print(p)

G = Zmod(p)

def function(X, Y, Z):
    def part(a, b, c):
        return vector([9 * a - 36 * c, 6 * a - 27 * c, b])
    def parts(n):
        Gx.<a, b, c> = G[]
        if n == 0: return vector([a, b, c])
        mid = parts(n // 2)
        result = mid(*mid)
        if n % 2 == 0: return result
        else: return part(*result)
    return parts(n)(X, Y, Z)

print(function(69, 48, 52))

#18497904729112673660453924568931260926987433082915122206570061299009611688118988
22553602045875909
```

```
#
(14319959658136174158606957484306445701189599912713951109955347046292413095975720
03500157255135707,
101156589113061173660082261838280146550665197237341096220581057007587080432597437
7971089090196019,
784497518859893244278116222363814433595961164446277297084989532659832474887622082
585456138030246)
```

这题是用jordan型正定矩阵来解决 (根据hint)

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix} * \begin{pmatrix} 9 & 6 & 0 \\ 0 & 0 & 1 \\ -36 & -27 & 0 \end{pmatrix}^{n-3} = \begin{pmatrix} a_{n-2} & b_{n-2} & c_{n-2} \\ a_{n-1} & b_{n-1} & c_{n-1} \\ a_n & b_n & c_n \end{pmatrix}$$

先尝试了一维的jordan矩阵, 发现不出来, 然后我们用 $n*n$ 型来进行计算, 将原jordan矩阵填充到 $3*3$ 型

我们先求出第二组和第三组解, 根据前面的公式就可以得出来

```
b2 = vector(GF(p),
[14319959658136174158606957484306445701189599912713951109955347046292413095975720
03500157255135707, 101156589113061173660082261838280146550665197237341096220581057
0075870804325974377971089090196019, 7844975188598932442781162223638144335959611644
46277297084989532659832474887622082585456138030246])
b1 = vector(GF(p), inv(b2))
b0 = vector(GF(p), inv(b1))
```

在解出前三组解之后, 我们就可以列出最后的表达式了

```
a = matrix(GF(p), [a0, a1, a2])
b = matrix(GF(p), [b0, b1, b2])
x = a.solve_right(b)
G = matrix(GF(p), [[9, 6, 0], [0, 0, 1], [-36, -27, 0]])
J, P = G.jordan_form(transformation=True)
H = ~P*x*P
long_to_bytes(int(3*H[0][1]/H[0][0])+3)
```

这样就解出了最后的结果

```
XYCTF{y0u_finally_f0und_t3h_s3cr3ts!!}
```

x0r

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
import os

flag = open("flag.txt", "rb").read()
key = os.urandom(16)
iv = os.urandom(16)
flag = pad(flag, 16)
```



```

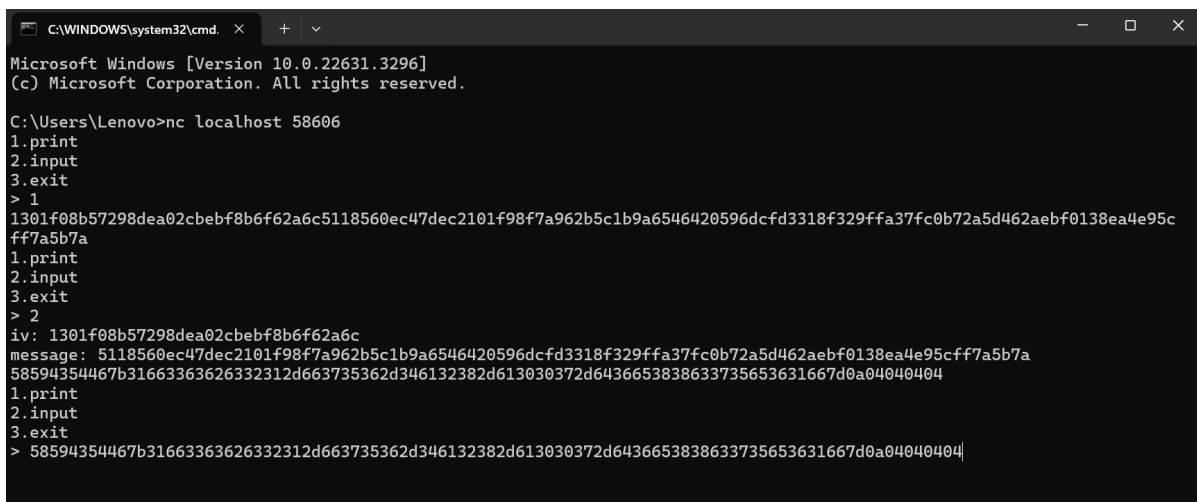
def aes_encrypt(key, plaintext):
    cipher = AES.new(key, AES.MODE_ECB)
    return cipher.encrypt(plaintext)

def encrypt(key, plaintext, iv):
    ciphertext = b""
    for i in range(0, len(plaintext), AES.block_size):
        key_block = aes_encrypt(key, iv)
        ciphertext_block = bytes(
            [plaintext[i + j] ^ key_block[j] for j in range(AES.block_size)]
        )
        ciphertext += ciphertext_block
        iv = key_block
    return ciphertext

while 1:
    try:
        print("1.print\n2.input\n3.exit")
        a = input("> ")
        if a == "1":
            print((iv + encrypt(key, flag, iv)).hex())
        elif a == "2":
            ivs = bytes.fromhex(input("iv: "))
            inputs = bytes.fromhex(input("message: "))
            print(encrypt(key, inputs, ivs).hex())
        elif a == "3":
            exit(0)
        else:
            print("You need input 1,2,3")
    except:exit(0)

```

这题的话就是一个AES加密体制，AES加密的特点就是对称加密，就是说每次加密，明文都是一样的，所以我们可以通过明文去破解，输出的是128位密码，但是我们测试 `iv = os.urandom(16)` 发现生成的iv是32位的，所以我们将前32位当成iv输入，后面的96位当作message输入，这样就能解出公钥



```

C:\WINDOWS\system32\cmd. X + -
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Lenovo>nc localhost 58606
1.print
2.input
3.exit
> 1
1301f08b57298dea02cbebf8b6f62a6c5118560ec47dec2101f98f7a962b5c1b9a6546420596dcfd3318f329ffa37fc0b72a5d462aebf0138ea4e95c
ff7a5b7a
1.print
2.input
3.exit
> 2
iv: 1301f08b57298dea02cbebf8b6f62a6c
message: 5118560ec47dec2101f98f7a962b5c1b9a6546420596dcfd3318f329ffa37fc0b72a5d462aebf0138ea4e95cff7a5b7a
58594354467b31663363626332312d663735362d346132382d613030372d6436653838633735653631667d0a04040404
1.print
2.input
3.exit
> 58594354467b31663363626332312d663735362d346132382d613030372d6436653838633735653631667d0a04040404

```

公钥是:

58594354467b31663363626332312d663735362d346132382d613030372d6436653838633735653
631667d0a04040404

然后我们对其进行逆向解密即可, 获得flag

```
XYCTF{1f3cbc21-f756-4a28-a007-d6e88c75e61f}
```

重生之我要当oi爷(未解决)

```
def f(a, b, p):
    t = 0
    for i in range(len(b)):
        t += pow(a, i, p) * b[i] % p
    return t % p

p = 1041231053

a = open("flag.png", "rb").read()
b = open("enc.txt", "w")
for i in range(len(a)):
    b.write(str(i) + str(f(i, a, p)) + "\n")
```

Complex_rsa

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

class Complex:
    def __init__(self, re, im):
        self.re = re
        self.im = im

    def __mul__(self, c):
        re_ = self.re * c.re - self.im * c.im
        im_ = self.re * c.im + self.im * c.re
        return Complex(re_, im_)

    def __str__(self):
        if self.im == 0:
            return str(self.re)
        elif self.re == 0:
            if abs(self.im) == 1:
                return f"{'-' if self.im < 0 else ''}i"
            else:
                return f"{self.im}i"
        else:
            return f"{self.re} + {self.im}i"
```

```

        return f"{self.re} {'+' if self.im > 0 else '-'} {abs(self.im)}i"

def complex_pow(c, exp, n):
    result = Complex(1, 0)
    while exp > 0:
        if exp & 1:
            result = result * c
            result.re = result.re % n
            result.im = result.im % n
        c = c * c
        c.re = c.re % n
        c.im = c.im % n
        exp >>= 1
    return result

m = bytes_to_long(flag)
key = getRandomNBitInteger(m.bit_length())
c = m ^ key
com = Complex(key, c)
p = getPrime(512)
q = getPrime(512)
e = 9
enc = complex_pow(com, e, p * q)
print(enc)
print(Complex(p, q) * Complex(p, q))
#
663509315281859813236494772639008445644945287478024372448892293435206485621649509
144334066045800380187657831835692767432398886689129489773701630462579173217424557
728527795515694461558273684532624793701033262862971641055991310908813061085463417
85251895116423206455175290083968296281375908109039893280371271943 +
652667306841292698066560188282651873840026566332312863371301783905179246117516979
653957449445413297935036178568967064398092417452068393281243486934867416561308905
938954366618576885229778664388055491449042965968872182754405428528871480718371534
36265722614658566275517205322945316112048487893204059562830581004i
#
-28814875173103880290298835537218644402443395484370652510062722255203946330565951
328874411874019897676900075613671629765922970689802650462822117767927082712245512
492082864958877932682404829188622269636302484189627580600076246836248427780151681
898051243380477561480415972565859837597822263289141887833338111120 +
235362412848885579543400940934854106052672292040465052424316433330114813432317923
674803623227280862945857543620663672974955235166884830751834386990766053503640556
408758413592161122945636548462064584183165189050320898315823173824074873376450569
212651128285746330837777597290934043912373820690250920839961482862i

```

首先就是根据我们给出的条件，将我们的p和q都计算出来，这个不难

然后根据我们在复数域上的欧拉函数，我们可以得到：

$$\phi = (p - 1) \times (q - 1)$$

然后我们进行计算，但是发现我们的phi和e居然有公因数3，这样我们的逆元就不存在了

但是我们可以升次，通过升次去找到我们可能的解

这里以模p为例：

$$enc = com^9 \pmod{p}$$

虽然不能直接求e关于 p^2-1 的逆元，但我们可以求出一个t使得它满足：

$$9t = 3 \pmod{p^2 - 1}$$

也就是说我们可以拥有：

$$com^3 = enc^t \pmod{p}$$

把com的实部虚部展开得到：

$$com^3 = (k + ci)^3 = (k^3 - 3kc^2) + (3k^2c - c^3)i \pmod{p}$$

而这个实部与虚部其实也就提供了关于k、c模p下的两个等式，也就是：

$$f_1 = k^3 - 3kc^2 - re$$

$$f_2 = 3k^2c - c^3 - im$$

所以可以resultant消去其中一个变量，然后再在模p下求根，就得到了k、c的候选值；对于模q下同理，然后将所有可能值crt起来并进行异或，检查是否含flag头即可。

```
from Crypto.Util.number import *
from gmpy2 import *

e = 9
pq2 =
235362412848885579543400940934854106052672292040465052424316433330114813432317923
674803623227280862945857543620663672974955235166884830751834386990766053503640556
408758413592161122945636548462064584183165189050320898315823173824074873376450569
21265112828574633083777597290934043912373820690250920839961482862
p2_q2 =
-28814875173103880290298835537218644402443395484370652510062722255203946330565951
328874411874019897676900075613671629765922970689802650462822117767927082712245512
492082864958877932682404829188622269636302484189627580600076246836248427780151681
898051243380477561480415972565859837597822263289141887833338111120

class Complex:
    def __init__(self, re, im):
        self.re = re
        self.im = im

    def __mul__(self, c):
        re_ = self.re * c.re - self.im * c.im
        im_ = self.re * c.im + self.im * c.re
        return Complex(re_, im_)

    def __str__(self):
        if self.im == 0:
            return str(self.re)
        elif self.re == 0:
            if abs(self.im) == 1:
```

```

        return f"{'-' if self.im < 0 else ''}i"
    else:
        return f"{self.im}i"
    else:
        return f"{self.re} {'+' if self.im > 0 else '-'} {abs(self.im)}i"

def complex_pow(c, exp, n):
    result = Complex(1, 0)
    while exp > 0:
        if exp & 1:
            result = result * c
            result.re = result.re % n
            result.im = result.im % n
        c = c * c
        c.re = c.re % n
        c.im = c.im % n
        exp >>= 1
    return result

p =
102055094560408234537828832918248297058162984509923361159025256765109863415324862
74067943978039013674207011185602314114359146043975207543018267242312660911
q = pq2 // 2 // p

com =
Complex(6635093152818598132364947726390084456449452874780243724488922934352064856
216495091443340660458003801876578318356927674323988866891294897737016304625791732
174245577285277955156944615582736845326247937010332628629716410559913109088130610
8546341785251895116423206455175290083968296281375908109039893280371271943 ,
652667306841292698066560188282651873840026566332312863371301783905179246117516979
653957449445413297935036178568967064398092417452068393281243486934867416561308905
938954366618576885229778664388055491449042965968872182754405428528871480718371534
36265722614658566275517205322945316112048487893204059562830581004)
re = int(complex_pow(com, inverse(3,(p^2-1)//3), p).re)
im = int(complex_pow(com, inverse(3,(p^2-1)//3), p).im)

PR.<a,b> = PolynomialRing(Zmod(p))
f1 = a^3 - 3*a*b^2 - re
f2 = 3*a^2*b - b^3 - im
h = f1.sylvester_matrix(f2, a).det()
res1 = h.univariate_polynomial().monic().roots()
print(res1)

re = int(complex_pow(com, inverse(3,(q^2-1)//3), q).re)
im = int(complex_pow(com, inverse(3,(q^2-1)//3), q).im)
PR.<a,b> = PolynomialRing(Zmod(q))
f1 = a^3 - 3*a*b^2 - re
f2 = 3*a^2*b - b^3 - im
h = f1.sylvester_matrix(f2, a).det()
res1 = h.univariate_polynomial().monic().roots()
print(res1)

```

这样就可以找出我们所有可能的解

然后用我们的CRT进行遍历，找出带flag头的就可以了

```
from Crypto.Util.number import *
from sympy.ntheory.modular import crt
#M = crt(n,c)[0]

pq2 =
235362412848885579543400940934854106052672292040465052424316433330114813432317923
674803623227280862945857543620663672974955235166884830751834386990766053503640556
408758413592161122945636548462064584183165189050320898315823173824074873376450569
21265112828574633083777597290934043912373820690250920839961482862
p2_q2 =
-28814875173103880290298835537218644402443395484370652510062722255203946330565951
328874411874019897676900075613671629765922970689802650462822117767927082712245512
492082864958877932682404829188622269636302484189627580600076246836248427780151681
898051243380477561480415972565859837597822263289141887833338111120
p =
102055094560408234537828832918248297058162984509923361159025256765109863415324862
74067943978039013674207011185602314114359146043975207543018267242312660911
q = pq2 // 2 // p

resp =
[(5741279734159523094711430662094390392066954482529914882030709450295074947781529
110509321968864008611691202383770840412010325910906148313511187473860891035, 1),
(44299277397806470981631822456958501117374743610734722164895140774521428277835321
66774438977867801037416377640995299677179237078298139871084989841206362410, 1),
(34301982100653260908270384034589202011869607388949017382302148763768565967424996
784183031307204025099431160836174025169583054770919358422089927245407466, 1)]

resq =
[(1073994073174582408141838171590134290130915976285654336264039690617110799629592
5365437945464907432967788383551341616951552057327658847904490010061152434332, 1),
(85865493636266108549525907337940440718995459152612166352068636212559156650435560
27382073545902963458915586846862230166123268280750109862724278599667847493, 1),
(70454470285672926934640368218960485370017780223499393128805999678254983011900147
24113928977446846014453607030578761699581827447107524459504793000974080961, 1),
(52769016690078624596064293114184167910670074121603973232337280607644828382725228
31695179159225558247215591099096136610833622276098301992005638773878483349, 1),
(40733059083434340014322347005057037395005380137014670775312176409065291937730379
71902079461771703238283853345951042314327641038705737553746426200870882295, 1),
(37357993339485442981178753995204212561692395192491200009074644073340654744189815
28427034590769440802753611282812668144292181442455716588786153175184716817, 1),
(37220394723685113176943138245817791190191950657328529209386804191399498996872119
60294412748248774573691326211744547172363931044270883035467641541946900209, 1),
(25322035732841158399436807886077082046027701207901897552049539874761118299194966
68633934893315585793821873529667573847786200205063152150526940602177115763, 1),
(41239177774976292234815240220615183818665656263203360896554485864851707291617876
4607518361571369361991330463978453617074285039619075164749001716157536065, 1)]

cp = []
cq = []
for i in resp:
    cp.append(i[0])

for i in resq:
```

```

cq.append(i[0])

KEY = []
for i in cp:
    for j in cq:
        n = [p,q]
        c = [i,j]
        key = crt(n,c)[0]
        KEY.append(key)

resp =
[(8001885192071749859699066040025029961395167917875583646644249565155819105797467
000967544636712092306411149367493633723366623965498649673837196063800580404, 1),
(16624355597809940043868421222045668368027940606674497538071745987398304371288281
79292947062605594949711892539092095892832053999186054026591407366803020136, 1),
(54118870418807958969697512959523290761833647244930271545110151261533679860619109
3807452278721326418083969279016584498160468079290503842589663811709060371, 1)]
resq =
[(8503432605028099223082740882529347146997316989367545950492333769305549393260049
047195629451308480848275471020811688622917531588079409913760096873405190234, 1),
(83666277596340512903332680120046023524050027403631361779004462984412009001641361
68489014946546974693508397220434695162939560760156985163750847606373150919, 1),
(78383363891459617889460082075885323603442132239575906039350326314020619395301408
01549439038388256401440065798242555755132126684810445868962728928464941121, 1),
(68573252805409661552095716296226735166287292330461532169192460049178262960641859
51208599616644606134509118662339264343895819605897243323286865301166906602, 1),
(61922290646588287210728389546818587299756254676361978703619448670143388423342777
05562409203724381687673713439770131476110414702628279278489497356226657489, 1),
(41216940296640419789277971652215466189392685891752978935136509419015213288874232
07726634825943843128912526687991267349501555811582062174969259522939168771, 1),
(24755867051769089110546279123148729885706808328539051599405631775137982316915601
11739604991279968415146174329518843070479843829399895584496027950700885139, 1),
(12172216203376189172437878047044987657740785418719442355017666584646843966575305
47334482771121693798142550712746858805371783011221995724541464038836673061, 1),
(55212540445548148310705512976368397912097477646198888894446552056119694292762230
1688292358201469351307145490177725937586378107953031679744096093896423948, 1)]
cp = []
cq = []
for i in resp:
    cp.append(i[0])

for i in resq:
    cq.append(i[0])

cc = []
for i in cp:
    for j in cq:
        n = [p,q]
        c = [i,j]
        cc = crt(n,c)[0]
        CC.append(cc)

for i in CC:
    for j in KEY:
        if(b"flag" in long_to_bytes(i ^ j)):

```

```
print(long_to_bytes(i ^ j))
print(len(cc))
```

这样就得到了我们的flag

```
flag{Complex_is_so_fun_and_I_think_you_know_Sylvester!}
```

Random_rr(未解决)

```
from Crypto.Util.number import *

from random import randint
flag=b'XYCTF{uuid}'
flag = bytes_to_long(flag)
n =
472993274721871037103726599805149366727531552333249750035977291933239067588481589
544777397613192273114354221827196579379954069925604091911249655707080927769808587
176515295614018992848517984372306879552247519117116110554431341268358177159108949
791969262793325836353834899335531293329721598226413212541536002401507477776699642
64734857611144570219748344977741566350285229621935507081895389023444249054515395
783080003733803406382744631528246608154546123270319561514117323480441428953306734
274538511770278887429407127143049023747710881993279361892937905382946820141513009
017756811296722630617325141162244806884220212939955235410280899112731530527048274
396186038160728562551536558223235783656985493518204710943916486379681906506757757
594165379493317173050550893487151879681122510523721157284728808336110950008840684
602353984682117748018347433177541603140491131603068512706893984834735290809952944
273565203183330739252949245209529232254867201402656024997949207918675051941911990
640248052951780195402390132237903538546705181463959793972284823588987652138458328
270662652334799233015314673544813649692428544375538627858921763941533600553536579
901589575693816746953261108022490849251974419402753031545629158199093099096735356
165044275617408697
rr =
118981410783452002362640814675858994572248094171084573145080724137925990393324395
47789237898270544336909458761754683941320649771736625000667170176071314483
def generate():
    fw = open("random", "w")
    for i in range(648):
        fw.write(str(random.getrandbits(32))+ "\n")
    fw.close()
generate()
key = str(random.getrandbits(32))
key1= int(str(key)[0])
ks = [randint(0, rr*(i+key1-1)) for i in range(16)]
c1 = pow(sum(k*flag**i for i, k in enumerate(ks)), 127, n)
c2 = pow(flag, 65537, n)
ks = [pow(69, k+key, rr*(i+key1)) for i, k in enumerate(ks)]
print(f"{ks = }")
print(f"{c1 = }")
print(f"{c2 = }")
```


反方向的密码 情难(未解决)

```
import hashlib
from Crypto.Util.number import *
from secrets import flag

def hash(x):
    return hashlib.sha512(x.encode()).digest() * 2

def pad(message):
    return (message[: len(message) // 2] + hash(str(len(message)))) +
    message[len(message) // 2 :])

p = getStrongPrime(1024)
q = getStrongPrime(1024)
n = p * q
e = 2
m = bytes_to_long(pad(flag))
print(pow(m, e, n))
print(n)
#
333529953751843435000867006727351402088326580978765890990083130320106922811166747
751228871562731337737437719206553193199183033126694028152942975893312564506862370
370443143293106251545930440712976483616963806866772346810990993268733572782445980
790355861715666113899197393328080515689312095113548816892342525811968999385989654
009731819704843667631833450205326973804627933804749725878374700708456481499480314
404936511757490470481654252301574639651969350516796324560004774245647854564033446
767855474822782302086255071208324901232974570813907033892873022689792388578578346
1594034339595106377701306570280371612953393097739
#
262786242991871484065597727708653362269346337342859797414248675408286703978651896
859668285271687956215434909791824175700789919308220414685398550065852336928842353
310849073403025300602617421007026583124351805273351012848006161068846924986033009
264883580179288676728619884484884393564485435278106205913247741113216193912641737
793120332525731400286304411352690560990745315028412599403796366993048106777161770
804862657213229668141046275259539741434764520586389275115948840021852190808474958
357273006700280110018531796592502702000208843338500830635148300950647309329975939
30711871108436386821290545084229347398808220810263
```

easy_ecc

```
from Crypto.Util.number import *
from hashlib import sha256
from secret import flag, secret, SECRET
```

```

assert flag[6:-1] == sha256(long_to_bytes(secret)).hexdigest().encode()

class ECC_easy:
    def __init__(self):
        self.a = 1365855822212045061018261334821659180641576788523935479
        self.b = 1732942721995516180470320010588432276870493483367341
        self.p = 1365855822212045061018261334821659180641576788523935481

    def add(self, P, Q):
        mul_inv = lambda x: pow(x, -1, self.p)
        x1, y1 = P
        x2, y2 = Q
        if P!=Q:
            l=(y2-y1)*inverse(x2-x1,self.p)%self.p
        else:l=(3*x1**2+2*self.a*x1+1)*inverse(2*self.b*y1,self.p)%self.p
        temp1 = (self.b*l**2-self.a-x1-x2)%self.p
        temp2 = ((2*x1+x2+self.a)*l-self.b*l**3-y1)%self.p
        x3 = temp1
        y3 = temp2
        return x3, y3

    def mul(self, x, P):
        Q = SECRET
        x = x % self.p
        while x > 0:
            if x & 1:
                Q = self.add(Q, P)
            P = self.add(P, P)
            x >>= 1
        return Q

    def ispoint(self, x, y):
        return (self.a * x ** 2 + x ** 3+x) % self.p == (self.b * y ** 2) %
self.p

ecc = ECC_easy()
LLLL =
(1060114032187482137663886206406014543797784561116139791,752764811411303365258802
649951280929945966659818544966)
assert ecc.ispoint(LLLL[0], LLLL[1])
END = ecc.mul(secret, LLLL)
print(END)

#
(695174082657148306737473938393010922439779304870471540,4146263570549585068674530
55549756701310099524292082869)

```

这题的话，我们首先将椭圆曲线转化成标准曲线，转化脚本在学习资料中已经给出。转完之后，我们尝试用hellman进行解密，发现报错，提示该点在奇异曲线上，于是我们选用奇异曲线的解密脚本来进行攻击，脚本如下所示。（实际上并不一定需要转化成标准曲线）

```
A=1098066776930223927329092382214459309226361965213
```

```

B=1263248714105743124314734095577181018742053879965591734
p=1365855822212045061018261334821659180641576788523935481
LLLL =
(804603363012007759329983017410816754946539644939,6683607008289577839808889388785
66241242911721895008218)
END=
(933414165833077907509715600260551365988944141925739220,1213467377002193380849948
30488363509910434835223666824)

```

```

def attack(p, a2, a4, a6, Gx, Gy, Px, Py):
    """
    Solves the discrete logarithm problem on a singular curve ( $y^2 = x^3 + a_2 * x^2 + a_4 * x + a_6$ ).
    :param p: the prime of the curve base ring
    :param a2: the a2 parameter of the curve
    :param a4: the a4 parameter of the curve
    :param a6: the a6 parameter of the curve
    :param Gx: the base point x value
    :param Gy: the base point y value
    :param Px: the point multiplication result x value
    :param Py: the point multiplication result y value
    :return: l such that  $l * G == P$ 
    """
    x = GF(p)["x"].gen()
    f = x ** 3 + a2 * x ** 2 + a4 * x + a6
    roots = f.roots()

    # Singular point is a cusp.
    if len(roots) == 1:
        alpha = roots[0][0]
        u = (Gx - alpha) / Gy
        v = (Px - alpha) / Py
        return int(v / u)

    # Singular point is a node.
    if len(roots) == 2:
        if roots[0][1] == 2:
            alpha = roots[0][0]
            beta = roots[1][0]
        elif roots[1][1] == 2:
            alpha = roots[1][0]
            beta = roots[0][0]
        else:
            raise ValueError("Expected root with multiplicity 2.")

        t = (alpha - beta).sqrt()
        u = (Gy + t * (Gx - alpha)) / (Gy - t * (Gx - alpha))
        v = (Py + t * (Px - alpha)) / (Py - t * (Px - alpha))
        return int(v.log(u))

    raise ValueError(f"Unexpected number of roots {len(roots)}.")

```

```

attack(p,0,A,B,804603363012007759329983017410816754946539644939,66836070082895778
3980888938878566241242911721895008218,9334141658330779075097156002605513659889441
41925739220,121346737700219338084994830488363509910434835223666824)

```

使用sage, 这样就解出了我们的sercet值, 再带回到原式中, 获得最后的flag

```
XYCTF{ec9a6e17537e81b7f593f65f7e2ca5d575e6b34c504c24e4afb40c1e9dc4be0d}
```

LCG_and_HNP(未解决)

```
from Crypto.Util.number import *
import random
from secrets import flag

class LCG:
    def __init__(self, seed, a, b, p):
        self.seed = seed
        self.a = a
        self.b = b
        self.p = p

    def next(self):
        self.seed = (self.seed * self.a + self.b) % self.p
        return self.seed >> (self.p.bit_length() - 8)

m = bytes_to_long(flag)
p = getPrime(128)
a = random.randint(1, p)
b = random.randint(1, p)
seed = random.randint(1, p)
out = []
lcg = LCG(seed, a, b, p)
for i in range(30):
    out.append(lcg.next())
key = ""
while 1:
    key += str(lcg.next())
    if int(key) >= m:
        break

with open("out.txt", "w") as f:
    f.write(f"p={p}\n")
    f.write(f"a={a}\n")
    f.write(f"b={b}\n")
    f.write(f"out={out}\n")
    f.write(f"c={int(key)^m}")
```

铜匠

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

```
from secrets import flag
```

```
m = bytes_to_long(flag)
m1 = getRandomRange(1, m)
m2 = getRandomRange(1, m)
m3 = m - m1 - m2
```

```
def task1():
    e = 149
    p = getPrime(512)
    q = getPrime(512)
    n = p * q
    d = inverse(e, (p-1)*(q-1))
    return (pow(m1, e, n), d >> 222 << 222, n)
```

```
def task2():
    e = 65537
    p = getPrime(1024)
    q = getPrime(1024)
    n = p * q
    return (pow(m2, e, n), (p + q) & ((1 << 624) - 1), n)
```

```
def task3():
    e = 65537
    p = getPrime(512)
    q = getPrime(512)
    n = p * q
    return (pow(m3, e, n), (p ^ q) >> 200, n)
```

```
c1, leak1, n1 = task1()
c2, leak2, n2 = task2()
c3, leak3, n3 = task3()
print(c1, leak1, n1)
print(c2, leak2, n2)
print(c3, leak3, n3)
```

```
#
```

```
(89623543982221289730635223555830551523170205418976759060541832483843039074358160
566735322009158596405712449020903311144480669706226166537602750967447480664875090
686428406188847601970724994074417752345460791736191511081890913041143570455636020
205647345764692062144226011846336769477026234106683791286490222089,
138474880017294332349992670187778287774153347391371789199005713563195654993662610
111557185709277805165708109047494971468100563949333712521647405765501704478862377
527360106399421209690341743821320754482338590671565510629203215009008479290995785
318405211007685664887994061938667418220613430135743123498167435264,
146331610798417415036517077006943013321623040860385791423062775325646472298267580
898028515394910588437521335092742913111680737790430660749825981979147191282007208
147041227246620008377726207734522466015971515317594545750944838673018946440525615
131606652748549901880641896940968837669894325535750125282351577689)
```

#

(54739611668213445766140030608978480144826727889040943407044478824900911154681809
445694091593432224598012351795877212321665765306217517488507634307880369358326535
351109751548001913233513338836614513319710422141434544416511657185041319769200777
688648500354717909111907725839771733889751059708665929740773611938223026530465114
181904100437298760625171484389232116733002682770376223374035263995847592020979259
835962612796761010797715301185255408424995176224788192115283456685136800647444436
287796233401755785094136369505451346410509481859442790299499577484645290758307706
17236599864271566534714755373562527112224962839980,629647935047329236503449756347
736881081355808260641340901141814496070624976901847188452954326446505809304300614
11603385577783897575232298084007041511817031640186953023971498914096209984730,207
486520696328484348979283144371383414362648598025869391545902371860292449079368704
778445631668275875361491707107203657601296830244019570954474660567464690551738972
346599112914436059124592712480593411473585118605377695879631890926484738948682098
386003461159197265898917773401661740173895132607378915577121528717143379466755335
970498741552020562001709540338491766559281443546655532717094420117230884484855703
942087287756657398195362299088470430074721788033940557835433789906998340666142620
501194434217098785985333295558389151582591382970605744250199232913530770802367695
86821808150397875920110335669136563171420068201)

#

(34640310217688693418173336791306022698488916505988760907493665070072623574363578
354500529023855888624978101429772253437880445815006839767802433777006836665709335
479076676231470534690281412388704665083311568028710188940132495410474044569100181
764053297307413009214044407982980600917158732129850605715306726034,
376358765177526195556604668489914624638748530752170855761001871193279163007352801
0801142052497,
948488691744300821732449660777365193967021412994299657250513142704430786218421667
910823545941935543802751678983424979988713662560448658799092183099605956910086636
324103560939667626393082538484501783103476126308148520547639330633135376944494750
61227647475480417779126252294793767003555255527593551612032661749)

这题就是一个coppersmith练习，只不过第一个形式貌似是对d的高位攻击，平时没有怎么见过

part2

题目泄露的信息是 $p+q$ 的低624位。由于有：

$$n = pq$$

这个等式自然在任何模数下都成立，也就有：

$$n = pq \pmod{2^{624}}$$

把已知条件放进来：

$$n = p(\text{leak} - p) \pmod{2^{624}}$$

所以可以在模 2^{624} 下解得 p 可能的低624bit，然后就转化成 p 低位泄露的问题了，对所有可能值逐一copper来得到正确解。

part3

题目泄露的信息是 p^q 的高312位，这里需要把剪枝和copper结合起来，做法可以参考鹏城杯的一道题目：

```

from Crypto.Util.number import *
from tqdm import *
import itertools

def small_roots(f, bounds, m=1, d=None):
    if not d:
        d = f.degree()

    R = f.base_ring()
    N = R.cardinality()

    k = ZZ(f.coefficients().pop(0))
    g = gcd(k, N)
    k = R(k/g)

    f *= 1/k
    f = f.change_ring(ZZ)

    vars = f.variables()
    G = Sequence([], f.parent())
    for k in range(m):
        for i in range(m-k+1):
            for subvars in itertools.combinations_with_replacement(vars[1:], i):
                g = f**k * prod(subvars) * N**(max(d-k, 0))
                G.append(g)

    B, monomials = G.coefficient_matrix()
    monomials = vector(monomials)

    factors = [monomial(*bounds) for monomial in monomials]
    for i, factor in enumerate(factors):
        B.rescale_col(i, factor)

    B = B.dense_matrix().LLL()
    B = B.change_ring(QQ)
    for i, factor in enumerate(factors):
        B.rescale_col(i, Integer(1)/factor)

    H = Sequence([], f.parent().change_ring(QQ))
    for h in filter(None, B*monomials):
        H.append(h)
        I = H.ideal()
        if I.dimension() == -1:
            H.pop()
        elif I.dimension() == 0:
            roots = []
            for root in I.variety(ring=ZZ):
                root = tuple(R(root[var]) for var in f.variables())
                roots.append(root)
            return roots

    return []

```

```

c1,leak1,n1 =
(89623543982221289730635223555830551523170205418976759060541832483843039074358160
566735322009158596405712449020903311144480669706226166537602750967447480664875090
686428406188847601970724994074417752345460791736191511081890913041143570455636020
205647345764692062144226011846336769477026234106683791286490222089,
138474880017294332349992670187778287774153347391371789199005713563195654993662610
111557185709277805165708109047494971468100563949333712521647405765501704478862377
527360106399421209690341743821320754482338590671565510629203215009008479290995785
318405211007685664887994061938667418220613430135743123498167435264,
146331610798417415036517077006943013321623040860385791423062775325646472298267580
898028515394910588437521335092742913111680737790430660749825981979147191282007208
147041227246620008377726207734522466015971515317594545750944838673018946440525615
131606652748549901880641896940968837669894325535750125282351577689)
c2,leak2,n2 =
(54739611668213445766140030608978480144826727889040943407044478824900911154681809
445694091593432224598012351795877212321665765306217517488507634307880369358326535
351109751548001913233513338836614513319710422141434544416511657185041319769200777
688648500354717909111907725839771733889751059708665929740773611938223026530465114
181904100437298760625171484389232116733002682770376223374035263995847592020979259
835962612796761010797715301185255408424995176224788192115283456685136800647444436
287796233401755785094136369505451346410509481859442790299499577484645290758307706
17236599864271566534714755373562527112224962839980, 629647935047329236503449756347
736881081355808260641340901141814496070624976901847188452954326446505809304300614
11603385577783897575232298084007041511817031640186953023971498914096209984730, 207
486520696328484348979283144371383414362648598025869391545902371860292449079368704
778445631668275875361491707107203657601296830244019570954474660567464690551738972
346599112914436059124592712480593411473585118605377695879631890926484738948682098
386003461159197265898917773401661740173895132607378915577121528717143379466755335
970498741552020562001709540338491766559281443546655532717094420117230884484855703
942087287756657398195362299088470430074721788033940557835433789906998340666142620
501194434217098785985333295558389151582591382970605744250199232913530770802367695
86821808150397875920110335669136563171420068201)
c3,leak3,n3 =
(34640310217688693418173336791306022698488916505988760907493665070072623574363578
354500529023855888624978101429772253437880445815006839767802433777006836665709335
479076676231470534690281412388704665083311568028710188940132495410474044569100181
764053297307413009214044407982980600917158732129850605715306726034,
376358765177526195556604668489914624638748530752170855761001871193279163007352801
0801142052497,
948488691744300821732449660777365193967021412994299657250513142704430786218421667
910823545941935543802751678983424979988713662560448658799092183099605956910086636
324103560939667626393082538484501783103476126308148520547639330633135376944494750
61227647475480417779126252294793767003555255527593551612032661749)
e1,e2,e3 = 149,65537,65537

```

```
##### task1
```

```

k1 = e1*leak1 // n1 + 1
PR.<x,y> = PolynomialRing(Zmod(e1*leak1))
f = 1 + k1*((n1+1)-y) - e1*x
bounds = (2^222,2^513)

res = small_roots(f,bounds,m=2,d=2)
leak = int(res[0][1])

PR.<x> = PolynomialRing(RealField(1000))

```



```

f = x*(leak-x) - n1
ph = int(f.roots()[0][0])

PR.<x> = PolynomialRing(Zmod(n1))
f = ph + x
res = f.small_roots(x=2^(222+6), beta=0.499, epsilon=0.02)[0]
p1 = int(ph + res)
q1 = n1 // p1
d1 = inverse(e1, (p1-1)*(q1-1))
m1 = pow(c1, d1, n1)

##### task1
if(0):
    PR.<x> = PolynomialRing(Zmod(n2))
    p1 = var('p1')
    p0 = solve_mod([p1*(leak2-p1) - n2 == 0], 2^624)
    for i in tqdm(p0):
        temp = int(i[0])
        f = 2^624*x + temp
        f = f.monic()
        res = f.small_roots(x=2^(1024-624), beta=0.499, epsilon=0.05)
        if(res != []):
            print(res, i)

p2 =
2^624*181885878066267467254651923462114482901163423945399153315880038460274990039
0376290902354433158869501621068317141437803547 +
30799806171826831530692477758473366665061074090587762392881036975218340474724228
413987808867958110969502274124994976148966054080915100124553907410082410345243978
97872311534501791582888829
q2 = n2 // p2
d2 = inverse(e2, (p2-1)*(q2-1))
m2 = pow(c2, d2, n2)

##### task1
leak3 = leak3 << 200
LEAK = bin(leak3)[2:].zfill(512)
def find(ph, qh, h):
    pM = int(ph + (512-h)*"1", 2)
    qM = int(qh + (512-h)*"1", 2)
    p0 = int(ph + (512-h)*"0", 2)
    q0 = int(qh + (512-h)*"0", 2)

    if(h == 512 - 200):
        PR.<x> = PolynomialRing(Zmod(n3))
        f = pM + x
        res = f.small_roots(x=2^200, beta=0.499)
        if(res != []):
            p3 = int(pM + int(res[0]))
            q3 = n3 // p3
            d3 = inverse(e3, (p3-1)*(q3-1))
            m3 = pow(c3, d3, n3)
            print(long_to_bytes(int(m1+m2+m3)))

```

```

if(pm*qM < n3 or pm*qm > n3):
    return

if(LEAK[h] == "0"):
    find(ph+"1",qh+"1",h+1)
    find(ph+"0",qh+"0",h+1)
else:
    find(ph+"1",qh+"0",h+1)
    find(ph+"0",qh+"1",h+1)

find("1","1",1)

```

这样就可以解出来了

```
XYCTF{__y0u_k0nw_h0w_t0_s0lv3_c0pP3r_th15_15_y0r_fl@9_hahahaha!!!__}
```

new_LCG(未解决)

```

from Crypto.Util.number import *
import random
from secrets import flag

p = getPrime(512)
q = getPrime(512)
r = getPrime(512)
n = p * q * r

class LCG1:
    def __init__(self, seed, a, b, m):
        self.seed = seed
        self.a = a
        self.b = b
        self.m = m
        self.sum = 0

    def next(self):
        old_seed = self.seed
        self.seed = (self.a * self.seed + self.b * self.sum) % self.m
        self.sum += old_seed
        return self.seed

class LCG2:
    def __init__(self, seed, q, A, B):
        self.E = EllipticCurve(GF(q), [A, B])
        self.P = self.E.lift_x(seed)
        self.b = self.E.random_point()

    def next(self):
        self.P = self.P + self.b

```

```

        return self.P[0]

a1 = random.randint(1, p)
b1 = random.randint(1, p)
seed1 = random.randint(1, p)
lcg1 = LCG1(seed1, a1, b1, p)
c1 = []
for _ in range(10):
    c1.append(lcg1.next())

a2 = random.randint(1, q)
b2 = random.randint(1, q)
seed2 = random.randint(1, q)
lcg2 = LCG2(seed2, q, a2, b2)
c2 = []
for _ in range(10):
    c2.append(lcg2.next())

c = bytes_to_long(flag)
e = 65537
print(n)
print(pow(c, e, n))
print(c1)
print(c2)
#
942554394956393500634473023924044851150783066435925660508624376974971108656861080
872622432250172571195245180102507380675343264066303507696268870722959770362631674
931170602993210221083436437860191861911457384526742569208842886612504579678703976
889217504241741044075116270718940025586448491942058697669033418724145042156461740
336592337509609124425828725269151627786668070531098444935129266626770404736852607
352075247856808563388127774523912002202264558855255067503
#
913511855200450258515359405375997856168901515704219399711463486182676567861100907
703218572447018201260262272839659342121359464316433203255135905051552140705406387
515651053003614302726389574202765960261614544137398235935065162752244446661874422
386248645482639272745912126145619169138518555525168643874609462676297946642162285
965346849337916077407251603948417115397679323392812146736495534074143472934805221
75832736301571839298158011533849603878482322619858083471

```

#

[69242290819763341801934779514171172753966564349680000322289082315112460530648332
36257422745299036986875244562025221130619850904830372215788197314906896784, 707045
810464337488125013716300756405499020414540801863434513087687914538170573314824134
496890600422837133767094273649381435979038909605432188919586751472, 56148766573911
177489716527456090848723245733374891046799834320209777869992565068270499644356022
4288857862513676706660506594465853704042586896087391214886, 6498834369085375452441
121562339384727357898216850540864190840194924515286757125433756518026123451611578
553656133012172518080309106483207567929943790044792, 55083454016849139406101839585
263986354061873490433688340808381536118108966296930272455116883667767491766528589
29409374912959736262162942801190344337268446, 760441008226521115410835744650729779
045176669817731313067295420281379688898871924295112744811222833296789279481941521
1618572734294964346056056171483002307, 7699815879242527638422887386550759485127768
822609011364311314299885418068889791030639324882736871756700299169635780542149704
586720216395186814592666988591, 82974813172038259969665335972261270851483090408460
504859088295130004970114888302128324350608130042704173329938532528439927063391794
1134488957784480285437, 7084115400374950827522650500486495223539292992998875483730
75809800503010605531028258934219353638197330992407404395522228738842206417828828
756951777504457, 74827960673142664262156483260369219551838077411347876135849773008
210233983757897258480566572500862886875708759790723680042177882225371152321912307
02833854]

#

[12573984103581521249597169818949825744876735847170990673204303602848066091917704
946653820386665801380026230957120354174481948164514637637956459534723582285,
644195440710999541385879210147208955810678062845999110166250756569966422234169723
0094798036088532393057448200220905589679548875702178737462785403325555226,
116842447456413671063861967744472046740898531239664223870249489217950991920250697
60680924547214190425118261001002764397184950251157744938735586522727499550,
103962434163733266954738434271393851167086528457896448619658763465537953134547736
68473030130335970707089781482749749170266279229263892370064669233541305377,
909074824136060637131028160881896685533811096939765972095395184580598377189406462
9343960530245792700858927510839835850767728294266738917884471006979663157,
114898480451047493327903332721286338850191561598058051597501747238080435697892576
25027794186359921080743368220606914862583644262009648261345452266067697628,
649349258009900424806913260265314442160901414078390702088746248078789581041616487
825633046538335114117254875547413590064940767523651802950986197978665018,
630213694034507794738227615165719412407345755948727442587988797838690105816238583
3414602346299055653879087077862824771863209271498552774720708307840705334,
584452639824464523647308950060873141506012556602752540561800058038114976165374414
2808567706048834618328671311836990498943019718294135733114642775270792763,
483454804399386865906160669982295742284004042597683362846209308933050758986540779
3951971491111695171201542345470702059513427268037868859282999222070054388]

Reverse

聪明的信使

一眼古典密码，直接打开ida，按f5，对字符串进行解密就可以了

```
ata:00404000 format      db 'Input your flag:',0 ; DATA XREF: _main+16To
ata:00404011 ; const char aS[]
ata:00404011 aS          db '%s',0 ; DATA XREF: _main+2Ato
ata:00404014 ; const char Str2[]
ata:00404014 Str2       db 'oujp{H0d_TwXf_Lahyc0_14_e3ah_Rvy0ac@wc!}',0
ata:00404014 ; DATA XREF: _main+4Ato
ata:0040403D ; const char aTryAgain[]
ata:0040403D aTryAgain   db 'Try again!',0 ; DATA XREF: _main+62to
ata:00404048 ; const char aGoodJob[]
ata:00404048 aGoodJob    db 'Good job!',0 ; DATA XREF: _main:loc_40162Dto
```

```
flag{Y0u_kn0w_Crypt0_14_v3ry_Imp0rt@nt!}
```

喵喵喵的flag碎了一地

在程序开头处找到第一处flag，说第二处flag在function中，我们很容易找到，然后根据提示，就可以知道第三处跟function互相引用（xref）了，所以找到对应的function718，是在一个数字下，读出flag即可

```
XYCTF{My_fl@g_h4s_br0ken_4parT_Bu7_Y0u_c@n_f1x_1t!}
```

Misc

game

熊博士

```
CBXGU{ORF_BV_NVR_BLF_CRZL_QQ}
```

看出来是古典加密，根据前五位XYCTF就可以推断出加密方式了，是两个位置上的ascii码加起来永远不变，容易得出flag

```
XYCTF{liu_ye_mei_you_xiao_jj}
```

签到

第一题！直接发给公众号获得flag

```
XYCTF{WELCOME_TO_XYCTF}
```

EZ_Base1024*2

base2048在线解密：

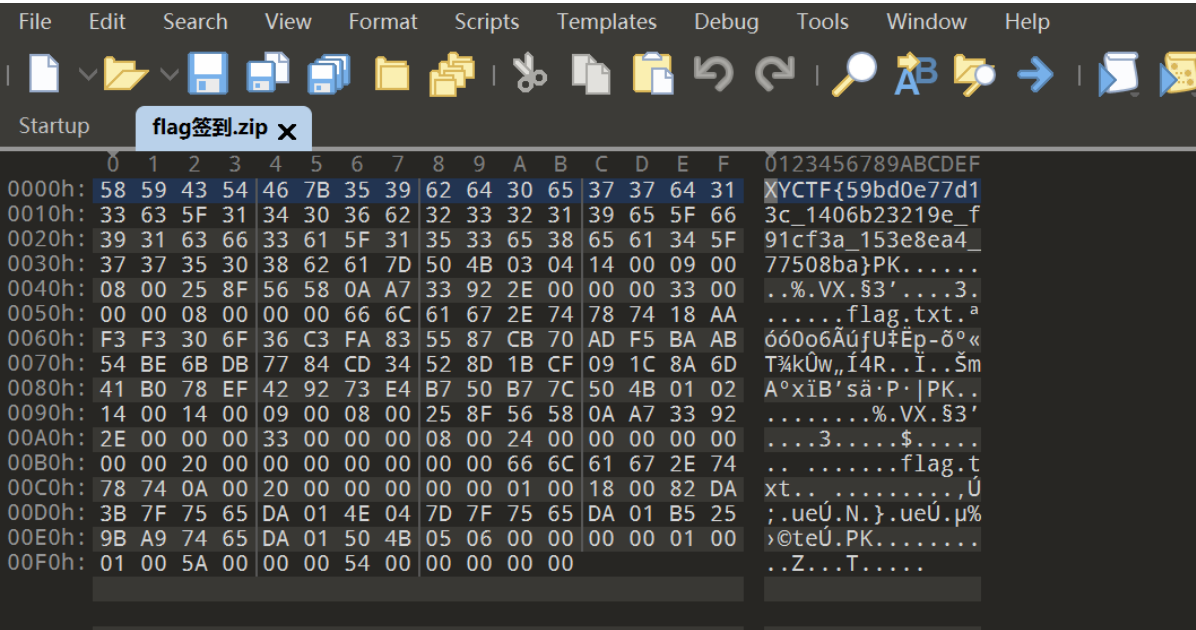
[Encode and Decode Base2048 Online Tool](#)

得出结果

```
XYCTF{84ca3a6e-3508-4e34-a5e0-7d0f03084181}
```

真>签到

有手就会，把附件放进01编辑器中，读取前面的flag就可以了



读取flag

```
XYCTF{59bd0e77d13c_1406b23219e_f91cf3a_153e8ea4_77508ba}
```

Osint1

这题给了一个滨海新区的照片，本来因为是天津市的，但是不是，后来在百度识图在一个网址上找到了用户的ip为江苏，再进一步查询发现该地点在江苏省南通市海安区，在滨海东路上，外面的海是黄海。

得到flag：

```
xyctf{江苏省|南通市|滨海东路|黄海}
```