0xGame2023 Week3 Writeup

> My vegetable has exploded :(

Web

GoShop

题目

Go Shop

Buy anything you want!

User: 8636e1eb-7d83-416c-9e87-68cfd44ffbac

Money: 100

Get Flag Reset

You don't have bought anything yet

Select a product to buy or sell

Name	Price	Number	Action
Apple	10		Buy Sell
Banana	50		Buy Sell
Orange	100		Buy Sell
Flag	999999999		Buy Sell

Hint

Hint 1: 注意 int64 类型的范围 两个 int 64 进行计算的时候会不会出现什么问题?

Hint 2: 整数溢出

在给出的代码中可以发现有两个int64相乘的情况出现，int64的范围是-9223372036854775808~9223372036854775807。根据提示可知，可在number输入较大的数使其发生整数溢出来刷money，从而可以买下flag，得到flag。

Go Shop

Buy Orange * 9000000000000000000 success

Buy anything you want!

User: 7f5cb715-98f3-47d2-904b-f17ac72bb6d8
Money: 3890459611768029000

Get Flag Reset

You have the following items:

- Orange: 9000000000000000000

Select a product to buy or sell

Name	Price	Number	Action
Apple	10		Buy Sell
Banana	50		Buy Sell
Orange	100	)0000000	Buy Sell
Flag	999999999		Buy Sell

Go Shop

Here is your flag: 0xGame{eaf905e4-28a1-4006-b8a0-e8ddc2d673bf}

Buy anything you want!

User: 7f5cb715-98f3-47d2-904b-f17ac72bb6d8
Money: 3890459610768029000

Get Flag Reset

You have the following items:

- Flag: 1
- Orange: 9000000000000000000

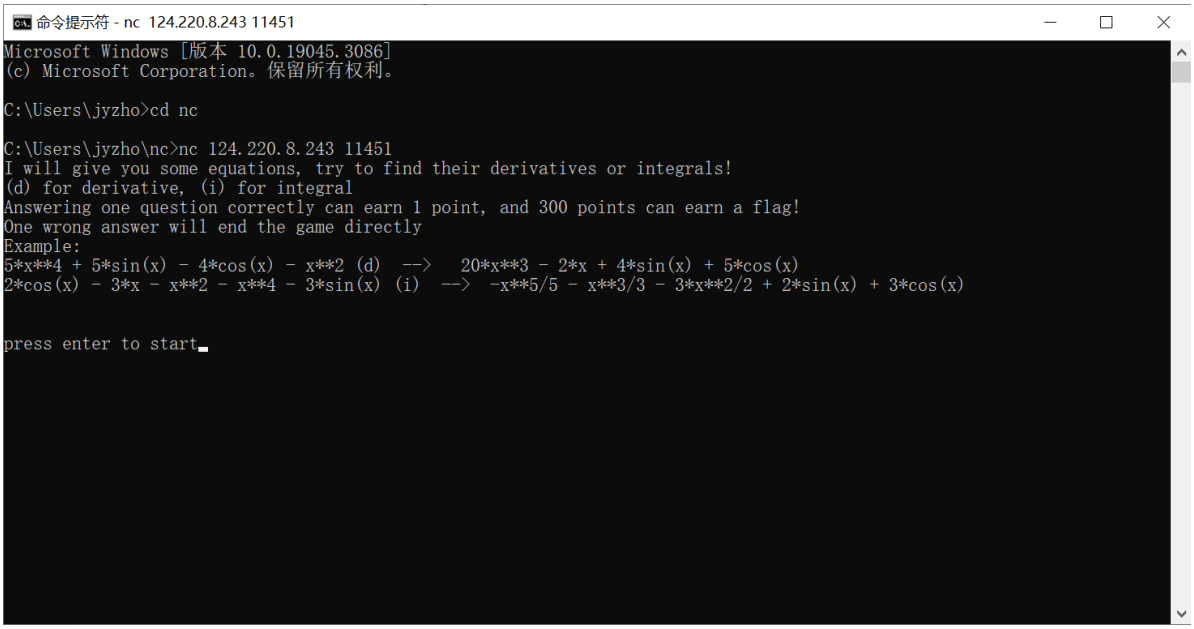
Select a product to buy or sell

Name	Price	Number	Action
Apple	10		Buy Sell
Banana	50		Buy Sell
Orange	100	)0000000	Buy Sell
Flag	999999999	1	Buy Sell

Misc

高数大师

题目（这里为了方便，我直接在windows上面装了nc指令。。。）



由于有300题，而且错一题就得重来，那么我肯定是不愿意口算的……于是本noob费劲千辛万苦写了一个交互脚本自动答题，这也是本noob第一次接触到pwntools。。。

```
from pwn import *
import sympy as sp
def gaoshu(io):
    io.sendline()
    io.recvuntil(b"press enter to start>")
    for i in range(299):
        f=str(io.recvline().strip())[2:-1]
        x=sp.symbols('x')
        if f[-2]=="d":
            f=f[0:-3]
            ans=str(sp.diff(f, x))
            print(ans,i,"d")
            io.sendlineafter(b"your answer >",ans)
        else:
            f=f[0:-3]
            ans=str(sp.integrate(f,x))
            print(ans,i,"i")
            io.sendlineafter(b"your answer >",ans)
        io.recvuntil(b">")
io = remote('124.220.8.243',11451)
gaoshu(io)
io.interactive()
```

先自动回答299题，然后为了防止出错，手动回答一题，得到flag如下

```python
from pwn import *
import sympy as sp
def gaoshu(io):
    io.sendline()
    io.recvuntil(b"press enter to start>")
    for i in range(299):
        f=str(io.recvline().strip())[2:-1]
        x=sp.symbols('x')
        if f[-2]=="d":
            f=f[0:-3]
            ans=str(sp.diff(f, x))
            print(ans,i,"d")
            io.sendlineafter(b"your answer >",ans)
        else:
            f=f[0:-3]
            ans=str(sp.integrate(f,x))
            print(ans,i,"i")
            io.sendlineafter(b"your answer >",ans)
        io.recvuntil(b">")
io = remote('124.220.8.243',11451)
gaoshu(io)
io.interactive()
```

```
问题  输出  调试控制台  终端  端口

10*x - 4*sin(x) 298 d
[*] Switching to interactive mode
  sin(x) + x (d)

your answer > cos(x)+1
correct!
0xGame{Y0u_Ar3_such_A_H1gh_M4th_M4s7er}
[*] Got EOF while reading in interactive
```

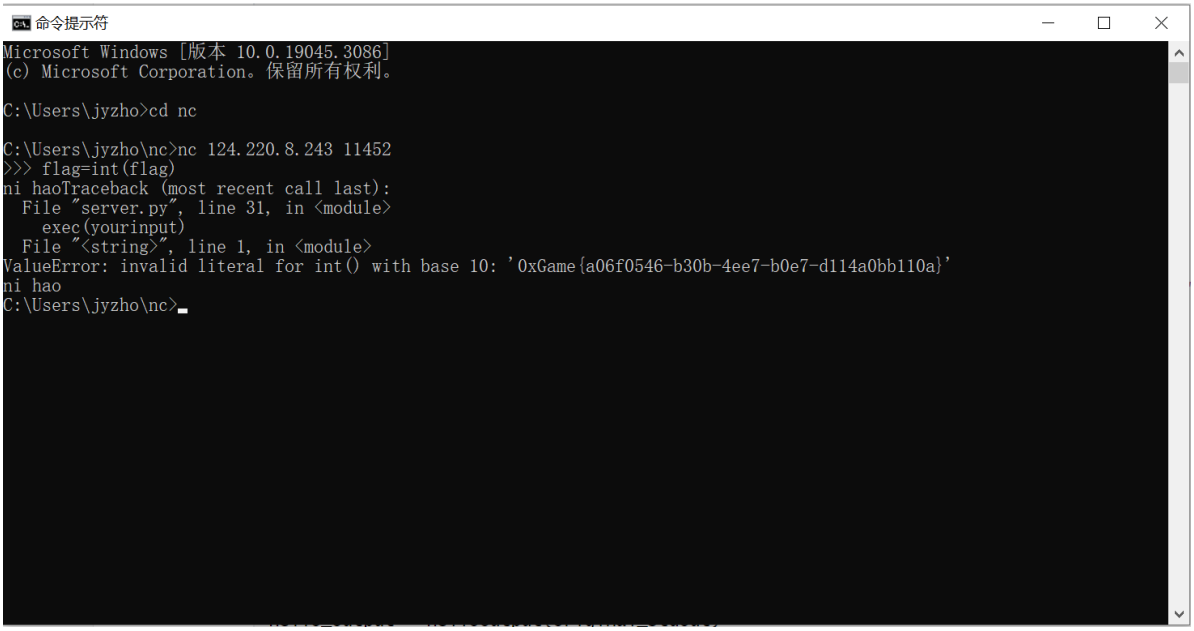
你好

题目源码，~~看来确实是一个微笑服务的python机器人，，，~~

```python
import string
import sys

class helloOutput:

    def __init__(self, original_stdout):
        self.original_stdout = original_stdout

    def write(self, text):
        self.original_stdout.write("ni hao")

    def flush(self):
        self.original_stdout.write("ni hao")

def check(input):
  if any(i in input for i in ["import", "_", "[", "]", "{", "}", ".", "eval",
"exec", "'", "\""]):
```

```
    print("hacker!!!")
    exit()
  if any(i not in string.printable for i in input):
    print("only ascii!!!")
    exit()

flag = open("flag.txt").read()

yourinput = input(">>> ")
check(yourinput)
original_stdout = sys.stdout
hello_output = helloOutput(original_stdout)
sys.stdout = hello_output
del sys
exec(yourinput)
```

从中可以看出，经过一番重定向和过滤，不管输入什么得到的都是若干个ni hao。于是可以考虑通过输入有关flag的错误指令，从报错中得到flag.txt的内容。我在这里输入了flag=int(flag)，得到了flag。



Crypto

> 这周的crypto本noob作为一个大一新生，知识有点不够做。。。以下几题基本上都是在碎片化的学习与盲猜中撞对的。。。

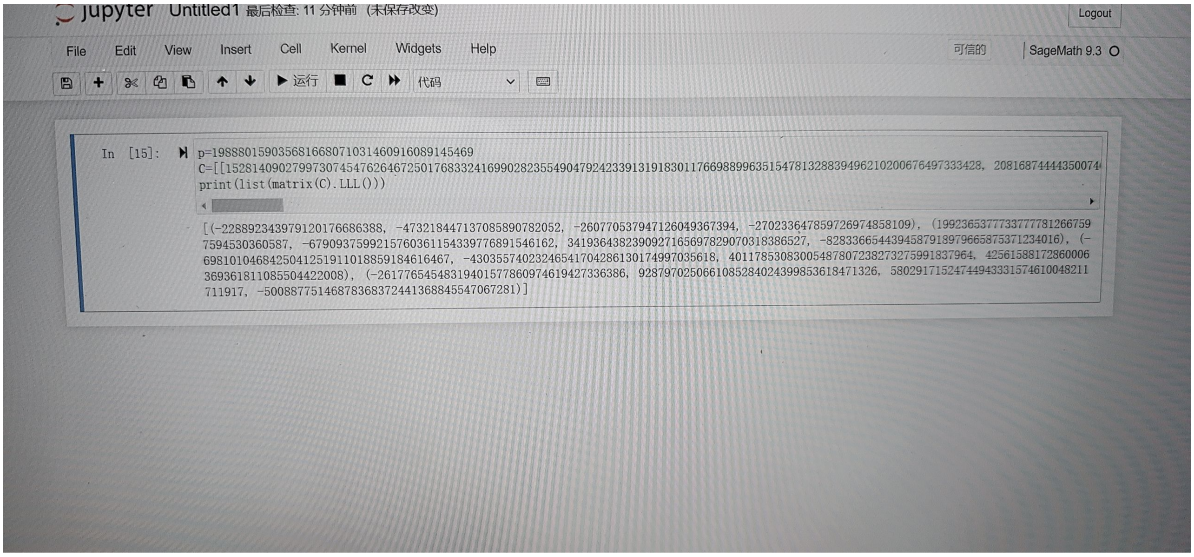
LLL-FirstBlood

题目

```
from random import randrange
from Crypto.Util.number import getPrime,bytes_to_long
from secret import flag
assert len(flag) % 4 == 0

length = len(flag)//4
m = [bytes_to_long(flag[i*length:(i+1)*length]) for i in range(4)]
p = getPrime(int(128))
```

```python
def MakeMask(n,p):
    upper = identity_matrix(n)
    low = identity_matrix(n)
    for i in range(n-1):
        for j in range(i+1, n):
            upper[i, j] = randrange(1, p)
            low[j, i] = randrange(1, p)
    result = upper * low
    assert det(result) == 1
    return result

def Matrix2List(x):return [list(i) for i in x]

noise = [[randrange(1, p) for i in range(4)] for _ in range(4)]
noise[0] = m
M = matrix(noise)
A = MakeMask(4,p)
C = A*M

print(f'p={p}')
print(f'C={Matrix2List(C)}')
'''
p=19888015903568166807103146091608914546 9
C=
[[15281409027997307454762646725017683324169902823554904792423391319183011766988 9635154781328839496210200676497333428,
20816874444350074678072503732785131140452725852438154588400834874597950213021800 774901340996449931200095671472027 72,
30808734094602990463394957507466321853072465728175347847039360448741068094136204 700064459849627337210295664402536 75,
34917343419951741836269919072926070702521975206314127679898794325987438511711753 691800803559775742965587344158234 58],
[23594095358090481273312446998671475468171348026100673294311352279914883241483740 659402383081475008095993957487567 98,
31911961991608214463510364603857919856826450404460225127908153488105557488254202 372918391707748722640974661832087 42,
46653465301553864572423453942842861983473362814515306708181138767677362880894001 194923177756482066432428394308992 83,
53693507460428502760673806385715654960879487997209689594262561929238521979593811 018394841964459958283894610044959 17],
[16414071110662654296029295602644431032859080726770654987605705145774129053922601 823347066355552565377459022831912 51,
21905361733991771670681533512719889312322728840285696692420623950879222750216283 347977292665609300401168071339772 44,
31275567591408454261323056994217071821083515169318814119287198028476284086568878 975964251335237825265614710504473 59,
37072399565292001593808706184717039210112760204393157063521835762899252633165804 089680920167824837703731219728354 10],
[98838145431958490135239344274514070195148076069934145696261426568571681653 39,
13190422499129347541373922929251088892868361241120937213742340947017395215646,
18832738552342488056498211782604832513006649329982003661701684946590064734701,
22323329751908690611034666068697427811613727429398087082295754189068333861152]]
'''
```

按照题意可得知对C进行LLL，取其结果矩阵的第一行的负数long_to_bytes即可得到flag



LLL-SecondBlood

题目

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

m = bytes_to_long(flag)
q = getPrime(512)
assert m.bit_length() == 318
```

```python
def encrypt(m):
    mask,noise = getPrime(511),getPrime(50)
    mask_.append(mask)
    noise_.append(noise)
    c = (mask*m + noise)%q
    return c

noise_,mask_  =[[] for _ in range(2)]
c_ = [encrypt(m) for i in range(4)]

print(f'q = {q}\nmask = {mask_}\nc_ = {c_}')

'''
q =
93424266017836508610201195685656564047152360599030090419771497782441539304359080
24696666887269890479558473622355346816236972767736577737321732137220122 53
mask =
[62371284452369929205772256448586626775759511264678888587824613340579700694689258338442311166474068339991426597513746202802132907361145760890693963312267 47,
63680313892139538894175452567501692337259752291974468038850291597677014794455768607045615932009074823726908511521267823911264625475245266319344089810708 41,
51064734609827911885782853974206421376303472892528520450440211979886070827772318398397301696821585078220784124498279766633852820219161208374081925063414 43,
63180908429503312280333495175428101235963168503536374215872648864138771426126861777960230493049086964133862189925111127527886407324108455896798200030476 67]
c_ =
[38235396647200290275869331524784927804385950044534892518441338309471653428393938788319148793346602506214228773330223211171203985284305197941096241862044 92,
17216596457502248199532449954605896911206726497325607684352146081678612467901362172193492346047241480399106565734366633793750481450454435272677903798164 25,
66863352007934483964895050238005931191610846880100938613881032425914652332370401449154714897383577491733133358147592080467739594985441189455670523857889 6,
49786058637998107649913028185198601088935625337119226626722033471341578240293931848392641821387734151199691818975059575537256034508589910930534433894406 6]
'''
```

根据题意构造矩阵，进行LLL后取第一行第五列(m)，进行long_to_bytes，得到flag

```
In [1]:
q = 9342426601783650861020119568565656404715236059903009041977149778244153930435908024696666887269890479558473622355346816236972767736577…
mask = [6237128445236992920577225644858662677575951126467888858782461334057970069468925833844231116647406833999142659751374620280213290736…
c_ = [3823539664720029027586933152478492780438595004453489251844133830947165342839393878831914879334660250621422877333022321117120398528430…
a=[[-mask[0],-mask[1],-mask[2],-mask[3],1,0],[q,0,0,0,0,0],[0,q,0,0,0,0],[0,0,q,0,0,0],[0,0,0,q,0,0],[c_[0],c_[1],c_[2],c_[3],0,pow(2,341)…
a=matrix(a)
print(a.LLL())
```

```
[                                                                                                    585427539127961
1080888257365151                                                                                            7030474740
69607                                                                                                1007577402866201
40441776610975277436599331102620625293782235942612008132308745772428788611527732901998961696447 7              447948948435560842
1114884561136888556243290994469299069799978201927583742360321890761754986543214231552]
[ -1910888333017412870916761846461328634000083605214274197589817937865130264202608301060350553832242468587737668218424655406 37   -1424568
0385142153621818418555956419083831972644618647016187096899025411678370004685365050252472369533758168079263167465226 7      684816891255828374
347115786749448280514029557187484047668002177359459919643038422187151436234620074760030128339133449311 29    871917803145379007707856495801
504132054861366321254049294537961570844125001106517007143545338224706543956699713545274083778    80890660336476437785548568437754565026219
387658496540505678081219142341384883436656924561192880092807226447976598762064658 2         -73029795619952342711500345141923131606704866 3
3919142913713009031326000902296407725703927102647364925776269681256038 4]
[  2929288228027271244041359761882861388043513170250148699714052651543906603082204104995716498385348592330568476581363998984 15   -25814870
1756219049139885245038773708709495955576941447075746919249565853036012642339310918818851825128555193186035432612156    29331011574876403 17
6032866756340860717538725281729723285308417332922142998804855828004511584416462093617655914871013644171 0    560537811455517097926702681720
623427486526254425412050211153681534638350531535932237976258197258468901497665455441620569432 -1126825964317483776941030192792516227101 79
5693124460427906583081458444780912087466823881819473077157014255592571931461675293         101732226601998777032133273302268888440881246 2
0557007169321837850936610955987186697379919327073703389577091645597286 4]
[  -154095630789219613542757703453615131826500875009551336175658280802112422852237733289695844597480809112653106162838243915 33   -21541319
0081726735512290319053131321721126382347386334897789644606049054224403868129468757099505703018388665343307879358518 -1264414715377635359 0
4173539914406953508147056877727008606265682580140837735266946465708270392135212890783995105607839446008 6     3618205654094162427287924673 4
103012778141302945589494283010663372438959220577486128010598577100825764761135145501988073412  -378746775172541103808076320919656126652 12
374553936424060108396994096490158789944906279756394730634872463449124928217722145 9         3419405833443553078684990089152889211469968 89
7408836456427932295953675450517188740660425954730330737942113481129984 0]
[ -36604882065025988920195149320215381310423034087738848387897637974326627809708866696896534172827302381339028254747953939644 3 -191460190
5109265177817965409595839470953215197036164630349009414052491693432429797135086391731852228497499731324407459102727   -2587751968137765808
631457786375124293007170929001037689960232938654814812379204003082012490111168404435856737215927741405 45   -548285023614391082648562176 72
94362055465761315481450766321131383587599568846714234676514895198700649953265939622124081155 8   306849156239812152316243266561082283111 37
478841702821109058716466107569091944070581018455579333093780245438433048037039716 1         -277029895331954183580256359104203062144334182
1328235007747788623754477002358720215202185129054922932800633450790912 0]
[-1964324791851501249527912021842001154536398005014266450826564918844325906459905836582931660660229762576215185286627145870502   19572795
7435281510322711338431161844931353493135595365394603581780167984701120955442786237297240386314925549338146615654985   619710932991450386 2
08637298096311601615269463800740123082910145465416951235468810189820697872998597770425900017697714409026  -56738914163701739911721561457 6
800959291763089056648685243336432654285320030959294516708374759466539869834017765245878269208  -10816810336263068897589164951147960964132
97289903875533642600696565151497014345403811001332364943189641439492877542889397 60        976564469656981790532288883746455413922856 33
375099194588753611303036803734653623658315802219709651553941291605688 32]
```

```python
from Crypto.Util.number import long_to_bytes
print(long_to_bytes(40441776610975277436599331102620625293782235942612008132308745772428788611527732901998961696447 7))
```

```
b'0xGame{19255b5c7b19c790e28d87c8a8bb1d33}'
```

EzMatrix

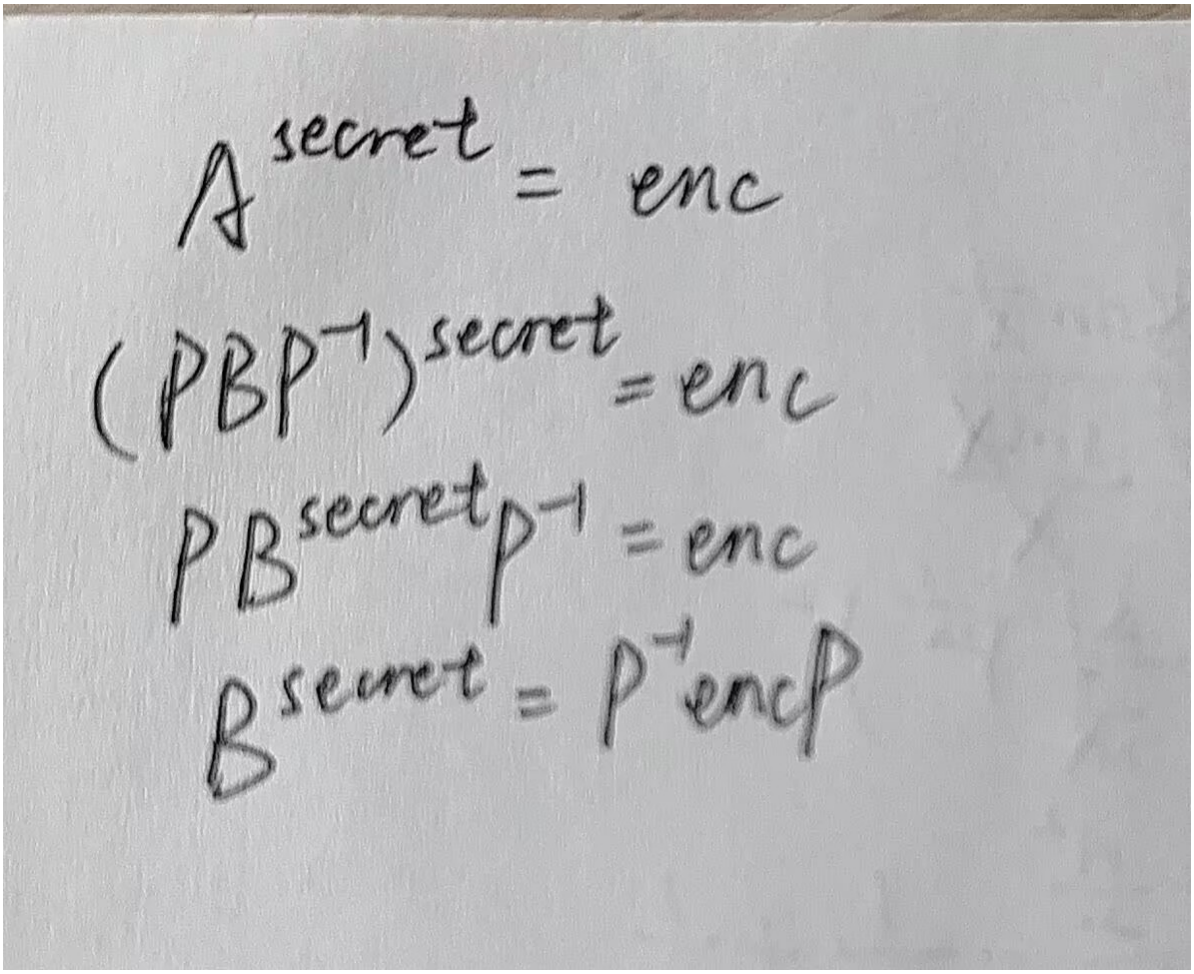
题目

```python
from Crypto.Util.number import getPrime
from random import randint
from secert import secert,flag
from hashlib import md5
def n2b(n):return md5(str(n).encode()).hexdigest()

assert secert < pow(2,64)
assert flag == '0xGame{'+n2b(secert)+'}'

def Martix2list(Martix):
    result = []
    Martix = list(Martix)
    for i in Martix:
        result.append(list(i))
    return result
```

```
A=[[12143520799533590286, 1517884368, 12143520745929978443, 796545089340,
12143514553710344843, 28963398496032, 12143436449354407235, 158437186324560,
12143329129091084963, 144214939188320, 12143459416553205779, 11289521392968],
[12143520799533124067, 1552775781, 12143520745442171123, 796372987410,
12143514596803995443, 28617862048776, 12143437786643111987, 155426784993480,
12143333265382547123, 140792203111560, 12143460985399172467, 10983300063372],
[12143520799533026603, 1545759072, 12143520746151921286, 781222462020,
12143514741528175043, 27856210942560, 12143440210529480891, 150563969013744,
12143339455702534403, 135941365971840, 12143463119774571623, 10579745342712],
[4857408319806885466, 2428704161425648657, 12143520747462241175, 758851601758,
12143514933292307603, 7286139389566980165, 9714738936567334300, 144947557513044,
12143346444338047691, 130561054163540, 4857352974113333366,
2428714303424782417],[12143520799533339320, 1476842796, 12143520749060275613,
733281428880, 12143515144091549812, 25896324662208, 12143446129977471347,
139126289668080, 12143353609086952433, 125093278125816, 12143467808884068695,
9705993135696],[34695773712880799926, 5204366058378782250, 12143520750775862343,
706665985740, 12143515359139397843, 24876891455539, 12143449149385190675,
5204499435641729607, 1734628523990131469, 119757210113970, 12143470097256549947,
9282407958928],[10986995009101166671, 1734788687033207505, 12143520752514668698,
680173911560, 12143515570582515443, 23883386182656, 12143452072344092516,
10408859957710764174, 8673790006740000925, 4047954924507284041,
12143472277719610437, 8879790035168],[12143520799534210329, 8095680534365818753,
12143520754224346525, 6071761054204856029, 12143515774342357443, 22931775530664,
12143454859049102627, 122586336122081, 12143373761302849103, 109840689548590,
8095634066844843878, 8500892291801],[2428704159899526175, 7286112481016467893,
12143520755876491019, 629765964828, 12143515968446948123, 9714838668887734012,
4857345013259425502, 117630592711632, 12143379764863568374, 105318302849760,
2428659620509049335, 7286120625945355053],[7286112479717322389,
7286112480971640825, 12143520757456628435, 606320684970, 12143516152115449139,
4857429497934652454, 4857347490735050126, 112978994964264, 12143385390297217523,
101086824360217, 7286069740980100293, 7286120294834973633],[7727695054246476847,
1202487728, 12143520758958480293, 584144077140, 12143516325240923843,
20377952745696, 12143462294760579275, 108622249048560, 12143390651947217363,
97133513961120, 12143479741445599772, 8831658996900830432],
[12143520799535388887, 1161628182, 12143520760380594623, 563225247585,
12143516488091679443, 19626876325056, 12143464472820678035, 104545135017180,
12143395570399006523, 93441517429260, 12143481309754543787, 7218375794633]]#
12*12
p = 12143520799543738643
A = Matrix(GF(p),A)
enc = A**secert

def Martix2list(Martix):
    result = []
    Martix = list(Martix)
    for i in Martix:
        result.append(list(i))
    return result

with open('enc.txt','w') as f:
    f.write(str(Martix2list(enc)))

'''
```

```
enc=[[11285847990515095003, 7585413350741918021, 11658254512436412666,
477577914899276103, 2941386515764607825, 11283325421744133699,
4096971712575507616, 8118672870538606033, 2377937081025778041,
6576171711896495163, 6152554374963853172, 5022013484610428974],
[8354008012616001452, 7787447107046065118, 9504997911333967278,
1082773427768571094, 6015520658629219637, 11244285744740006951,
4493944053220750368, 3504246247470690014, 1738582001618280397,
2330057776906622572, 3043456814665571080, 2981613454022714952],
[2508674373714509177, 3544963739532775937, 7952732753025175616,
11161786730565526285, 3397123486689639675, 6454135592624912854,
6613201018024296927, 9748485344986779929, 1819761609989340766,
1259944825407465767, 1596049024644778041, 7769939905324967788],
[4200851163596876950, 11960539098651202761, 3303721151143544462,
2532304102428121556, 11083895221097319129, 1171933471304558017,
1549099593543874478, 6088238862927163233, 6459553630361959801,
947358195425767572, 2090533922210134578, 9023030120605201052],
[2271102089902208138, 1614812525306266829, 1546249462332047661,
3168333397191737100, 7678980468150522028, 3128939172985153696,
1146041044751755224, 11870173227065140617, 8351303466095252790,
694704483676649448, 7944218023016968278, 583421745603756386],
[10309472503110333289, 1100598261990718822, 10235859400888405310,
910925705831020921, 10771855884237562064, 9970830255165655653,
11678899608458971536, 4368822164222204233, 3104861419162339779,
4540709628196554222, 7851809145727500968, 12086896840826708824],
[10973051751637593366, 5039073157846327641, 4855314857834773443,
4416954195828423951, 8243966437000815560, 8250554263390748131,
8093181066366682440, 1145520354143718292, 294729013023637045,
10115389386419597159, 2767140395261835843, 6724257139233017485],
[6878768250003631244, 10834164422364241529, 6946589221005878489,
539734218479521833, 2691724062063066048, 3989403041446358401, 815244541494093987,
11168528286389981272, 2021358468726921955, 1123433019094267521,
524639025046508882, 5720273332497702547], [6688451244183880831,
10892730373179989558, 6987453292894341174, 5572212176769878684,
11332149024403380575, 3944612864568504791, 6768594304071589280,
10526434024562201079, 10241323610053039912, 1120473558410865753,
306153635148226248, 36066660630742221O4], [7556871914690327290,
11353594909211427742, 747771112781361153, 1245068803956910299,
2831489557155431404, 18000356209488765551, 1050411779595241927,
5665981688041778089, 2028968510484240787, 4386552235402890530,
10334391443650474796, 3883841302951550608], [4485787817401669404,
184501191500952934, 3690661645276970957, 6263309802498749034,
6484490370652685031, 9743108369653588026, 3045941510087387269,
5870433915209047275, 4679598273992216016, 11839352681285251516,
4957980185504231911, 7925596893607015470], [1000449712878466719,
7022601702937838844, 1095849907482791166, 11989051568709522226,
6768031250066783733, 185945517026191241, 4280928696740160411,
5633542561098902406, 10176177574499086410, 5782837249861240943,
7406530879613861823, 19718582248395209l6]]
'''
```

由于还没学到相似矩阵，本noob赶紧恶补了一下，在一番艰难困苦的理解后，终于推算出了下面的式子o(╥﹏╥)o



B为A的特征值，P为A的特征向量，于是写出了以下代码

```
In [2]:
A=[[12143520799533590286, 1517884368, 12143520745929978443, 796545089340, 12143514553710344843, 28963398496032, 12143436449354407235, 1584
p = 12143520799543738643
A = Matrix(GF(p),A)
enc=[[11285847990515095003, 7585413350741918021, 11658254512436412666, 477577914899276103, 2941386515764607825, 11283325421744133699, 4096
enc=Matrix(enc)
B=A.eigenmatrix_right()[0]
P=A.eigenmatrix_right()[1]
PP=P.inverse()
encenc=PP*enc*P
print(discrete_log(mod(encenc[0][0],p),mod(B[0][0],p)))
```

```
6208835615336459559
```

然后通过题目中所给的flag计算方法得到了flag

matrix.py > ...

```python
from hashlib import md5
def n2b(n):return md5(str(n).encode()).hexdigest()
flag = '0xGame{'+n2b(6208835615336459559)+'}'
print(flag)

'''
A=[[12143520799533590286, 1517884368, 12143520745929978443, 796545089340, 12143514553710344843, 28963398496032, 121434364493544072351
p = 12143520799543738643
A = Matrix(GF(p),A)
enc=[[11285847990515095003, 7585413350741918021, 11658254512436412666, 477577914899276103, 2941386515764607825, 11283325421744133699
enc=Matrix(enc)
B=A.eigenmatrix_right()[0]
P=A.eigenmatrix_right()[1]
PP=P.inverse()
encenc=PP*enc*P
print(discrete_log(mod(encenc[0][0],p),mod(B[0][0],p)))
'''
```

问题 输出 调试控制台 终端 端口

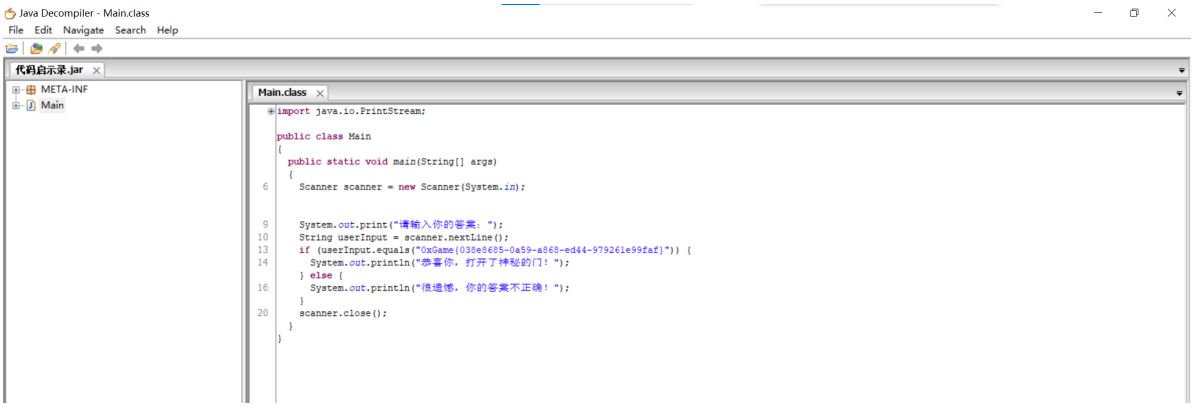
```
版权所有 (C) Microsoft Corporation。保留所有权利。

尝试新的跨平台 PowerShell https://aka.ms/pscore6

PS C:\Users\jyzho\Desktop\编程\Python>  & 'C:\Users\jyzho\AppData\Local\Programs\Python\Python38\python.exe' 'c:\Users\jyzho\.vscode\extensions
\ms-python.python-2023.18.0\pythonFiles\lib\python\debugpy\adapter/../..\debugpy\launcher' '1146' '--' 'C:\Users\jyzho\Desktop\编程\Python\matr
ix.py'
0xGame{06450201eb6171d40151563d967e59ea}
PS C:\Users\jyzho\Desktop\编程\Python>
```

Reverse

> 总体来说Reverse是这三周五个方向中最简单的，~~连本noob每周都能做出来三道。。。~~

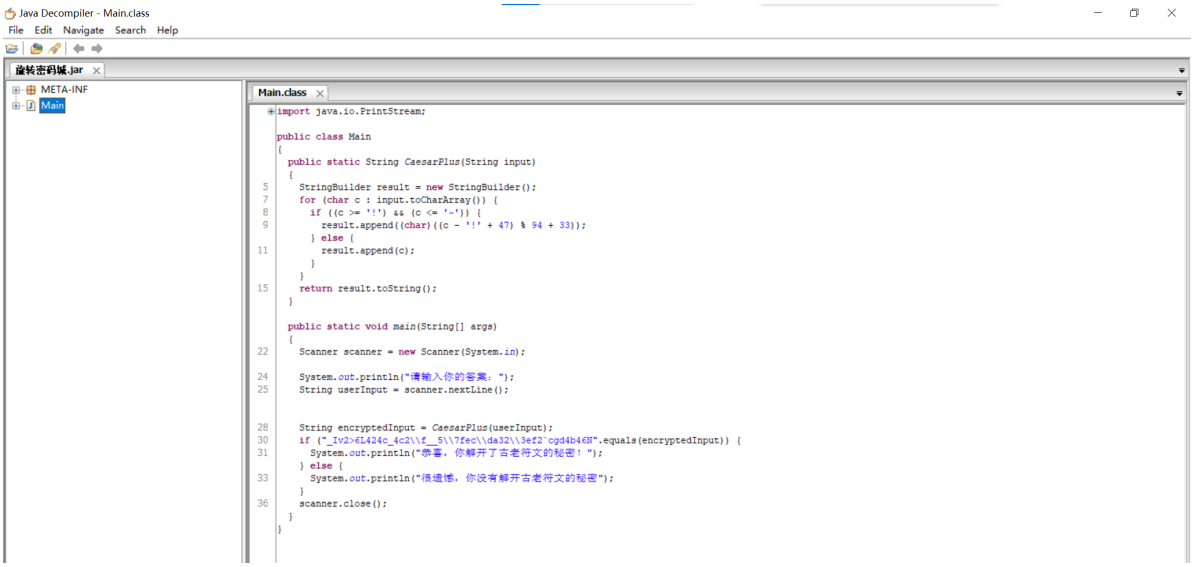
代码启示录

下载附件得到一个.jar文件，将其放到jd-gui中逆向解析得到源码，即可看到flag



旋转密码城

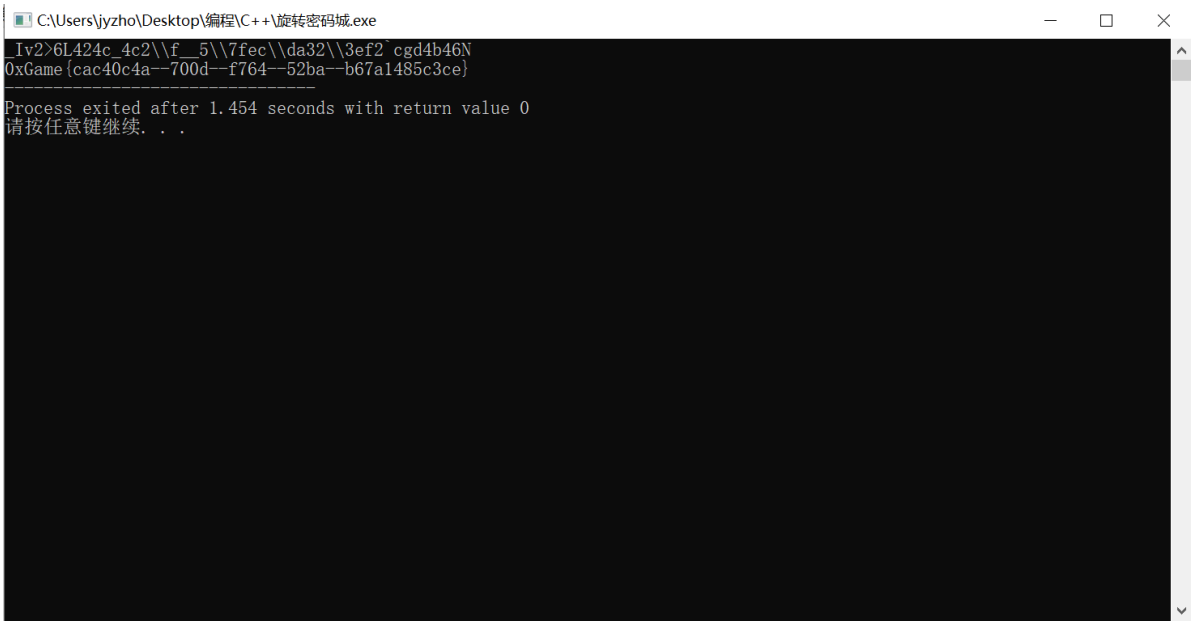
下载附件后仍是一个.jar文件，像上一题一样放到jd-gui中打开查看源码，发现是对flag进行了一番类似凯撒密码的~~出题人自定义的~~加密，并给出了密文。



于是自己编写相应的代码对其进行解密

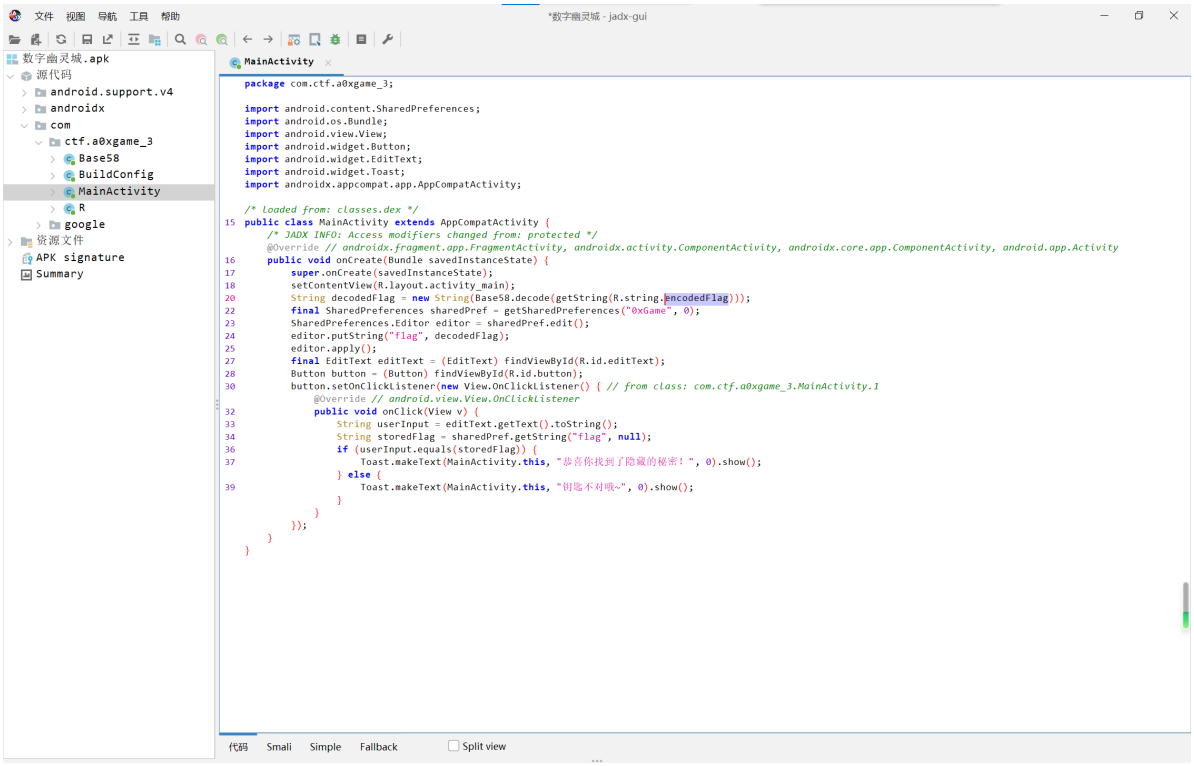
```
#include<bits/stdc++.h>
using namespace std;
char c[200];
char a[200];
 int main()
{
    for(int i=33;i<=126;i++)
    c[i]=(i+14)%94+33;
    cin>>a;
    for(int i=0;i<=strlen(a)-1;i++)
    {
        if(a[i]>=33&&a[i]<=126)
        {
            int j=33;
            while(c[j]!=a[i])
            j++;
            a[i]=j;
        }
        else
        continue;
    }
    for(int i=0;i<=strlen(a)-1;i++)
    cout<<a[i];
    return 0;
}
```

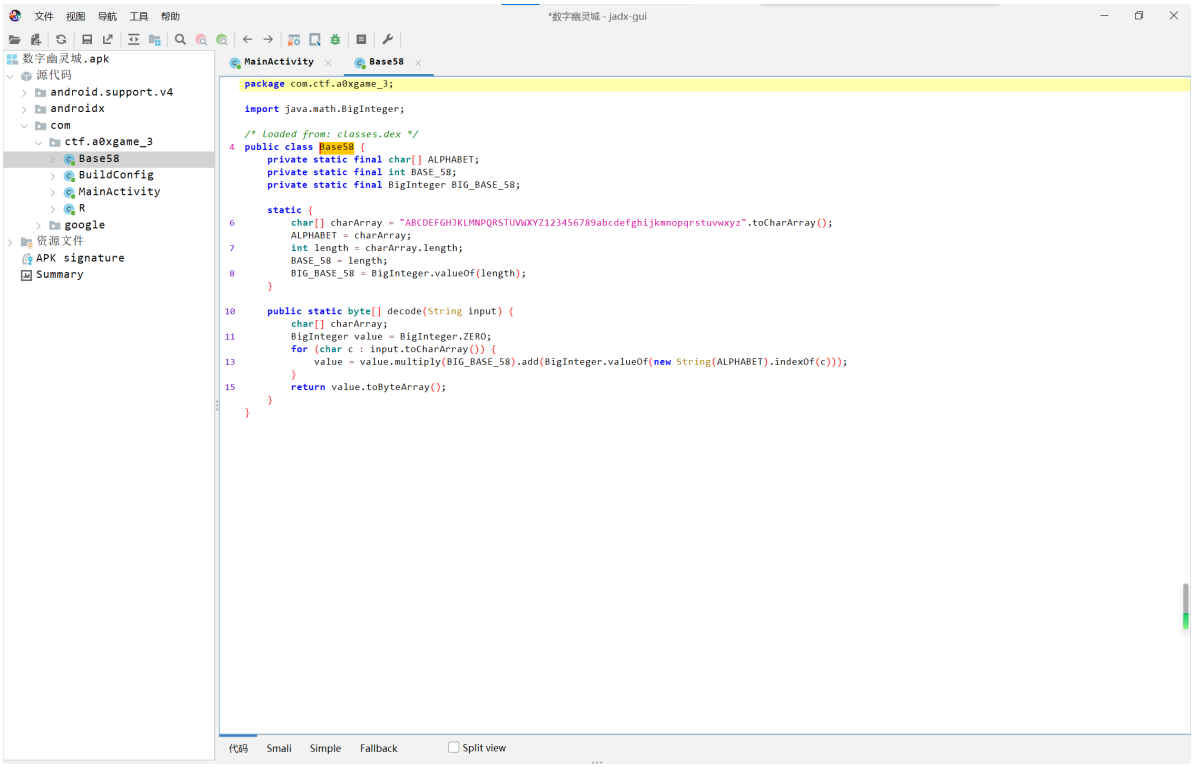
输入密文，得到的运行结果即为flag



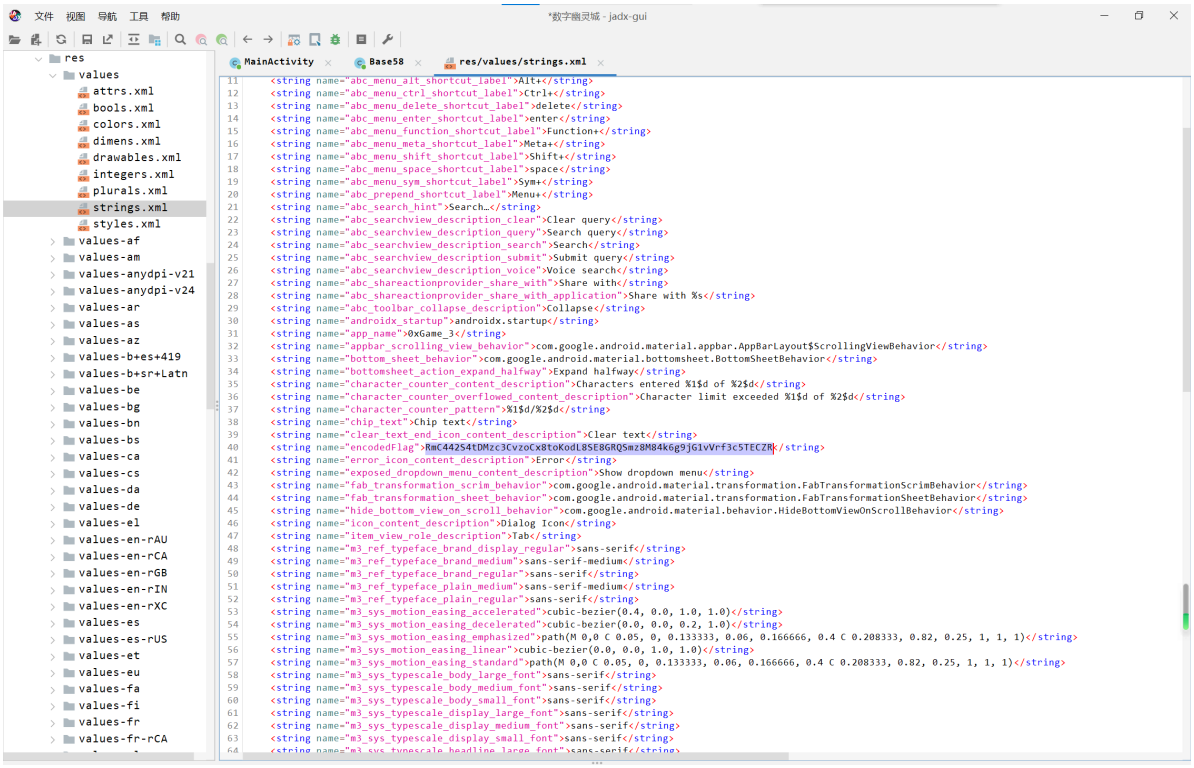
数字幽灵城

下载附件，这次发现是一个.apk文件，用jadx-gui对其进行逆向解析，找到MainActivity文件，发现flag被进行了~~出题人自定义的~~Base58加密（可在Base58文件中看到加密算法的源码）





在资源文件中找到密文



编写程序将其解密

```
import java.math.BigInteger;
public class Base58 {
    private static final char[] ALPHABET;
    private static final int BASE_58;
    private static final BigInteger BIG_BASE_58;

    static {
        char[] charArray =
"ABCDEFGHJKLMNPQRSTUVWXYZ123456789abcdefghijkmnopqrstuvwxyz".toCharArray();
        ALPHABET = charArray;
```

```
        int length = charArray.length;
        BASE_58 = length;
        BIG_BASE_58 = BigInteger.valueOf(length);
    }

    public static byte[] decode(String input) {
        char[] charArray;
        BigInteger value = BigInteger.ZERO;
        for (char c : input.toCharArray()) {
            value = value.multiply(BIG_BASE_58).add(BigInteger.valueOf(new
String(ALPHABET).indexOf(c)));
        }
        return value.toByteArray();
    }
}
```

```
public class oout{
    public static void main(String[] args){
        String decodedFlag = new
String(Base58.decode("RmC442S4tDMzc3CvzoCx8toKodL8SE8GRQSmz8M84k6g9jG1vVrf3c5TEC
ZR"));
        System.out.printf("%s",decodedFlag);
    }
}
```

输出得到flag