[buuctf] crypto全解——85-120（不建议直接抄flag）

原创

置顶 咸鱼壹号 于 2020-11-24 11:20:09 发布 3543 收藏 21
分类专栏： buuctf 密码学 文章标签： 密码学

本文链接：https://blog.csdn.net/ao52426055/article/details/109502365

85.[ACTF新生赛2020]crypto-rsa3

查看题目

```
//output.txt
17760650483649924697095903022687160888596932177821105108052463408451697333144164499389802957361229009585306926403653045925365287558626794687783105514754691022710056649665814838183468303736613455384801190325125272647404766127422313772768868953582353304677879313190214344440873561082116783871748885990224286368314573903785113823547710005409453611689847750526930736416823750714074908512897030709057495258304830359887371176539714284246123320209259266173955588681603806019124982999228259142295101669579104518417300289198838076344898341288308014072284472217752647113499281562901027823743794067192921160475815605303822100 49
```

```
//rsa3.py
from flag import FLAG
from Cryptodome.Util.number import *
import gmpy2
import random

e=65537
p = getPrime(512)
q = int(gmpy2.next_prime(p))
n = p*q
m = bytes_to_long(FLAG)
c = pow(m,e,n)
print(n)
print(c)
```

很明显文本里面的一堆数字就是n和c的值
那就很简单了，我们用yafu来分解N，也可以在线分解

```
factor(17760650483649924697095903022687160888596932177821105108052463408451697333144164499389802957361229009585306926403653045925365287558626794687783105514754691022710056649665814838183468303736613455384801190325125272647404766127422313772768868953582353304677879313190214344440873561082116783871748885990224286368 3)


fac: factoring 17760650483649924697095903022687160888596932177821105108052463408451697333144164499389802957361229009585306926403653045925365287558626794687783105514754691022710056649665814838183468303736613455384801190325125272647404766127422313772768868953582353304677879313190214344440873561082116783871748885990224286368 3
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
fmt: 1000000 iterations
Total factoring time = 0.6794 seconds


***factors found***

P155 = 13326909050357447643526585836833969378078147057723054701432842192988717649385731430095055622303549577233495793715580004801634268505725255565021519817179293
P155 = 13326909050357447643526585836833969378078147057723054701432842192988717649385731430095055622303549577233495793715580004801634268505725255565021519817179231

ans = 1
```

pq的值youlepy文件里面还有e
那么我们直接带入脚本就可以了（前面写了那么多脚本，我觉得你可以创一个文件夹，到时候把这些带入就可以了，而不是每次都要重新敲）

```
import gmpy2

p = 13326909050357447643526585836833969378078147057723054701432842192988717649385731430095055622303549577233495793715580004801634268505725255565021519817179293
q = 13326909050357447643526585836833969378078147057723054701432842192988717649385731430095055622303549577233495793715580004801634268505725255565021519817179231
c=14573903785113823547710005409453611689847750526930736416823750714074908512897030709057495258304830359887371176539714284246123320209259266173955588681603806019124982999228259142295101669579104518417300289198838076344898341288308014072284472217752647113499281562901027823743794067192921160475815605303822100 49
n=17760650483649924697095903022687160888596932177821105108052463408451697333144164499389802957361229009585306926403653045925365287558626794687783105514754691022710056649665814838183468303736613455384801190325125272647404766127422313772768868953582353304677879313190214344440873561082116783871748885990224286368 3


e=65537


phin = (p-1)*(q-1)


d=gmpy2.invert(e,phin)

print(d)

m = pow(c,d,n)

print(m)
print(hex(m))
print(bytes.fromhex(hex(m)[2:]))
```

运行得到

```
1142110340697533342134076532322394538183055540690133289094348258061535212792815766150851970390819896211244297276
8902360002419925826186240372948581699098548699684783004877571490662028879663621214255323576437411112094114014989
5595289559560740951667825123772519353303069022288432998660657137564563315041240411393
8934415128636956678674546293145489137505762887266286461062465751947313436077176801261068105535068444770563810x616374667b705f616e645f715f73686f756c645f6e6f745f62655f736f5f636c6f73655f696e5f76616c75657d
b'actf{p_and_q_should_not_be_so_close_in_value}'
```

86.[RoarCTF2019]babyRSA

查看题目

```
import sympy
import random

def myGetPrime():
    A= getPrime(513)
    print(A)
    B=A-random.randint(1e3,1e5)
    print(B)
    return sympy.nextPrime((B!)%A)
p=myGetPrime()
#A1=2185696345246163043734827843419143400006607675041902749385246351346986526206434083661383106660230095977263239777348731756033905665829995446416926446723440 7
#B1=2185696345246163043734827843419143400006607675041902749385246351346986526206434083661383106660230095977263239777348731756033905665829995446416926446714059 6

q=myGetPrime()
#A2=1646611311583922811976788789930882002574926093386344688822416716985761217866413954572634086740679075456022751601379626994143807681819461703030485185841892 7
#B2=1646611311583922811976788789930882002574926093386344688822416716985761217866413954572634086740679075456022751601379626994143807681819461703030485185835102 6

r=myGetPrime()

n=p*q*r
#n=8549266378627529215983160339108387617514935430932767300871662765071816058563972310079334753464962833041663125566090130753390990043141344752426233223265915304706790869348194712106907045156282241735765643217187095118467313255421369012330804269736196998636037506095470292065636414415414581283855836533417293593144142409627020614069181466231856269692576799193736978262790840823908735803316541002069015206771571111273225203858843289675840589870901034246788226436273 3
c=pow(flag,e,n)
#e=0x1001
#c=7570088302166957773932931679545070620450263580231073147715699883471082077024521946870324530200999893206708038397756029970806047622208963020997262975596514031752603468045248336091737881224436588452718605634188861556433556076505355015575836227162233001743340302726112756122558591248477782958850121396111069045198762550270133148514163968435642731690512299575982524113387273436271604181981994864566280329241880220443087452134210841362363515047596312122009523677642 8
#so,what is the flag?
```

做这题看到阶乘一下想到了 gxzy2020 的一题，也是考到了威尔逊定理（Wilson’s theorem）：当且仅当p为素数时：(p -1)! ≡ -1 (mod p)。

阶乘只乘到 B ，所以把（B+1）乘到（A-1）这一段也补上就得到了威尔逊公式，反之我们可以由用 -1 乘这一段的模反数，就得到了题目中的 (B!)%A 。

```
from Crypto.Util.number import *
from sympy import nextprime
A1=2185696345246163043734827843419143400006607675041902749385246351346986526206434083661383106660230095977263239
7773487317560339056658299954464169264467234407
B1=2185696345246163043734827843419143400006607675041902749385246351346986526206434083661383106660230095977263239
77734873175603390566582999544641692644671405 96

A2=1646611311583922811976788789930882002574926093386344688822416716985761217866413954572634086740679075456022751
6013796269941438076818194617030304851858418927
B2=1646611311583922811976788789930882002574926093386344688822416716985761217866413954572634086740679075456022751
6013796269941438076818194617030304851858351026

def f(a,b,P):# a*(a+1)*...*b  (mod P)
    ans=1
    for i in range(a,b+1):
        ans*=i
        ans%=P
    return ans%P

inv1=inverse(f(B1+1,A1-1,A1),A1)
ans1=((A1-1)*inv1)%A1
p=nextprime(ans1)

inv2=inverse(f(B2+1,A2-1,A2),A2)
ans2=((A2-1)*inv2)%A2
q=nextprime(ans2)

e=0x1001
c=7570088302166957773932931679545070620450263580231073147715699883471082077024521946870324530200999893206708038397756029970806047622208963020997262975596514031752603468045248336091737881224436588452718605634188861556433556076505355015575836227162233001743340302726112756122558591248477782958850121396111069045198762550270133148514163968435642731690512299575982524113387273436271604181981994864566280329241880220443087452134210841362363515047596312122009523677642 8
n=8549266378627529215983160339108387617514935430932767300871662765071816058563972310079334753464962833041663125566090130753390990043141344752426233223265915304706790869348194712106907045156282241735765643217187095118467313255421369012330804269736196998636037506095470292065636414415414581283855836533417293593144142409627020614069181466231856269692576799193736978262790840823908735803316541002069015206771571111273225203858843289675840589870901034246788226436273 3
r=(n//p)//q
assert isPrime(r)
d=inverse(e,(p-1)*(q-1)*(r-1))
m=pow(c,d,n)
print(long_to_bytes(m))
```

运行得到

```
b'RoarCTF{wm-CongrAtu1ation4-1t4-ju4t-A-bAby-R4A}'
```

87.坏蛋是雷宾

查看题目

老牌刺客之王混进了女王的住所。一天，女王得到了一个匿名举报，说她的侍卫里有一个刺客，叫做Rabin，而他的信息就在一份文件里，文件中有附带一个Pk，是523798549，密文是162853095，校验码二进制值是110001，根据说明是放在明文后一起加密的，明文与密文长度相同。加密算法和这位老牌刺客同名。快拯救女王，答案是求得的明文，进行32位md5小写哈希字符串，提交即可。 注意：得到的 flag 请包上 flag{} 提交

n分解然后写脚本

```
'''// python2'''
from gmpy2 import *
import hashlib
n=523798549
p=10663
q=49123
e=2
c=162853095
inv_p = invert(p, q)
inv_q = invert(q, p)


mp = pow(c, (p + 1) / 4, p)
mq = pow(c, (q + 1) / 4, q)


a = (inv_p * p * mq + inv_q * q * mp) % n
b = n - int(a)
c = (inv_p * p * mq - inv_q * q * mp) % n
d = n - int(c)

for i in (a, b, c, d):
    print(bin(i)[2:])
#在得出的四个解中，找到二进制符合题目的解
#10010011100100100101010110001
m='10010011100100100101010'
mc=str(int(m,2))
md=hashlib.md5()
md.update(mc.encode("utf8"))
flag = md.hexdigest()
print("flag{"+str(flag)+'}')
#flag{ca5cec442b2734735406d78c88e90f35}
```

88.[AFCTF2018]Single

查看题目

```
Jmqrida rva Lfmz (JRL) eu m uqajemf seny xl enlxdomrexn uajiderc jxoqarerexnu. Rvada mda rvdaa jxooxn rcqau xl J
RLu: Paxqmdyc, Mrrmjs-Yalanja mny oekay.

Paxqmdyc-urcfa JRLu vmu m jxiqfa xl giaurexnu (rmusu) en dmnza xl jmrazxdeau. Lxd akmoqfa, Wab, Lxdanuej, Jdcqrx
, Benmdc xd uxoarvenz afua. Ramo jmn zmen uxoa qxenru lxd atadc uxftay rmus. Oxda qxenru lxd oxda jxoqfejmray rm
usu iuimffc. Rva nakr rmus en jvmen jmn ba xqanay xnfc mlrad uxoa ramo uxfta qdatexiu rmus. Rvan rva zmoa reoa e
u xtad uio xl qxenru uvxwu cxi m JRL wenad. Lmoxiu akmoqfa xl uijv JRL eu Yaljxn JRL gimfu.

Waff, mrrmjs-yalanja eu mnxrvad enradaurenz seny xl jxoqarerexnu. Vada atadc ramo vmu xwn narwxds(xd xnfc xna vx
ur) werv tifnmdmbfa uadtejau. Cxid ramo vmu reoa lxd qmrjvenz cxid uadtejau mny yatafxqenz akqfxeru iuimffc. Ux,
 rvan xdzmnehadu jxnnajru qmdrejeqmnru xl jxoqarerexn mny rva wmdzmoa urmdru! Cxi uvxify qdxrajr xwn uadtejau lx
d yalanja qxenru mny vmjs xqqxnanru lxd mrrmjs qxenru. Veurxdejmffc rveu eu m ledur rcqa xl JRLu, atadcbxyc snxw
u mbxir YAL JXN JRL - uxoarvenz fesa m Wxdfy Jiq xl mff xrvad jxoqarerexnu.

Oekay jxoqarerexnu omc tmdc qxuuebfa lxdomru. Er omc ba uxoarvenz fesa wmdzmoa werv uqajemf reoa lxd rmus-bmuay
afaoanru (a.z. IJUB eJRL).

JRL zmoau xlran rxijv xn omnc xrvad muqajru xl enlxdomrexn uajiderc: jdcqrxzdmqvc, urazx, benmdc mnmfcueu, datad
ua anzanaadenz, oxbefa uajiderc mny xrvadu. Zxxy ramou zanadmffc vmta urdxnz useffu mny akqadeanja en mff rvaua
euuiau.

Iuimffc, lfmz eu uxoa urdenz xl dmnyxo ymrm xd rakr en uxoa lxdomr. Akmoqfa mljrl{Xv_I_lxiny_er_neja_rDc}
```

我看到这我就直接去在线爆破了

爆破结果

```
0 -1.448 Capture the Flag (CTF) is a special kind of information security competitions. There are three common types of CTFs: Jeopardy, Attack-Defence and mixed. Jeopardy-style CTFs has a couple of questions (tasks) in range of categories. For example, Web, Forensic, Crypto, Binary or something else. Team can gain some points for every solved task. More points for more complicated tasks usually. The next task in chain can be opened only after some team solve previous task. Then the game time is over sum of points shows you a CTF winer. Famous example of such CTF is Defcon CTF quals. Well, attack-defence is another interesting kind of competitions. Here every team has own network(or only one host) with vulnarable services. Your team has time for patching your services and developing exploits usually. So, then organizers connects participants of competition and the wargame starts! You should protect own services for defence points and hack opponents for attack points. Historically this is a first type of CTFs, everybody knows about DEF CON CTF - something like a World Cup of all other competitions. Mixed competitions may vary possible formats. It may be something like wargame with special time for task-based elements (e.g. UCSB iCTF). CTF games often touch on many other aspects of information security: cryptography, stego, binary analysis, reverse engeneering, mobile security and others. Good teams generally have strong skills and experience in all these issues. Usually, flag is some string of random data or text in some format. Example afctf{Oh_U_found_it_nice_tRy}
1 -3.331 Capture the Flag (CTF) is a special ?ind of information security competitions. There are three common types of CTFs: ?eopardy, Attac?-Defence and mixed. ?eopardy-style CTFs has a couple of ?uestions (tas?s) in range of categories. For example, ?eb, Forensic, Crypto, Binary or something else. Team can gain some points for every solved tas?. More points for more complicated tas?s usually. The next tas? in chain can be opened only after some team solve previous tas?. Then the game time is over sum of points sho?s you a CTF ?iner. Famous example of such CTF is Defcon CTF ?uals. ?ell, attac?-defence is another interesting ?ind of competitions. Here every team has o?n net?or?(or only one host) ?ith vulnarable services. Your team has time for patching your services and developing exploits usually. So, then organi?ers connects participants of competition and the ?argame starts! You should protect o?n services for defence points and hac? opponents for attac? points. Historically this is a first type of CTFs, everybody ?no?s about DEF CON CTF - something li?e a ?orld Cup of all other competitions. Mixed competitions may vary possible formats. It may be something li?e ?argame ?ith special time for tas?-based elements (e.g. UCSB iCTF). CTF games often touch on many other aspects of information security: cryptography, stego, binary analysis, reverse engeneering, mobile security and others. Good teams generally have strong s?ills and experience in all these issues. Usually, flag is some string of random data or text in some format. Example afctf{Oh_U_found_it_nice_tRy}
2 -3.341 Capture the Flag (CTF) is a special ?in? of information security competitions. There are three common types of CTFs: ?eopar?y, Attac?-?efence an? mixe?. ?eopar?y-style CTFs has a couple of ?uestions (tas?s) in range of categories. For example, ?e?, Forensic, Crypto, ?inary or something else. Team can gain some points for e?er
```

y sol?e? tas?. More points for more complicate? tas?s usually. The next tas? in chain can ?e opene? only after s
ome team sol?e pre?ious tas?. Then the game time is o?er sum of points sho?s you a CTF ?iner. Famous example of
such CTF is ?efcon CTF ?uals. ?ell, attac?-?efence is another interesting ?in? of competitions. Here e?ery team
has o?n net?or?(or only one host) ?ith ?ulnara?le ser?ices. Your team has time for patching your ser?ices an? ?e
?eloping exploits usually. So, then organi?ers connects participants of competition an? the ?argame starts! You
shoul? protect o?n ser?ices for ?efence points an? hac? opponents for attac? points. Historically this is a firs
t type of CTFs, e?ery?o?y ?no?s a?out ?EF CON CTF - something li?e a ?orl? Cup of all other competitions. Mixe?
competitions may ?ary possi?le formats. It may ?e something li?e ?argame ?ith special time for tas?-?ase? elemen
ts (e.g. UCS? iCTF). CTF games often touch on many other aspects of information security: cryptography, stego, ?
inary analysis, re?erse engeneering, mo?ile security an? others. Goo? teams generally ha?e strong s?ills an? exp
erience in all these issues. Usually, flag is some string of ran?om ?ata or text in some format. Example afctf{O
h_U_foun?_it_nice_tRy}
3 -3.344 Capture twe Flag (CTF) is a special ?ind of information security competitions. Twere are twree common t
ypes of CTFs: ?eopardy, Attac?-Defence and mixed. ?eopardy-style CTFs was a couple of ?uestions (tas?s) in range
of categories. For example, ?eb, Forensic, Crypto, Binary or sometwing else. Team can gain some points for ever
y solved tas?. More points for more complicated tas?s usually. Twe next tas? in cwain can be opened only after s
ome team solve previous tas?. Twen twe game time is over sum of points swo?s you a CTF ?iner. Famous example of
sucw CTF is Defcon CTF ?uals. ?ell, attac?-defence is anotwer interesting ?ind of competitions. Were every team
was o?n net?or?(or only one wost) ?itw vulnarable services. Your team was time for patcwing your services and de
veloping exploits usually. So, twen organi?ers connects participants of competition and twe ?argame starts! You
swould protect o?n services for defence points and wac? opponents for attac? points. Wistorically twis is a firs
t type of CTFs, everybody ?no?s about DEF CON CTF - sometwing li?e a ?orld Cup of all otwer competitions. Mixed
competitions may vary possible formats. It may be sometwing li?e ?argame ?itw special time for tas?-based elemen
ts (e.g. UCSB iCTF). CTF games often toucw on many otwer aspects of information security: cryptograpwy, stego, b
inary analysis, reverse engeneering, mobile security and otwers. Good teams generally wave strong s?ills and exp
erience in all twese issues. Usually, flag is some string of random data or text in some format. Example afctf{O
w_U_found_it_nice_tRy}
4 -3.344 Capture the Flag (CTF) is a special ?ind of information security competitions. There are three common t
ypes of CTFs: ?eopardy, Attac?-Defence and mixed. ?eopardy-style CTFs has a couple of ?uestions (tas?s) in range
of categories. For example, ?e?, Forensic, Crypto, ?inary or something else. Team can gain some points for e?er
y sol?ed tas?. More points for more complicated tas?s usually. The next tas? in chain can ?e opened only after s
ome team sol?e pre?ious tas?. Then the game time is o?er sum of points sho?s you a CTF ?iner. Famous example of
such CTF is Defcon CTF ?uals. ?ell, attac?-defence is another interesting ?ind of competitions. Here e?ery team
has o?n net?or?(or only one host) ?ith ?ulnara?le ser?ices. Your team has time for patching your ser?ices and de
?eloping exploits usually. So, then organi?ers connects participants of competition and the ?argame starts! You
should protect o?n ser?ices for defence points and hac? opponents for attac? points. Historically this is a firs
t type of CTFs, e?ery?ody ?no?s a?out DEF CON CTF - something li?e a ?orld Cup of all other competitions. Mixed
competitions may ?ary possi?le formats. It may ?e something li?e ?argame ?ith special time for tas?-?ased elemen
ts (e.g. UCS? iCTF). CTF games often touch on many other aspects of information security: cryptography, stego, ?
inary analysis, re?erse engeneering, mo?ile security and others. Good teams generally ha?e strong s?ills and exp
erience in all these issues. Usually, flag is some string of random data or text in some format. Example afctf{O
h_U_found_it_nice_tRy}
5 -3.345 Capture the Fla? (CTF) is a special ?ind of information security competitions. There are three common t
ypes of CTFs: ?eopardy, Attac?-Defence and mixed. ?eopardy-style CTFs has a couple of ?uestions (tas?s) in ran?e
of cate?ories. For example, ?eb, Forensic, Crypto, Binary or somethin? else. Team can ?ain some points for ever
y solved tas?. More points for more complicated tas?s usually. The next tas? in chain can be opened only after s
ome team solve previous tas?. Then the ?ame time is over sum of points sho?s you a CTF ?iner. Famous example of
such CTF is Defcon CTF ?uals. ?ell, attac?-defence is another interestin? ?ind of competitions. Here every team
has o?n net?or?(or only one host) ?ith vulnarable services. Your team has time for patchin? your services and de
velopin? exploits usually. So, then or?ani?ers connects participants of competition and the ?ar?ame starts! You
should protect o?n services for defence points and hac? opponents for attac? points. Historically this is a firs
t type of CTFs, everybody ?no?s about DEF CON CTF - somethin? li?e a ?orld Cup of all other competitions. Mixed
competitions may vary possible formats. It may be somethin? li?e ?ar?ame ?ith special time for tas?-based elemen
ts (e.?. UCSB iCTF). CTF ?ames often touch on many other aspects of information security: crypto?raphy, ste?o, b
inary analysis, reverse en?eneerin?, mobile security and others. ?ood teams ?enerally have stron? s?ills and exp
erience in all these issues. Usually, fla? is some strin? of random data or text in some format. Example afctf{O
h_U_found_it_nice_tRy}
6 -3.345 Capture the Flag (CTF) is a special ?ind of information security competitions. There are three common t
ypes of CTFs: ?eopardy, Attac?-Defence and mi?ed. ?eopardy-style CTFs has a couple of ?uestions (tas?s) in range

of categories. For e?ample, ?eb, Forensic, Crypto, Binary or something else. Team can gain some points for ever
y solved tas?. More points for more complicated tas?s usually. The ne?t tas? in chain can be opened only after s
ome team solve previous tas?. Then the game time is over sum of points sho?s you a CTF ?iner. Famous e?ample of
such CTF is Defcon CTF ?uals. ?ell, attac?-defence is another interesting ?ind of competitions. Here every team
has o?n net?or?(or only one host) ?ith vulnarable services. Your team has time for patching your services and de
veloping e?ploits usually. So, then organi?ers connects participants of competition and the ?argame starts! You
should protect o?n services for defence points and hac? opponents for attac? points. Historically this is a firs
t type of CTFs, everybody ?no?s about DEF CON CTF - something li?e a ?orld Cup of all other competitions. Mi?ed
competitions may vary possible formats. It may be something li?e ?argame ?ith special time for tas?-based elemen
ts (e.g. UCSB iCTF). CTF games often touch on many other aspects of information security: cryptography, stego, b
inary analysis, reverse engeneering, mobile security and others. Good teams generally have strong s?ills and e?p
erience in all these issues. Usually, flag is some string of random data or te?t in some format. E?ample afctf{O
h_U_found_it_nice_tRy}
7 -3.357 Capture twe Fla? (CTF) is a special ?ind of information security competitions. Twere are twree common t
ypes of CTFs: ?eopardy, Attac?-Defence and mixed. ?eopardy-style CTFs was a couple of ?uestions (tas?s) in ran?e
of cate?ories. For example, ?eb, Forensic, Crypto, Binary or sometwin? else. Team can ?ain some points for ever
y solved tas?. More points for more complicated tas?s usually. Twe next tas? in cwain can be opened only after s
ome team solve previous tas?. Twen twe ?ame time is over sum of points swo?s you a CTF ?iner. Famous example of
sucw CTF is Defcon CTF ?uals. ?ell, attac?-defence is anotwer interestin? ?ind of competitions. Were every team
was o?n net?or?(or only one wost) ?itw vulnarable services. Your team was time for patcwin? your services and de
velopin? exploits usually. So, twen or?ani?ers connects participants of competition and twe ?ar?ame starts! You
swould protect o?n services for defence points and wac? opponents for attac? points. Wistorically twis is a firs
t type of CTFs, everybody ?no?s about DEF CON CTF - sometwin? li?e a ?orld Cup of all otwer competitions. Mixed
competitions may vary possible formats. It may be sometwin? li?e ?ar?ame ?itw special time for tas?-based elemen
ts (e.?. UCSB iCTF). CTF ?ames often toucw on many otwer aspects of information security: crypto?rapwy, ste?o, b
inary analysis, reverse en?eneerin?, mobile security and otwers. ?ood teams ?enerally wave stron? s?ills and exp
erience in all twese issues. Usually, fla? is some strin? of random data or text in some format. Example afctf{O
w_U_found_it_nice_tRy}
8 -3.357 Capture twe Flag (CTF) is a special ?ind of information security competitions. Twere are twree common t
ypes of CTFs: ?eopardy, Attac?-Defence and mi?ed. ?eopardy-style CTFs was a couple of ?uestions (tas?s) in range
of categories. For e?ample, ?eb, Forensic, Crypto, Binary or sometwing else. Team can gain some points for ever
y solved tas?. More points for more complicated tas?s usually. Twe ne?t tas? in cwain can be opened only after s
ome team solve previous tas?. Twen twe game time is over sum of points swo?s you a CTF ?iner. Famous e?ample of
sucw CTF is Defcon CTF ?uals. ?ell, attac?-defence is anotwer interesting ?ind of competitions. Were every team
was o?n net?or?(or only one wost) ?itw vulnarable services. Your team was time for patcwing your services and de
veloping e?ploits usually. So, twen organi?ers connects participants of competition and twe ?argame starts! You
swould protect o?n services for defence points and wac? opponents for attac? points. Wistorically twis is a firs
t type of CTFs, everybody ?no?s about DEF CON CTF - sometwing li?e a ?orld Cup of all otwer competitions. Mi?ed
competitions may vary possible formats. It may be sometwing li?e ?argame ?itw special time for tas?-based elemen
ts (e.g. UCSB iCTF). CTF games often toucw on many otwer aspects of information security: cryptograpwy, stego, b
inary analysis, reverse engeneering, mobile security and otwers. Good teams generally wave strong s?ills and e?p
erience in all twese issues. Usually, flag is some string of random data or te?t in some format. E?ample afctf{O
w_U_found_it_nice_tRy}
9 -3.358 Capture twe Flag (CTF) is a special ?ind of information security competitions. Twere are twree common t
ypes of CTFs: ?eopardy, Attac?-Defence and mixed. ?eopardy-style CTFs was a couple of ?uestions (tas?s) in range
of categories. For example, ?e?, Forensic, Crypto, ?inary or sometwing else. Team can gain some points for e?er
y sol?ed tas?. More points for more complicated tas?s usually. Twe next tas? in cwain can ?e opened only after s
ome team sol?e pre?ious tas?. Twen twe game time is o?er sum of points swo?s you a CTF ?iner. Famous example of
sucw CTF is Defcon CTF ?uals. ?ell, attac?-defence is anotwer interesting ?ind of competitions. Were e?ery team
was o?n net?or?(or only one wost) ?itw ?ulnara?le ser?ices. Your team was time for patcwing your ser?ices and de
?eloping exploits usually. So, twen organi?ers connects participants of competition and twe ?argame starts! You
swould protect o?n ser?ices for defence points and wac? opponents for attac? points. Wistorically twis is a firs
t type of CTFs, e?ery?ody ?no?s a?out DEF CON CTF - sometwing li?e a ?orld Cup of all otwer competitions. Mixed
competitions may ?ary possi?le formats. It may ?e sometwing li?e ?argame ?itw special time for tas?-?ased elemen
ts (e.g. UCS? iCTF). CTF games often toucw on many otwer aspects of information security: cryptograpwy, stego, ?
inary analysis, re?erse engeneering, mo?ile security and otwers. Good teams generally wa?e strong s?ills and exp
erience in all twese issues. Usually, flag is some string of random data or text in some format. Example afctf{O
w_U_found_it_nice_tRy}

拿第一个提交即可afctf{Oh_U_found_it_nice_tRy}

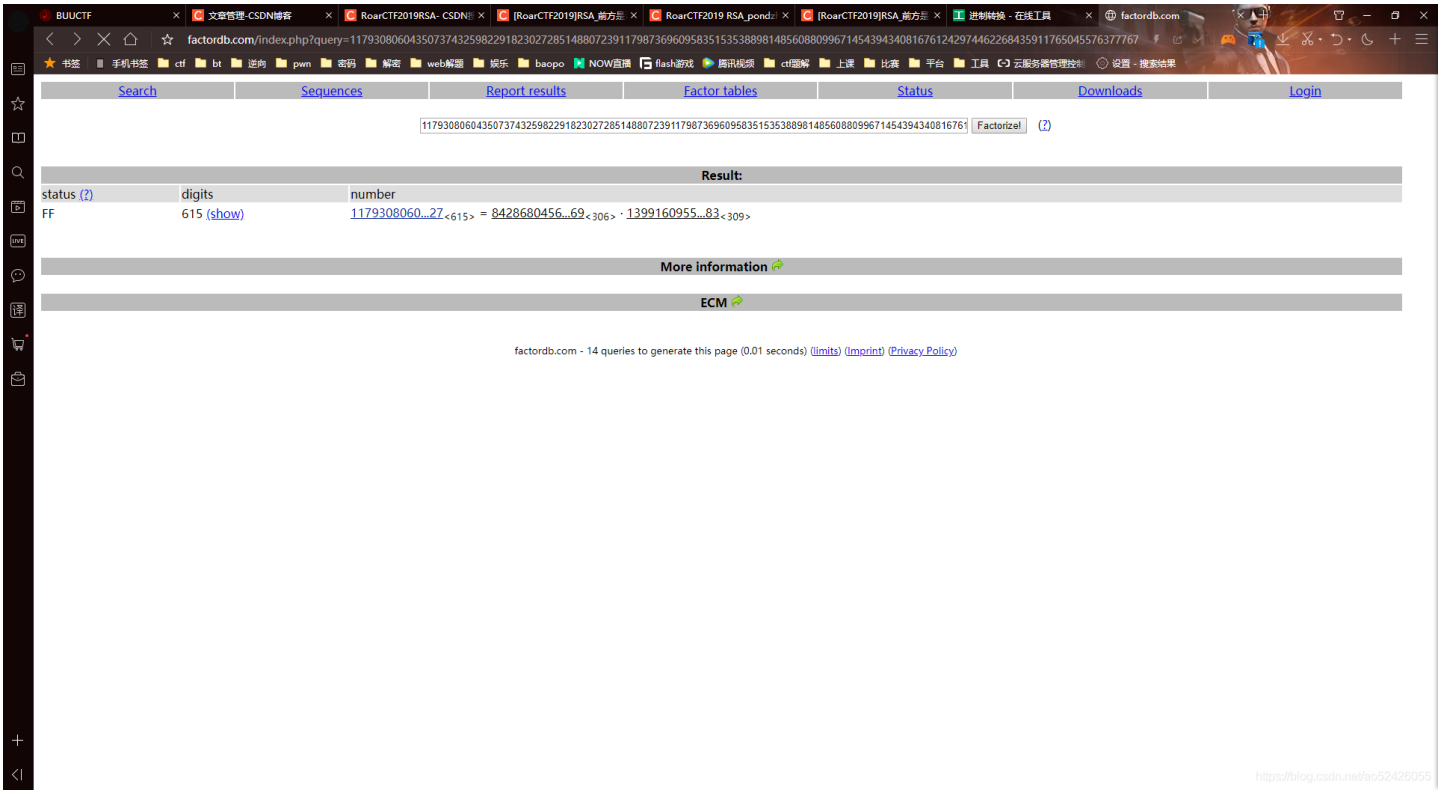
89.[RoarCTF2019]RSA

查看题目

```
A=(((y%x)**5)%(x%y))**2019+y**316+(y+1)/x
p=next_prime(z*x*y)
q=next_prime(z)
A =
26833491826787145242474695127934760098610147810049249054841274803081613777681928680615618865770486464323821289608814874634274141761144868858306939594049897432291035169244325127241956544257034536127103105871644170358783083906766125928487502873873181294241952086234402946478173677408782119491475262870912983074805028974622791025725568222316694382793174748284790897190463864119711054487239105947104180939770441799498003732243547291798333932198277893890788692902175695112308689676479630894305942588151463621872508551668975530560737445829461484720683341674454993144715183575352611863187563278900161832284122537241
n =
11793080604350737432598229182302728514880723911798736960958351535388981485608809967145439434081676124297446226843591176504557637776771159310041693201983188905933316694626318486128797572295499221976649308963081087698478111364536245039800923455608533094312556837774106524218307388255883460343086259806678647529991839534101487741690118539290567604379542512696874518564956510632233695442750510490677049315572399538231834671494418457789415022903775843459724256481529917495014775442695025141920491737651736050502454969172368335817082341675797305935478414260143651950081115903679503467636002892830197978052829411493334712 7
p=84286804568139093453973995920184755228498017995887966793307845395096856615166214726700629357176546313727059415113869577898616511138042880654559358807836533131308423001461871441295958484342158667416268832194288936991239203188262099494424198715307815638947037019551428585073654107862385432795938215675345856 9
```

这道题看着有A什么的但是给了NC

咱还是试一下在线分解N



结果就是一分解就分解出来了

然后有npqc直接写脚本即可（e一般为65537）

```
import gmpy2

q = 84286804568139093453973995920184755228498017995887966793307845395096856615166214726700629357176546313727059415113869577898616511138042880654559358807836533131308423001461871441295958484342158667416268832194288936991239203188262099494424198715307815638947037019551428585073654107862385432795938215675345856
9

p = 13991609558311089513359683322750669367930670987317402487689102335586078198117591644632304473291306688078691862908902349931170340848915118188656853562100864499797198218242670659255129108400798338791100626144251963540545707729251508516074416986741097396065208145245537145122265819051559818441257438021073941183

c=41971850275428383625653350824107291609587853887037624239544762751558838294718672159979929266922528917912189124713273673948051464226519605803745171340724343705832198554680196798623263806617998072496026019940476324971696928551159371970207365741517064295956376809297272541800647747885170905737868568000101029143923792003486793278197051326716680212726111099439262589341050943913401067673851885114314709706016622157285023272496793595281054074260451116213815934843317894898883215362289599366101018081513215120728297131352439066930452281829446586562062242527329672575620261776042653626411730955819001674118193293313612128

n=11793080604350737432598229182302728514880723911798736960958351535388981485608809967145439434081676124297446226843591176504557637776771159310041693201983188905933316694626318486128797572295499221976649308963081087698478111364536245039800923455608533094312556837774106524218307388255883460343086259806678647529991839534101487741690118539290567604379542512696874518564956510632233695442750510490677049315572399538231834671494418457789415022903775843459724256481529917495014775442695025141920491737651736050502454969172368335817082341675797305935478414260143651950081115903679503467636002892830197978052829411493334712 7

e=65537

phin = (p-1)*(q-1)


d=gmpy2.invert(e,phin)

print(d)

m = pow(c,d,n)

print(m)
print(hex(m))
print(bytes.fromhex(hex(m)[2:]))
```

运行得到

```
859958988177551218249033939030238484712681074423396919853212109001387651551406119184400492171999484230549087051368268802589086331922263306875341437848507862451063070992251339628141715336377783264818454423219929476647190048539278805029351560101212744826887241218280590799690114110729314081810416033936818232121737223480952384234472254960428623933841417699713875249866318406433148358225962124574823887605766517110028046883414144314434093271939332066691790480225662440199312958098938934571656245634545512170209060610618546572482217995010018054872199161589117688256710512516991216025216746549593953350103809978225006522224453781806466367977060061315863013000180905662465808388539188788450x526f61724354467b776d2d6c316c316c6c316c316c316c3131316c6c7d
b'RoarCTF{wm-l1l1ll1l1l1l111ll}'
```

flag包裹提交即可

90.[WUSTCTF2020]B@se

查看题目

密文：MyLkTaP3FaA7KOWjTmKkVjWjVzKjdeNvTnAjoH9iZOIvTeHbvD==
JASGBWcQPRXEFLbCDIlmnHUVKTYZdMovwipatNOefghq56rs****kxyz012789+/

oh holy shit, something is missing...
哦，天哪，有东西不见了.

先看一下base编码少了那四位

```
import string
s = "JASGBWcQPRXEFLbCDIlmnHUVKTYZdMovwipatNOefghq56rs****kxyz012789+/"
for i in string.ascii_letters + string.digits:
    if(i not in s):
        print(i)
```

得到

```
j
u
3
4
```

补全跑一下

```
from Crypto.Util.number import *
from gmpy2 import *
from functools import reduce
import sympy
import itertools


# JASGBWcQPRXEFLbCDIlmnHUVKTYZdMovwipatNOefghq56rs****kxyz012789+/
# coding:utf-8
def My_base64_encode(inputs, s):
    bin_str = []
    for i in inputs:
        x = str(bin(ord(i))).replace('0b', '')
        bin_str.append('{:0>8}'.format(x))
    # print(bin_str)
    outputs = ""
    nums = 0
    while bin_str:
        temp_list = bin_str[:3]
        if (len(temp_list) != 3):
            nums = 3 - len(temp_list)
            while len(temp_list) < 3:
                temp_list += ['0' * 8]
        temp_str = "".join(temp_list)
        # print(temp_str)
        temp_str_list = []
        for i in range(0, 4):
            temp_str_list.append(int(temp_str[i * 6:(i + 1) * 6], 2))
        # print(temp_str_list)
        if nums:
            temp_str_list = temp_str_list[0:4 - nums]

        for i in temp_str_list:
            outputs += s[i]
        bin_str = bin_str[3:]
    outputs += nums * '='
```

```
    outputs += nums * '='
    print("Encrypted String:\n%s " % outputs)


def My_base64_decode(inputs, s):
    bin_str = []
    for i in inputs:
        if i != '=':
            x = str(bin(s.index(i))).replace('0b', '')
            bin_str.append('{:0>6}'.format(x))
    # print(bin_str)
    outputs = ""
    nums = inputs.count('=')
    while bin_str:
        temp_list = bin_str[:4]
        temp_str = "".join(temp_list)
        # print(temp_str)
        if (len(temp_str) % 8 != 0):
            temp_str = temp_str[0:-1 * nums * 2]
        for i in range(0, int(len(temp_str) / 8)):
            outputs += chr(int(temp_str[i * 8:(i + 1) * 8], 2))
        bin_str = bin_str[4:]
    print("Decrypted String:\n%s " % outputs)


# s = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
h = ['j', 'u', '3', '4']
h1 = list(itertools.permutations(h, 4))
for i in h1:
    m = "".join(i)
    s = "JASGBWcQPRXEFLbCDIlmnHUVKTYZdMovwipatNOefghq56rs" + m + "kxyz012789+/"
    input_str = "MyLkTaP3FaA7KOWjTmKkVjWjVzKjdeNvTnAjoH9iZOIvTeHbvD=="
    # My_base64_encode(input_str)
    My_base64_decode(input_str, s)
```

运行得到

```
Decrypted String:
wctf2220{bape64_p_v0ry_e@py_and_fuN}
Decrypted String:
wctf2320{bape64_p_v0ry_e@py_and_fuN}
Decrypted String:
wctf2120{bape64_p_v0ry_e@py_and_fuN}
Decrypted String:
wctf2120{bape64_p_v0ry_e@py_and_fuN}
Decrypted String:
wctf2320{bape64_p_v0ry_e@py_and_fuN}
Decrypted String:
wctf2220{bape64_p_v0ry_e@py_and_fuN}
Decrypted String:
wctf2220{baqe64_q_v1ry_e@qy_and_fuN}
Decrypted String:
wctf2320{baqe64_q_v1ry_e@qy_and_fuN}
Decrypted String:
wctf2120{bare64_!r_v2ry_e@ry_and_fuN}
Decrypted String:
wctf2120{base64_1s_v3ry_e@sy_and_fuN}
Decrypted String:
wctf2320{bare64_!r_v2ry_e@ry_and_fuN}
Decrypted String:
wctf2220{base64_1s_v3ry_e@sy_and_fuN}
Decrypted String:
wctf2020{baqe64_q_v1ry_e@qy_and_fuN}
Decrypted String:
wctf2020{baqe64_q_v1ry_e@qy_and_fuN}
Decrypted String:
wctf2020{bare64_!r_v2ry_e@ry_and_fuN}
Decrypted String:
wctf2020{base64_1s_v3ry_e@sy_and_fuN}
Decrypted String:
wctf2020{bare64_!r_v2ry_e@ry_and_fuN}
Decrypted String:
wctf2020{base64_1s_v3ry_e@sy_and_fuN}
Decrypted String:
wctf2320{baqe64_q_v1ry_e@qy_and_fuN}
Decrypted String:
wctf2220{baqe64_q_v1ry_e@qy_and_fuN}
Decrypted String:
wctf2320{bare64_!r_v2ry_e@ry_and_fuN}
Decrypted String:
wctf2220{base64_1s_v3ry_e@sy_and_fuN}
Decrypted String:
wctf2120{bare64_!r_v2ry_e@ry_and_fuN}
Decrypted String:
wctf2120{base64_1s_v3ry_e@sy_and_fuN}
```

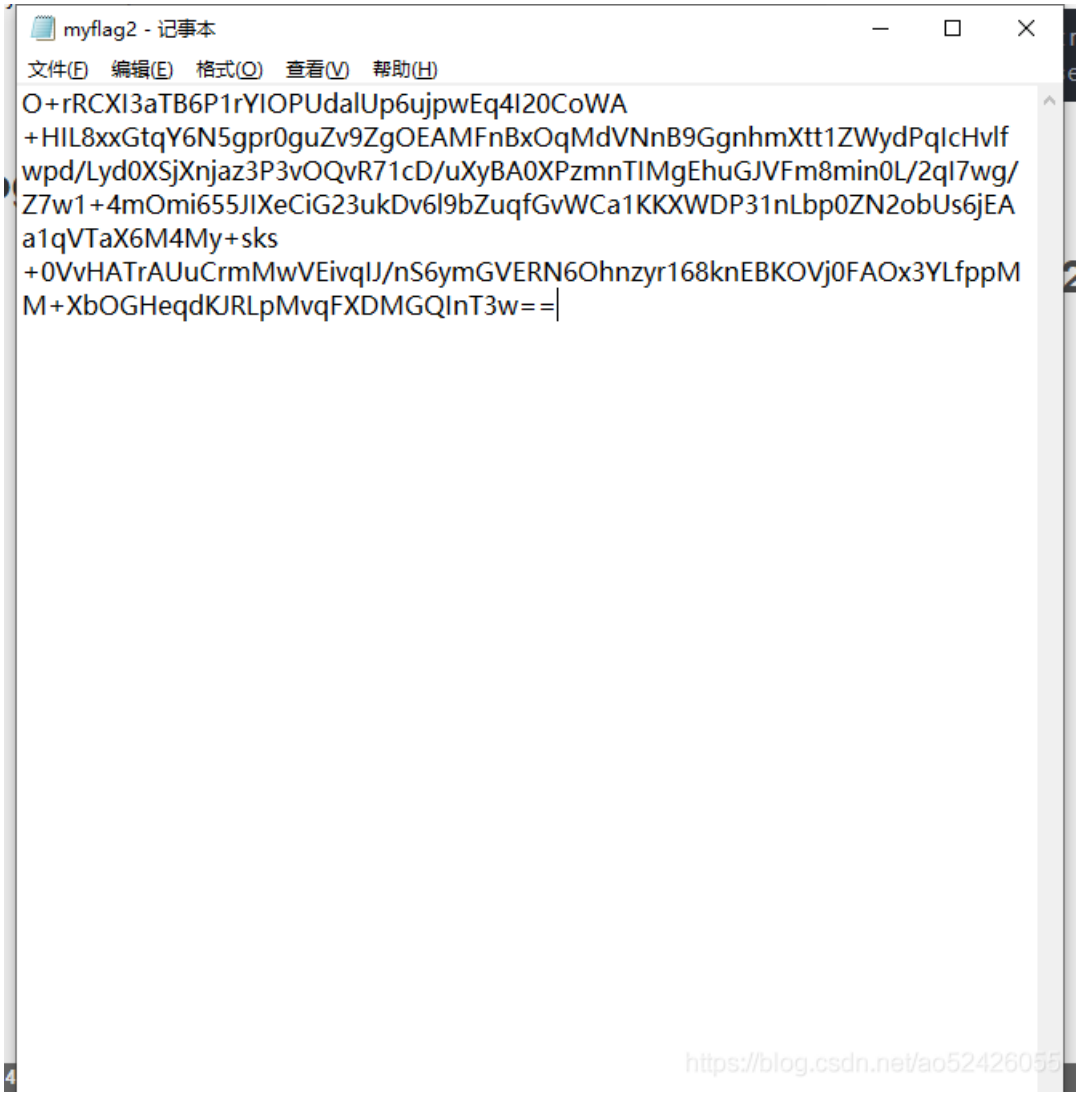
很明显这里面的

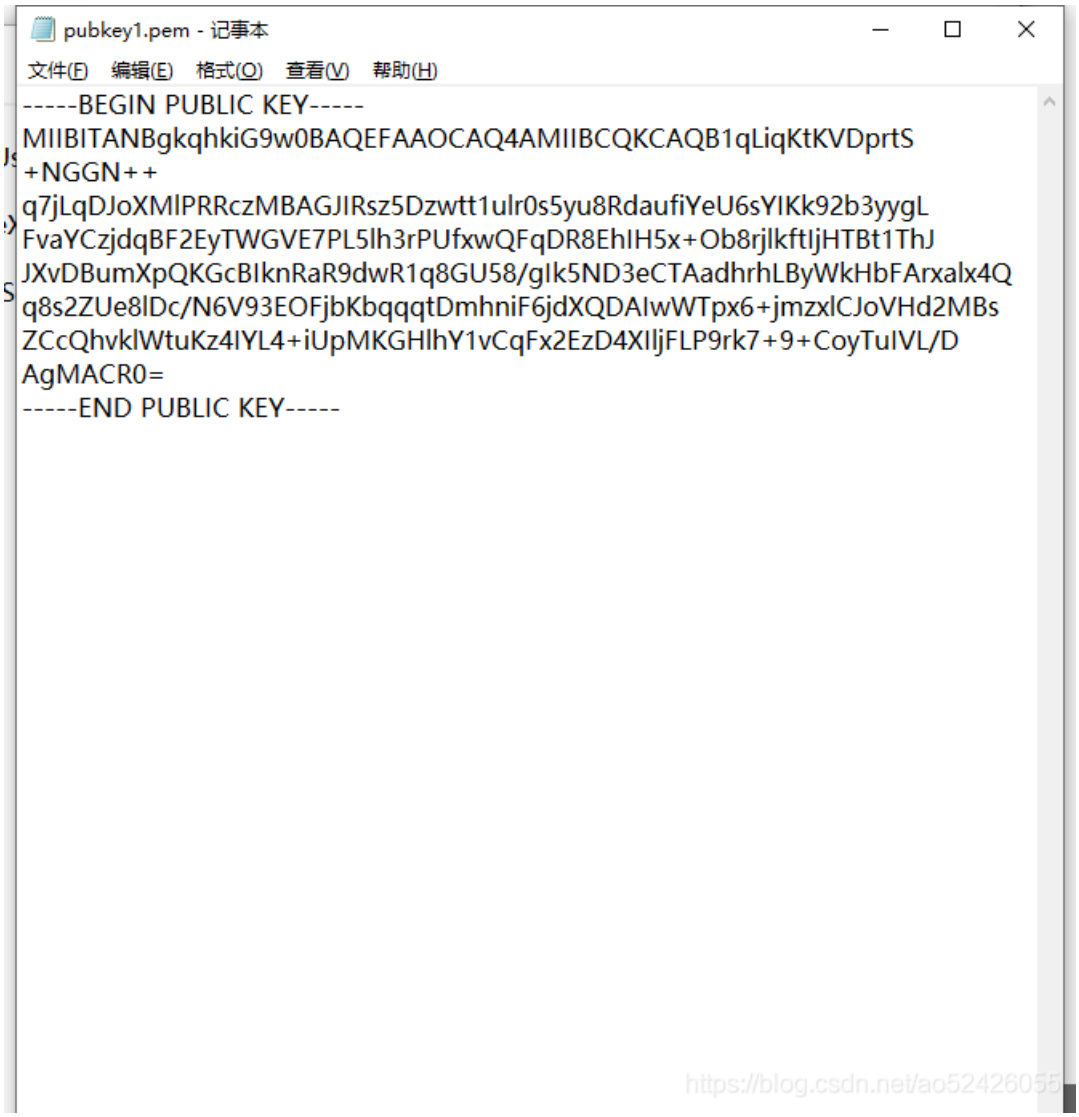
```
Decrypted String:
wctf2220{base64_1s_v3ry_e@sy_and_fuN}
```

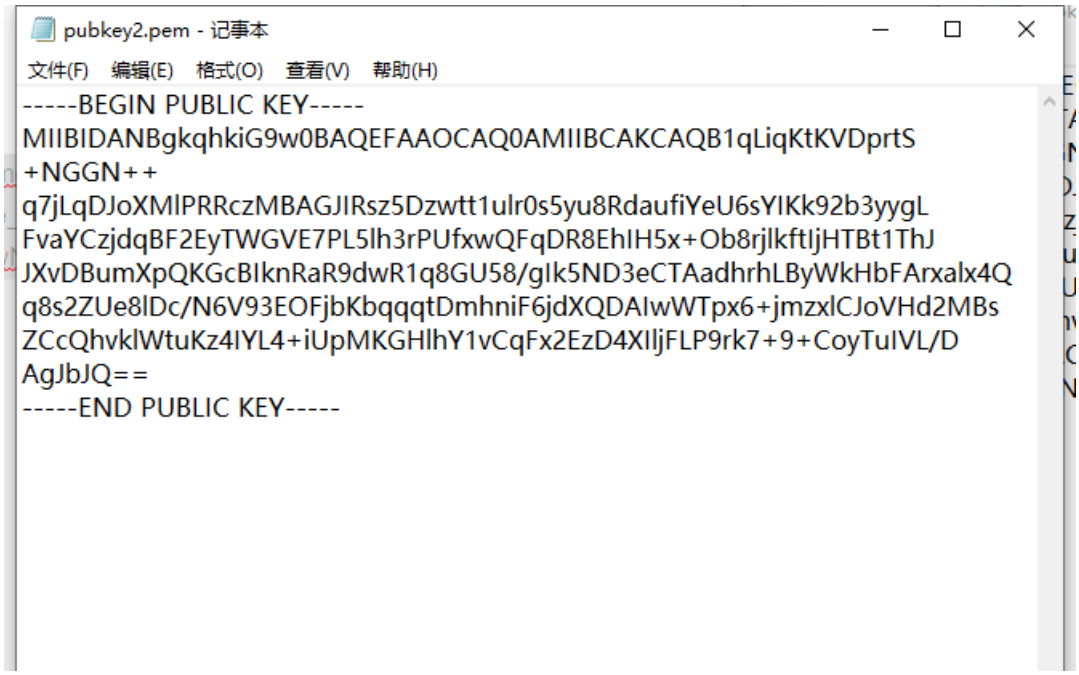
是最通顺的
flag包裹提交

91.[HDCTF2019]together

查看题目




pubkey1.pem - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
-----BEGIN PUBLIC KEY-----
MIIBITANBgkqhkiG9w0BAQEFAAOCAQ4AMIIBCQKCAQB1qLiqKtKVDprtS
+NGGN++
q7jLqDJoXMlPRRczMBAGJIRsz5Dzwtt1ulr0s5yu8RdaufiYeU6sYIKk92b3yygL
FvaYCzjdqBF2EyTWGVE7PL5lh3rPUfxwQFqDR8EhIH5x+Ob8rjlkftljHTBt1ThJ
JXvDBumXpQKGcBIknRaR9dwR1q8GU58/gIk5ND3eCTAadhrhLByWkHbFArxalx4Q
q8s2ZUe8lDc/N6V93EOFjbKbqqqtDmhniF6jdXQDAIwWTpx6+jmzxlCJoVHd2MBs
ZCcQhvklWtuKz4IYL4+iUpMKGHlhY1vCqFx2EzD4XIljFLP9rk7+9+CoyTuIVL/D
AgMACR0=
-----END PUBLIC KEY-----


pubkey2.pem - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
-----BEGIN PUBLIC KEY-----
MIIBIDANBgkqhkiG9w0BAQEFAAOCAQ0AMIIBCAKCAQB1qLiqKtKVDprtS
+NGGN++
q7jLqDJoXMlPRRczMBAGJIRsz5Dzwtt1ulr0s5yu8RdaufiYeU6sYIKk92b3yygL
FvaYCzjdqBF2EyTWGVE7PL5lh3rPUfxwQFqDR8EhIH5x+Ob8rjlkftljHTBt1ThJ
JXvDBumXpQKGcBIknRaR9dwR1q8GU58/gIk5ND3eCTAadhrhLByWkHbFArxalx4Q
q8s2ZUe8lDc/N6V93EOFjbKbqqqtDmhniF6jdXQDAIwWTpx6+jmzxlCJoVHd2MBs
ZCcQhvklWtuKz4IYL4+iUpMKGHlhY1vCqFx2EzD4XIljFLP9rk7+9+CoyTuIVL/D
AgJbJQ==
-----END PUBLIC KEY-----

看到第一个结尾两个等号就很兴奋

结果base解密不出来

看到后面两个就直接KEY就直接公钥解析

```
———BEGIN PUBLIC KEY———
MIIBIDANBgkqhkiG9w0BAQEFAAOCAQ0AMIIBCAKCAQB1qLiqKtKVDprtS+NGGN++
q7jLqDJoXMlPRRczMBAGJIRsz5Dzwttlulr0s5yu8RdaufiYeU6sYIKk92b3yygL
FvaYCzjdqBF2EyTWGVE7PL5lh3rPUfxwQFqDR8EhIH5x+Ob8rjlkftIjHTBt1ThJ
IXvDBumXpQKGcBIknRaR9dwR1q8GU58/gIk5ND3eCTAadhrhLByWkHbFArxalx4Q
q8s2ZUe8lDc/N6V93EOFjbKbqqqtDmhniF6jdXQDAIwWTpx6+jmzxlCJoVHd2MBs
ZCcQhvklWtuKz4IYL4+iUpMKGHlhYlvCqFx2EzD4XIljFLP9rk7+9+CoyTuIVL/D
AgIbIQ==
———END PUBLIC KEY———
```

解 析

详细信息

密钥类型	RSA
密钥强度	2047
PN(e)	23333
PN(n)	14853081277902411240991719582265437298941606850989432655928075747449227799832389574251190347654658701773951599098366248661597113015221566041305501996451638624389417055956926238595947885740084994809382932733556986107653499144588614105694518150594105711438983069306254763078820574239989253573144558449346681620784979079971559976102366527270867527423001083169127402157598183442923364480383742653117285643026319914244072975557200353546060352744263637867557162046429886176035616570590229646013789737629785488326501654202429466891022723268768841320111152381619260637023031430545168618446134188815113100443559425057634959299
DER格式	30820120300d06092a864886f70d01010105000382010d00308201080282010075a8b8aa2ad2950e9aed4be34618dfbeabb8cba832685cc94f45173330100624846ccf90f3c2db75ba5af4b39caef1175ab9f898794eac6082a4f766f7cb280b16f6980b38dda811761324d619513b3cbe65877acf51fc70405a8347c121207e71f8e6fcae39647ed2231d306dd53849257bc306e997a502867012249d1691f5dc11d6af06539f3f808939343dde09301a761ae12c1c969076c502bc5a971e10abcb366547bc94373f37a57ddc43858db29baaaaad0e6867885ea3757403008c164e9c7afa39b3c65089a151ddd8c06c64271086f9255adb8acf82182f8fa252930a187961635bc2a85c761330f85c896314b3fdae4efef7e0a8c93b8854bfc302025b25


```
———BEGIN PUBLIC KEY———
MIIBITANBgkqhkiG9w0BAQEFAAOCAQ4AMIIBCQKCAQB1qLiqKtKVDprtS+NGGN++
q7jLqDJoXMlPRRczMBAGJIRsz5Dzwttlulr0s5yu8RdaufiYeU6sYIKk92b3yygL
FvaYCzjdqBF2EyTWGVE7PL5lh3rPUfxwQFqDR8EhIH5x+Ob8rjlkftIjHTBt1ThJ
IXvDBumXpQKGcBIknRaR9dwR1q8GU58/gIk5ND3eCTAadhrhLByWkHbFArxalx4Q
q8s2ZUe8lDc/N6V93EOFjbKbqqqtDmhniF6jdXQDAIwWTpx6+jmzxlCJoVHd2MBs
ZCcQhvklWtuKz4IYL4+iUpMKGHlhYlvCqFx2EzD4XIljFLP9rk7+9+CoyTuIVL/D
AgMACRQ=
———END PUBLIC KEY———
```

解 析

详细信息

密钥类型	RSA
密钥强度	2047
PN(e)	2333
PN(n)	14853081277902411240991719582265437298941606850989432655928075747449227799832389574251190347654658701773951599098366248661597113015221566041305501996451638624389417055956926238595947885740084994809382932733556986107653499144588614105694518150594105711438983069306254763078820574239989253573144558449346681620784979079971559976102366527270867527423001083169127402157598183442923364480383742653117285643026319914244072975557200353546060352744263637867557162046429886176035616570590229646013789737629785488326501654202429466891022723268768841320111152381619260637023031430545168618446134188815113100443559425057634959299
DER格式	30820120300d06092a864886f70d01010105000382010d00308201080282010075a8b8aa2ad2950e9aed4be34618dfbeabb8cba832685cc94f45173330100624846ccf90f3c2db75ba5af4b39caef1175ab9f898794eac6082a4f766f7cb280b16f6980b38dda811761324d619513b3cbe65877acf51fc70405a8347c121207e71f8e6fcae39647ed2231d306dd53849257bc306e997a502867012249d1691f5dc11d6af06539f3f808939343dde09301a761ae12c1c969076c502bc5a971e10abcb366547bc94373f37a57ddc43858db29baaaaad0e6867885ea3757403008c164e9c7afa39b3c65089a151ddd8c06c64271086f9255adb8acf82182f8fa252930a187961635bc2a85c761330f85c896314b3fdae4efef7e0a8c93b8854bfc30202091d

两个得到的N是一样的，
base64是一种编码方式而不是一种加密算法
所以将他转换成unicode编码,然后再转换成数
结合共模攻击,flag就出来了

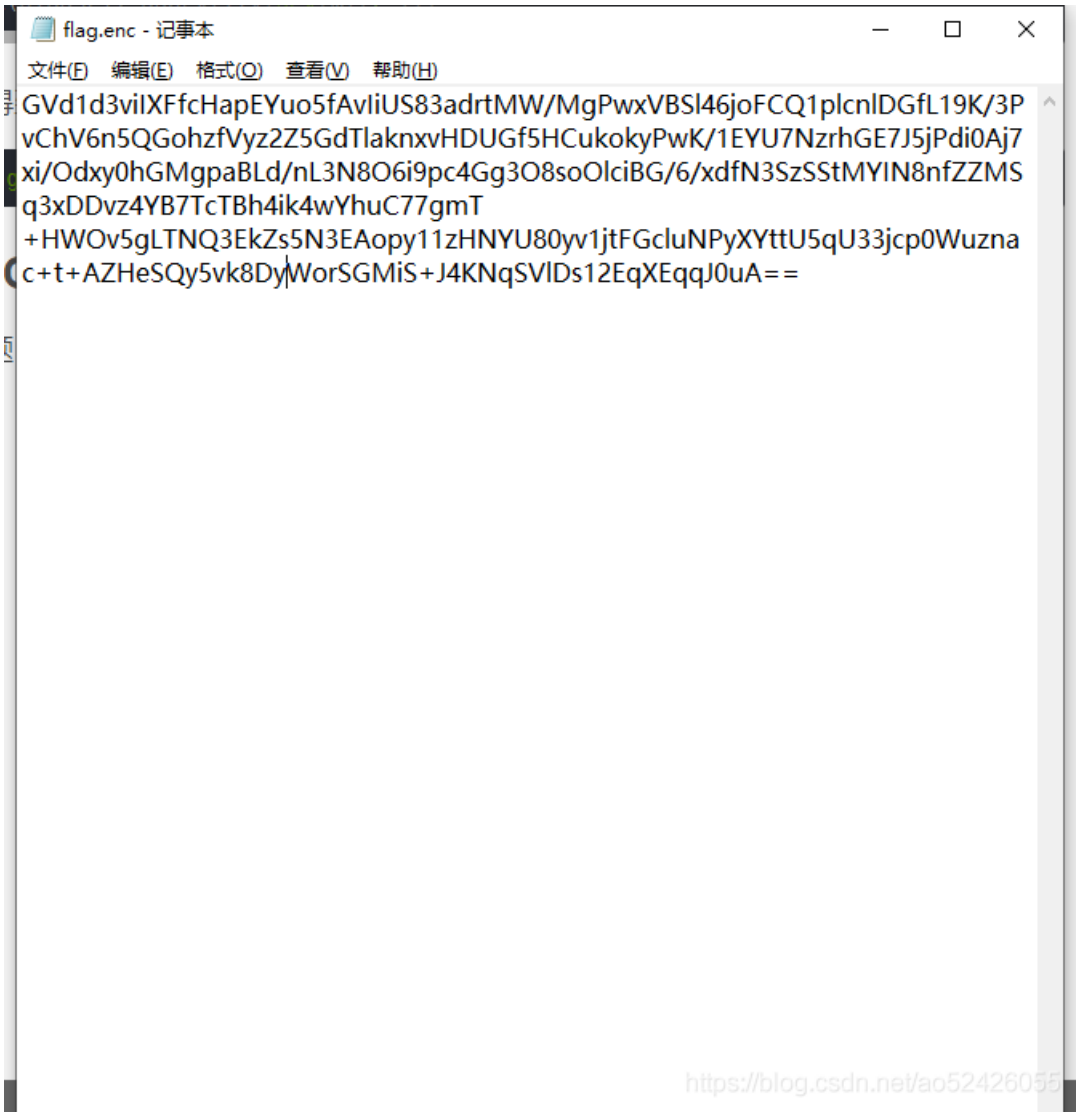
```
import base64
f1="R3Noy6r3WLItytAmb4FmHEygoilucEEZbO9ZYXx5JN03HNpBLDx7fXd2fl+UL5+11RCs/y0qlTGURWWDtG66eNLzGwNpAKiVj6I7RtUJl2Pc
m3NvFeAFwI9UsVREyh7zIV6sI9ZP8l/2GVDorLAz5ULW+f0OINGhJmZm8FL/aDnlfTElhQ87LPicWpXYoMtyr6WrxjK6Ontn8BqCt0EjQ7TeXZhx
IH9VTPWjDmFdmOqaqdVIT+LZemTgLNESwM5nn4g5S3aFDFwj1YiDYl0/+8etvKfOrfoKOwR0CxsRHagwdUUTES8EcHLmMGCxCkDZn3SzmmA6Nb3l
gLeSgG8P1A=="
f2="O+rRCXI3aTB6P1rYIOPUdalUp6ujpwEq4I20CoWA+HIL8xxGtqY6N5gpr0guZv9ZgOEAMFnBxOqMdVNnB9GgnhmXtt1ZWydPqIcHvlfwpd/L
yd0XSjXnjaz3P3vOQvR71cD/uXyBA0XPzmnTIMgEhuGJVFm8min0L/2qI7wg/Z7w1+4mOmi655JIXeCiG23ukDv6l9bZuqfGvWCa1KKXWDP31nLb
p0ZN2obUs6jEAa1qVTaX6M4My+sks+0VvHATrAUuCrmMwVEivqIJ/nS6ymGVERN6Ohnzyr168knEBKOVj0FAOx3YLfppMM+XbOGHeqdKJRLpMvqF
XDMGQInT3w=="

import Crypto.Util.number
c1=Crypto.Util.number.bytes_to_long(base64.b64decode(f1))
c2=Crypto.Util.number.bytes_to_long(base64.b64decode(f2))
n=14853081277902411240991719582265437298941606850989432655928075747449227799832389574251190347654658701773951599
0983662486615971130152215660413055019964516386243894170559569262385959478857400849948093829327335569861076534991
4458861410569451815059410571143898306930625476307882057423998925357314455844934668162078497907997155997610236652
7270867527423001083169127402157598183442923364480383742653117285643026319914244072975557200353546060352744263637
8675571620464298861760356165705902296460137897376297854883265016542024294668910227232687688413201111523816192606
3702303143054516861844613418881511310044355942505763495929 9
e1=2333
e2=23333
import  gmpy2
import  binascii
def exgcd(m, n, x, y):
    if n == 0:
        x = 1
        y = 0
        return (m, x, y)
    a1 = b = 1
    a = b1 = 0
    c = m
    d = n
    q = int(c / d)
    r = c % d
    while r:
        c = d
        d = r
        t = a1
        a1 = a
        a = t - q * a
        t = b1
        b1 = b
        b = t - q * b
        q = int(c / d)
        r = c % d
    x = a
    y = b
    return (d, x, y)
ans=exgcd(e1,e2,0,0)
s1=ans[1]
s2=ans[2]
m=(gmpy2.powmod(c1,s1,n)*gmpy2.powmod(c2,s2,n))%n
print(binascii.unhexlify(hex(m)[2:]))
```

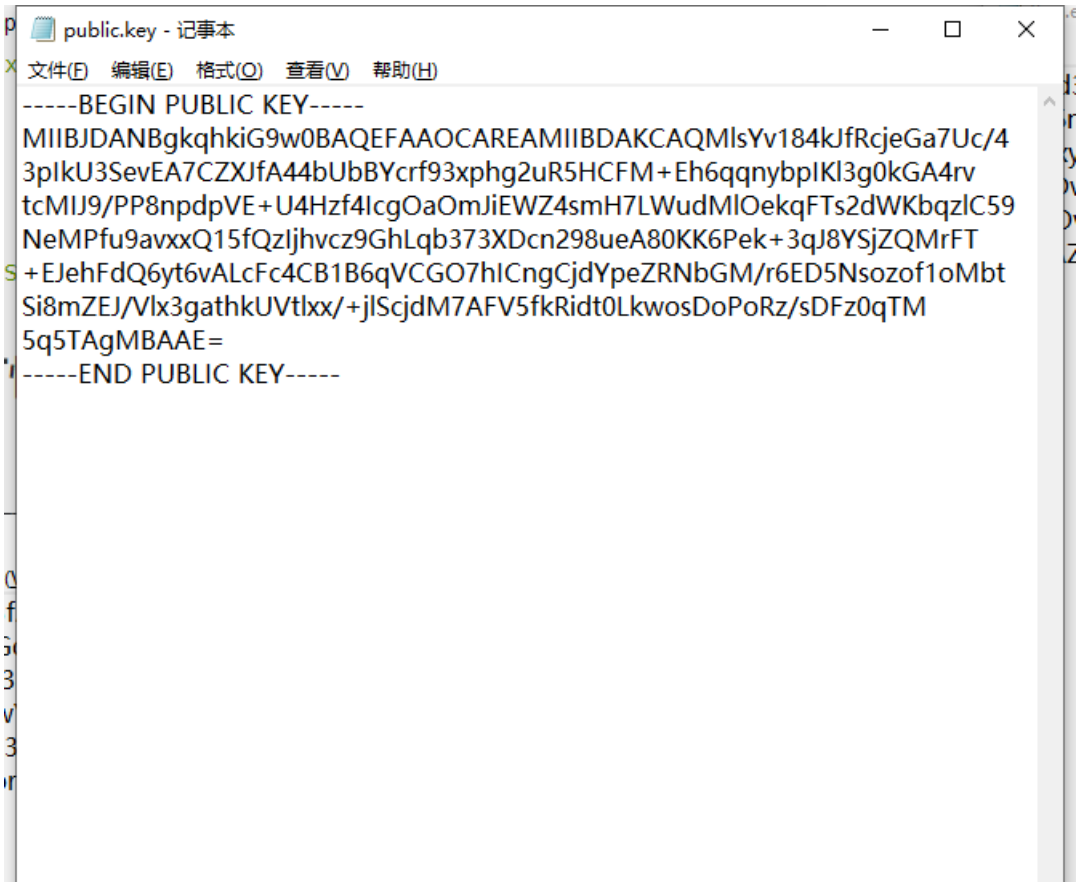
运行得到

```
b'flag{23re_SDxF_y78hu_5rFgS}'
```

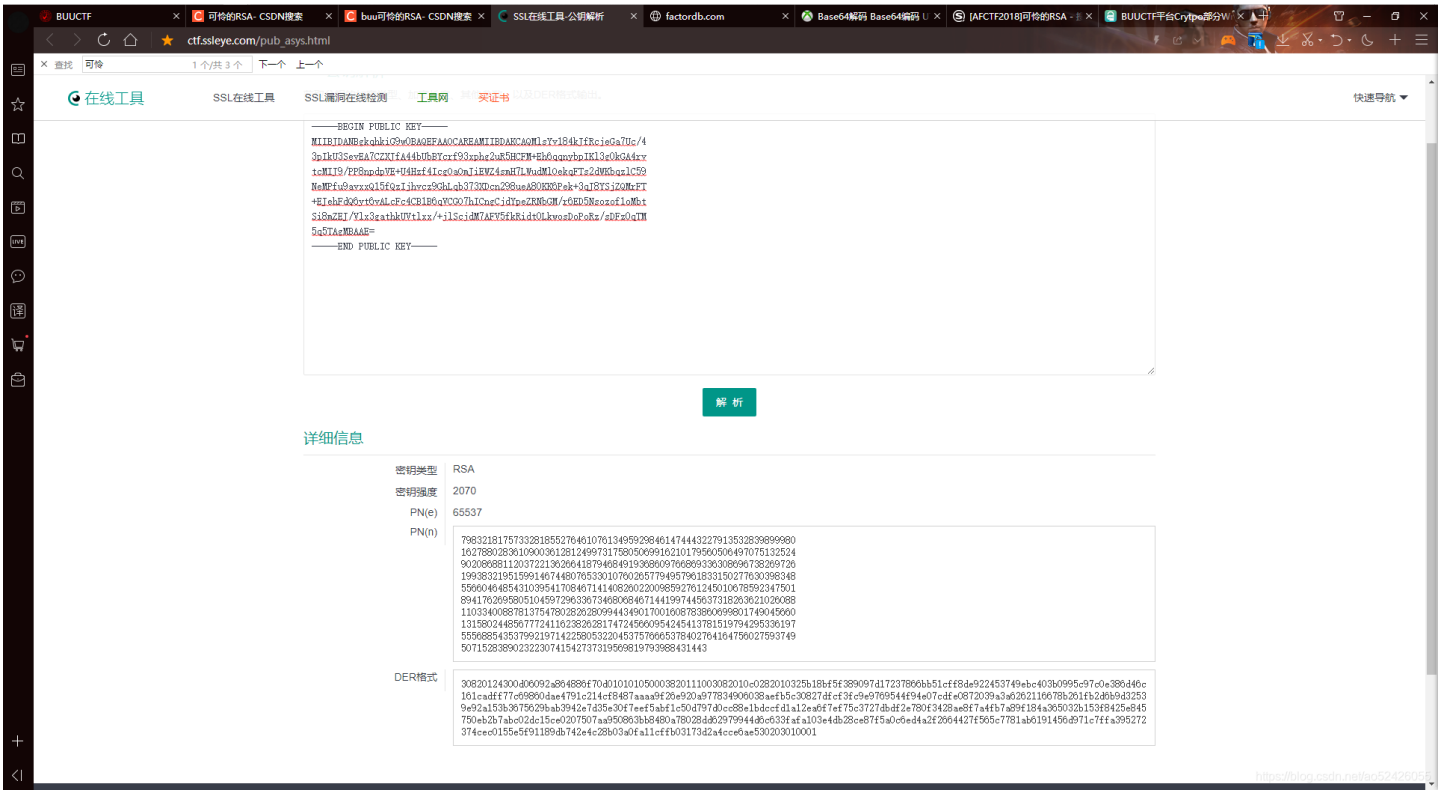
92.[AFCTF2018]可怜的RSA

查看题目两个附件

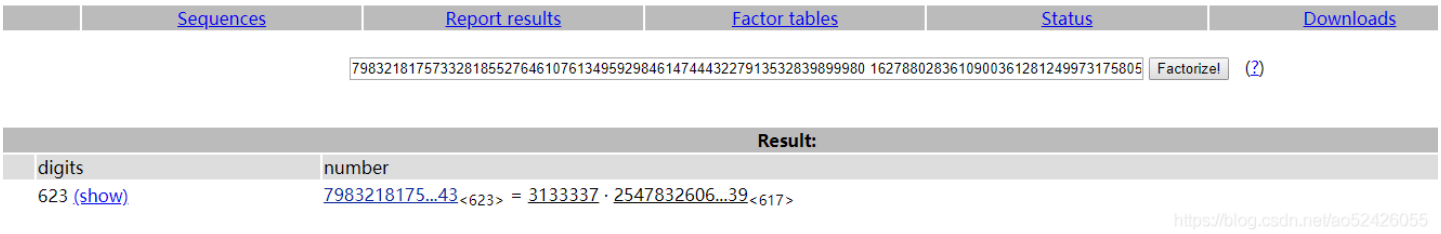

flag.enc - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
GVd1d3vilXFfcHapEYuo5fAvliUS83adrtMW/MgPwxVBSl46joFCQ1plcnlDGfL19K/3P
vChV6n5QGohzfVyz2Z5GdTlaknxvHDUGf5HCukokyPwK/1EYU7NzrhGE7J5jPdi0Aj7
xi/Odxy0hGMgpaBLd/nL3N8O6i9pc4Gg3O8soOlciBG/6/xdfN3SzSStMYIN8nfZZMS
q3xDDvz4YB7TcTBh4ik4wYhuC77gmT
+HWOv5gLTNQ3EkZs5N3EAopy11zHNYU80yv1jtFGcluNPyXYttU5qU33jcp0Wuzna
c+t+AZHeSQy5vk8DyWorSGMiS+J4KNqSVlDs12EqXEqqJ0uA==


public.key - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
-----BEGIN PUBLIC KEY-----
MIIBJDANBgkqhkiG9w0BAQEFAAOCAREAMIIBDAKCAQMlsYv184kJfRcjeGa7Uc/4
3pIkU3SevEA7CZXJfA44bUbBYcrf93xphg2uR5HCFM+Eh6qqnybpIKl3g0kGA4rv
tcMIJ9/PP8npdpVE+U4Hzf4IcgOaOmJiEWZ4smH7LWudMlOekqFTs2dWKbqzlC59
NeMPfu9avxxQ15fQzljhvcz9GhLqb373XDcn298ueA80KK6Pek+3qJ8YSjZQMrFT
+EJehFdQ6yt6vALcFc4CB1B6qVCGO7hICngCjdYpeZRNbGM/r6ED5Nsozof1oMbt
Si8mZEJ/Vlx3gathkUVtlxx/+jlScjdM7AFV5fkRidt0LkwosDoPoRz/sDFz0qTM
5q5TAgMBAAE=
-----END PUBLIC KEY-----

很明显还是一个base一个公钥解密



而且这个N也是可以分解的



分解网站我上面有就不设置超链接了

然后就是常规的求私钥

这边因为flag.enc是RSA的PKCS1_OAEP加密得来的。所以我们这边也是给生成一个私钥文件。

在做到这边的时候，如何导出一个私钥文件。找了一下百度上的方法。都是先generate后给参数分别赋值的。但是我发现我并不行，试了一下python3和python2下的Crypto库都得到一个报错

Exception has occurred: AttributeError

can't set attribute

也就是现在无法通过这么直接赋值了。

这种情况的话，可以去看下python调用的Crypto库里面的RSA模块的一个底层的实现。

发现有一个construct函数，传入一个rsa_components参数，是一个元组型的数据，也就是tuple类型的，分别是(n,e,d,p,q)

```
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
rsa_components=(n,e,int(d),p,q)
arsa=RSA.construct(rsa_components)
arsa.exportKey()
```

然后导出的私钥，对加密后的密文，使用PKCS1_OAEP模块进行解密即可。

脚本如下

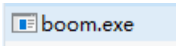
```
`from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
f=open("public.key","r")
key=RSA.import_key(f.read())
f.close()
e=key.e
n=key.n
import base64
from Crypto.Util.number import *
import gmpy2
p= 3133337
q=254783260649374192922001721363994977190818429145282283164559062116931183219713999360047291348411629741442462714864396957860365881174246118819559509962196468073788222782856382615820991083394389495730341012151411561564087428438200480668308638143623798857203950823184628500029016056897618763191511473527300909575569408421442998873946787436077669378280944783364011594490358783068537162165483742734623865083073677131120730040113834189678949305540675824532489810220119228833744427368480459206763413618712317871634414675330768900817218821793691687872877247696426653999925560521448458786001262839688902730675753420617762449 39
print(p*q==n)
f=open("flag.enc","r")
c_base64=f.read().strip("\n")
c_bytes=base64.b64decode(c_base64)
c=bytes_to_long(c_bytes)
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
rsa_components=(n,e,int(d),p,q)
arsa=RSA.construct(rsa_components)
rsakey = RSA.importKey(arsa.exportKey())
rsakey = PKCS1_OAEP.new(rsakey)
decrypted = rsakey.decrypt(c_bytes)
print(decrypted)`
```

运行得到

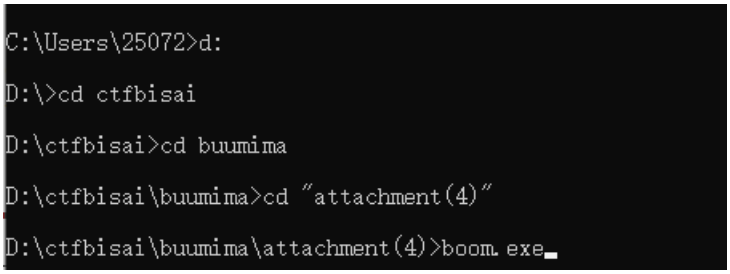
```
b'afctf{R54_|5_$0_B0rin9}'
```

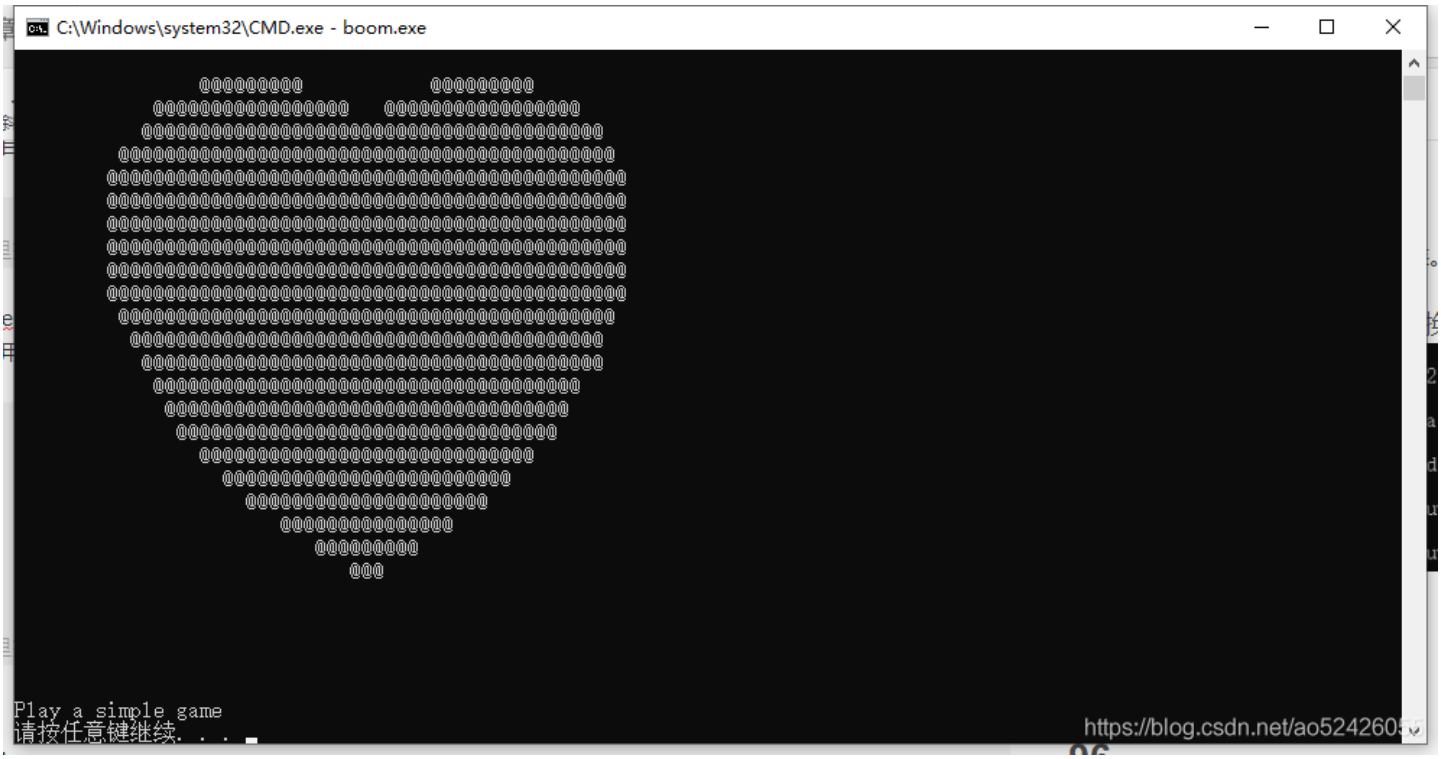
93.[网鼎杯 2020 青龙组]boom

查看题目



一个exe，先说一下我翻车的时候。。我一开始是直接打开的这个exe但是你写到最后一步，崩就没了
我这边用min+r打开cmd船靠在切换到exe的






```
first:this string md5:46e5efe6165a5afb361217446a2dbd01
```

md5在线解密

```
en5oy
```

一个三元一次方程

```
This time:Here are have some formulas
3x-y+z=185
2x+3y-z=321
x+y+z=173
input: x =
```

脚本：

```
from sympy import *
x = Symbol('x')
y = Symbol('y')
z = Symbol('z')
print(solve([3*x-y+z-185,2*x+3*y-z-321,x+y+z-173],[x, y,z]))
```

运行得到

```
{x: 74, y: 68, z: 31}
```

一元二次方程

```
Last time: Kill it
x*x+x-7943722218936282=0
input x:
```

脚本

```
import math
print("ax*x+bx+c=0")
a = float(input("input a: "))
b = float(input("input b: "))
c = float(input("input c: "))
p = b*b-4*a*c
if p < 0:
    print("None")
    exit()
else:
    X1 = (-b+math.sqrt(p))/(2*a)
    X2 = (-b-math.sqrt(p))/(2*a)
print("X1: " + str(X1))
print("X2: " + str(X2))
```

运行得到

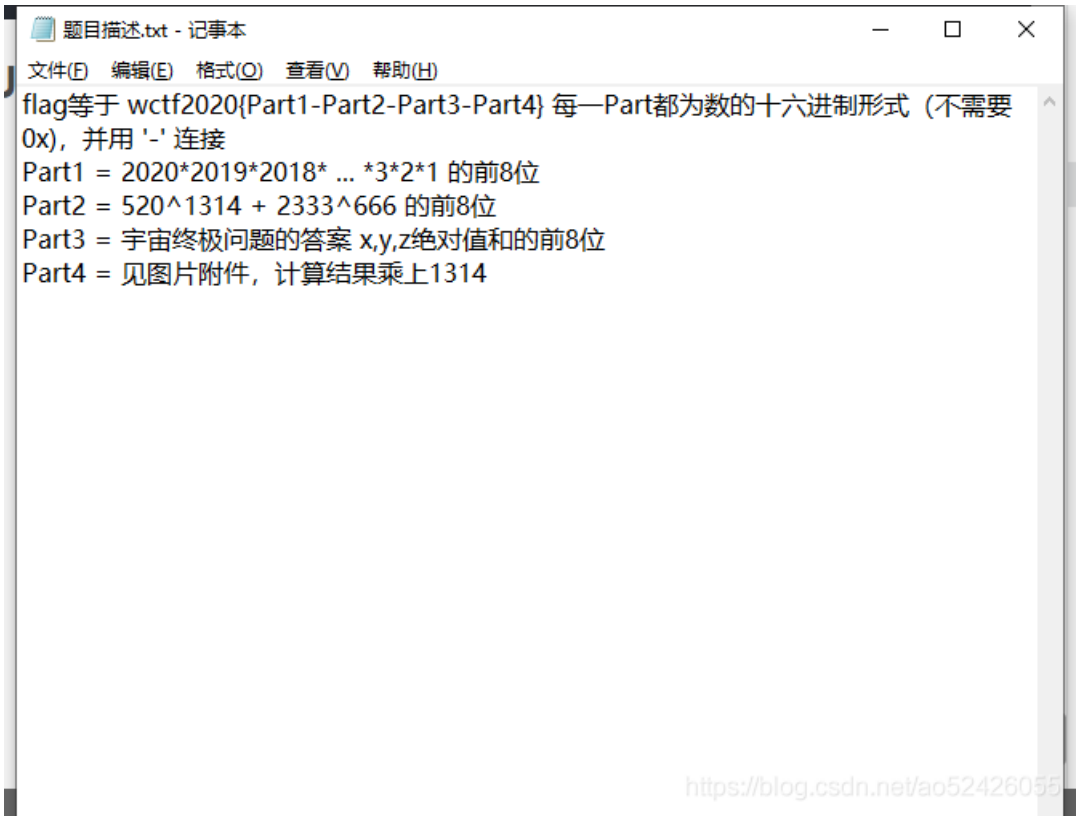
```
ax*x+bx+c=0
input a: 1
input b: 1
input c: -7943722218936282
X1: 89127561.0
X2: -89127562.0
```

在cmd框里最后得到

```
Last time: Kill it
x*x+x-7943722218936282=0
input x: 89127561
Great This is your FLAG
flag{en5oy_746831_89127561}
D:\ctfbisai\buumima\attachment(4)>
```

94.[WUSTCTF2020]大数计算

查看题目



$$\int_0^{22} 2x\,dx + 36$$

我都不知道这个wp该怎么写。。。
脚本运行吧

```
#Part1 = 2020*2019*2018* ... *3*2*1 的前8位
p1=1
for i in range(1,2021):
 p1*=i
p1=str(p1)[0:8]
#print(p1)
part1=hex(int(p1))
print(part1)
#Part2 = 520^1314 + 2333^666 的前8位
p2=520**1314 + 2333**666
p2=str(p2)[0:8]
#print(p2)
part2=hex(int(p2))
print(part2)
#Part3 = 宇宙终极问题的答案 x,y,z绝对值和的前8位
p3=80538738812075974 + 80435758145817515 + 12602123297335631
p3=str(p3)[0:8]
#print(p3)
part3=hex(int(p3))
print(part3)
#Part4 = 见图片附件，计算结果乘上1314
p4=(22**2+36)*1314
p4=str(p4)[0:8]
#print(p4)
part4=hex(int(p4))
print(part4)
print('wctf2020{'+str(part1)+'-'+str(part2)+'-'+str(part3)+'-'+str(part4)+'}')
```

运行得到

```
0x24d231f
0x403cfd3
0x108db5e
0xa6d10
wctf2020{0x24d231f-0x403cfd3-0x108db5e-0xa6d10}
```

去掉0x提交

95.RSA & what

查看题目

HUB1

```
78509541971826828686650821430481698544707729376681939872804641116691781082048475931429102897649822366122939500947406317370516262703761099353961775190544303927822758350460480825193108381890946761327758787454576107436442754996655551937191385987531357728224305315005627466779804969469570366031353293316544931294972558170896541727305558221629599458760097597012481149627008089697707694600010270103026099059818146644720805471339152631370068134109392224031742817359903162412515518821648947682560619152118203496912034328769118130039968351541480926270045752587669180818025773035170767366038069897388464230689881000063368487871540282314354913985073298289745969808964956119074685069813029945808025558231269687314921002824089813782288849255995766506793657335636758978459311901662407243387274453743200591166849445573333068938514121465309188801778204904343486262030678343616985656417592987110066991343898089921957932989775323345093477019391543479142772863658621804987461723170530800372006626931272913576417569861106880840405412558154011495646360324022249791938469171874401400255420160239596931299999415959953602635987906021805649634574545749391977133760117744989906657985763003635087109045264983077502969548857557498507842856005425318086372536414716971697412629526163150748619328091306742267675740578011800062477174189782151273970783531227579758540364970485350157944321579108232221072397135934034064481497887079641131808838242743811511451355024436983050572020925065644355566436462
```

5618133203024215941534926113892937988520918939061441606915556516246057349589921494351383160036280826024605351878
4080561809077599738041172630025549230417505875488197463468139666730341829133255078262199619239321005263052898949
6521660825425218839858013954518968187582408945619504498458582493838452190533428990642245415297683486730469329246
6676355760173232407753256256317546190171995276258924613533179898467683358934751999655196790168438343198229183747
0911082629887776598586097447093245718502622932949753366282347672588588738393425968871937726150006764015224315183
1064830397559358296502118918224698695734925315673652607163997384403906899640429054847464066885185607820109333542
5412842295604919065487301340901573809617549185106072798799159726375235125260509158832996701927878713084753334549
1295809124121685941706596054217502048359702319095910634076127793374780651759883654015903962475767093437271961060
5847716694567011786898902590302399814285033895698581613180534954905937704747713127084757909562838456964563682165
0
4946443479437105452246788319415890865727007924654595587707822135500697094585683496869866054181016687203404158476
6748715014011115178822146002789719324827346160741102781598488396939622062635862504178155827780493021265429670405
5890683796941327712758797770820006623289146990000114915293539639766846910274034245607746230740851938158390562286
0570022231776096063763290076768454501425379307981482584287014664154832326706598157910646813844064943882377423307
8622555730398802546803682008295971205073309586054686046857585708461606913205109488291925374523476202975912477634
8047587755897123575123506976140900565238840752841856713613368250071926171873213897914794115466890719123299469964
0194508992914107607621798369465709455552952881846981845550183686877084326122862484760737580671754817711990665815
7287017546001601710041447934643703429178483713224089132193160149441490892771320844892722109574580238001444184113
9882391378410438764884597938773868771896252329517440068673532468372840830510218585255432000690265226016573313570
9779450838792149613940870655583761588269382576648405709522338328528693287855681754345162477203565202426022995103
7431748818273873270007887966574590960376648210013800141702368064771782432314338885781759576617215288348427471824
8
1529422835997283071681441373701272126726118940720387321260410981026288310530009867592602712106719220705559480236
8859657541582298402615901057440435947467042867851826217503388051398437290974899272782838169441677674098102173054
5374002974037896534944567124543272737618380646771071804878796585983783360553761828325817820260204820004421979881
8710272555626909523349006166756065249335574402636482335147572002635214995083739750034313068474530467140276871083
9694571980344444495407930840494712621639552655129210472204787817837320788603307185727785799793225525131598283789
2164421298202073945919187779856785892717251746704537315003771369737854896595170485152591013676942418134278534037
6544678406335289168122752672301553520777365831309925876709416546953822870239712615299873845208438296957780293117
8643122740918901920581835191157275714555699360664346433619680235020461605628649724601680010500314304612060867349
6196758720552776772796609670537056331996894322779267635281472481559819839042424017171718303214059720568484939239
3701440381615413542541827699797719487594131029339877734016445069302051648917738265131617837363866047834844463457
4495711946979923179636832492757069449667945331392756234565669024041462443130464624859922604652470236413109596433
5
7971798893624795126548915758369795603189347785885418699105152916187947848828174406231860047090612096000228288651
1477294555606503083169449335174864424180701080203993329996226566203834693869525797695969610065991941396723959032
6800190825068164430415983004776257934330806643464705864163858546921244263485872110265686676948058495547807940337
6471401652171146755728484673723637499012131680983381999682159283263902402641152040733020628126539013076394816569
4574512140518775603040182029818771866749548761938870605590174330887949847420877829240131490902432602005681085180
8072941768376460625680948757669458903829717900154901633850881446735490850796350832629751542062696791424128974382
3171970493325866077931073730268026544543777197774995911074495936858629308201606792754856496740084599238007610752
2755566531760628823374519718763740378295585535591752887339222947397184116326706799921515431185636740825707782742
3737834757810526742572929102138439861329874668100272750524167746933634461845189018992025028286703094526223475329
3267887499080993068257573865387628938415149680719414630861436882100666062687098978469704516023106942845896110775
1207771093777394616856305293335603892178327520756554333365975114235981173451368131680404850832773147333013716920
1231113536504011585566399834598706630572978719929270538869712247735296365251106281837157487959875251131775400928
1411992870827229037033653711038102313463775974071614096966218326937067663032558338528499494316469239745910319543
4968057377474610500216801375394703781249039351368816958227409657934091741509357152328382960684515093945552479461
3822819139619567451542606860299978275650757687037748957505615751551436062971163916663857058991380856939132463137
7803362721031226895973739455351089472009916519398133377590753110723255690947815644145789979751569434881696176279
6703443502856101079430585547997496001098926600499728389113862894833789669213630332988693669889340482430613291490
6138032044847514706766860410027725561172136121523226067371508581161229365391317951112635131145697945328058866430
8729991819663511303777713866691429698604054927455983521450530061825610550876402646151887657938715988198354466725
8537064954616097750399839661065797883103731694314852301848272092388637114950059216922969842082648527035538090054
0938903656476761197489952434163378056665575013452340569684761426084918304380654012197516886873737093900575219109
4273663212672971160625615839996368299088147317821606082702137377659890128195852765554331841366427792149272318598
4
3686980681593604691184819581740581735025989087148306318437372839796890945843262504602537629021472991403838753473
1762237978339011724858818860181178811639468996206294711495853807311240013786226884265118119546377272154555615363
1052361928782927033314735476230217443170348194166245628962261945236397935730280066662362718123907590362358674958

```
0325590584363644725222541387103876265780134564758449391757626347158734720266439190857014038912690320460239109399
0827188675090199750617303773574821926387194478875191828814971296674530519321530805302667925998711835019806761133
0784032814048893746638750773391689012978194364999209582684836843359983010560683802288735248003839114024908071392
6896409516506961045467755880875644438154217378281522792090622493102845707365245377742438787353328045594464659299
6920617956675786286711447540353883400282402551158169958389450168079568459656526911857835375748015814860506707921
8529970961562758049559899642150777336217699380754130078042232170916046131322530463994567475953004045641722243339
3640554592181965443543707213338752353356847244353220006913302297919568568350829733796170116939479496625641511224
6587706103819620428258245999539040721929317130088874161577093962579487428358736401687123174207198251449851429295
```

HUB2

```
7850954197182682868665082143048169854470772937668193987280464111669178108204847593142910289764982236612293950094
7406317370516262703761099353961775190544303927822758350460480825193108381890946761327758787454576107436442754996
6555519371913859875313577282243053150056274667798049694695703660313532933165449312949725581708965417273055582216
2959945876009759701248114962700808969770769460001027010302609905981814664472080547133915263137006813410939222403
1742817359903162412515518821648947682560619152118203496912034328769118130039968351541480926270045752587669180818
0257730351707673660380698973884642306898810000633684878715402823143549139850732982897459698089649561190746850698
1302994580802555823126968731492100282408981378228884925599576650679365733563675897845931190166240724338727445374
3200591166849445573333068938514121465309188801778204904343486262030678343616985656417592987110066991343898089921
9579329897753233450934770193915434791427728636586218049874617231705308003720066269312729135764175698611068808404
0541255815401149564636032402224979193846917187440140025542016023959693129999941595995360263598790602180564963457
4545749391977133760117744989906657985763003635087109045264983077502969548857557498507842856005425318086372536414
7
599

5921690793720937273061002160113958578256463239342894809760736290375439229020981209011384544621771599963766541762
4823897913252872832759030109896613998315798061232056349654612864496773100071669770510407903915627671487214746335
0811303393260622707024952543509891692246246277965823414460326811240048060543656588688604452353899779068825120910
2821670047153397631877347971803269761322133250546971653204791663565625180298059277416566051748097263975657722715
6206607807610549174590398659787740037020671895497528872107204833367860905500813580908930422901536434849092497409
7403734627265297637171818849461766523691595241613878709865506436588268999163342945070495338153600520537498539457
3965828046929592966127157525731402961357849332061460914366179795997497743306999466375914063562894097160840344510
4909471520219620348608836879174410762927164732027325983691531279429724658950100866629916571772250770286603345421
5783240025504356157664454861755286285777763585177517962526550082063830247078830775137458633120793497902750940807
0750239286694632579691445060226446258872205229743082768175082734909432396833767031127293378583885064937611566722
3821665435911506351891489985627506615492005617098615432522564204152887767244129985681083657783356557756654335186
3739406464168327408787332557075677530337165834484020007892027675119202103828303439555536541114867283339805573197
9936251496062787901679749138981200776883273097991623064764187275900190684674797763167570431017944885712816038570
1185892914523053669366534408863734305635222625590986006420486092550427301086984563126480814987024980594613542978
3101292476788266914183353005775775279516236964264354978352281670847380077509142702510019213295214790476628486508
0898999608560019730936141086323852680212787752376726292151515098499856013664715486579116331650307328522396621644
1025637452229043510097323724381056976302288136843260163922706692913035222445496716008888946581535004546355744211
6803907312573099419025873033531399511022448652702954144744887983354046304584897066398051865738748145867367462323
5884967747753367196834415424296328941556948757989591066099904357873746130040693782892481800265829276988218166878
4501439254131996848948120781562158861495883827848139425862249576454689133681009549361314460818658995959098228995
7022022686496353631055499759323953350765211376042885200820401212866149229865546527000561489665141789359523630369
6321761987989967138360463841656795042135054620443490211315672000628272088959128885027107607494192771567830605717
6
5276309264606229365713856498417582144534168490394124010874434443171018570909047114855381070588230560858405390733
4592079287136823235547539457109838059683546850999734050560433373054779956099882298974747378030777971771552278772
4471724766494090783971030594671013168209717686720448579582618378459567979027822271918653169622428153856198907810
0402243402703624134324950296721232613754009271598315377607099747787081605832526137843582348585831745447779792428
8793882757360483776680199838137999907641644468389107809388968605548270983866835612091604035212301901925508451376
9603803814947774554028717814638951416291274696771515474086351482107953150253616922787262398450376249126999644026
3824784130809739331730791113051427161333891113992355452792590174247226018486700615568591639438954665539279464125
2375016658273400573337832846825056894494591223849587792971710172231467812017222849378796490407258390572107476671
1732215815561012960394537195757832959268603775112932862105945720853959285187521763557915356428113876893276879775
6032177189818521145997066995245519739342420457431227441463615969712450340593459153154952321354834644961147703575
```

```
365762005114909224132081781498693478029887865134514864114098871645160650620849175561207124650742064358314981136005
878643717869894032287788980700995761677735184497986972696235655324405091128398428096066576164931089523045507297743141510205398773596932655397899485316248305154465687329455511600999559204318307020870625816484054059957707209710413950585751766327392985120262885418535618564719493380008423050341303785889330771303714930747783053675828368109351761782016918142079610533868158223078831810842813205179376101495283733045626227282862735570146474057819796633261312730703725564728682349635591764235332791244001962183887038809182474862963742575912521463988513016318375237890872977351705325921252549455588092105267951258205151660429709820436352508103938235848392672700867932771908313886596929191186363038209716023096073804357555933026401821277442452771915324856387676006793149902938422899325386250193933775851437747201193327927318114483038116984938789379939075505209306917960557948571034365557002859259588243663242652765445289543175871512658016490241028642263721509847631604236791677943105226754576949599472372112994361629487964230554589491291463298045503175587908740157531069976540847360616672713793422451599841662512221305620880009507793310315069927265011615167470243846306273447271400492610366837850680400274004554796471669353634944766085058020531496220451150035285837225413253316754996082549894961851484157070319926486743158075467427599055447814063704142784211139174688325744712003594762145686389093406204401079544305928173634697617577241503483833468272663526343265553785294217733488802528374861157617153425146184734956650562829058722415086964038643762337124974316526039667522068330214280564636890693057514062861000391913199929585550121511139329481821879998270328930459698907047500008117551008543229026450202373689910474631683074222694639502702982082579183187085738264722132273460502621007309391833124749430755560033555094234052653628137203661213871388109886630316942550199897840000882987308096559200937117620866829007428890368141793365747227967068859786283562750634016997845091878853927034634038592884029957388929218953173808216640873404638142351646769432897138542190731481428348932261938657004618355657238398077727717334920933068342434365817978101507225937857613044222298496307116620764258558982206159728246785086805073795772642371376169423187949703717562754642744973063821621482846300348340892837562031519329087130031693013926052138253327976766383927867937504094194932807533684515088026582722207676247663906392853084336072552532827023837621497559355189220755846375124948195
271453634732502613378948161256470991260052778799128789839624515809143527363206813219580098196957510291648493698144497567392065251244844074992734669490296293997386198359280316655904691639367482203210051809125904410431506925233748438563432432765082806410596909389309574744345183086466189590042168311300998735327143724021177966665606776248225091592876754324130164789485946408720916884821490044263639460485174800529063062901262428660342494780404063519400882310814561091957994429967996416471675526895646133464152479068520555884983056659284508287561521030966292747606015287376394153614679413499822136414549679627238750326382673119350423345849138973385539539618774393895887930742115025972384655428893353635590523681802120132061727125612213528338916406590202535275847064652054864089907627592308421920283810485634377245284091747900227525575127957827131251661583298807027307699571854285220114301448402322564191136316793431716806316307752664887381737073571231393688250870437858421690499432375371881293672757309847894799091033979371138378245751370210123334615521766875700104457442683738407428992999773728340419251028537189648312252504072795784650085375426596736856862427733791319048908651106991904515344454345339191276589768747210295861681062067
```

README.txt

```
素数生成算法太麻烦了，有没有取巧的方法呢？
诶，这里好像有个不错的想法哟。
看起来节约了不少时间呢，嘿嘿嘿……
顺便问问，应该大家都知道base64吧，用来编码还是很方便的呢！
```

rsa.py

```python
from Crypto.Util.number import bytes_to_long, getPrime
from random import randint
from gmpy2 import powmod

p = getPrime(2048)
q = getPrime(2048)
N = p*q
Phi = (p-1)*(q-1)
def get_enc_key(N,Phi):
    e = getPrime(N)
    if Phi % e == 0:
        return get_enc_key(N, Phi)
    else:
        return e
e1 = get_enc_key(randint(10, 12), Phi)
e2 = get_enc_key(randint(10, 12), Phi)

fr = open(r"./base64", "rb")#flag is in this file
f1 = open(r"./HUB1", "wb")
f2 = open(r"./HUB2", "wb")
base64 = fr.read(255)
f1.write("%d\n%d\n" % (N, e1))
f2.write("%d\n%d\n" % (N, e2))
while len(base64)>0:
    pt = bytes_to_long(base64)
    ct1 = powmod(pt, e1, N)
    ct2 = powmod(pt, e2, N)
    f1.write("\n%d" % ct1)
    f2.write("\n%d" % ct2)
    base64 = fr.read(255)
fr.close()
f1.close()
f2.close()
```

N一样 很容易想到共模攻击

```python
from Crypto.Util.number import*
import base64

def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)

def CMA(n,e1,e2,c1,c2):
    s = egcd(e1, e2)
    s1 = s[1]
    s2 = s[2]
    if s1<0:
        s1 = - s1
        c1 = inverse(c1, n)
    elif s2<0:
        s2 = - s2
        c2 = inverse(c2, n)
    m = pow(c1,s1,n)*pow(c2,s2,n) % n
    return m

f1=open("HUB1")
f2=open("HUB2")
N=f1.readline()
N=f2.readline()
e1,e2=f1.readline(),f2.readline()
f1.readline()
f2.readline()
c1,c2=f1.readline(),f2.readline()
ans=b''
cnt=0
while len(c1)!=0:
    cnt+=1
    ans+=long_to_bytes(CMA(int(N),int(e1),int(e2),int(c1),int(c2)))
    #print(base64.b64decode(temp))
    c1,c2=f1.readline(),f2.readline()
temp=b''
M=b''
print(ans)
for i in ans:
    k=long_to_bytes(i)
    #print(i," ",end="")
    if k==b'\n':
        M+=base64.b64decode(temp)
        temp=b''
        continue
    temp+=k
print(M)
```

运行得到

```
b'VEhJUz==\nRkxBR3==\nSVN=\nSElEREVOLo==\nQ0FO\nWU9V\nRklORM==\nSVT=\nT1VUP4==\nRE8=\nWU9V\nS05PV9==\nQkFTRTY0P5
==\nWW91bmdD\nVEhJTku=\nWU9V\nQVJF\nTk9U\nVEhBVE==\nRkFNSUxJQVI=\nV0lUSO==\nQkFTRTY0Lh==\nQmFzZTY0\naXO=\nYW==\n
Z3JvdXA=\nb2b=\nc2ltaWxhcn==\nYmluYXJ5LXRvLXRleHR=\nZW5jb2Rpbme=\nc2NoZW1lc0==\ndGhhdD==\ncmVwcmVzZW50\nYmluYXJ5
\nZGF0YW==\naW5=\nYW6=\nQVNDSUl=\nc3RyaW5n\nZm9ybWF0\nYnk=\ndHJhbnNYXRpbmd=\naXS=\naW50b1==\nYT==\ncmFkaXgtNjQ=
\ncmVwcmVzZW50YXRpb24u\nVGhl\ndGVybc==\nQmFzZTY0\nb3JpZ2luYXRlc8==\nZnJvbd==\nYY==\nc3BlY2lmaWN=\nTUlNRT==\nY29u
dGVudI==\ndHJhbnNmZXI=\nZW5jb2Rpbmcu\nVGhl\ncGFydGljdWxhct==\nc2V0\nb2b=\nNjR=\nY2hhcmFjdGVyc5==\nY2hvc2Vu\ndG+=
\ncmVwcmVzZW50\ndGhl\nNjQ=\ncGxhY2UtdmFsdWVz\nZm9y\ndGhl\nYmFzZd==\ndmFyaWVz\nYmV0d2Vlbt==\naW1wbGVtZW50YXRpb25z
Lp==\nVGhl\nZ2VuZXJhbI==\nc3RyYXRlZ3n=\naXO=\ndG9=\nY2hvb3Nl\nNjR=\nY2hhcmFjdGVyc5==\ndGhhdA==\nYXJl\nYm90aN==\n
bWVtYmVyc5==\nb2a=\nYS==\nc3Vic2V0\nY29tbW9u\ndG8=\nbW9zdM==\nZW5jb2RpbmdzLA==\nYW5k\nYWxzb8==\ncHJpbnRhYmxlLg==
\nVGhpc9==\nY29tYmluYXRpb25=\nbGVhdmVz\ndGhl\nZGF0YW==\ndW5saWtlbHk=\ndG/=\nYmV=\nbW9kaWZpZWS=\naW5=\ndHJhbnNpdE
==\ndGhyb3VnaN==\naW5mb3JtYXRpb26=\nc3lzdGVtcyw=\nc3VjaN==\nYXM=\nRS1tYWlsLD==\ndGhhdA==\nd2VyZQ==\ndHJhZGl0aW9u
YWxseQ==\nbm90\nOC1iaXQ=\nY2xlYW4uWzFd\nRm9y\nZXhhbXBsZSw=\nTUlNRSdz\nQmFzZTY0\naW1wbGVtZW50YXRpb24=\ndXNlcw==\n
QahDWiw=\nYahDeiw=\nYW5k\nMKhDOQ==\nZm9y\ndGhl\nZmlyc3Q=\nNjI=\ndmFsdWVzLg==\nT3RoZXI=\ndmFyaWF0aW9ucw==\nc2hhcm
U=\ndGhpcw==\ncHJvcGVydHk=\nYnV0\nZGlmZmVy\naW4=\ndGhl\nc3ltYm9scw==\nY2hvc2Vu\nZm9y\ndGhl\nbGFzdA==\ndHdv\ndmFs
dWVzOw==\nYW4=\nZXhhbXBsZQ==\naXM=\nVVRGLTcu'
b"THISFLAGISHIDDEN.CANYOUFINDITOUT?DOYOUKNOWBASE64?YoungCTHINKYOUARENOTTHATFAMILIARWITHBASE64.Base64isagroupofsi
milarbinary-to-textencodingschemesthatrepresentbinarydatainanASCIIstringformatbytranslatingitintoaradix-64repres
entation.ThetermBase64originatesfromaspecificMIMEcontenttransferencoding.Theparticularsetof64characterschosentor
epresentthe64place-valuesforthebasevariesbetweenimplementations.Thegeneralstrategyistochoose64charactersthatareb
othmembersofasubsetcommontomostencodings,andalsoprintable.Thiscombinationleavesthedataunlikelytobemodifiedintran
sitthroughinformationsystems,suchasE-mail,thatweretraditionallynot8-bitclean.[1]Forexample,MIME'sBase64implement
ationusesA\xa8CZ,a\xa8Cz,and0\xa8C9forthefirst62values.Othervariationssharethispropertybutdifferinthesymbolschos
enforthelasttwovalues;anexampleis"
```

再看第三个提示，就知道是base隐写
我白嫖了一个隐写脚本

```
from Crypto.Util.number import*
import base64
c = b'VEhJUz==\nRkxBR3==\nSVN=\nSElEREVOLo==\nQ0FO\nWU9V\nRklORM==\nSVT=\nT1VUP4==\nRE8=\nWU9V\nS05PV9==\nQkFTRTY0P5==\nWW91bmdD\nVEhJTku=\nWU9V\nQVJF\nTk9U\nVEhBVE==\nRkFNSUxJQVI=\nV0lUSO==\nQkFTRTY0Lh==\nQmFzZTY0\naXO=\nYW==\nZ3JvdXA=\nb2b=\nc2ltaWxhcn==\nYmluYXJ5LXRvLXRleHR=\nZW5jb2Rpbme=\nc2NoZW1lc0==\ndGhhdD==\ncmVwcmVzZW50\nYmluYXJ5\nZGF0YW==\naW5=\nYW6=\nQVNDSUl=\nc3RyaW5n\nZm9ybWF0\nYnk=\ndHJhbnNsYXRpbmd=\naXS=\naW50b1==\nYT==\ncmFkaXgtNjQ=\ncmVwcmVzZW50YXRpb24u\nVGhl\ndGVybc==\nQmFzZTY0\nb3JpZ2luYXRlc8==\nZnJvbd==\nYY==\nc3BlY2lmaWN=\nTUlNRT==\nY29udGVudI==\ndHJhbnNmZXI=\nZW5jb2Rpbmcu\nVGhl\ncGFydGljdWxhct==\nc2V0\nb2b=\nNjR=\nY2hhcmFjdGVyc5==\nY2hvc2Vu\ndG+=\ncmVwcmVzZW50\ndGhl\nNjQ=\ncGxhY2UtdmFsdWVz\nZm9y\ndGhl\nYmFzZd==\ndmFyaWVz\nYmV0d2Vlbt==\naW1wbGVtZW50YXRpb25zLp==\nVGhl\nZ2VuZXJhbI==\nc3RyYXRlZ3n=\naXO=\ndG9=\nY2hvb3Nl\nNjR=\nY2hhcmFjdGVyc5==\ndGhhdA==\nYXJl\nYm90aW==\nbWVtYmVyc5==\nb2a=\nYS==\nc3Vic2V0\nY29tbW9u\ndG8=\nbW9zdM==\nZW5jb2RpbmdzLA==\nYW5k\nYWxzb8==\ncHJpbnRhYmxlLg==\nVGhpc9==\nY29tYmluYXRpb25=\nbGVhdmVz\ndGhl\nZGF0YW==\ndW5saWtlbHk=\ndG/=\nYmV=\nbW9kaWZpZWS=\naW5=\ndHJhbnNpdE==\ndGhyb3VnaN==\naW5mb3JtYXRpb26=\nc3lzdGVtcyw=\nc3VjaN==\nYXM=\nRS1tYWlsLD==\ndGhhdA==\nd2VyZQ==\ndHJhZGl0aW9uYWxseQ==\nbm90\nOC1iaXQ=\nY2xlYW4uWzFd\nRm9y\nZXhhbXBsZSw=\nTUlNRSdz\nQmFzZTY0\naW1wbGVtZW50YXRpb24=\ndXNlcw==\nQahDWiw=\nYahDeiw=\nYW5k\nMKhDOQ==\nZm9y\ndGhl\nZmlyc3Q=\nNjI=\ndmFsdWVzLg==\nT3RoZXI=\ndmFyaWF0aW9ucw==\nc2hhcmU=\ndGhpcw==\ncHJvcGVydHk=\nYnV0\nZGlmZmVy\naW4=\ndGhl\nc3ltYm9scw==\nY2hvc2Vu\nZm9y\ndGhl\nbGFzdA==\ndHdv\ndmFsdWVzOw==\nYW4=\nZXhhbXBsZQ==\naXM=\nVVRGLTcu'

def get_base64_diff_value(s1, s2):
    base64chars = b'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
    res = 0
    for i in range(len(s2)):
        if s1[i] != s2[i]:
            return abs(base64chars.index(s1[i]) - base64chars.index(s2[i]))
    return res

def solve_stego():
    line=b''
    bin_str=''
    for i in c:
        k=long_to_bytes(i)
        if k==b'\n':
            steg_line = line
            norm_line = base64.b64encode(base64.b64decode(line))
            diff = get_base64_diff_value(steg_line, norm_line)
            #print(diff)
            pads_num = steg_line.count(b'=')
            if diff:
                bin_str += bin(diff)[2:].zfill(pads_num * 2)
            else:
                bin_str += '0' * pads_num * 2
            print(goflag(bin_str))
            line=b''
            continue
        line+=k

def goflag(bin_str):
    res_str = ''
    for i in range(0, len(bin_str), 8):
        res_str += chr(int(bin_str[i:i + 8], 2))
    return res_str


if __name__ == '__main__':
    solve_stego()
```

运行得到

```
7c86d8f7d6de33a87f7f9d6b005ce640
```

flag包裹提交即可 注意（共模攻击py要和H1，2放在一个文件夹）

96.[MRCTF2020]babyRSA

查看题目一个加密脚本

```
'''
import sympy
import random
from gmpy2 import gcd, invert
from Crypto.Util.number import getPrime, isPrime, getRandomNBitInteger, bytes_to_long, long_to_bytes
from z3 import *
flag = b"MRCTF{xxxx}"
base = 65537


def GCD(A):
    B = 1
    for i in range(1, len(A)):
        B = gcd(A[i-1], A[i])
    return B


def gen_p():
    P = [0 for i in range(17)]
    P[0] = getPrime(128)
    for i in range(1, 17):
        P[i] = sympy.nextprime(P[i-1])
    print("P_p :", P[9])
    n = 1
    for i in range(17):
        n *= P[i]
    p = getPrime(1024)
    factor = pow(p, base, n)
    print("P_factor :", factor)
    return sympy.nextprime(p)


def gen_q():
    sub_Q = getPrime(1024)
    Q_1 = getPrime(1024)
    Q_2 = getPrime(1024)
    Q = sub_Q ** Q_2 % Q_1
    print("Q_1: ", Q_1)
    print("Q_2: ", Q_2)
    print("sub_Q: ", sub_Q)
    return sympy.nextprime(Q)


if __name__ == "__main__":
    _E = base
    _P = gen_p()
    _Q = gen_q()
    assert (gcd(_E, (_P - 1) * (_Q - 1)) == 1)
    _M = bytes_to_long(flag)
    _C = pow(_M, _E, _P * _Q)
    print("Ciphertext = ", _C)
'''
```

拿到加密脚本以后,发现p,q的生成方式不同,其中,q的生成方式比较简单.三数相乘以后取结果的下一个素数.p的求解则需要先求出n的欧拉函数值,然后求出base对应的逆元然后求解p,p求出来以后就可以解密密文了.

脚本如下

```
P_p=20602792684730861271967757255499114342 1
P_factor=213671742765908980787116579976289600595864704574134469173111790965233629909513884704158446946409910475727584342641848597858942209151114627306286393390259700239698869487469080881267182803062488043469138252786381822646126962323295676431679988602406971858136496624861228526070581338082202663895710929460596143281673761666804565161435963957655012011051936180536581488499059517946308650135300428672486819645279969693519039407892941672784362868653243632727928279698588177694171797254644864554162848696210763681197279758130811723700154618280764123396312330032986093579531909363210692564988076206283296967165522152288770019720928264542910922693728918198338839
Q_1=10376643984946558808462504949579385763455651706456348843314822452463810597116105176312771843806286254818481474760129949405281366285145974012749955778539871448190946163199602004831579016796769993296797448448120987966417300958523146978514162898202184788394587120143015507125780316352361286311396749596957860552 1
Q_2=15101073427691693979059146127898148644254803503235079730649610513635872358695312348408786017643862984368846267168177751365294755532560741485851456605351324308362781068608489026112064116198761443511488756549186612050784456621056162050396120585140938604119432672843707399537232243303515351975017396063066469743
sub_Q=16899252979359331575789599510143024199495363833091931480013053680980182497111203957256238944958435064392439198480097819370779590995647299263100429047927352511695946185622726223260008917695081072947505826033217762696128600987663034094509362995930280318966890412389099106911382624149778366699575139136102894965 1
Ciphertext=17091872405163671414608621877494510476440948857917616735746743308408427921897950499683941222168544917579226476564309085870599970704886742203308478718118367245419076669830423762164115618266400607343070134587949250256840628045894398430272902820349996179151242318385245936070803773009851521798281995694742416786515597717633955966971402060725376881297901264720539873915382800070822030063480291257296502076613623719361967895626584587783125335059388589596445412335786543409259019639579800476391141700339365700602504389061305913779041821116222365675070227111764573014765434616005249930453007284328156720773998796682764718321 1
import gmpy2
import Crypto
import sympy
base = 65537
q=sympy.nextprime(gmpy2.powmod(sub_Q,Q_2,Q_1))
print(q)
P=[]
for i in range(9):
    P_p=sympy.prevprime(P_p)
P.append(P_p)
for i in range(1,17):
    P.append(sympy.nextprime(P[i-1]))
n=1
phi=1
for i in range(17):
    n*=P[i]
    phi*=(P[i]-1)
based=gmpy2.invert(base,phi)
_p=gmpy2.powmod(P_factor,based,n)
p=sympy.nextprime(_p)
print(p)
d=gmpy2.invert(base,(p-1)*(q-1))
print(d)
plaintext=gmpy2.powmod(Ciphertext,d,p*q)
import binascii
print(binascii.unhexlify(hex(plaintext)[2:]))
```

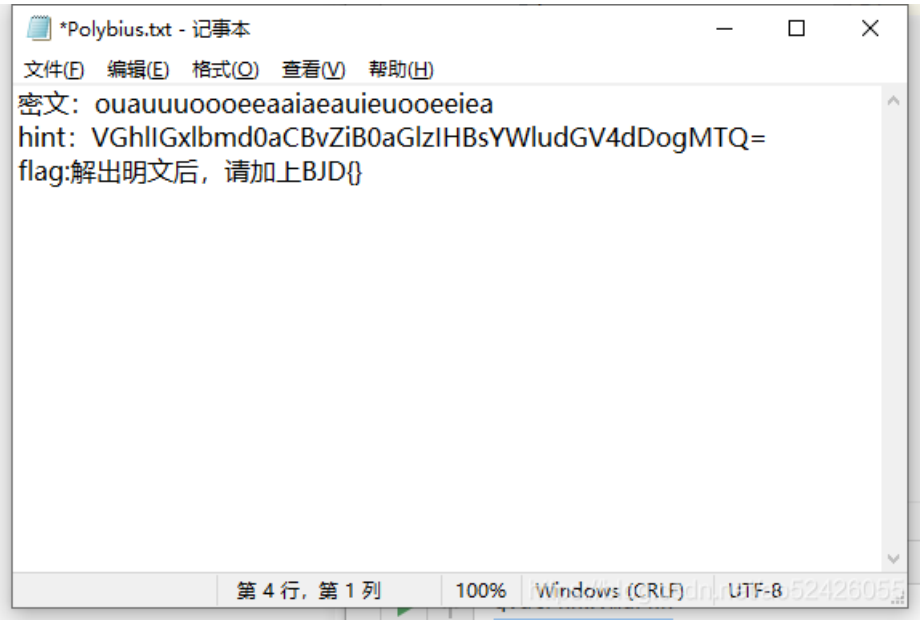
我喜欢输出pq

运行得到

```
95170653714081687088760585440906768700419459767774333757336842864507607081809193370870747769993218256925111100260761958233280546585624501259121060195932474781731613458132842656517609786144352755126076860272047457230913808406105832246663969943550533958139118721153456230616182820319799156494938586844573835221160735380264118564161835536782782924160005620631679929855445290207351945863258282088265202232862202180668844947205806261323713945818872852303248590355632665886900928520533421774721590935485773234619558181513033385642711706205607543347313747616062185115981201425568780146693758544521883683953378438266703113683351195574433742815103197994136129621630647393746927135052107013181699117871005570486273370360071247634235798286933692480392574891137618457458491875894195315046267512673111735111634700308391191031435323720919776532671492685243496013965439642965417782657024324810813460943009803523746219386004686158627347562391960350225884309848872664977762173337218436656877529787718596024111549639127697401644536052416977012986784355398576981328286309770831513614006593341425479503207045189256599702671665740566948401749025685499816072328652483318792816801473438354457054937294730403181276751936682799532579536740319581925345930379 3
b'MRCTF{sti11_@_b@by_qu3st10n}'
```

97.[BJDCTF2020]Polybius

查看题目



hint用base64解密出来The length of this plaintext: 14

,结合题目polybius 猜测这是波利比奥斯方阵密码.

但是a,e,o,i,u这五个字符的代表顺序却不知道,因此可能有54321种情况,在结合刚才所说的i,j同时占一个位置,所以情况数要再乘上2,将这些情况全部都打印出来,然后去找有真实语义的句子就可以了.

```
import itertools
s="aeoiu"
sumresult=[]
numsumresult=[]
ciper="ouauuuoooeeaaiaeauieuooeeiea"


for i in itertools.permutations(s,5):#找出所有全排列
    sumresult.append("".join(i))
for i in sumresult:
    temp=""
    for j in ciper:
        temp+=str(i.index(j)+1)
    numsumresult.append(temp)
for i in numsumresult:
    ans_=""
    for j in range(0, len(i),2):
        xx=(int(i[j])-1)*5+int(i[j+1])+96
        if xx>ord('i'):
            xx+=1
        ans_+=chr(xx)
    print(ans_)
```

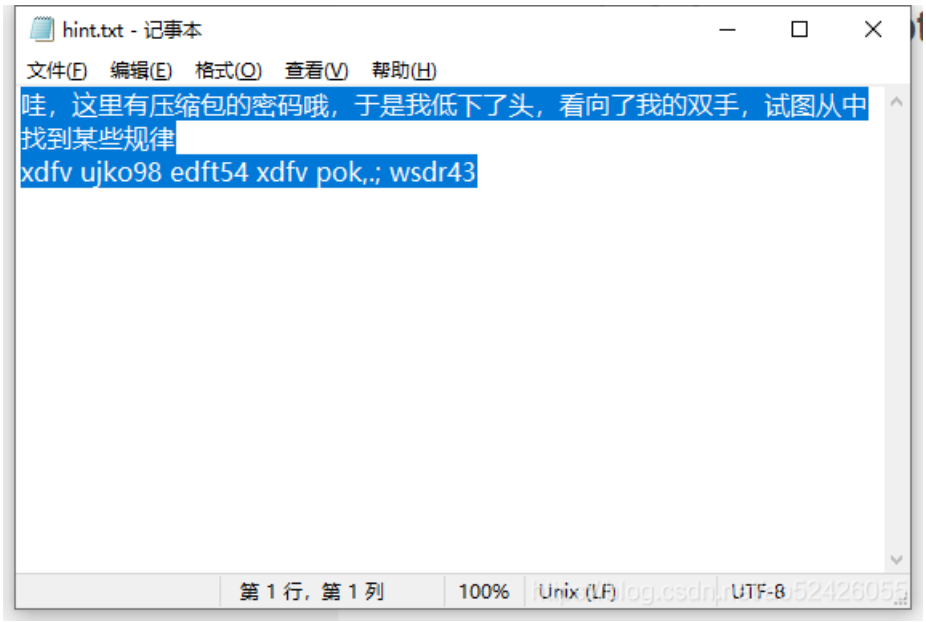
运行得到

```
flagispolybius
flagkxoplubkyx
fqaghousqxbhpo
fvaghpyxvsbhop
```

很多的结果里面找到flagispolybius

98.[ACTF新生赛2020]crypto-classic1

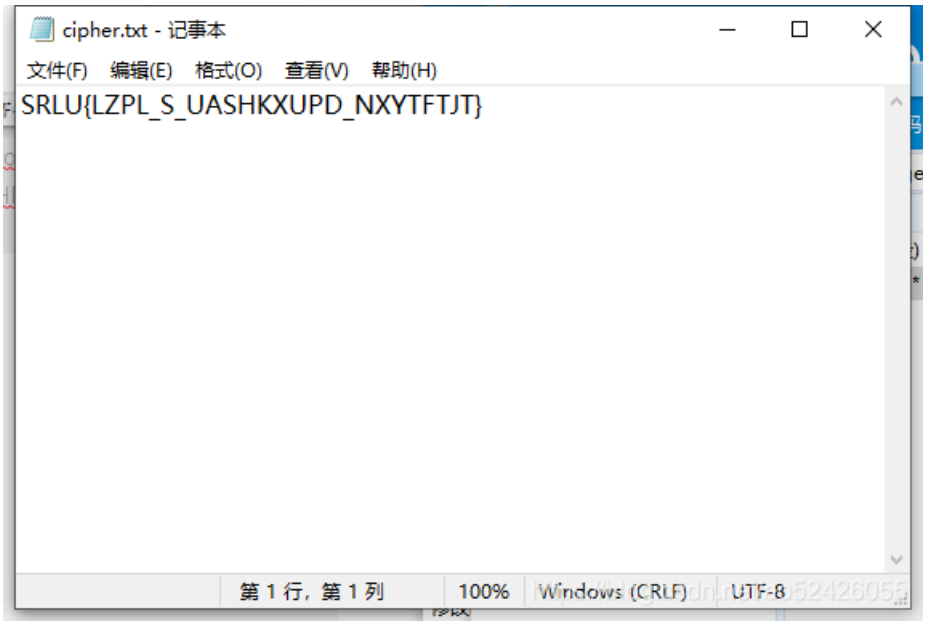
查看题目



低下头看，就是看键盘，比如xdfv包围的就是c

按照这个规律把键盘密码解出来就是circle

这就是压缩包密码，得到



脚本如下

```
s = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
s1 = 'ACTF'
s2 = 'SRLU'
key = ''
for i in range(len(s1)):
    key += s[(s.find(s2[i]) - s.find(s1[i])) % 26]
print(key)

# 解密
cipher = 'SRLU{LZPL_S_UASHKXUPD_NXYTFTJT}'
key = 'SPSP'
# decode
flag = ''
for i in range(0, len(cipher)):
    flag += s[(s.find(cipher[i]) + 26 - s.find(key[i % len(key)])) % 26]
print(flag)
```

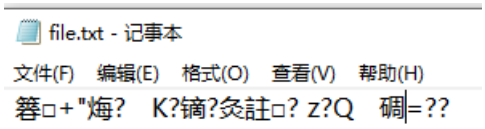
运行得到

```
SPSP
ACTFHZEDQKAKZOXVPLZDIKVIGENEREH
```

按照原题顺序把{_}这些换回来，ACTF{WHAT_A_CLASSICAL_VIGENERE}提交发现不对
然后就越走越远了，我看别的大佬的wp，还有说题出错的直接改题目的等等
我发现他们的我提交也不行啊
最后想起一个大小写转换
flag{what_a_classical_vigenere}提交这个成功了，顿时觉得自己好沙雕

99.EasyProgram

查看题目



附件.txt

```
get buf unsign s[256]
get buf t[256]
we have key:whoami
we have flag:????????????????????????????????

for i:0 to 256
    set s[i]:i

for i:0 to 256
    set t[i]:key[(i)mod(key.lenth)]

for i:0 to 256
    set j:(j+s[i]+t[i])mod(256)
        swap:s[i],s[j]

for m:0 to 38
    set i:(i + 1)mod(256)
    set j:(j + S[i])mod(256)
    swap:s[i],s[j]
    set x:(s[i] + (s[j]mod(256))mod(256))
    set flag[m]:flag[m]^s[x]

fprint flagx to file
```

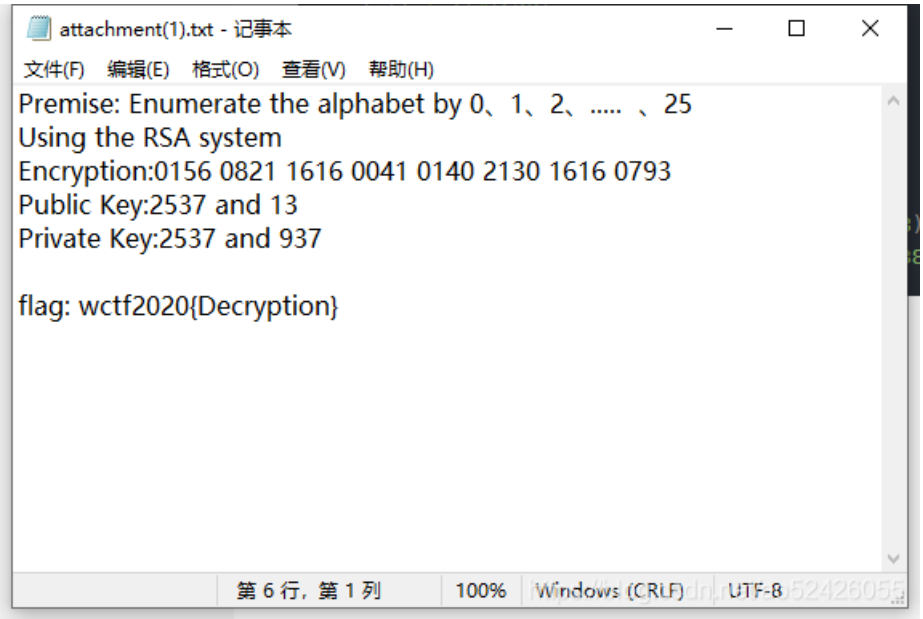
代码是一系列运算后，最后进行异或。解题思路就是，还原源代码，再进行一次异或就可以了

```
#读文件方法一：
def filehex(file):
    fhex=[]
    f = open(file,'rb')
    ff = f.read().hex()

    for i in range(0,len(str(ff)),2):
        fhex.append(int(str(ff)[i:i+2],16))
    return fhex

flagx2=filehex('file.txt')
print(flagx2)

#读文件方法二：使用010edit 读取'file.txt'文件16进制
flagx=[0x00,0xBA,0x8F,0x11,0x2B,0x22,0x9F,0x51,0xA1,0x2F,0xAB,0xB7,0x4B,0xD7,0x3F,0xEF,0xE1,0xB5,0x13,0xBE,0xC4,
0xD4,0x5D,0x03,0xD9,0x00,0x7A,0xCA,0x1D,0x51,0xA4,0x73,0xB5,0xEF,0x3D,0x9B,0x31,0xB3]

s=[]
t=[]
key='whoami'
j=0
for i in range(0,256):
    s.append(i)

for i in range(0,256):
    t.append(key[i % len(key)])

for i in range(0,256):
     j=(j+int(s[i])+int(ord(t[i])))%256
     s[i], s[j] = s[j], s[i]

i=0;j=0;x=0
for m in range(0,38):
    i=(i + 1)%(256)
    j=(j + s[i])%(256)
    s[i],s[j] = s[j],s[i]
    x=(s[i] + (s[j]%(256)))%(256)
    flagx[m] = flagx[m] ^ s[x]
    flagx2[m]=flagx2[m]^s[x]

print(''.join(chr(flagx[i]) for i in range(0,38)))
print(''.join(chr(flagx2[i]) for i in range(0,38)))
```

得到flag{f238yu28323uf28u2yef2ud8uf289euf}

100.[WUSTCTF2020]情书

查看题目



可以看到RSA加密系统

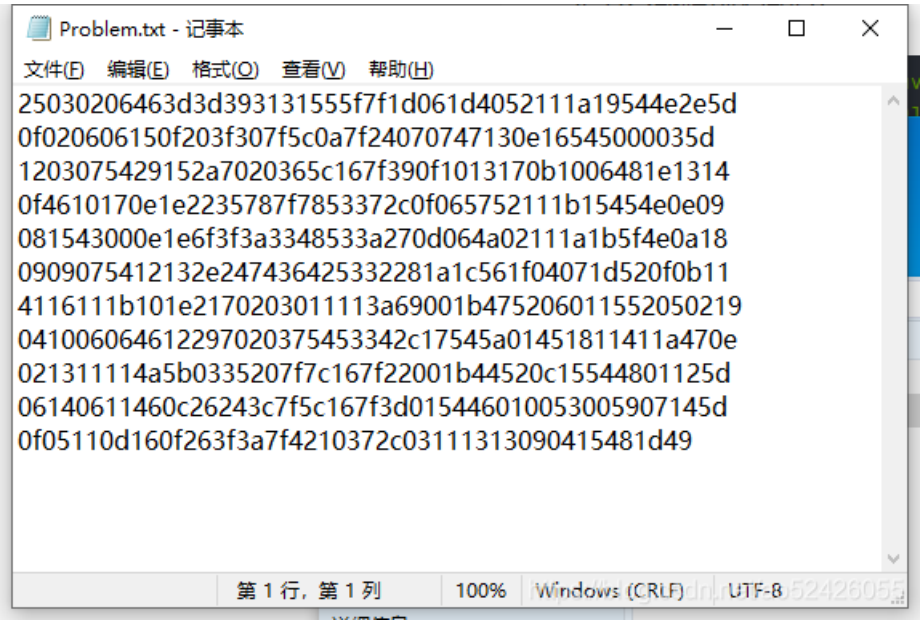
2537分解得到43和59

```
a = "abcdefghijklmnopqrstuvwxyz"
c = "0156 0821 1616 0041 0140 2130 1616 0793".split(" ")
N = 2537
e = 13
d = 937
p = 43
q = 59
phi_N = (p-1)*(q-1)

m = "".join(a[pow(int(i),d,N)] for i in c)
print (m)
```

运行得到iloveyou

101.[AFCTF2018]你听过一次一密么？

查看题目



加密方法

一次一密的加密算法就是异或加密。将明文分组，与密钥按位异或即可。

我这里出的题目是用同一个密钥去加密多条明文，当密文条数较多时就很容易被攻击，例如Many Time Pad。

Many Time Pad攻击

这个攻击的原理是c1⊕c2 = m1⊕m2，而通过m1⊕m2可以分析出m1和m2，因此m1与m2不再安全。

代码搬运自GitHub。

```
#!/usr/bin/python
## OTP - Recovering the private key from a set of messages that were encrypted w/ the same private key (Many tim
e pad attack) - crypto100-many_time_secret @ alexctf 2017
# Original code by jwomers: https://github.com/Jwomers/many-time-pad-attack/blob/master/attack.py)

import string
import collections
import sets, sys

# 11 unknown ciphertexts (in hex format), all encrpyted with the same key

c1='25030206463d3d393131555f7f1d061d4052111a19544e2e5d'
c2='0f020606150f203f307f5c0a7f24070747130e16545000035d'
c3='1203075429152a7020365c167f390f1013170b1006481e1314'
c4='0f4610170e1e2235787f7853372c0f065752111b15454e0e09'
c5='081543000e1e6f3f3a3348533a270d064a02111a1b5f4e0a18'
c6='0909075412132e247436425332281a1c561f04071d520f0b11'
c7='4116111b101e2170203011113a69001b475206011552050219'
c8='041006064612297020375453342c17545a01451811411a470e'
c9='021311114a5b0335207f7c167f22001b44520c15544801125d'
c10='06140611460c26243c7f5c167f3d0154460100530059071 45d'
c11='0f05110d160f263f3a7f4210372c03111313090415481d49'
ciphers = [c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11]
# The target ciphertext we want to crack
#target_cipher = "0529242a631234122d2b36697f13272c207f2021283a6b0c7908"

# XORs two string
def strxor(a, b):     # xor two strings (trims the longer input)
    return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b)])

def target_fix(target_cipher):
```

```
def target_fix(target_cipher):
    # To store the final key
    final_key = [None]*150
    # To store the positions we know are broken
    known_key_positions = set()

    # For each ciphertext
    for current_index, ciphertext in enumerate(ciphers):
        counter = collections.Counter()
        # for each other ciphertext
        for index, ciphertext2 in enumerate(ciphers):
            if current_index != index: # don't xor a ciphertext with itself
                for indexOfChar, char in enumerate(strxor(ciphertext.decode('hex'), ciphertext2.decode('hex'))):
 # Xor the two ciphertexts
                    # If a character in the xored result is a alphanumeric character, it means there was probabl
y a space character in one of the plaintexts (we don't know which one)
                    if char in string.printable and char.isalpha(): counter[indexOfChar] += 1 # Increment the co
unter at this index
        knownSpaceIndexes = []

        # Loop through all positions where a space character was possible in the current_index cipher
        for ind, val in counter.items():
            # If a space was found at least 7 times at this index out of the 9 possible XORS, then the space cha
racter was likely from the current_index cipher!
            if val >= 7: knownSpaceIndexes.append(ind)
        #print knownSpaceIndexes # Shows all the positions where we now know the key!

        # Now Xor the current_index with spaces, and at the knownSpaceIndexes positions we get the key back!
        xor_with_spaces = strxor(ciphertext.decode('hex'),' '*150)
        for index in knownSpaceIndexes:
            # Store the key's value at the correct position
            final_key[index] = xor_with_spaces[index].encode('hex')
            # Record that we known the key at this position
            known_key_positions.add(index)

    # Construct a hex key from the currently known key, adding in '00' hex chars where we do not know (to make a
 complete hex string)
    final_key_hex = ''.join([val if val is not None else '00' for val in final_key])
    # Xor the currently known key with the target cipher
    output = strxor(target_cipher.decode('hex'),final_key_hex.decode('hex'))

    print "Fix this sentence:"
    print ''.join([char if index in known_key_positions else '*' for index, char in enumerate(output)])+"\n"

    # WAIT.. MANUAL STEP HERE
    # This output are printing a * if that character is not known yet
    # fix the missing characters like this: "Let*M**k*ow if *o{*a" = "cure, Let Me know if you a"
    # if is too hard, change the target_cipher to another one and try again
    # and we have our key to fix the entire text!

    #sys.exit(0) #comment and continue if u got a good key

    target_plaintext = "cure, Let Me know if you a"
    print "Fixed:"
    print target_plaintext+"\n"

    key = strxor(target_cipher.decode('hex'),target_plaintext)

    print "Decrypted msg:"
    for cipher in ciphers:
```

```
        print strxor(cipher.decode('hex'),key)

    print "\nPrivate key recovered: "+key+"\n"

for i in ciphers:
    target_fix(i)
```

运行得到

```
'afctf{OPT_1s_Int3rest1ng}'
```

102.[BJDCTF2020]编码与调制

查看题目 曼彻斯特

6进制024A447B4469664D616E6368657374657263 6F64657D
通过解码16进制即可得到flag{DifManchestercode}

103.[网鼎杯 2020 青龙组]you_raise_me_up

查看题目

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from Crypto.Util.number import *
import random

n = 2 ** 512
m = random.randint(2, n-1) | 1
c = pow(m, bytes_to_long(flag), n)
print 'm = ' + str(m)
print 'c = ' + str(c)

# m = 391190709124527428959489662565274039318305952172936859403855079581402770986890308469084735451207885386318986881041563704825943945069343345307381099559075
# c = 666585139420321424585678945072365863252081679162179677590976689523300023402364287878602564495379799537321130848560539702412318008592411761080248597258449 9
```

mbytes_to_long(flag) %n=c的flag值.

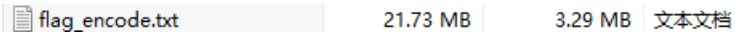
```
m = 391190709124527428959489662565274039318305952172936859403855079581402770986890308469084735451207885386318986
881041563704825943945069343345307381099559075
c = 666585139420321424585678945072365863252081679162179677590976689523300023402364287878602564495379799537321130
848560539702412318008592411761080248597258449 9
n = 2 ** 512
import sympy
flag=sympy.discrete_log(2**512,c,m)
import binascii
print(binascii.unhexlify(hex(flag)[2:]))
```

运行得到

```
b'flag{5f95ca93-1594-762d-ed0b-a9139692cb4a}'
```

104.[AFCTF2018]BASE

查看题目

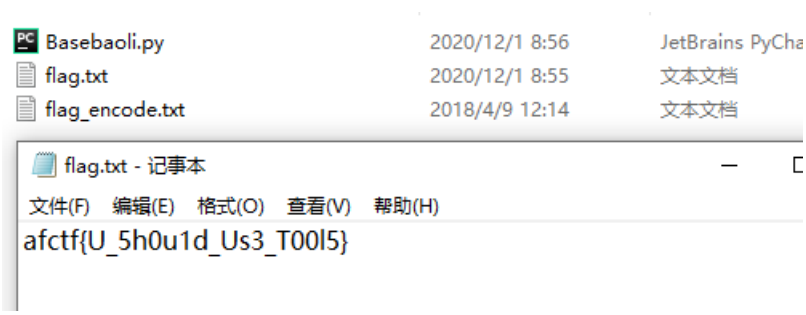


拿到附件是一个flag加密后的文本，其实是flag经过很多次base系列编码后的一个内容。文件有20+MB，文本编辑器发现是一行编码后的内容。

这题主要是三种编码，base64/base32/base16。base16也就是16进制。
写了一个很简单的脚本

```
import base64
file = open("flag_encode.txt",'r')
file2 = open("flag.txt",'w')
base = file.read()
while(1):
    try:
        base = base64.b32decode(base).decode()
    except:
        try:
            base = base64.b64decode(base).decode()
        except:
            try:
                base = base64.b16decode(base).decode()
            except:
                print("完成！")
                file2.write(base)
                break
```

运行得到

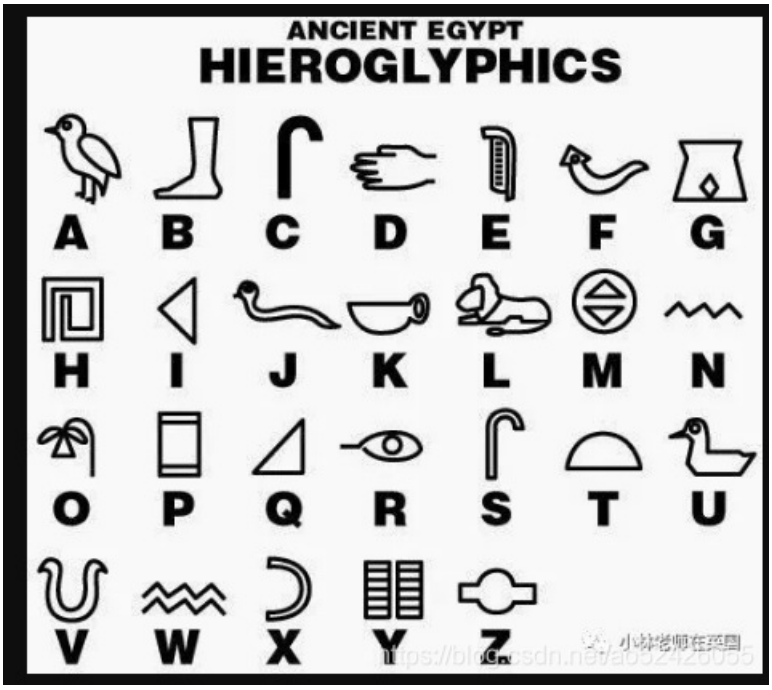


```
afctf{U_5h0u1d_Us3_T00l5}
```

105.[NPUCTF2020]Classical Ciphe

放在https://quipqiup.com/上解，试出来压缩包密码：the_key_is_atdash
解出密码得到图片

是象形文字和猪圈密码的结合

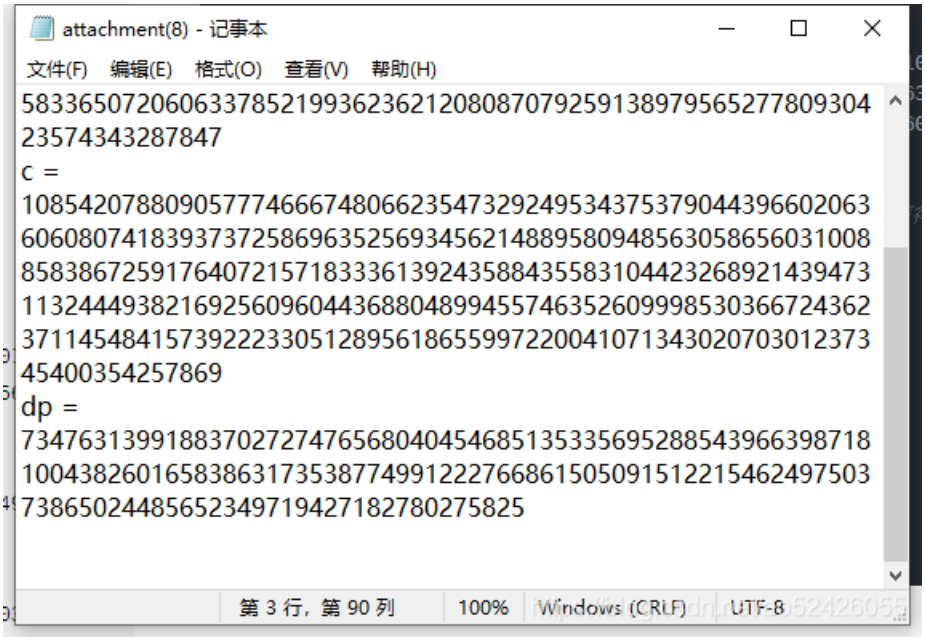


flag{classicalcode}

解压得到一张图，变形猪圈很容易看出来（那个鸟和狗是啥真没见过…）,根据题目名称里的classical还是很容易顺出来的，答案是 classicalcode，flag{classicalcode}

106.[WUSTCTF2020]dp_leaking_1s_very_d@angerous

这个一看到题目我相信有的大佬就直接把题目flag包裹提交了哈哈哈
没错题目就是答案
查看题目



bp泄露，把以前的脚本带一下数值就可以了

```
import gmpy2 as gp

e = 65537
n = 156808343598578774957375696815188980682166740609302831099696492068246337198792510898818496239166339015207305102101431634283168544492984586566799996471150252382144148257236707247267506165670877506370253127695314163987084076462560095456635833650720606337852199362362120808707925913897956527780930423574343287847
dp = 734763139918837027274765680404546851353356952885439663987181004382601658386317353877499122276686150509151221546249750373865024485652349719427182780275825
c = 108542078809057774666748066235473292495343753790443966020636060807418393737258696352569345621488958094856305865603100885838672591764072157183336139243588435583104423268921439473113244493821692560960443688048994557463526099985303667243623711454841573922233051289561865599722004107134302070301237345400354257869
for i in range(1, e):  # 在范围(1,e)之间进行遍历
    if (dp * e - 1) % i == 0:
        if n % (((dp * e - 1) // i) + 1) == 0:  # 存在p，使得n能被p整除
            p = ((dp * e - 1) // i) + 1
            q = n // (((dp * e - 1) // i) + 1)
            phi = (q - 1) * (p - 1)  # 欧拉定理
            d = gp.invert(e, phi)  # 求模逆
            m = pow(c, d, n)  # 快速求幂取模运算

print(m)  # 10进制明文
print('------------')
print(hex(m)[2:])  # 16进制明文
print('------------')
print(bytes.fromhex(hex(m)[2:]))  # 16进制转文本
```

运行得到

```
389117854707293786592764169809997493124376786541753609962979508064887479585043981668293367897 3
------------
776374663230323076 4705f6c65616b696e675f31735f766572795f6440616e6765726f75737d
------------
b'wctf2020{dp_leaking_1s_very_d@angerous}'
```

脚本这些东西该存就存一下

107.[UTCTF2020]basic-crypto

题目是一堆二进制

```
str='01010101 01101000 00101101 01101111 01101000 00101100 00100000 01101100 01101111 01101111 01101011 01110011
 00100000 01101100 01101001 01101011 01100101 00100000 01110111 01100101 00100000 01101000 01100001 01110110 011
00101 00100000 01100001 01101110 01101111 01110100 01101000 01100101 01110010 00100000 01100010 01101100 0110111
1 01100011 01101011 00100000 01101111 01100110 00100000 01110100 01100101 01111000 01110100 00101100 00100000 01
110111 01101001 01110100 01101000 00100000 01110011 01101111 01101101 01100101 00100000 01110011 01101111 011100
10 01110100 00100000 01101111 01100110 00100000 01110011 01110000 01100101 01100011 01101001 01100001 01101100 0
0100000 01100101 01101110 01100011 01101111 01100100 01101001 01101110 01100111 00101110 00100000 01000011 01100
001 01101110 00100000 01111001 01101111 01110101 00100000 01100110 01101001 01100111 01110101 01110010 01100101
00100000 01101111 01110101 01110100 00100000 01110111 01101000 01100001 01110100 00100000 01110100 01101000 0110
1001 01110011 00100000 01100101 01101110 01100011 01101111 01100100 01101001 01101110 01100111 00100000 01101001
 01110011 00111111 00100000 00101000 01101000 01101001 01101110 01110100 00111010 00100000 01101001 01100110 001
00000 01111001 01101111 01110101 00100000 01101100 01101111 01101111 01101011 00100000 01100011 01100001 0111001
0 01100101 01100110 01110101 01101100 01101100 01111001 00101100 00100000 01111001 01101111 01110101 00100111 01
101100 01101100 00100000 01101110 01101111 01110100 01101001 01100011 01100101 00100000 01110100 01101000 011000
01 01110100 00100000 01110100 01101000 01100101 01110010 01100101 00100000 01101111 01101110 01101100 01111001 0
0100000 01100011 01101000 01100001 01110010 01100001 01100011 01110100 01100101 01110010 01110011 00100000 01110
000 01110010 01100101 01110011 01100101 01101110 01110100 00100000 01100001 01110010 01100101 00100000 01000001
00101101 01011010 00101100 00100000 01100001 00101101 01111010 00101100 00100000 00110000 00101101 00111001 0010
1100 00100000 01100001 01101110 01100100 00100000 01110011 01101111 01101101 01100101 01110100 01101001 01101101
 01100101 01110011 00100000 00101111 00100000 01100001 01101110 01100100 00100000 00101011 00101110 00100000 010
10011 01100101 01100101 00100000 01101001 01100110 00100000 01111001 01101111 01110101 00100000 01100011 0110000
1 01101110 00100000 01100110 01101001 01101110 01100100 00100000 01100001 01101110 00100000 01100101 01101110 01
100011 01101111 01100100 01101001 01101110 01100111 00100000 01110100 01101000 01100001 01110100 00100000 011011
00 01101111 01101111 01101011 01110011 00100000 01101100 01101001 01101011 01100101 00100000 01110100 01101000 0
1101001 01110011 00100000 01101111 01101110 01100101 00101110 00101001 00001010 01010100 01101101 01010110 00110
011 01001001 01000111 01001110 01101111 01011001 01010111 01111000 01110011 01011010 01010111 00110101 01101110
01011010 01010011 01000101 01100111 01010001 00110010 01000110 01110101 01001001 01001000 01101100 01110110 0110
0100 01010011 01000010 01101101 01100001 01010111 01100100 00110001 01100011 01101101 01010101 01100111 01100010
 00110011 01010110 00110000 01001001 01001000 01100100 01101111 01011001 01011000 01010001 01101110 01100011 011
11001 01000010 01101110 01100010 00110010 01101100 01110101 01011010 01111001 01000010 01110110 01100010 0110100
1 01000010 01101111 01011010 01011000 01001010 01101100 01010000 01111001 01000010 01001010 01100100 01000011 01
000010 01110011 01100010 00110010 00111001 01110010 01100011 01111001 01000010 01110011 01100001 01010111 011101
00 01101100 01001001 01001000 01010010 01101111 01011010 01010011 01000010 01110011 01011010 01011000 01010010 0
0110000 01011010 01011000 01001010 01111010 01001001 01000111 01000110 01111001 01011010 01010011 01000010 01111
010 01100001 01000111 01101100 01101101 01100100 01000111 01010110 01101011 01001001 01000111 01001010 00110101
01001001 01001000 01001110 01110110 01100010 01010111 01010101 01100111 01011001 00110010 00111001 01110101 0110
0011 00110011 01010010 01101000 01100010 01101110 01010001 01110101 01001001 01000011 01101000 01101111 01100001
 01010111 00110101 00110000 01001111 01101001 01000010 00110101 01100010 00110011 01010101 01100111 01100010 010
10111 01101100 01101110 01100001 01001000 01010001 01100111 01100100 00110010 01000110 01110101 01100100 0100001
1 01000010 00110000 01100010 01111001 01000010 01111010 01100100 01000111 01000110 01111001 01100100 01000011 01
000010 01110011 01100010 00110010 00111001 01110010 01100001 01010111 00110101 01101110 01001001 01001000 010101
10 01110111 01001001 01000110 01001010 01110110 01100010 01010111 01000110 01110101 01001001 01001000 01000010 0
1101100 01100010 00110011 01000010 01110011 01011010 01010011 01101011 01110101 01000011 01101101 01110100 00110
010 01011001 01101110 01001110 01111000 01100011 01101101 01010001 01110011 01001001 01000111 01101100 00110101
01011010 01010011 01100100 01101001 01100010 01111001 01000010 01110010 01100100 01101110 01100100 00110101 0101
1001 00110010 01010001 01100111 01011010 01001000 01001010 01110110 01011001 01101101 00111000 01101000 01001001
 01000110 01101000 00110101 01011010 01111001 01000010 01110111 01100101 01010111 01001001 01100111 01011010 010
01000 01001010 01110110 01001001 01001000 01000010 01111010 01100101 01000111 01110100 00110010 01001001 0100001
1 01101000 01110010 01100101 01000111 00110100 01100111 01100100 00110010 01110100 01110000 01100010 01000111 00
111000 01100111 01011010 01001000 01001010 01110110 01001001 01001000 01001010 01110010 01011001 01101101 001101
01 01110110 01011001 00110010 01010001 01110101 01001100 01101001 00110100 01110000 01001001 01001000 01110000 0
1110010 01011001 01101101 01010001 00110110 01001001 01000111 01110011 01100111 01011001 00110010 01010110 01110
011 01011001 00110010 01010010 01111010 01011010 01000111 01010110 01101011 01100011 00110011 01101100 00110100
```

10 01101011 01100011 01101101 01110100 01101011 01001001 01000011 01100100 01110110 01001010 01111001 01000010 0
1101010 01100011 01101110 01101100 01101110 01011001 01111001 01000010 01101100 01100101 01101001 01000010 00110
011 01100101 01010111 01001110 01101011 01001001 01001000 01101100 01110111 01011010 01000111 00111001 00110100
01001001 01001000 01001110 00110100 01001001 01000111 01010010 01111001 01100010 01111001 01000010 01110010 0110
0100 01101110 01110000 01111001 01100001 00110010 01111000 01110110 01011010 01000011 01110111 01100111 01011001
00110011 01101011 01100111 01011010 01001000 01001010 01110010 01011010 01000011 01100100 01101010 01001001 010
01000 01110000 01101001 01100101 01010111 01111000 01110010 01100010 01001000 01011010 01110000 01001001 0100011
1 01010010 01111001 01100010 01111001 01000010 00110011 01100101 01010111 01001110 01101011 01001001 01000111 00
110001 00110101 01100100 00110011 01100100 00110101 01100101 01000011 01000010 01110100 01100011 01101101 011101
00 01101001 01100001 00110010 00110001 01101011 01100010 00110010 01001001 01100111 01100011 00110011 01100111 0
1100111 01011010 01001000 01001010 01110110 01001001 01000111 01010010 01110110 01100001 01000111 01010001 01110
011 01001001 01001000 01000010 00110101 01100100 01101110 01011010 00110101 01011010 00110010 00111001 01110101
01001001 01000111 01111000 01110000 01001001 01000011 01100100 01101011 01001010 01111001 01110111 01100111 0110
0001 00110011 01101000 01110101 01001001 01000111 01001110 00110101 01001001 01001000 01101100 00110100 01001100
01101001 01000010 01011010 01100101 01000111 00110001 01110110 01001001 01000111 01101100 00110101 01011010 010
10011 01000010 00110001 01100101 01001000 01101100 01101110 01001001 01000111 01110011 01100111 01100011 0100011
1 00111001 01101110 01001001 01000111 00110001 01111001 01100001 00110010 01001010 01110010 01100010 01010111 01
010010 01110110 01011001 01101101 01001101 01110011 01001001 01000111 01101100 00110101 01011010 01010011 010000
10 01110100 01100001 00110011 01100111 01100111 01100011 00110011 01101000 01110111 01100010 00110010 01001001 0
1100111 01011010 01001000 01001010 01110110 01001001 01000111 01001010 01110110 01011001 00110010 01010001 01100
111 01100101 01011000 01000001 01100111 01011010 01001000 01001010 01110110 01001001 01000111 01100100 00110101
01011001 01101101 00110101 01101010 01001001 01000111 01111000 01110010 01011001 00110010 00111001 01110101 0100
1001 01001000 01101100 00110100 01001001 01000111 00110001 00110101 01100100 00110011 01100100 00110101 01100101
01000011 01000010 01101110 01100101 01010111 01001010 01110101 01011001 01111001 01000010 01101011 01100011 011
01101 01110100 01101011 01001001 01000111 01001110 01111001 01100101 01010111 01100011 01100111 01011010 0101100
0 01101111 01100111 01100011 00110011 01100111 01100111 01011010 01001000 01001010 01110110 01001001 01000101 00
111001 00110100 01100011 01011000 01011010 01111010 01011001 00110011 01001001 01100111 01100100 01101101 011101
00 00110100 01100011 01010111 01010110 01110010 01100011 01010111 00111000 01110101 01000011 01101110 01001010 0
1101110 01100001 01000111 00110101 00110100 01100011 00110010 01010010 01101101 01100101 01011000 01001110 01101
011 01100100 01000111 01100100 01101111 01100100 01010011 01000101 01100111 01100011 01010111 01100100 01101101
01001001 01000111 01101100 01111010 01011001 01010111 01110011 01100111 01011001 00110011 01010010 01101111 0110
0100 01001000 01010110 01110000 01100001 00110010 01010101 01100111 01011010 01000111 01101100 01110010 01001001
01001000 01110000 01110010 01100010 01101110 01010010 01101111 01100001 01000111 01110100 00110100 01001001 010
01000 01001010 00110100 01100011 01010111 01111000 01101011 01011010 00110010 00110101 00110100 01100011 0011001
0 01111000 01110000 01100011 01010011 01000010 01111001 01100001 01011000 01001110 00110101 01100101 01010111 01
110100 01101111 01100010 01101101 01110011 01110101 01001001 01000111 01101100 01110010 01100101 01000111 011100
11 01100111 01100100 01001000 01010101 01100111 01100011 01111001 01000010 01101010 01100101 01011000 01001110 0
1110101 01001001 01000111 01001110 01101110 01100101 01000011 01000010 01111010 01100101 01011000 01101011 01100
111 01100011 01010111 01100100 01101101 01100101 01000011 01000010 01110000 01100011 00110011 01101000 01101100
01001001 01000111 01110100 01101010 01011001 00110010 01100100 00110100 01011010 01001000 01010101 00110110 0100
1001 01000111 01011010 01101011 01011001 00110011 01101100 01111010 01100010 01101110 01110100 01101111 01001101
01001000 01011010 01100110 01011010 01000111 01101011 00110000 01011010 01001000 01010110 01100110 01100100 011
01101 01101011 00110000 01011010 01000110 00111001 00110000 01011000 00110011 01001001 00110000 01100101 0101100
0 01101100 01100110 01100011 01101110 01101000 01111000 01100010 01000111 01010001 01110111 01100110 01010011 00
110100 01100111 01100011 01010111 01100100 01101101 01001001 01001000 01011010 00110000 01100101 01011000 011010
11 01100111 01011001 00110011 01010010 01101111 01011010 01010011 01000010 01101011 01100001 01011000 01001110 0
1101011 01001001 01001000 01001101 01100111 01100101 01010111 01100100 01101011 01001001 01000111 01100100 01101
010 01001001 01001000 01001010 00110100 01100011 01010111 01111000 01101011 01011010 00110010 00110101 00110100
01100011 00110010 01111000 01110000 01100011 01010011 01000010 00110000 01100100 01010011 01000010 01110111 0101
1010 01101110 01010110 01101011 01001001 01001000 01110000 01101101 01100100 01001000 01101100 01101100 01100100
01000111 01101000 01110101 01001001 01000111 01100100 01101010 01011001 01111001 01000010 01101011 01100001 010
11000 01010010 00110001 01001001 01001000 01010110 01101110 01100101 01000111 01010001 01100111 01011010 0011001
0 01001101 01100111 01100101 01101110 01001110 00110001 01100100 01001000 01001001 01100111 01011001 01101101 01
101000 01101110 01100100 01101110 01101100 01110010 01011010 01010111 00110101 01110010 01001100 01000011 010000
10 01111010 01100001 01000111 01010101 01100111 01100100 01000111 01010001 01100111 01100101 01000111 01110100 0
1111010 01100101 01011000 01101100 01111000 01001001 01001000 01010010 00110001 01001001 01000111 01101000 01101
110 01011010 01000011 01000010 00110001 01011010 01111001 01000010 00110110 01100011 00110010 01010101 01100111
01100011 00110010 01001110 01101011 01100001 00110011 01100111 01100111 01100011 00110011 01101100 00110101 0100
1100 01101001 01000010 01110000 01011010 00110010 01111000 01110010 01001001 01001000 01000110 01101110 01011010

```
'01101001 01000010 01110010 01100001 01001000 01000010 01101110 01100011 01010111 01110100 01101100 01001001 010
00111 01010010 01110000 01100001 01111001 01000010 01111001 01100001 01011000 01001110 00110101 01100101 0101011
1 01110100 01101111 01100010 01101101 01110011 01101000'
li=str.split(' ')
print(li)
flag_=''
for i in li:
    flag_+=chr(eval('0b'+i))
print(flag_)
base_='TmV3IGNoYWxsZW5nZSEgQ2FuIHlvdSBmaWd1cmUgb3V0IHdoYXQncyBnb2luZyBvbiBoZXJlPyBJdCBsb29rcyBsaWtlIHRoZSBsZXR0Z
XJzIGFyZSBzaGlmdGVkIGJ5IHNvbWUgY29uc3RhbnQuIChoaW50OiB5b3UgbWlnaHQgd2FudCB0byBzdGFydCBsb29raW5nIHVwIFJvbWFuIHBlb
3BsZSkuCmt2YnNxcmQsIGl5ZSdiby Brdnd5Y2QgZHJvYm8hIFh5ZyBweWIgZHJvIHBzeGt2IChreG4gd2tpbG8gZHJvIHJrYm5vY2QuLi4pIHprY
mQ6IGsgY2VsY2RzZGVkc3l4IG1zenJvYi4gU3ggZHJvIHB5dnZ5Z3N4cSBkb2hkLCBTJ2ZvIGRrdW94IHdpIHdvY2NrcW8ga3huIGJvenZrbW9uI
G9mb2JpIGt2enJrbG9kc20gbXJrYmttZG9iIGdzZHIgayBteWJib2N6eXhub3htbyBkeSBrIG5zcHBvYm94ZCBtcmtia21kb2IgLSB1eHlneCBrY
yBrIGNlbGNkc2RlZHN5eCBtc3pyb2IuIE1reCBpeWUgcHN4biBkcm8gcHN4a3YgcHBvcmtkIGRybyBwdmtxIHNjI
HF5c3hxIGR5IGxvIHlwIGRybyBweWJ3a2QgZWRwdmtxey4uLn0gLSBncnNtciB3b2t4YyBkcmtkIHNwIGl5ZSBjb28gZHJrZCB6a2Rkb2J4LCBpe
WUgdXh5ZyBncmtkIGRybyBteWJib2N6eXhub3htb2MgcHliIGUsIGQsIHAsIHYgaywga3huIHEga2JvLiBJeWUgbWt4IHpieWxrbHZpIGd5YnUge
WVkIGRybyBib3drc3hzeHEgbXJrYmttZG9iYyBsaSBib3p2a21zeHEgZHJvdyBreG4gc3hwb2Jic3hxIG15d3d5eCBneWJuYyBzeCBkcm8gT3hxd
nNjciB2a3hxZWtxby4gS3h5ZHJvYiBxYm9rZCB3b2RyeW4gc2MgZHkgZWNvIHBib2Flb3htaSBreGt2aWNzYzogZ28gdXh5ZyBkcmtkICdvJyBjc
nlnYyBleiB3eWNkIHlwZG94IHN4IGRybyBrdnpya2xvZCwgY3kgZHJrZCdjIHpieWxrbHZpIGRybyB3eWNkIG15d3d5eCBtcmtia21kb2Igc3ggZ
HJvIGRvaGQsIHB5dnZ5Z29uIGxpICdkJywga3huIGN5IH14LiBZeG1vIGl5ZSB1eHlnIGsgcG9nIG1ya2JrbWRvYmMsIGl5ZSBta3ggc3hwb2IgZ
HJvIGJvY2QgeXAgZHJvIGd5Ym5jIGxrY29uIHl4IG15d3d5eCBneWJuYyBkcmtkIGNyeWcgZXogc3ggZHJvIE94cXZzY3Igdmt4cWVrcW8uCnJna
G54c2RmeXNkdGdodSEgcWdmIGlzYWsgY3RodHVpa2UgZGlrIHprbnRoaGt4IHJ4cWxkZ254c2xpcSByaXN5eWtobmsuIGlreGsgdHUgcyBjeXNuI
GNneCBzeXkgcWdmeCBpc3hlIGtjY2d4ZHU6IGZkY3lzbntoMHZfZGk0ZHVfdmk0ZF90X3I0eXlfcnhxbGQwfS4gcWdmIHZ0eXkgY3RoZSBkaXNkI
HMgeWdkIGdjIHJ4cWxkZ254c2xpcSB0dSBwZnVkIHpmdHlldGhuIGdjYyBkaXR1IHVneGQgZ2MgenN1dHIgYmhndnlrZW5rLCBzaGUgdGQgeGtze
XlxIHR1IGhnZCB1ZyB6c2Ugc2Nka3ggc3l5LiBpZ2xrIHFnZiBraHBncWtlIGRpayByaXN5eWtobmsh'
import base64
flag_=base64.b64decode(base_).decode()
print(flag_)
```

运行得到

```
['01010101', '01101000', '00101101', '01101111', '01101000', '00101100', '00100000', '01101100', '01101111', '01
101111', '01101011', '01110011', '00100000', '01101100', '01101001', '01101011', '01100101', '00100000', '011101
11', '01100101', '00100000', '01101000', '01100001', '01110110', '01100101', '00100000', '01100001', '01101110',
 '01101111', '01110100', '01101000', '01100101', '01110010', '00100000', '01100010', '01101100', '01101111', '01
100011', '01101011', '00100000', '01101111', '01100110', '00100000', '01110100', '01100101', '01111000', '011101
00', '00101100', '00100000', '01110111', '01101001', '01110100', '01101000', '00100000', '01110011', '01101111',
 '01101101', '01100101', '00100000', '01110011', '01101111', '01110010', '01110100', '00100000', '01101111', '01
100110', '00100000', '01110011', '01110000', '01100101', '01100011', '01101001', '01100001', '01101100', '001000
00', '01100101', '01101110', '01100011', '01101111', '01100100', '01101001', '01101110', '01100111', '00101110',
 '00100000', '01000011', '01100001', '01101110', '00100000', '01111001', '01101111', '01110101', '00100000', '01
100110', '01101001', '01100111', '01110101', '01110010', '01100101', '00100000', '01101111', '01110101', '011101
00', '00100000', '01110111', '01101000', '01100001', '01110100', '00100000', '01110100', '01101000', '01101001',
 '01110011', '00100000', '01100101', '01101110', '01100011', '01101111', '01100100', '01101001', '01101110', '01
100111', '00100000', '01101001', '01110011', '00111111', '00100000', '00101000', '01101000', '01101001', '011011
10', '01110100', '00111010', '00100000', '01101001', '01100110', '00100000', '01111001', '01101111', '01110101',
 '00100000', '01101100', '01101111', '01101111', '01101011', '00100000', '01100011', '01100001', '01110010', '01
100101', '01100110', '01110101', '01101100', '01101100', '01111001', '00101100', '00100000', '01111001', '011011
11', '01110101', '00100111', '01101100', '01101100', '00100000', '01101110', '01101111', '01110100', '01101001',
 '01100011', '01100101', '00100000', '01110100', '01101000', '01100001', '01110100', '00100000', '01110100', '01
101000', '01100101', '01110010', '01100101', '00100000', '01101111', '01101110', '01101100', '01111001', '001000
00', '01100011', '01101000', '01100001', '01110010', '01100001', '01100011', '01110100', '01100101', '01110010',
 '01110011', '00100000', '01110000', '01110010', '01100101', '01110011', '01100101', '01101110', '01110100', '00
100000', '01100001', '01110010', '01100101', '00100000', '01000001', '00101101', '01011010', '00101100', '001000
00', '01100001', '00101101', '01111010', '00101100', '00100000', '00110000', '00101101', '00111001', '00101100',
 '00100000', '01100001', '01101110', '01100100', '00100000', '01110011', '01101111', '01101101', '01100101', '01
110100', '01101001', '01101101', '01100101', '01110011', '00100000', '00101111', '00100000', '01100001', '011011
10', '01100100', '00100000', '00101011', '00101110', '00100000', '01010011', '01100101', '01100101', '00100000',
```

'01101001', '01100110', '00100000', '01111001', '01101111', '01110101', '00100000', '01100011', '01100001', '01
101110', '00100000', '01100110', '01101001', '01101110', '01100100', '00100000', '01100001', '01101110', '001000
00', '01100101', '01101110', '01100011', '01101111', '01100100', '01101001', '01101110', '01100111', '00100000',
'01110100', '01101000', '01100001', '01110100', '00100000', '01101100', '01101111', '01101111', '01101011', '01
110011', '00100000', '01101100', '01101001', '01101011', '01100101', '00100000', '01110100', '01101000', '011010
01', '01110011', '00100000', '01101111', '01101110', '01100101', '00101110', '00101001', '00001010', '01010100',
'01101101', '01010110', '00110011', '01001001', '01000111', '01001110', '01101111', '01011001', '01010111', '01
111000', '01110011', '01011010', '01010111', '00110101', '01101110', '01011010', '01010011', '01000101', '011001
11', '01010001', '00110010', '01000110', '01110101', '01001001', '01001000', '01101100', '01110110', '01100100',
'01010011', '01000010', '01101101', '01100001', '01010111', '01100100', '00110001', '01100011', '01101101', '01
010101', '01100111', '01100010', '00110011', '01010110', '00110000', '01001001', '01001000', '01100100', '011011
11', '01011001', '01011000', '01010001', '01101110', '01100011', '01111001', '01000010', '01101110', '01100010',
'00110010', '01101100', '01110101', '01011010', '01111001', '01000010', '01110110', '01100010', '01101001', '01
000010', '01101111', '01011010', '01011000', '01001010', '01101100', '01010000', '01111001', '01000010', '010010
10', '01100100', '01000011', '01000010', '01110011', '01100010', '00110010', '00111001', '01110010', '01100011',
'01111001', '01000010', '01110011', '01100001', '01010111', '01110100', '01101100', '01001001', '01001000', '01
010010', '01101111', '01011010', '01010011', '01000010', '01110011', '01011010', '01011000', '01010010', '001100
00', '01011010', '01011000', '01001010', '01111010', '01001001', '01000111', '01000110', '01111001', '01011010',
'01010011', '01000010', '01111010', '01100001', '01000111', '01101100', '01101101', '01100100', '01000111', '01
010110', '01101011', '01001001', '01000111', '01001010', '00110101', '01001001', '01001000', '01001110', '011101
10', '01100010', '01010111', '01010101', '01100111', '01011001', '00110010', '00111001', '01110101', '01100011',
'00110011', '01010010', '01101000', '01100010', '01101110', '01010001', '01110101', '01001001', '01000011', '01
101000', '01101111', '01100001', '01010111', '00110101', '00110000', '01001111', '01101001', '01000010', '001101
01', '01100010', '00110011', '01010101', '01100111', '01100010', '01010111', '01101100', '01101110', '01100001',
'01001000', '01010001', '01100111', '01100100', '00110010', '01000110', '01110101', '01100100', '01000011', '01
000010', '00110000', '01100010', '01111001', '01000010', '01111010', '01100100', '01000111', '01000110', '011110
01', '01100100', '01000011', '01000010', '01110011', '01100010', '00110010', '00111001', '01110010', '01100001',
'01010111', '00110101', '01101110', '01001001', '01001000', '01010110', '01110111', '01001001', '01000110', '01
001010', '01110110', '01100010', '01010111', '01000110', '01110101', '01001001', '01001000', '01000010', '011011
00', '01100010', '00110011', '01000010', '01110011', '01011010', '01010011', '01101011', '01110101', '01000011',
'01101101', '01110100', '00110010', '01011001', '01101110', '01001110', '01111000', '01100011', '01101101', '01
010001', '01110011', '01001001', '01000111', '01101100', '00110101', '01011010', '01010011', '01100100', '011010
01', '01100010', '01111001', '01000010', '01110010', '01100100', '01101110', '01100100', '00110101', '01011001',
'00110010', '01010001', '01100111', '01011010', '01001000', '01001010', '01110110', '01011001', '01101101', '00
111000', '01101000', '01001001', '01000110', '01101000', '00110101', '01011010', '01111001', '01000010', '011101
11', '01100101', '01010111', '01001001', '01100111', '01011010', '01001000', '01001010', '01110110', '01001001',
'01001000', '01000010', '01111010', '01100101', '01000111', '01110100', '00110010', '01001001', '01000011', '01
101000', '01110010', '01100101', '01000111', '00110100', '01100111', '01100100', '00110010', '01110100', '011100
00', '01100010', '01000111', '00111000', '01100111', '01011010', '01001000', '01001010', '01110110', '01001001',
'01001000', '01001010', '01110010', '01011001', '01101101', '00110101', '01110110', '01011001', '00110010', '01
010001', '01110101', '01001100', '01101001', '00110100', '01110000', '01001001', '01001000', '01110000', '011100
10', '01011001', '01101101', '01010001', '00110110', '01001001', '01000111', '01110011', '01100111', '01011001',
'00110010', '01010110', '01110011', '01011001', '00110010', '01010010', '01111010', '01011010', '01000111', '01
010110', '01101011', '01100011', '00110011', '01101100', '00110100', '01001001', '01000111', '00110001', '011110
10', '01100101', '01101110', '01001010', '01110110', '01011001', '01101001', '00110100', '01100111', '01010101',
'00110011', '01100111', '01100111', '01011010', '01001000', '01001010', '01110110', '01001001', '01001000', '01
000010', '00110101', '01100100', '01101110', '01011010', '00110101', '01011010', '00110011', '01001110', '001101
00', '01100011', '01010011', '01000010', '01101011', '01100010', '00110010', '01101000', '01101011', '01001100',
'01000011', '01000010', '01010100', '01001010', '00110010', '01011010', '01110110', '01001001', '01000111', '01
010010', '01110010', '01100100', '01010111', '00111001', '00110100', '01001001', '01001000', '01100100', '011100
00', '01001001', '01001000', '01100100', '01110110', '01011001', '00110010', '01001110', '01110010', '01100011',
'01010111', '00111000', '01100111', '01100001', '00110011', '01101000', '01110101', '01001001', '01000111', '01
001010', '01110110', '01100101', '01101110', '01011010', '01110010', '01100010', '01010111', '00111001', '011101
01', '01001001', '01000111', '00111001', '01101101', '01100010', '00110010', '01001010', '01110000', '01001001',
'01000111', '01110100', '00110010', '01100101', '01101110', '01001010', '01110010', '01100010', '01000111', '00
111001', '01101011', '01100011', '00110010', '00110000', '01100111', '01100010', '01011000', '01001010', '011100
10', '01011001', '01101101', '01110100', '01110100', '01011010', '01000111', '00111001', '01101001', '01001001',
'01000111', '01100100', '01111010', '01011010', '01001000', '01001001', '01100111', '01100001', '01111001', '01
000010', '01110100', '01100101', '01010111', '01001010', '01101001', '01100010', '00110010', '01001110', '001101

10', '01100101', '01011000', '01101000', '01110101', '01100010', '00110011', '01101000', '01110100', '01100010',
'01111001', '01000010', '01101011', '01100101', '01010011', '01000010', '01110010', '01001001', '01000111', '00
110101', '01111010', '01100011', '01001000', '01000010', '01110110', '01011001', '01101101', '00111001', '001101
00', '01011010', '01000011', '01000010', '01110100', '01100011', '01101101', '01110100', '01101001', '01100001',
'00110010', '00110001', '01101011', '01100010', '00110010', '01001001', '01100111', '01001100', '01010011', '01
000010', '00110001', '01100101', '01001000', '01101100', '01101110', '01100101', '01000011', '01000010', '011100
10', '01011001', '01111001', '01000010', '01110010', '01001001', '01000111', '01001110', '01101100', '01100010',
'01000111', '01001110', '01101011', '01100011', '00110010', '01010010', '01101100', '01011010', '01001000', '01
001110', '00110101', '01100101', '01000011', '01000010', '01110100', '01100011', '00110011', '01110000', '011110
01', '01100010', '00110010', '01001001', '01110101', '01001001', '01000101', '00110001', '01110010', '01100101',
'01000011', '01000010', '01110000', '01100101', '01010111', '01010101', '01100111', '01100011', '01001000', '01
001110', '00110100', '01100010', '01101001', '01000010', '01101011', '01100011', '01101101', '00111000', '011001
11', '01100011', '01001000', '01001110', '00110100', '01100001', '00110011', '01011001', '01100111', '01100011',
'01001000', '01011010', '01110010', '01100011', '01010100', '00111000', '01100111', '01100011', '01101110', '01
001110', '00110100', '01011010', '01000100', '01101111', '01100111', '01010010', '00110010', '00111000', '011001
11', '01100100', '01011000', '01101000', '00110101', '01011010', '01111001', '01000010', '01101011', '01100011',
'01101101', '01110100', '01101011', '01001001', '01000111', '01010010', '01111001', '01100010', '01111001', '01
000010', '01110111', '01100100', '01101101', '01110100', '01111000', '01001001', '01001000', '01001110', '011010
10', '01001001', '01001000', '01000110', '00110101', '01100011', '00110011', '01101000', '01111000', '01001001',
'01000111', '01010010', '00110101', '01001001', '01000111', '01111000', '01110110', '01001001', '01001000', '01
101100', '01110111', '01001001', '01000111', '01010010', '01111001', '01100010', '01111001', '01000010', '011101
11', '01100101', '01010111', '01001010', '00110011', '01100001', '00110010', '01010001', '01100111', '01011010',
'01010111', '01010010', '01110111', '01100100', '01101101', '01110100', '01111000', '01100101', '01111001', '00
110100', '01110101', '01001100', '01101110', '00110000', '01100111', '01001100', '01010011', '01000010', '011011
10', '01100011', '01101110', '01001110', '01110100', '01100011', '01101001', '01000010', '00110011', '01100010',
'00110010', '01110100', '00110100', '01011001', '01111001', '01000010', '01101011', '01100011', '01101101', '01
110100', '01101011', '01001001', '01001000', '01001110', '01110111', '01001001', '01000111', '01101100', '001101
01', '01011010', '01010011', '01000010', '01101010', '01100010', '00110010', '00111000', '01100111', '01011010',
'01001000', '01001010', '01110010', '01011010', '01000011', '01000010', '00110110', '01100001', '00110010', '01
010010', '01101011', '01100010', '00110010', '01001010', '00110100', '01001100', '01000011', '01000010', '011100
00', '01100101', '01010111', '01010101', '01100111', '01100100', '01011000', '01101000', '00110101', '01011010',
'01111001', '01000010', '01101110', '01100011', '01101101', '01110100', '01101011', '01001001', '01000111', '01
010010', '01111001', '01100010', '01111001', '01000010', '01110100', '01100101', '01010111', '01001010', '011010
01', '01100010', '00110010', '01001110', '00110110', '01100101', '01011000', '01101000', '01110101', '01100010',
'00110011', '01101000', '01110100', '01100010', '00110010', '01001101', '01100111', '01100011', '01001000', '01
101100', '01101001', '01001001', '01000111', '01010101', '01110011', '01001001', '01000111', '01010001', '011100
11', '01001001', '01001000', '01000001', '01110011', '01001001', '01001000', '01011001', '01100111', '01100001',
'01111001', '01110111', '01100111', '01100001', '00110011', '01101000', '01110101', '01001001', '01001000', '01
000101', '01100111', '01100001', '00110010', '01001010', '01110110', '01001100', '01101001', '01000010', '010010
10', '01100101', '01010111', '01010101', '01100111', '01100010', '01010111', '01110100', '00110100', '01001001',
'01001000', '01110000', '01101001', '01100101', '01010111', '01111000', '01110010', '01100010', '01001000', '01
011010', '01110000', '01001001', '01000111', '01100100', '00110101', '01011001', '01101110', '01010101', '011001
11', '01100101', '01010111', '01010110', '01101011', '01001001', '01000111', '01010010', '01111001', '01100010',
'01111001', '01000010', '01101001', '01100010', '00110011', '01100100', '01110010', '01100011', '00110011', '01
101000', '01111010', '01100101', '01001000', '01000101', '01100111', '01100010', '01011000', '01001010', '011100
10', '01011001', '01101101', '01110100', '01110100', '01011010', '01000111', '00111001', '01101001', '01011001',
'01111001', '01000010', '01110011', '01100001', '01010011', '01000010', '01101001', '01100010', '00110011', '01
110000', '00110010', '01100001', '00110010', '00110001', '01111010', '01100101', '01001000', '01000101', '011001
11', '01011010', '01001000', '01001010', '01110110', '01100100', '01111001', '01000010', '01110010', '01100101',
'01000111', '00110100', '01100111', '01100011', '00110011', '01101000', '01110111', '01100010', '00110010', '01
001010', '01101001', '01100011', '00110011', '01101000', '01111000', '01001001', '01000111', '00110001', '001101
01', '01100100', '00110011', '01100100', '00110101', '01100101', '01000011', '01000010', '01101110', '01100101',
'01010111', '01001010', '01110101', '01011001', '01111001', '01000010', '01111010', '01100101', '01000011', '01
000010', '01101011', '01100011', '01101101', '00111000', '01100111', '01010100', '00110011', '01101000', '011110
00', '01100100', '01101110', '01001110', '01101010', '01100011', '01101001', '01000010', '00110010', '01100001',
'00110011', '01101000', '01111000', '01011010', '01010111', '01110100', '01111000', '01100010', '01111001', '00
110100', '01100111', '01010011', '00110011', '01101000', '00110101', '01011010', '01001000', '01001010', '011101
10', '01011001', '01101001', '01000010', '01111000', '01011001', '01101101', '00111001', '01110010', '01011010',
'01000011', '01000010', '00110011', '01100010', '00110010', '01010010', '01111001', '01100101', '01010111', '00
110100', '01100111', '01100011', '00110010', '01001101', '01100111', '01011010', '01001000', '01101011', '011001

```
11', '01011010', '01010111', '01001110', '01110110', '01001001', '01001000', '01000010', '01101001', '01100010',
 '00110010', '01000110', '01101100', '01100010', '00110011', '01101000', '01110100', '01100001', '01010011', '01
000010', '01110010', '01100101', '01000111', '01110100', '00110010', '01100001', '01010111', '01001110', '011110
10', '01011001', '01111010', '01101111', '01100111', '01011010', '00110010', '00111000', '01100111', '01100100',
 '01011000', '01101000', '00110101', '01011010', '01111001', '01000010', '01101011', '01100011', '01101101', '01
110100', '01101011', '01001001', '01000011', '01100100', '01110110', '01001010', '01111001', '01000010', '011010
10', '01100011', '01101110', '01101100', '01101110', '01011001', '01111001', '01000010', '01101100', '01100101',
 '01101001', '01000010', '00110011', '01100101', '01010111', '01001110', '01101011', '01001001', '01001000', '01
101100', '01110111', '01011010', '01000111', '00111001', '00110100', '01001001', '01001000', '01001110', '001101
00', '01001001', '01000111', '01010010', '01111001', '01100010', '01111001', '01000010', '01110010', '01100100',
 '01101110', '01110000', '01111001', '01100001', '00110010', '01111000', '01110110', '01011010', '01000011', '01
110111', '01100111', '01011001', '00110011', '01101011', '01100111', '01011010', '01001000', '01001010', '011100
10', '01011010', '01000011', '01100100', '01101010', '01001001', '01001000', '01110000', '01101001', '01100101',
 '01010111', '01111000', '01110010', '01100010', '01001000', '01011010', '01110000', '01001001', '01000111', '01
010010', '01111001', '01100010', '01111001', '01000010', '00110011', '01100101', '01010111', '01001110', '011010
11', '01001001', '01000111', '00110001', '00110101', '01100100', '00110011', '01100100', '00110101', '01100101',
 '01000011', '01000010', '01110100', '01100011', '01101101', '01110100', '01101001', '01100001', '00110010', '00
110001', '01101011', '01100010', '00110010', '01001001', '01100111', '01100011', '00110011', '01100111', '011001
11', '01011010', '01001000', '01001010', '01110110', '01001001', '01000111', '01010010', '01110110', '01100001',
 '01000111', '01010001', '01110011', '01001001', '01001000', '01000010', '00110101', '01100100', '01101110', '01
011010', '00110101', '01011010', '00110010', '00111001', '01110101', '01001001', '01000111', '01111000', '011100
00', '01001001', '01000011', '01100100', '01101011', '01001010', '01111001', '01110111', '01100111', '01100001',
 '00110011', '01101000', '01110101', '01001001', '01000111', '01001110', '00110101', '01001001', '01001000', '01
101100', '00110100', '01001100', '01101001', '01000010', '01011010', '01100101', '01000111', '00110001', '011101
10', '01001001', '01000111', '01101100', '00110101', '01011010', '01010011', '01000010', '00110001', '01100101',
 '01001000', '01101100', '01101110', '01001001', '01000111', '01110011', '01100111', '01100011', '01000111', '00
111001', '01101110', '01001001', '01000111', '00110001', '01111001', '01100001', '00110010', '01001010', '011100
10', '01100010', '01010111', '01010010', '01110110', '01011001', '01101101', '01001101', '01110011', '01001001',
 '01000111', '01101100', '00110101', '01011010', '01010011', '01000010', '01110100', '01100001', '00110011', '01
100111', '01100111', '01100011', '00110011', '01101000', '01110111', '01100010', '00110010', '01001001', '011001
11', '01011010', '01001000', '01001010', '01110110', '01001001', '01000111', '01001010', '01110110', '01011001',
 '00110010', '01010001', '01100111', '01100101', '01011000', '01000001', '01100111', '01011010', '01001000', '01
001010', '01110110', '01001001', '01000111', '01100100', '00110101', '01011001', '01101101', '00110101', '011010
10', '01001001', '01000111', '01111000', '01110010', '01011001', '00110010', '00111001', '01110101', '01001001',
 '01001000', '01101100', '00110100', '01001001', '01000111', '00110001', '00110101', '01100100', '00110011', '01
100100', '00110101', '01100101', '01000011', '01000010', '01101110', '01100101', '01010111', '01001010', '011101
01', '01011001', '01111001', '01000010', '01101011', '01100011', '01101101', '01110100', '01101011', '01001001',
 '01000111', '01001110', '01111001', '01100101', '01010111', '01100011', '01100111', '01011010', '01011000', '01
101111', '01100111', '01100011', '00110011', '01100111', '01100111', '01011010', '01001000', '01001010', '011101
10', '01001001', '01000101', '00111001', '00110100', '01100011', '01011000', '01011010', '01111010', '01011001',
 '00110011', '01001001', '01100111', '01100100', '01101101', '01110100', '00110100', '01100011', '01010111', '01
010110', '01110010', '01100011', '01010111', '00111000', '01110101', '01000011', '01101110', '01001010', '011011
10', '01100001', '01000111', '00110101', '00110100', '01100011', '00110010', '01010010', '01101101', '01100101',
 '01011000', '01001110', '01101011', '01100100', '01000111', '01100100', '01101111', '01100100', '01010011', '01
000101', '01100111', '01100011', '01010111', '01100100', '01101101', '01001001', '01000111', '01101100', '011110
10', '01011001', '01010111', '01110011', '01100111', '01011001', '00110011', '01010010', '01101111', '01100100',
 '01001000', '01010110', '01110000', '01100001', '00110010', '01010101', '01100111', '01011010', '01000111', '01
101100', '01110010', '01001001', '01001000', '01110000', '01110010', '01100010', '01101110', '01010010', '011011
11', '01100001', '01000111', '01110100', '00110100', '01001001', '01001000', '01001010', '00110100', '01100011',
 '01010111', '01111000', '01101011', '01011010', '00110010', '00110101', '00110100', '01100011', '00110010', '01
111000', '01110000', '01100011', '01010011', '01000010', '01111001', '01100001', '01011000', '01001110', '001101
01', '01100101', '01010111', '01110100', '01101111', '01100010', '01101101', '01110011', '01110101', '01001001',
 '01000111', '01101100', '01110010', '01100101', '01000111', '01110011', '01100111', '01100100', '01001000', '01
010101', '01100111', '01100011', '01111001', '01000010', '01101010', '01100101', '01011000', '01001110', '011101
01', '01001001', '01000111', '01001110', '01101110', '01100101', '01000011', '01000010', '01111010', '01100101',
 '01011000', '01101011', '01100111', '01100011', '01010111', '01100100', '01101101', '01100101', '01000011', '01
000010', '01110000', '01100011', '00110011', '01101000', '01101100', '01001001', '01000111', '01110100', '011010
10', '01011001', '00110010', '01100100', '00110100', '01011010', '01001000', '01010101', '00110110', '01001001',
 '01000111', '01011010', '01101011', '01011001', '00110011', '01101100', '01111010', '01100010', '01101110', '01
```

110100', '01101111', '01001101', '01001000', '01011010', '01100110', '01011010', '01000111', '01101011', '001100
00', '01011010', '01001000', '01010110', '01100110', '01100100', '01101101', '01101011', '00110000', '01011010',
 '01000110', '00111001', '00110000', '01011000', '00110011', '01001001', '00110000', '01100101', '01011000', '01
101100', '01100110', '01100011', '01101110', '01101000', '01111000', '01100010', '01000111', '01010001', '011101
11', '01100110', '01010011', '00110100', '01100111', '01100011', '01010111', '01100100', '01101101', '01001001',
 '01001000', '01011010', '00110000', '01100101', '01011000', '01101011', '01100111', '01011001', '00110011', '01
010010', '01101111', '01011010', '01010011', '01000010', '01101011', '01100001', '01011000', '01001110', '011010
11', '01001001', '01001000', '01001101', '01100111', '01100101', '01010111', '01100100', '01101011', '01001001',
 '01000111', '01100100', '01101010', '01001001', '01001000', '01001010', '00110100', '01100011', '01010111', '01
111000', '01101011', '01011010', '00110010', '00110101', '00110100', '01100011', '00110010', '01111000', '011100
00', '01100011', '01010011', '01000010', '00110000', '01100100', '01010011', '01000010', '01110111', '01011010',
 '01101110', '01010110', '01101011', '01001001', '01001000', '01110000', '01101101', '01100100', '01001000', '01
101100', '01101100', '01100100', '01000111', '01101000', '01110101', '01001001', '01000111', '01100100', '011010
10', '01011001', '01111001', '01000010', '01101011', '01100001', '01011000', '01010010', '00110001', '01001001',
 '01001000', '01010110', '01101110', '01100101', '01000111', '01010001', '01100111', '01011010', '00110010', '01
001101', '01100111', '01100101', '01101110', '01001110', '00110001', '01100100', '01001000', '01001001', '011001
11', '01011001', '01101101', '01101000', '01101110', '01100100', '01101110', '01101100', '01110010', '01011010',
 '01010111', '00110101', '01110010', '01001100', '01000011', '01000010', '01111010', '01100001', '01000111', '01
010101', '01100111', '01100100', '01000111', '01010001', '01100111', '01100101', '01000111', '01110100', '011110
10', '01100101', '01011000', '01101100', '01111000', '01001001', '01001000', '01010010', '00110001', '01001001',
 '01000111', '01101000', '01101110', '01011010', '01000011', '01000010', '00110001', '01011010', '01111001', '01
000010', '00110110', '01100011', '00110010', '01010101', '01100111', '01100011', '00110010', '01001110', '011010
11', '01100001', '00110011', '01100111', '01100111', '01100011', '00110011', '01101100', '00110101', '01001100',
 '01101001', '01000010', '01110000', '01011010', '00110010', '01111000', '01110010', '01001001', '01001000', '01
000110', '01101110', '01011010', '01101001', '01000010', '01110010', '01100001', '01001000', '01000010', '011011
10', '01100011', '01010111', '01110100', '01101100', '01001001', '01000111', '01010010', '01110000', '01100001',
 '01111001', '01000010', '01111001', '01100001', '01011000', '01001110', '00110101', '01100101', '01010111', '01
110100', '01101111', '01100010', '01101101', '01110011', '01101000']
Uh-oh, looks like we have another block of text, with some sort of special encoding. Can you figure out what thi
s encoding is? (hint: if you look carefully, you'll notice that there only characters present are A-Z, a-z, 0-9,
 and sometimes / and +. See if you can find an encoding that looks like this one.)
TmV3IGNoYWxsZW5nZSEgQ2FuIHlvdSBmaWd1cmUgb3V0IHdoYXQncyBnb2luZyBvbiBoZXJlPyBJdCBsb29rcyBsaWtlIHRoZSBsZXR0ZXJzIGFy
ZSBzaGlmdGVkIGJ5IHNvbWUgY29uc3RhbnQuIChoaW50OiB5b3UgbWlnaHQgd2FudCB0byBzdGFydCBsb29raW5nIHVwIFJvbWFuIHBlb3BsZSku
Cmt2YnNxcmQsIGl5ZSdibyBrdnd5Y2QgZHJvYm8hIFh5ZyBweWIgZHJvIHBzeGt2IChreG4gd2tpbG8gZHJvIHJrYm5vY2QuLi4pIHprYmQ6IGsg
Y2VsY2RzZGVkc3l4IG1zenJvYi4gU3ggZHJvIHB5dnZ5Z3N4cSBkb2hkLCBTJ2ZvIGRrdW94IHdpIHdvY2NrcW8ga3huIGJvenZrbW9uIG9mb2Jp
IGt2enJrbG9kc20gbXJrYmttZG9iIGdzZHIgayBteWJib2N6eXhub3htbyBkeSBrIG5zcHBvYm94IGQgbXJrYmttZG9iIC0gdXh5Z3gga2Mgk2Nl
bGNkc2RlZHN5eCBtc3pyb2IuIE1reCBpeWUgcHN4biBkcm8gcHN4a3YgcHZrcT8gcnN4ZDogR28gdXh5ZyBkcmtkIGRybyBwdmtxIHNjIHF5c3hx
IGR5IGxvIHlwIGRybyBweWJ3a2QgZWRwdmtxey4uLn0gLSBncnNtciB3b2t4YyBkcmtkIHNwIGl5ZSBjb28gZHJrZCB6a2Rkb2J4LCBpeWUgdXh5
ZyBncmtkIGRybyBteWJib2N6eXhub3htb2MgcHliIGUsIGQsIHAsIHYgaywga3huIHEga2JvLiBJeWUgbWt4IHpieWxrbHZpIGd5YnUgeWVkIGRy
byBib3drc3hzeHEgbXJrYmttZG9iYyBsaSBib3p2a21zeHEgZHJvdyBreG4gc3hwb2Jic3hxIG15d3d5eCBneWJuYyBzeCBkcm8gT3hxdnNjciB2
a3hxZWtxby4gS3h5ZHJvYiBxYm9rZCB3b2RyeW4gc2MgZHkgZWNvIHBib2Flb3htaSBreGt2aWNzYzogZ28gdXh5ZyBkcmtkICdvJyBjcnlnYyBl
eiB3eWNkIHlwZG94IHN4IGRybyBrdnpya2xvZCwgY3kgZHJrZCdjIHpieWxrbHZpIGRybyB3eWNkIG15d3d5eCBtcmtia21kb2Igc3ggZHJvIGRv
aGQsIHB5dnZ5Z29uIGxpICdkJywga3huIGN5IHl4LiBZeG1vIGl5ZSB1eHlnIGsgcG9nIG1ya2JrbWRvYmMsIGl5ZSBta3ggc3hwb2IgZHJvIGJv
Y2QgeXAgZHJvIGd5Ym5jIGxrY29uIHl4IG15d3d5eCBneWJuYyBkcmtkIGNyeWcgZXogc3ggZHJvIE94cXZzY3Igdmt4cWVrcW8uCnJnaG54c2Rm
eXNkdGdodSEgcWdmIGlzYWsgY3RodHVpa2UgZGlrIHprbnRoaGt4IHJ4cWxkZ254c2xpcSByaXN5eWtobmsuIGlreGsgdHUgcyBjeXNuIGNneCBz
eXkgcWdmeCBpc3hlIGtjY2d4ZHU6IGZkY3lzbntoMHZfZGk0ZHVfdmk0ZF90X3I0eXlfcnhxbGQwfS4gcWdmIHZ0eXkgY3RoZSBkaXNkIHMgeWdk
IGdjIHJ4cWxkZ254c2xpcSB0dSBwZnVkIGdjYyBkaXR1IHVneGQgZ2Ygc2dzdHUgcmhndnlrZW5rLCBzaGUgdGQgeGtlcXlxIHR1IGhnZCB2ZXJ1
IGhnZCB1ZyB6c2Ugc2Nka3ggc3l5LiBpZ2xrIHFnZiBraHBncWtlIGRpayByaXN5eWtobmsh
New challenge! Can you figure out what's going on here? It looks like the letters are shifted by some constant.
(hint: you might want to start looking up Roman people).
kvbsqrd, iye'bo kvwycd drobo! Xyg pyb dro psxkv (kxn wkilo dro rkbnocd...) zkbd: k celcdsdedsyx mszrob. Sx dro p
yvvygsxq dohd, S'fo dkuox wi wocckqo kxn bozvkmon ofobi kvzrklodsm mrkbkmdob gsdr k mybboczyxnoxmo dy k nsppobox
d mrkbkmdob - uxygx kc k celcdsdedsyx mszrob. Mkx iye psxn dro psxkv pvkq? rsxd: Go uxyg drkd dro pvkq sc qysxq
dy lo yp dro pybwkd edpvkq{...} - grsmr wokxc drkd sp iye coo drkd zkddobx, iye uxyg grkd dro mybboczyxnoxmoc py
b e, d, p, v k, kxn q kbo. Iye mkx zbylklvi gybu yed dro bowksxsxq mrkbkmdobc li bozvkmsxq drow kxn sxpobbsxq my
wwyx gybnc sx dro Oxqvscr vkxqekqo. Kxydrob qbokd wodryn sc dy eco pboaeoxmi kxkvicsc: go uxyg drkd 'o' crygc ez
 wycd ypdox sx dro kvzrklod, cy drkd'c zbylklvi dro wycd mywwyx mrkbkmdob sx dro dohd, pyvvygon li 'd', kxn cy y
x. Yxmo iye uxyg k pog mrkbkmdobc, iye mkx sxpob dro bocd yp dro gybnc lkcon yx mywwyx gybnc drkd cryg ez sx dro
 Oxqvscr vkxqekqo.
rghnxsdfysdtghu! qgf isak cthtuike dik zknthhkx rxqldgnxslig risyvkhnk. ikxk tu s cysn cgx syy qgfx isxe kccgxdu

```
: fdcysn{h0v_di4du_vi4d_t_r4yy_rxqld0}. qgf vtyy cthe disd s ygd gc rxqldgnxsliq tu pfud zftyethn gcc ditu ugxd
gc zsutr bhgvykenk, she td xksyyq tu hgd ug zse scdkx syy. iglk qgf khpgqke dik risyykhnk!
```

词频分析得到

```
congratulations! you have finished the beginner cryptography challenge. here is a flag for all your hard efforts
: utflag{n0w_th4ts_wh4t_i_c4ll_crypt0}. you will find that a lot of cryptography is just building off this sort
of basic knowledge, and it really is not so bad after all. hope you enjoyed the challenge!
```

flag{n0w_th4ts_wh4t_i_c4ll_crypt0}这个就是flag了

108.[GKCTF2020]Backdoor

查看题目 是一个cve漏洞
前面具体过程就不写了 那个流量的用记事本打开，然后是一个base64解密，另一个是公钥解密即可
脚本如下

```
from Crypto.Util import number
from gmpy2 import *
'''#密文c
02142af7ce70fe0ddae116bb7e96260274ee9252a8cb528e7fdd29809c2a6032727c05526133ae4610ed944572ff1abfcd0b17aa22ef44a2
'''
'''e = 65537
n = 155189610416250748761824045853940987814871410592854559270243212767838311221687450763597803430780112164805875
75072479784829258678691739
'''
'''p:33866199770511146373033285191736271658178321798452126407671970019 41
q:45824335611278553108052944566579932817826626451165430245370516824 79
'''
vals=39
M=1
n = mpz(155189610416250748761824045853940987814871410592854559270243212767838311221687450763597803430780112164805875757507247978482925867869 1739)
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 1
03, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227
, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353,
359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 49
1, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641,
 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 7
97, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947
, 953, 967, 971, 977, 983, 991, 997, 1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049, 1051, 1061, 1063, 1069, 108
7, 1091, 1093, 1097, 1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163, 1171, 1181, 1187, 1193, 1201, 1213, 1217, 1
223, 1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283, 1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321, 1327, 1361,
 1367, 1373, 1381, 1399, 1409, 1423, 1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459, 1471, 1481, 1483, 1487, 148
9, 1493, 1499, 1511, 1523, 1531, 1543, 1549, 1553, 1559, 1567, 1571, 1579, 1583, 1597, 1601, 1607, 1609, 1613, 1
619, 1621, 1627, 1637, 1657, 1663, 1667, 1669, 1693, 1697, 1699, 1709, 1721, 1723, 1733, 1741, 1747, 1753, 1759,
 1777, 1783, 1787, 1789, 1801, 1811, 1823, 1831, 1847, 1861, 1867, 1871, 1873, 1877, 1879, 1889, 1901, 1907, 191
3, 1931, 1933, 1949, 1951, 1973, 1979, 1987, 1993, 1997, 1999, 2003, 2011, 2017, 2027, 2029, 2039, 2053, 2063, 2
069, 2081, 2083, 2087, 2089, 2099, 2111, 2113, 2129, 2131, 2137, 2141, 2143, 2153, 2161, 2179, 2203, 2207, 2213,
 2221, 2237, 2239, 2243, 2251, 2267, 2269, 2273, 2281, 2287, 2293, 2297, 2309, 2311, 2333, 2339, 2341, 2347, 235
1, 2357, 2371, 2377, 2381, 2383, 2389, 2393, 2399, 2411, 2417, 2423, 2437, 2441, 2447, 2459, 2467, 2473, 2477, 2
503, 2521, 2531, 2539, 2543, 2549, 2551, 2557, 2579, 2591, 2593, 2609, 2617, 2621, 2633, 2647, 2657, 2659, 2663,
 2671, 2677, 2683, 2687, 2689, 2693, 2699, 2707, 2711, 2713, 2719, 2729, 2731, 2741, 2749, 2753, 2767, 2777, 278
9, 2791, 2797, 2801, 2803, 2819, 2833, 2837, 2843, 2851, 2857, 2861, 2879, 2887, 2897, 2903, 2909, 2917, 2927, 2
939, 2953, 2957, 2963, 2969, 2971, 2999]

for x in range(0, vals):
```

```
    M=M*primes[x]


for a in range(1,20):
    for k in range(50):
        p=mpz(k*M+(65537**a %M))
        if is_prime(p):
            q = mpz(n//p)
            if is_prime(q):
                print('p=%d\nq=%d'%(p,q))

import gmpy2
p =gmpy2.mpz(458243356112785531080529445665799328178266264511654302453705168247 9)
q =gmpy2.mpz(338661997705111463730332851917362716581783217984521264076719700194 1)
e =gmpy2.mpz(65537)
phi_n= (p - 1) * (q - 1)
d = gmpy2.invert(e, phi_n)
print("d is:%d"%(d))
# d is:1149956978599018129014215044754098629972931368939804379486522291475145627109733710462288499234512027895921314033386053756334771174215 3
import gmpy2,binascii

from Crypto.Util.number import long_to_bytes
c = int("02142af7ce70fe0ddae116bb7e96260274ee9252a8cb528e7fdd29809c2a6032727c05526133ae4610ed944572ff1abfcd0b17aa22ef44a2",16)

d = 1149956978599018129014215044754098629972931368939804379486522291475145627109733710462288499234512027895921314033386053756334771174215 3
n = 1551896104162507487618240458539409878148714105928545592702432127678383112216874507635978034307801121648058757507247978482925867869173 9
m = pow(c,d,n)
print('m = ',m)
print('m = ',hex(m))
print('long_to_bytes(m) =',long_to_bytes(m))
```

运行得到

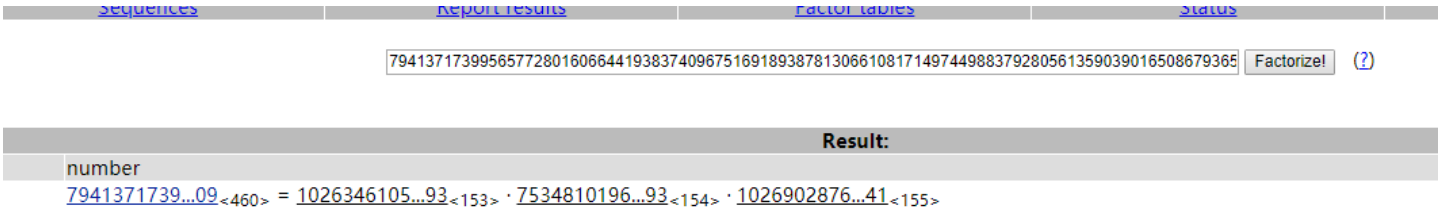
```
p=458243356112785531080529445665799328178266264511654302453705168247 9
q=338661997705111463730332851917362716581783217984521264076719700194 1
d is:1149956978599018129014215044754098629972931368939804379486522291475145627109733710462288499234512027895921314033386053756334771174215 3
m =  560063927934065528831067449817712559161537148281180970991300144074213308328500823539642621456572222 69
m =  0x666c61677b37363039353863392d636361392d343538622d396362652d656130376161313636386 5347d
long_to_bytes(m) = b'flag{760958c9-cca9-458b-9cbe-ea07aa1668e4}'
```

109.[V&N2020 公开赛]easy_RSA

查看题目

```python
from random import randint
from gmpy2 import *
from Crypto.Util.number import *

def getprime(bits):
    while 1:
        n = 1
        while n.bit_length() < bits:
            n *= next_prime(randint(1,1000))
        if isPrime(n - 1):
            return n - 1

m = bytes_to_long(b'flag{**********************************}')

p = getprime(505)
q = getPrime(512)
r = getPrime(512)
assert m < q

n = p * q * r
e = 0x10001
d = invert(q ** 2, p ** 2)
c = pow(m, 2, r)
cipher = pow(c, e, n)

print(n)
print(d)
print(cipher)


'''

794137173995657728016066441938374096751691893878130661081714974498837928056135903901650867936580610872219815719905880789270383755828067871142041124291405965805536634812310647333518650561741895663078064989494523334598527947110688635177256011468979083320605103256178446993230320443790240285158260236926519042413378204298514714890725325831769281505530787739922007367026883959544239568886349070557272869042275528961483412544495589811933856131557221673534170105409
751598784279417094944451720215806702111845455836014503039945348760369352269574673254722410084557011937597762907070230899122138872195225896975230590437872440200254594718252985960458440004898309186159472029979174388752122849271413544958400305438645775193309590298384124604895215509766824532266431851886144016181552339237189663931240329994319347050264087484514363884830125849837531400402896667129165106174033562061127306134852270841283140436659133571063017368170624129271357162815443486121503288672265151840789663971807716241487975280365482433433165015033647830925504804397494043011222770567328573994138052938992493130456846621463334486682095678988310912749300531477997566228441194639420871600623535260568794369980618031873434310815044745848165901997680344500054482 00

'''
```

还是用分解N的网站在线分解得到pqr



脚本如下

```
from Crypto.Util.number import *
from gmpy2 import *
from sympy.ntheory.residue_ntheory import nthroot_mod

n = 7941371739956577280160664419383740967516918938781306610817149744988379280561359039016508679365806108722198157199058807892703837558280678711420411242914059658055366348123106473335186505617418956630780649894945233345985279471106888635177256011468979083320605103256178446993230320443790240285158260236926519042413378204298514714890725325831769281505530787739922007367026883959544239568886349070557272869042275528961483412544495589811933856131557221673534170105409
d = 7515987842794170949444517202158067021118454558360145030399453487603693522695746732547224100845570119375977629070702308991221388721952258969752305904378724402002545947182529859604584400048983091861594720299791743887521228492714135449584003054386457751933095902983841246048952155097668245322664318518861440
cipher = 1618155233923718966393124032999431934705026408748451436388483012584983753140040289666712916510617403356206112730613485227084128314043665913357106301736817062412927135716281544348612150328867226515184078966397180771624148797528036548243343316501503364783092550480439749404301122277056732857399413805293899249313045684662146333448668209567898831091274930053147799756622844119463942087160062353526056879436998061803187343431081504474584816590199768034450005448200
#分解得到的pqr
p = 1026346105594789189708609579182599810573279493669493441371048048647682379616621361898271663175241512887996577585362569246097978101643970050817330394155393
q = 7534810196420932552168708937019691994681052660068275906973480617604535381306041583841106383688654426129050931519275383386503174076258645141589911492908993
r = 1026902876775430621756372166497626192440794088378419381778666041374486618464598423886646371187338007280374709236104124542234888363993371273305100579154384 1
phn=(p-1)*(q-1)*(r-1)
e = 0x10001
d = invert(e,phn)
print(d)
#696507389127827123706661871654450815284301235334466795840166268431469309011195076922454367593796687564813693144028411855398655084684946537577925810218190635193690686941033400271584581743511646525701770230697389808139688889055897738680146145489141461623210306152017482372340288056132947782754463871096351562634396868323861335480091797855608845695507892749062713128285934572259972389743839646950269761373893877746254763641876362977045864056456449244814748352513

c=pow(cipher,d,n)
print(c)
#808109245511251639736110581690049008535531557408753834078830988533410679632559382367878788756992040481498664381989876382887271652233886471418275706521368 3

m=nthroot_mod(c,2,r)
print(m)
#560063927934300164682519716465273289957181002071254323934339008750917393911906838117835749912363260 13

print(long_to_bytes(m))
```

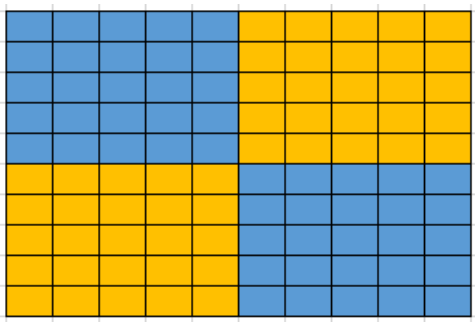
运行得到

```
69650738912782712370666187165445081528430123533446679584016626843146930901119507692245436759379668756481369314402841185539865508468494653757792581021819063519369068694103340027158458174351164652570177023069738980813968888905589773868014614548914146162321030615201748237234028805613294778275446387109635156263439686832386133548009179785560884569550789274906271312828593457225997238974383964695026976137389387774625476364187636297704586405645644924481474835251380810924551125163973611058169004900853553155740875383407883098853341067963255938236787878875699204048149866438198987638288727165223388647141827570652136835600639279343001646825197164652732899571810020712543239343390087509173939119068381178357499123632601 3b'flag{fd462593-25e4-4631-a96a-0cd5c72b2d1b}'
```

四方密码简介

四方密码是用四个方块，然后配合字母来完成加密。四方密码需要密钥，要用任意两个单词作为密钥。比如说题目中的security和information来作为密钥，然后去处单词里面的重复字母作为真正的密钥，也就是security和informat。

然后就是介绍四方了，四方，也就是四个方块，为4个5x5的方块，方块的左上角和右下角，也就是图中的蓝色方块，是用来按顺序存放26个字母的。



然而这里只有25个小方格，所以一般是把字母q或z删去不要，或者是把i、j放在同一个格子里面。

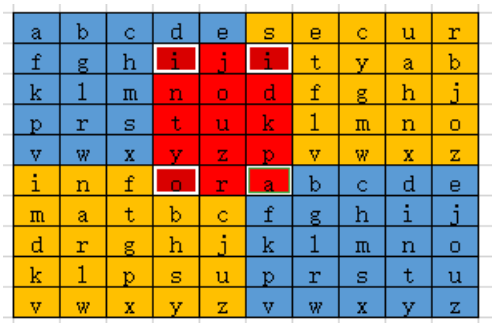
然后右上角和左下角的方块存放的是我们的密钥，然后分别把密钥写在田字格的右上角和左下角，然后再按字母排列表的顺序写下来。注意，密钥中已经有的字母就去除不写，不要再次出现！

加密

首先第一步，填写好这个方格，就拿security和information来说，填写之后的内容是这样的，这里是去除了q。

a	b	c	d	e	s	e	c	u	r
f	g	h	i	j	i	t	y	a	b
k	l	m	n	o	d	f	g	h	j
p	r	s	t	u	k	l	m	n	o
v	w	x	y	z	p	v	w	x	z
i	n	f	o	r	a	b	c	d	e
m	a	t	b	c	f	g	h	i	j
d	r	g	h	j	k	l	m	n	o
k	l	p	s	u	p	r	s	t	u
v	w	x	y	z	v	w	x	y	z

假设我们需要加密的内容是iamafool。首先把他拆分为两个两个一组，拆分完毕之后是ia ma fo ol。这里需要注意：四方密码加密只对偶数个字符有效，最后余下的一个字符将无法加密。然后按照顺序来，首先是ia，找到i在左上角方格中的位置和a在右下角方格中的位置。然后画出以他们连线为对角线的矩形，即为下图中的红色方块。



加密得到的字母与和明文在同一行的边角点，这里也就是i对应i，a对应o，ia加密得到的内容就是io，剩下来的字符也是同样的操作方式。

解密

揭秘也很简单了，就是加密的逆过程，比如说对应上面加密的例子，密文是io，明文就是ia，也就是现在左上角的方格中找到i，左下角找到o，然后取同一行的边角点就好了。

flag{youngandsuccessful}

110[De1CTF2019]babyrsa

查看题目

```
import binascii
from data import e1,e2,p,q1p,q1q,hint,flag

n =  [201296153524917654993401129431883171805487615978613008473058271415104656196705368446345582464392303716588369281030634328702457071803559071942848615109060712653524095794410481010849959239621485270973707054520705770987802462828200655737110156642919913720851570169012091141910685742086803977100428428359404284519495006076136346826841320876669402878927574852825428770575952849898630649426781719834065824187302480033601394629489168759101341493523782129180512328590533576271982377164785337889286889607842457223293436094067296243684952391556332877994213450449956886613526662807848523209820823703672412148183503573120138342 3L, 312216501556278499644664137494147006138238410601495244512349016771600090990140189265810948798400972485434119805330668319766170236762256250678540033170187940417236125560084715790604288981177905879910556813804082633827618416257144158790874780727719681603849099199580109836936836078850528885594612415951311884774799865642252141498029521264667585069093788376400057166757438141914437282421179801858680467482456460612259248328657580068523212827382008779181166387805782738637978788296276329006607223124881492046826474165408601107263821107544544784369104984726248575939329085311707286840686184079389581621595686952328923142 1L, 299445375153979533615209227741241926055247113067538353037034788904141635107774605597983343130212163893562518749177920076382992258210188496485206738137867724528228095465711298163102072328832397713241228848049934189583094600094063428721731890084492379595774691141589912024334767105813562438157137628024784543902738083774306851571100954967279663080012541075179675593840197342798618409972391762542360690014535445597860639159700711300878111239120443122195355138806639138313587903766504390836660611831156205113873793106880255882114422025746986403355066996567909581710647746463994280444700922867397754748628425967488232530303L, 257034378556001352151857784535839254469127316616040541841638832722655033230162957003572531053011467266678974974355325799749514783545704155542214017785361047372961543160563140394491163864943236684837498331478005574033684895422731694890802220093689039936584982639055675167986842114626070697966134346611481869018920162820659161909204433787561672508098724835017122257820043969699969830574239426071743141325984212691697225182244782488368810764846398373430793246369971451998350348333677430799353612761499909978759053136427752144860463813686196385518922927877831376222614335289152693334267689473585529197409018609826791807911]
c =  [191314326612179084702623384212996919985261577905835441567419812388221585639885202259869152345700373838881127244083929181139427219941255050147275459461333073297817476003028295882480429226357143910334319304111805450853164380843179273487052419275704327578929850913960449500854624295754400606529672538450413983996484423400429708144155719040576670281575129710793846017248163080786318444801102017873435830738151867717904777120400511571803188044221204720076367220639893153208635806313306471169938197777506841509504162980852614788411776816778672368656666207391847
```

```
046483954029213495373613490690687473081930148461830425717614569L, 153418984332266382351600720298757338269567999829581079102500559583349224602025549247431441221700183551174524594720171336146422424114798493690614828605702798636924256215260568628084251352676085448558333583140712006873404425128565752787129866415730124567294026605973396094437711453471812682850507289259935187048990054161872500033045812307014447051574127907870279268107109986461914671305507136007658982343923501539658115950606567537112783080051933709362961247907726894337734147036457039107421938984718000813214690552117093398463925007065236701452590242678583682169021764898147896794722273433630354285419151183781630120 31L, 187150650716480400179672112972311065381399850876853585556505670577155505864648147636836882990378971828450075785714013590612137776451144146429030770035681555084658196285537471732442359365868124454400954507551543576467370870716058119841634165902783526054333623279490482437225562629799094882024425303075058193715947479362238352335869454235222569387010023706463820978461050149817633077292346757377022521551308371548768318858886691504188850880893245348925061997244867834462673367898727821378955525093535833058801449477141100098931341621853823099926044356647774361975873123172248627238135109744930874502817554524287461944 46L, 228228456122485829313848044746331926247491884763014877011247270312854903259218779728996559261519970985787900827176643346203232849858034096887126018966970751855715783659242497325733436293163983107258482410312348652258253166615236387439648274456175813365540641036444217498322700550186092782087126071186100883012061705688351452579870960174408813599946559833863579427512314916549893358015994503236388061352492191302334120943965714596233221346857340286379692057181241820081481708623426228033822116162278951682936380508471565212173903618326402612086875652377019628414227184987900320219096615039006119546935171681953918379 7L]
f=lambda m,e,n,c:pow(m,e,n)==c
assert(sum(map(f,[p]*4,[4]*4,n,c))==4)

ee1 = 42
ee2 = 3
ce1 =  45722651786340123946960815003059322528810481841378247280642868553607692149509126962872583037142461398806689489141741494974836882341505234255325683219092163052843461632338442529011502378931140356111756932712822516814023166068902569458299933391973504078898958921809723346229893913662577294963528318424676803942288386430172430880307619748186863890050113934573820505570928109017842647598266634344447182347849367714564686341871007505886728393751147033556889217604647355628557502208364412269944908011305064122941446516990168924709684092200183860653173856272384
ce2 =  13908468332333567158469136439932325992349696889129103935400760239319454409539725389747059213835238373047899198211128689374049729578146875309231962936554403287882999967840346216695208424582739777034261079550395918048421086843927009452479936045850799096750074359160775182238980989229190157551197830879877097703347301072427149474991803868325769967332356950863518504965486565464059770451458557744949735282131727956056279292800694203866167270268988437389945703117070604488999247750139568614939965885211276821987586882908159585863514561191905040244967655444219603287214405014887994238259270716355378069726760953320025828158
tmp =  86407877807860983516777956598254075768407045069785430900517174281341496344746255499901271896092508162157148744725528982424037419052194840720949809891134854871222612682162490991065015935449289960707882463387
n  =  15911581555796798614711625288508309704791837516232122410440958830726078821069050404012820896260071751380436992710638364294658173571101596931605797509712839622479368850251206419748090059752427303611760004621378226431226983665746837779056271530181865648115862947527212787824629516204832313026456390047768174765687040950636530480549014401279054346098030395100387004111574278813749630986724706263655166289586230453975953773791945408589484679371854113457758157492241225180907090235116325034822993748409011554673180494306003272836905082473475046277554085737627846557240367696214081276345071055578169299060706794192776825039
assert(pow(e1,ee1,n)==ce1)
assert(pow(e2+tmp,ee2,n)==ce2)

e = 46531
n = 16278524034278364842964386062476113517067911891699789991355982121084973951738324063305190630865511554888330215827724887964565979607808294168282995825864982603759381323048907814961279012375346497781046417204954101076457350988751188332353062731641153547102721113593787978587135707313755661153376485647168543680503160420091693269984008764444291289486805840439906620313162344057956594836197521501755378387944609246120662335790110901623740990451586621846212047950084207251595169141015645449217847180683357626383565631317253913942886396494396189837432429078251573229378917400841832190737517876329732390158686666459532785060 3
c = 14992132140996160330967307558503117255626925777426611978518339050671013041490724616892634911030918360867974894371539160853827180596100892180735770688723270765387697604426715670445270819626709364566478781273676115921657967761494619448095207169386364541164659123273236874649888236433399127407801843412677293516986398190165291102109310458304626261648346825196743539220198199366711858135271877662410355585767124059539217274691606825103355310348607611233052725805236763220343249873849646219850954945346791015858261715967952461021650307307454434510851869862964236227932964442289459508441345652423088404453536608812799355469
hint=int(binascii.hexlify(hint),16)
```

```
hint=int(binascii.hexlify(hint),16)
assert(q1p*q1q==n)
assert(q1p<q1q)
assert(c==pow(hint,e,n))

flag=int(binascii.hexlify(flag),16)
q1=q1p
q2 =  1144011882274795846808840461512997046569205361687671329165891823575834610533363869961237832949325665677736
954266894474103119694564585747311875129748682970926386775152835849944163828724501670464165734726588416276909872 2
852879835689480355927830870263528853765319209851496608916812371085467963867142497822195951 3
c1 =  262739975753930281690942784321252339035906196846340713237510382364557685379543498765074448825799342194332 6
811811297700460750181220334219832278877196101120282306031665273030210363863507814144473471503837838168697840065 9
822558337545860958645085460286256902257167204915880987476381283404425741919963121752736704662488883775531121508 1
173386523806086783266198390289097231168172692326653657393522561741947951887577156666663584249108899327053951891 4
863551799397701505509958124783277359170061945744125188192993037832438869624553997836012292277187870817853910104 2
403050993740360035141417613812470516800228862066480927004612 4
c2 =  739559112922887664903081961668582189920483268499575772492445081297747078782226638712233472213276047091159 9
176362617225218345404468270014548817267727669872896838106451520392806497466576907063295603746660003188440170919 4
901572508293081733107153189257716431050648826207461712664998590490380169021625992614090509071408233529907502982 3
950835576723857570980316767681045655966547612114976694785191106470664650670539709162664871368451178045695545355 2
020460909638016134124590438425738826828694773960514221910109473941451471431637903182205738738109429736425025621 3
083008954731863818267566506678426560504162991663173727077095 96
assert(c1==pow(flag,e1,p*q1))
assert(c2==pow(flag,e2,p*q2))
```

这是一道综合的rsa
包括中国剩余定理特解为解
小指数攻击(枚举爆破)
在线质因数分解factordb
rsa中e关于(p-1)*(q-1)的逆元不唯一,同时兼有中国剩余定理方程合并.

得到两个方程

c1=me1 %(pq1)

c2=me2 %(pq2)

在通常情况下,我们只需要一个方程就可以求得一个满足条件的最小解作为解

这道题目最后给了两个方程,显然我们求得的解必须满足这两个方程.

首先看第一个方程.

通过计算得知gcd(e1,(p-1)(q1-1))=14.

可以知道该rsa式子并不能用常规方法.通过变形.

c1=me1 %(pq1)=(m14)e1/14%(pq1) ,我们可以在此条件下求出m14 的通解(显然最小特解很大可能不是答案,因为这个解还需要满足第二个方程)

记pq1=n1,pq2=n2,记e1/14的逆元为d1,方程2中e2/14的逆元为d2

记该通解为方程

m14 ☰c1d1mod n1

同理可得:

gcd(e2,(p-1)(q2-1))=14

则:

m14☰c2d2 mod n2

此时尝试用中国剩余定理来求出一个特解,但是遗憾的是该特解不是最终解.

于是尝试爆破,但是同样发现过了很长时间还是爆破不了.

于是转变方法.将该同余方程组进行细化.

记c1d =a1,c2d2 =a2

得到

m14 ☰a1 mod p

m14 ☰ a1 mod q1

m14 ☰ a2 mod p

m 14 ☰ a2 mod q2

由于m的指数过大,我们尝试通过构造一个新的rsa式子来降解m的指数.

理论上4个方程有6种合并方式.但是通过计算

gcd(p-1,7)!=1所以如果选择p的话显然是行不通的.

于是舍弃p

选择q1,q2进行合并.

合并函数

脚本如下

```
import gmpy2
import math
def  merge(a1,n1,a2,n2):
    d = math.gcd(n1,n2)
    c = a2-a1
    if c%d!=0:
        return 0
    c = (c%n2+n2)%n2
    c = c//d
    n1 = n1//d
    n2 = n2//d
    c *= gmpy2.invert(n1,n2)
    c %= n2
    c *= n1*d
    c += a1
    global n3
    global a3
    n3 = n1*n2*d
    a3 = (c%n3+n3)%n3
    return 1
```

```
def exCRT(a,n):
    a1=a[0]
    n1=n[0]
    le= len(a)
    for i in range(1,le):
        a2 = a[i]
        n2=n[i]
        if not merge(a1,n1,a2,n2):
            return -1
        a1 = a3
        n1 = n3
    global mod
    mod=n1
    return (a1%n1+n1)%n1
def exCRT_getequation(a,n):
    a1=a[0]
    n1=n[0]
    le= len(a)
    for i in range(1,le):
        a2 = a[i]
        n2=n[i]
        if not merge(a1,n1,a2,n2):
            return -1
        a1 = a3
        n1 = n3
    return (a1,n1)
#a为余数列表
#n为模数列表
n =  [201296153524917654993401129431883171805487615978613008473058271415104656196705368446345582464392303716588369281030634328702457071803559071942848615109060712653524095794410481010849959239621485270973707054520705770987802462828200655737110156642919913720851570169012091141910685742086803977100428428359404284519495006076136346826841132087666940287892757485282542877057595284989863064942678171983406582418730248003360139462948916875910134149352378212918051232859053357627198237716478533788928688960784245722329343609406729624368495239155633287799421345044995688661352666280784852320982082370367241214818350357312013834231, 312216501556278499644664137494147006138238410601495244512349016771600090990140189265810948798400972485434119805330668319766170236762256250678540033170187940417236125560084715790604288981177905879910556813804082633827618416257144158790874780727719681603849099199580109836693683607885052888559461241595131188477479986564225214149802952126466758506909378837640005716675743814191443728242117980185868046748245646061225924832865758006852321282738200877918116638780578273863797878829627632900660722312488149204682647416540860110726382110754454478436910498472624857593932908531170728684068618407938958162159568695232892314219, 299445375153979533615209227741241926055247113067538353037034788904141635107774605597983343130212163893562518749177920076382992258210188496485206738137867724528228095465711298163102072328832397713241228848049934189583094600094063428721731890084492379595774691141589912024334767105813562438157137628024784543902738083774306851571100954967279663080012541075179675593840197342798618409972391762542360690014535445597860639159700711300878111239120443122195355138806639138313587903766504390836606118311562051138737931068802558821144220257469864033550669965679095817106477464639942804447009228673977547486284259674882325303032, 257034378556001352151857784535839254469127316616040541841638832722655033230162957003572531053011467266678974974355325799749514783545704155542214017785361047372961543160563140394491163864943236684837498331478005574033684895422731694890802220093689039936584982639055675167986842114626070697966134346611481869018920162820659161909204433787561672508098724835017122257820043969699969830574239426071743141325984212691697225182244782488368810764846398373430793246369971451998350348333677430799353612761499909978759053136427752144860463813686196385518922927877831376222614335289152693334267689473585529197409018609826791807915]
c =  [191314326612179084702623384212996919985261577905835441567419812388221585639885202259869152345700373838881127244083929181139427219941255050147275459461333073297817476003028295882480429226357143910334319304111805450853164380843179273487052419275704327578929850913960449500854624295754400606529672538450413983996484423400429708144155719040576670281575129710793846017248163080786318444801102017873435830738151867717904777120400511571803188044221204720076367220639893153208635806313306471169938197777506841509504162980852614788411776816778672368656666207391847046483954029213495373613490690687473081930148461830425717614569, 153418984332266382351600720298757338269567999829581079102500559583349224602025549247431441221700183551174524594720171336146422424114798493690614828605702798636924256215260568628084251352676085448558333583140712006873404425128565752787129866415730124567294026605973396094400]
```

```
377114534718126828505072892599351870489900541618725000330458123070144470515741279078702792681071099864619146713055071360076589823439235015396581159506065675371127830800519337093629612479077268943377341470364570391074219389847180008132146905521170933984639250070652367014525902426785836821690217648981478967947222734336303542854191511837816301203 1, 187150650716480400179672112972311065381399850876853585556505670577155505864648147636836882990378971828450075785714013590612137776451144146429030770035681555084658196285537471732442359365868124454400954507551543576467370870716058119841634165902783526054333623279490482437225562629799094882024425303075058193715947479362238352335869454235222569387010023706463820978461050149817633077292346757377022521551308371548768318858886691504188850880893245348925061997244867834462673367898727821378955525093535833058801449477141100098931341621853823099926044356647774361975873123172248627238135109744930874502817554524287461944461884763014877011247270312854903259218779728996559261519970985787900827176643346203232849858034096887126018966970751855715783659242497325733436293163983107258482410312348652258253166615236387439648274456175813365540641036444217498322700550186092782087126071186100883012061705688351452579870960174408813599946559833863579427512314916549893358015994503236388061352492191302334120943965714596233221346857340286379692057181241820081481708623426228033822116162278951682936380508471565212173903618326402612086875652377019628414227184987900320219096615039006119546935171681953918379 7]
p_4=exCRT(c,n)
p=gmpy2.iroot(p_4,4)[0]
print(p)
#中国剩余定理
ee1 = 42
ee2 = 3
ce1 =  45722651786340123946960815003059322528810481841378247280642868553607692149509126962872583037142461398806689489141741494974836882341505234255325683219092163052843461632338442529011502378931140356111756932712822516814023166068902569458299933391973504078898958921809723346229893913662577294963528318424676803942288386430172430880307619748186863890050113934573820505570928109017842647598266634344471823478493677145646863418710075058867283937511470335568892176046473556285575022083644122699449080113050641229414465169901689247096840922001838606531738562723842ce2 =  13908468332333567158469136439932325992349696889129103935400760239319454409539725389747059213835238373047899198211128689374049729578146875309231962936554403287882999967840346216695208424582739777034261079550395918048421086843927009452479936045850799096750074359160775182238980989229190157551197830879877097703347301072427149474991803868325769967332356950863518504965486565464059770451458557744949735282131727956056279292800694203866167270268988437389945703117070604488999247750139568614939965885211276821987586882908159585863514561191905040244967655444219603287214405014887994238259270716355378069726760953320025828158tmp =  86407877807860983516777956598254075768407045069785430900517174281341496344746255499901271896092508162157148744472552898242403741905219484072094980989113485487122261268216249099106501593544928996070788246338748 = 15911581555796798614711625288508309704791837516232122410440958830726078821069050404012820896260071751380436992710638364294658173571101596931605797509712839622479368850251206419748090059752427303611760004621378226431226983665746837779056271530181865648115862947527212787824629516204832313026456390047768174765687040950636530480549014401279054346098030395100387004111574278813749630986724706263655166289586230453975953773791945408589484679371854113457758157492241225180907090235116325034822993748409011554673180494306003272836905082473475046277554085737627846557240367696214081276345071055578169299060706794192776825039#assert(pow(e1,ee1,n)==ce1)#assert(pow(e2+tmp,ee2,n)==ce2)#n  =  15911581555796798614711625288508309704791837516232122410440958830726078821069050404012820896260071751380436992710638364294658173571101596931605797509712839622479368850251206419748090059752427303611760004621378226431226983665746837779056271530181865648115862947527212787824629516204832313026456390047768174765687040950636530480549014401279054346098030395100387004111574278813749630986724706263655166289586230453975953773791945408589484679371854113457758157492241225180907090235116325034822993748409011554673180494306003272836905082473475046277554085737627846557240367696214081276345071055578169299060706794192776825039def doit(ee,n,ce):
    k=0
    while True:
        x=ce+k*n
        if gmpy2.iroot(x,ee)[1]:
            return gmpy2.iroot(x,ee)[0]
        k=k+1
e1=doit(ee1,n,ce1)
e2=doit(ee2,n,ce2)-tmp
##小指数爆破
q1p=12758731925343664356931214205855970681549721166108386659253421707931049726036530742609566128110371004239277
```

```
453866174657404985539066741684196020137840472950102380232067786400322600902938984916355631714439668326671310160916766472897536055371474076089779472372913037040153356437528808922911484049460342088834871
q1=q1p
##factordb爆破
q2 =  1144011882274795846808840461512997046569205361687671329165891823575834610533363869961237832949325665677736954266894474103119694564585747311875129748682970926386775152835849944163828724501670464165734726588416276909872285287983568948035592783087026352885376531920985149660891681237108546796386714249782219595513
c1 =  2627399757539302816909427843212523390359061968463407132375103823645576853795434987650744488257993421943326811811297700460750181220334219832278877196101120282306031665273030210363863507814144473471503837838168697840065982255833754586095864508546028625690225716720491588098747638128340442574191996312175273670466248888377553112150811733865238060867832661983902890972311681726923266536573935225617419479518875771566666635842491088993270539518914863551799397701505509958124783277359170061945744125188192993037832438869624553997836012292277187870817853910104240305099374036003514141761381247051680022886206648092700461246
c2 =  7395591129228876649030819616685821899204832684995757724924450812977470787822266387122334722132760470911599176362617225218345404468270014548817267727669872896838106451520392806497466576907063295603746660003188440170919490157250829308173310715318925771643105064882620746171266499859049038016902162599261409050907140823352990750298239508355767238575709803167676810456559665476121149766947851911064706646506705397091626648713684511780456955453552020460909638016134124590438425738826828694773960514221910109473941451471431637903182205738738109429736425025621308300895473186381826756650667842656050416299166317372707709596
#assert(c1==pow(flag,e1,p*q1))
#assert(c2==pow(flag,e2,p*q2))
phi1=(p-1)*(q1-1)
phi2=(p-1)*(q2-1)
xx1=gmpy2.gcd(e1,phi1)
xx2=gmpy2.gcd(e2,phi2)
d1=gmpy2.invert(e1//xx1,phi1)
d2=gmpy2.invert(e2//xx2,phi2)
nn=[]
aa=[]
nn.append(q1)
nn.append(q2)
a1=gmpy2.powmod(c1,d1,p*q1)%q1
a2=gmpy2.powmod(c2,d2,p*q2)%q2
aa.append(a1)
aa.append(a2)
last=exCRT_getequation(aa,nn)#最终方程组 aa=n^14%q1*q2
new_e=7
new_phi=(q1-1)*(q2-1)
new_d=gmpy2.invert(new_e,new_phi)
m_2=gmpy2.powmod(last[0],new_d,last[1])#特解m_2
flag=gmpy2.iroot(m_2,2)[0]
import binascii
print(binascii.unhexlify(hex(flag)[2:]))
```

运行得到

```
109935857933867829728985398563235455481120300859311421762540858762721955038310117609456763338082237907005937380873151279351831600225270995344096532750271070807051984097524900957809427861441436796934012393707770012556604479065826879107677002380580866325868240270494148512743861326447181476633546419262340100453
b'de1ctf{9b10a98b-71bb-4bdf-a6ff-f319943de21f}'
```

111.[UTCTF2020]hill

查看题目

```
wznqca{d4uqop0fk_q1nwofDbzg_eu}
```

这题就是个二阶希尔密码加密。知道密文前6位wznqca和明文前6位utflag，等于4个未知数6个一阶方程，居然还有解，那就是二阶跑不了了。求个逆乘回来就好。

```
from mpmath.libmp.backend import xrange

a00, a01, a10, a11 = 13, 6, 3, 21
dic = 'abcdefghijklmnopqrstuvwxyz'
c = 'wznqcaduqopfkqnwofdbzgeu'
m = ''
for i in xrange(0, len(c), 2):
    a, b = dic.index(c[i]), dic.index(c[i+1])
    a0, b0 = (a00*a+a01*b)%26, (a10*a+a11*b)%26
    m += dic[a0]
    m += dic[b0]
print(m)
```

记得把中间下划线、空格、数字之类的补回去，还有一个字母是大写别忘了
utflag{d4nger0us_c1pherText_qq}

112.[V&N2020 公开赛]CRT

查看题目

```
import hashlib
from functools import reduce
from Crypto.Util.number import *

ms = [getRandomNBitInteger(128) for i in range(8)]
p = reduce(lambda x,y: x*y, ms)
x = getRandomRange(1, p)
cs = [x % m  for m in ms]

flag = "flag{" + hashlib.sha256(str(x).encode()).hexdigest() + "}"
# assert("4b93deeb" in flag)

# ms = [284461942441737992421992210219060544764, 218436209063777179204189567410606431578, 288673438109933649911276214358963643204, 239232622368515797881077917549177081575, 206264514127207567149705234795160750411, 338915547568169045185589241329271490503, 246545359356590592172327146579550739141, 219686182542160835171493232381209438048]
# cs = [273520784183505348818648859874365852523, 128223029008039086716133583343107528289, 511109102540677127116777269686608341 9, 334623355951168204235878787846644484 39, 145377705960376589843356778052388633917, 128158421725856807614557926615949143594, 230664008267846531848877293149791626711, 945490199664809596889192333437939100003]
```

中国剩余定理，模不互质
根据assert里的 说明这道题需要爆破

```
import hashlib
import gmpy2
import math
def  merge(a1,n1,a2,n2):
    d = math.gcd(n1,n2)
    c = a2-a1
    if c%d!=0:
        return 0
    c = (c%n2+n2)%n2
    c = c//d
    n1 = n1//d
    n2 = n2//d
    c *= gmpy2.invert(n1,n2)
    c %= n2
    c *= n1*d
    c += a1
    global n3
    global a3
    n3 = n1*n2*d
    a3 = (c%n3+n3)%n3
    return 1
def exCRT(a,n):
    a1=a[0]
    n1=n[0]
    le= len(a)
    for i in range(1,le):
        a2 = a[i]
        n2=n[i]
        if not merge(a1,n1,a2,n2):
            return -1
        a1 = a3
        n1 = n3
    global mod
    mod=n1
    return (a1%n1+n1)%n1
ms = [284461942441737992421992210219060544764, 218436209063777179204189567410606431578, 288673438109933649911276214358963643204, 239232622368515797881077917549177081575, 206264514127207567149705234795160750411, 3389155475681690451855892413291271490503, 246545359356590592172327146579550739141, 219686182542160835171493232381209438048]
cs = [273520784183505348818648859874365852523, 128223029008039086716133583343107528289, 511109102540677127116777269686608341 9, 334623355951168204235878787846644484 39, 145377705960376589843356778052388633917, 1281584217258568076145579266159491435 94, 230664008267846531848877293149791626711, 945490199664809596889192333437939100 03]
x=exCRT(cs,ms)
flag=hashlib.sha256(str(x).encode()).hexdigest()
while "4b93deeb" not in flag:
    x=x+mod
    flag = hashlib.sha256(str(x).encode()).hexdigest()
print(flag)
```

python代码需要声明
运行得到

```
fa71921acdc2a756897d6b0c7ee41a7397386de2e7cde5b6adb525414b93deeb
```

113.[MRCTF2020]Easy_RSA

查看题目

```
import sympy
from gmpy2 import gcd, invert
```

```python
from random import randint
from Crypto.Util.number import getPrime, isPrime, getRandomNBitInteger, bytes_to_long, long_to_bytes
import base64

from zlib import *
flag = b"MRCTF{XXXX}"
base = 65537

def gen_prime(N):
    A = 0
    while 1:
        A = getPrime(N)
        if A % 8 == 5:
            break
    return A

def gen_p():
    p = getPrime(1024)
    q = getPrime(1024)
    assert (p < q)
    n = p * q
    print("P_n = ", n)
    F_n = (p - 1) * (q - 1)
    print("P_F_n = ", F_n)
    factor2 = 2021 * p + 2020 * q
    if factor2 < 0:
        factor2 = (-1) * factor2
    return sympy.nextprime(factor2)


def gen_q():
    p = getPrime(1024)
    q = getPrime(1024)
    assert (p < q)
    n = p * q
    print("Q_n = ", n)
    e = getRandomNBitInteger(53)
    F_n = (p - 1) * (q - 1)
    while gcd(e, F_n) != 1:
        e = getRandomNBitInteger(53)
    d = invert(e, F_n)
    print("Q_E_D = ", e * d)
    factor2 = 2021 * p - 2020 * q
    if factor2 < 0:
        factor2 = (-1) * factor2
    return sympy.nextprime(factor2)


if __name__ == "__main__":
    _E = base
    _P = gen_p()
    _Q = gen_q()
    assert (gcd(_E, (_P - 1) * (_Q - 1)) == 1)
    _M = bytes_to_long(flag)
    _C = pow(_M, _E, _P * _Q)
    print("Ciphertext = ", _C)
'''
P_n =  14057332139537395701238463644827948204030576528558543283405966933509944444681257521108769303999679955371474546213196051386802936343092965202519504111238572269823072199039812208100301939365080328518578704076769147484922508482686658959347725753762078590928561862163337382463252361958145933210306431342748775024336556028267742021320
```

```
8916817625436604684840186868658910731107573941540248335525588636715374910899570386483289737906923560147784203338
9670559525271151411747807282888019850618766792402026060012471724306742087636398053899410192943797866870912865258
7073901337310278665778299513763593234951137512120572797739181693
P_F_n =  1405733213953739570123846364482794820403057652855854328340596693350994444468125752110876930399967995537
1474546213196051386802936343092965202519504111238572269823072199039812208100301939365080328518578704076769147484
9225084826866589593477257537620785909285618621633373824632523619581459332103064313427487750240994273639673211101
2756203987901861608292693556795137818528088242690306459837666810661669462354007405721043279030957101877828172371
0994930151635857933293394780142192586806292968028305922173313521186946635709194350912242693822450297748434301924
9503585618598042567880980334265379562529649766823279914276267357 40
Q_n =  2071429833816044974954536074368801884287727405454085209645948528393680234127136376615797611252503400431993
8054034934880860956966585051684483662535780621673316774842614701726445870630109196016676725183412879870463432277
6299166691304940404037332955936553061041763679023524843675202629179431004676975405939257071621626166355335502627
1880874625459945628657840918789517101579699191012380452982551951927838891048313381333090253016044897292606083990
2082432745485612382530027894749207307600011040480932956805930333278188212553008934234121922658144185461340155575
9236219461780344469127987669565138930308525189944897421753947
Q_E_D =  1007720792222981345861161568507428178554081277169628919292598687466725726023339189580755826717524936182
5951828633612277270333018303722110505829865349079433788509849907358382183253279830951353838317523342953346734839
0389323225198805294950484802068148590902907221150968539067980432831310376368202773212266320112670699737501054831
6462865851422814192375722227139756468435550247318556885738341087118744061495400782537743497081580630557549328126
7578612370076828804844532619988098371750453882549810378930487368219105305036680682580260265867426844084457795549
9368404019114913934477160428428662847012289516655310680119638600315228284298935201
Ciphertext =  4085593735522843852536116152444127463417535684595088488933863081318260748591009467790977912655026330
4194796000904384775495000943424070396334435810126536165332565417336797036611773382728344687175253081047586602838
6850274282926215579145146290243247942757725220131264649269906201404064129994857287503858768681150917354255775550
2739403341664303264477433964465401168671663976051235335571906579522220116721983178096130822578047848246729441082
8543488412258764446494815238766185728454416691898859462532083437213793104823759147317613637881419787581920745151
4303945267127906084429601065375391218805142698306963417375077174489469620 21
'''
```

114.[MRCTF2020]Easy_RSA

查看题目

```
import sympy
from gmpy2 import gcd, invert
from random import randint
from Crypto.Util.number import getPrime, isPrime, getRandomNBitInteger, bytes_to_long, long_to_bytes
import base64

from zlib import *
flag = b"MRCTF{XXXX}"
base = 65537

def gen_prime(N):
    A = 0
    while 1:
        A = getPrime(N)
        if A % 8 == 5:
            break
    return A

def gen_p():
    p = getPrime(1024)
    q = getPrime(1024)
    assert (p < q)
    n = p * q
    print("P_n = ", n)
    F_n = (p - 1) * (q - 1)
```

```
    print("P_F_n = ", F_n)
    factor2 = 2021 * p + 2020 * q
    if factor2 < 0:
        factor2 = (-1) * factor2
    return sympy.nextprime(factor2)


def gen_q():
    p = getPrime(1024)
    q = getPrime(1024)
    assert (p < q)
    n = p * q
    print("Q_n = ", n)
    e = getRandomNBitInteger(53)
    F_n = (p - 1) * (q - 1)
    while gcd(e, F_n) != 1:
        e = getRandomNBitInteger(53)
    d = invert(e, F_n)
    print("Q_E_D = ", e * d)
    factor2 = 2021 * p - 2020 * q
    if factor2 < 0:
        factor2 = (-1) * factor2
    return sympy.nextprime(factor2)


if __name__ == "__main__":
    _E = base
    _P = gen_p()
    _Q = gen_q()
    assert (gcd(_E, (_P - 1) * (_Q - 1)) == 1)
    _M = bytes_to_long(flag)
    _C = pow(_M, _E, _P * _Q)
    print("Ciphertext = ", _C)
'''
P_n =  140573321395373957012384636448279482040305765285585432834059669335099444446812575211087693039996799553714
745462131960513868029363430929652025195041112385722698230721990398122081003019393650803285185787040767691474849 2
250848268665895934772575376207859092856186216333738246325236195814593321030643134274877502433655602826774202132 0
891681762543660468484018686865891073110757394154024833552558863671537491089957038648328973790692356014778420333 8
967055952527115141174780728288801985061876679240202606001247172430674208763639805389941019294379786687091286525 8
707390133731027866577829951376359323495113751212057279773918169 3
P_F_n =  140573321395373957012384636448279482040305765285585432834059669335099444446812575211087693039996799553 7
147454621319605138680293634309296520251950411123857226982307219903981220810030193936508032851857870407676914748 4
922508482686658959347725753762078590928561862163337382463252361958145933210306431342748775024099427363967321110 1
275620398790186160829269355679513781852808824269030645983766681066166946235400740572104327903095710187782817237 1
099493015163585793329339478014219258680629296802830592217331352118694663570919435091224269382245029774843430192 4
950358561859804256788098033426537956252964976682327991427626735740 
Q_n =  207142983381604497495453607436880188428772740545408520964594852839368023412713637661579761125250340043199 3
805403493488086095696658505168448366253578062167331677484261470172644587063010919601667672518341287987046343227 7
629916669130494040403733295593655306104176367902352484367520262917943100467697540593925707162162616635533502627 1
880874625459945628657840918789517101579699191012380452982551951927838891048313381333090253016044897292606908399 0
208243274548561238253002789474920730760001104048093295680593033327818821255300893423412192265814418546134015557 5
792362194617803444691279876695651389303085251899448974217539477 
Q_E_D =  100772079222298134586116156850742817855408127716962891929259868746672572602333918958075582671752493618 2
595182863361227727033301830372211050582986534907943378850984990735838218325327983095135383831752334295334673483 9
038932322519880529495048480206814859090290722115096853906798043283131037636820277321226632011267069973750105483 1
646286585142281419237572222713975646843555024731855688573834108711874406149540078253774349708158063055754932812 6
757861237007682880484453261998809837175045388254981037893048736821910530503668068258026026586742684408445779554 9
936840401911491393447716042842866284701228951665531068011963860031522828429893520 1 
Ciphertext =  408559373552284385253611615244412746341753568459508848893386308131826074859100946779097791265502 63
```

```
3041947960009043847754950009434240703963344358101265361653325654173367970366117733827283446871752530810475866028
3868502742829262155791451462902432479427577252201312646492699062014040641299948572875038587686811509173542557755
5027394033416643032644774339644654011686716639760512353355719065795222201167219831780961308225780478482467294410
8285434884122587644464948152387661857284544166918988594625320834372137931048237591473176136378814197875819207451
5143039452671279060844296010653753912188051426983069634173750771744894696 2021
'''
```

脚本如下

```
P_n =  1405733213953739570123846364482794820403057652855854328340596693350994444468125752110876930399967995537147454621319605138680293634309296520251950411123857226982307219903981220810030193936508032851857870407676914748492250848268665895934772575376207859092856186216333738246325236195814593321030643134274877502433655602826774202132089168176254366046848401868686589107311075739415402483355255886367153749108995703864832897379069235601477842033389670559525271151411747807282888019850618766792402026060012471724306742087636398053899410192943797866870912865258707390133731027866577829951376359323495113751212057279773918169 3
P_F_n =  1405733213953739570123846364482794820403057652855854328340596693350994444468125752110876930399967995537147454621319605138680293634309296520251950411123857226982307219903981220810030193936508032851857870407676914748492250848268665895934772575376207859092856186216333738246325236195814593321030643134274877502409942736396732111012756203987901861608292693556795137818528088242690306459837666810661669462354007405721043279030957101877828172371099493015163585793329339478014219258680629296802830592217331352118694663570919435091224269382245029774843430192495035856185980425678809803342653795625296497668232799142762673574 0
Q_n =207142983381604497495453607436880188428772740545408520964594852839368023412713637661579761125250340043199380540349348808609569665850516844836625357806216733167748426147017264458706301091960166767251834128798704634322776299166691304940404037332955936553061041763679023524843675202629179431004676975405939257071621626166355335502627188087462545994562865784091878951710157969919101238045298255195192783889104831338133309025301604489729260960839902082432745485612382530027894749207307600011040480932956805930333278188212553008934234121922658144185461340155575792362194617803444691279876695651389303085251899448974217539 47
Q_E_D =100772079222298134586116156850742817855408127716962891929259868746672572602333918958075582671752493618259518286336122772703330183037221105058298653490794337885098499073583821832532798309513538383175233429533467348390389323225198805294950484802068148590902907221150968539067980432831310376368202773212266320112670699737501054831646286585142281419237572222713975646843555024731855688573834108711874406149540078253774349708158063055754932812675786123700768288048445326199880983717504538825498103789304873682191053050366806825802602658674268440844577955499368404019114913934477160428428662847012289516655310680119638600315228284298935201Ciphertext =  4085593735522843852536116152444127463417535684595088488933863081318260748591009467790977912655026330419479600090438477549500094342407039633443581012653616533256541733679703661177338272834468717525308104758660283868502742829262155791451462902432479427577252201312646492699062014040641299948572875038587686811509173542557755502739403341664303264477433964465401168671663976051235335571906579522220116721983178096130822578047848246729441082854348841225876444649481523876618572845441669189885946253208343721379310482375914731761363788141978758192074515143039452671279060844296010653753912188051426983069634173750771744894696 2021import gmpy2import sympybase = 65537def getp(n,phi):    p_q=n+1-phi    p_q_2=p_q**2-4*n    p_q_2=gmpy2.iroot(p_q_2,2)[0]    q=(p_q_2+p_q)//2    p = p_q - q    factor2 = 2021 * p + 2020 * q    if factor2 < 0:        factor2 = (-1) * factor2    P = sympy.nextprime(factor2)    return Pdef getq(n,phi):    p_q=n+1-phi    p_q_2=p_q**2-4*n    p_q_2=gmpy2.iroot(p_q_2,2)[0]    q=(p_q_2+p_q)//2    p = p_q - q
```

```
    factor2 = 2021 * p -2020 * q
    if factor2 < 0:
        factor2 = (-1) * factor2
    Q = sympy.nextprime(factor2)
    return Q
p=getp(P_n,P_F_n)
K=((Q_E_D-1)//Q_n)+1
phi=(Q_E_D-1)//K
q=getq(Q_n,phi)
e=base
d=gmpy2.invert(e,(p-1)*(q-1))
M=gmpy2.powmod(Ciphertext,d,p*q)
import binascii
print(binascii.unhexlify(hex(M)[2:]))

#b'MRCTF{Ju3t_@_31mp13_que3t10n}'
```

115.[NPUCTF2020]EzRSA

查看题目

```
from gmpy2 import lcm , powmod , invert , gcd , mpz
from Crypto.Util.number import getPrime
from sympy import nextprime
from random import randint
p = getPrime(1024)
q = getPrime(1024)
n = p * q
gift = lcm(p - 1 , q - 1)
e = 54722
flag = b'NPUCTF{******************}'
m = int.from_bytes(flag , 'big')
c = powmod(m , e , n)
print('n: ' , n)
print('gift: ' , gift)
print('c: ' , c)

#n:  170839412302134897004266364844877382824264714946070988472953353396381775836854579211985691054177346686920727275913935820766724870395243668018315332760614742193236588998334728204643915617668576514362063710734787040194694650162053166557366806834908041080799658229750588994620505287900202893612531531225647058362291364631977912555969127091606458868499738245141274743272296691951341370998735303837547717838512545356711196525972148499715679935561764213156909581030407713105358848305724434074275180493549408768736341692131404154709311856576760966703385958312527532207761757678324785371851616674385826529113535389523998112 1
#gift:  213549265377668621255332956056096728530330893682588735591191691745477219796068224014982113817721683358650909096989241977595840608799405458502289416595076842774154573624791841025580489452208572064295257963841848380024336831270256645819670850854363505135099957278718823624327563160987525361701566441403205882291946944328445340306407623276502424843554332659741885175158630851454012457130915278755971295020935782557689613227804511217791026601974101399510657948486876825108445333841711548351513286959471216205236208341416395468130625913705758103665744189742843257592401895096114182255425136926224836889997733788619011410 4
#c:  373896063919473795766768414356500550359627645161792247466974552929992939550797143531118157838722332342932328692737057695507861833575750816126358516412604754541302882987326934292409233929895763507973644685183741435775731252515835657960721249606024440376582263651534719221181765817082231364674352083197767386186963751984313386328855005835942945505267632319672828040850861452795305721477916545035657782037881046752700637729619410267136030205990189797733972829234513282718422715506132632858564001991632884737229575447283231825863605466309147580123505065740185726296041589848371307413921259668536578026966750027110853831 9
```

由于gift较大，直接遍历
得到p,q后，变形c=me %n=>c=(m2)e/2 %n(因为e和φ(n)不互质）
得到m2 后尝试直接开平方看是否默认最小值为解

```
gift=213549265377668621255332956056096728530330893682588735591191691745477219796068224014982113817721683358650909096989241977595840608799405458502289416595076842774154573624791841025580489452208572064295257963841848380024336831270256645819670850854363505135099957278718823624327563160987525361701566441403205882291946944328445340306407623276502424843554332659741885175158630851454012457130915278755971295020935782557689613227804511217791026601974101399510657948486876825108445333841711548351513286959471216205236208341416395468130625913705758103665744189742843257592401895096114182255425136926224836889997733788619011410 4
import gmpy2
e=54722
c=373896063919473795766768414356500550359627645161792247466974552929992939550797143531118157838722332342932328692737057695507861833575750816126358516412604754541302882987326934292409233929895763507973644685183741435775731252515835657960721249606024440376582263651534719221181765817082231364674352083197767386186963751984313386328855005835942945505267632319672828040850861452795305721477916545035657782037881046752700637729619410267136030205990189797733972829234513282718422715506132632858564001991632884737229575447283231825863605466309147580123505065740185726296041589848371307413921259668536578026966750027110853831 9
n=170839412302134897004266364844877382824264714946070988472953353396381775836854579211985691054177346686920727277591393582076672487039524366801831533276061474219323658899833472820464391561766857651436206371073478704019469465016205316655736680683490804108079965822975058899462050528790020289361253153122564705836229136463197791255596912709160645886849973824514127474327229669195134137099873530383754771783851254535671119652597214849971567993556176421315690958103040771310535884830572443407427518049354940876873634169213140415470931185657676096670338595831252753220776175767832478537185161667438582652911353538952399811 21
for i in range(100):
    phi=gift*i
    pq=n+1-phi
    p_q=gmpy2.iroot(pq**2-4*n,2)[0]
    if gmpy2.is_prime((pq+p_q)//2):
        p=(pq+p_q)//2
        q=pq-p
        break
new_e=e//2
new_d=gmpy2.invert(new_e,(p-1)*(q-1))
m_2=gmpy2.powmod(c,new_d,n)
m=gmpy2.iroot(m_2,2)[0]
import binascii
print(binascii.unhexlify(hex(m)[2:]))
```

运行得到 `b'NPUCTF{diff1cult_rsa_1s_e@sy}'`

116.[AFCTF2018]花开藏宝地

查看题目 一个zip压缩包一个txt

```
第80804238007977405688648566160504278593148666302626415149704905628622876270862865768337953835725801963142685182
51081293807211599635578239631830392702070562312065201408003280942118040098424206159252073371024348394723096263194
45045134540159517488288781666622635328316972979183761952842010806304748313326215619695085380586052550443025074501
971925005072999275628549710915357400946408857号藏宝图

我把我的宝藏都藏在了那里！
那个神秘的地方！
于是我把藏宝图分成了5份，交给五位贤者让他们帮我妥善保管，并且只要搜集3份就可以获得宝藏的地址。

第一位贤者将藏宝图放进时空门中说道：
“那么口令就是我的生日吧，那可是个好数字呢。”

第二位贤者将藏宝图放进宝箱，【小】声念着自己的名字锁上了宝箱。

第三位贤者将藏宝图施上咒语丢进大海：“只要【大】声喊出那句咒语就可以把水驱逐！”

第四位贤者找了个破锁锁上了宝箱，狡黠地笑着：“谁知道它是坏的呢？”

第五位贤者给藏宝图裹上了隐身衣，放入了一个匣子里

据说，只有拥有【智慧】与【力量】就可以获得宝藏了呢！~
你是这样的勇者吗？
```

那么我们就直接爆破第一个zip的密码
过程不写了
• secret1是爆破生日数字：19260817
• secret2爆破英文字母：alice
• secret4伪加密
• secret5 ntfs隐写
得到四组x和m，题目说只要3组就行了
应该是e=3，然后共模攻击。即明文m不变，给出改变的c和n。可是如果是这样，为什么还需要提示一个大数？
而且实际运算以后，并无法得到flag
百度了一下，原来这是标准的门限秘密共享方案(threshold secret sharing scheme),简称门限方案：
http://www.matrix67.com/blog/archives/1261
假设公司董事会共五个人，每个人保存秘钥的一部分。要求三个人在场就可以拿到秘钥打开保险箱，而且保险箱打开后，无法知道到底是哪三个人提供的秘钥。
和这题提示正好对应：
于是我把藏宝图分成了5份，交给五位贤者让他们帮我妥善保管，并且只要搜集3份就可以获得宝藏的地址。
门限加密有多种方案，举个例子，三个平面能确定一个点。而有无数平面通过同一个点。
题目标题为花开，即Asmnth-Bloom方案
利用的就是中国剩余数定理，可以给出很多组n和c，满足m mod n =c，然后根据其中的几组就可以找到解，但要注意crt的有多解，只要是m+kn1n2……nn的都满足。
这题同理：

```
rom Crypto.Util.number import *

z=808042380079774056886485661605042785931486663026264151497049056286228762708628657683379538357258019631426851825108129380721159963557823963183039270207056231206520140800328094211804009842420615925207337102434839472309626319450451345401595174882887816666226353283169729791837619528420108063047483133262156196950853805860525504430250745019719250050729992756285497109153574009464088570
x5 = 230502064382947282343660159791611936696520807970361139469603458689311286041516767875903549263861950740778705012699983268093626403307298415066249636346303539570207577050391796770068203937723627361951969413683246596072925692670365490970847825269581004483964261491917680759091791653759514213188778401968676433284753781006738293752440186858616315727565803777032119737689210471541053061940547213
m5 = 347051559622463144539669950096658163425646411435797691973701513725701575100810446175849424000000075855070430240507732735393411493866540572679626172742301366146501862670272443070970511943485865887494229487420503750457974262802053722093905126235340380261828593508455621667309946361705530667957484731929151875527489478449361198648310684702574627199321092927111137398333029697068474762820822249
x4 = 100459779913520540098065407420629954816677926423356769524759072632219106155849450125185205557491138357760494272691949199099803239098119602186117878931534968435982565071570831032814288620974807498206233914826253433847572703407678712965098320122549759579566316372220959610814573945698083909575005303253205653244238542300266460559790606278310650849881421791081944960157781855164700773081375247
m4 = 347051559622463144539669950096658163425646411435797691973701513725701575100810446175849424000000075855070430240507732735393411493866540572679626172742301366146501862670272443070970511943485865887494229487420503750457974262802053722093905126235340380261828593508455621667309946361705530667957484731929151875527489478449361198648310684702574627199321092927111137398333029697068474762820820091
x2 = 152012681270682340051690627924586232702552460810030322267827401771304907469802591861912921281833890613186317787813611372838066924894691892444503039545946728621696590087591246339208248647926966446848123290344911662916758039134817404720512465817867255277476717353439505243247568126193361558042940352204093381260402400739429050280526212446967632582771424597203000629197487733610187359662268583
m2 = 347051559622463144539669950096658163425646411435797691973701513725701575100810446175849424000000075855070430240507732735393411493866540572679626172742301366146501862670272443070970511943485865887494229487420503750457974262802053722093905126235340380261828593508455621667309946361705530667957484731929151875527489478449361198648310684702574627199321092927111137398333029697068474762820818553
x2 = 152012681270682340051690627924586232702552460810030322267827401771304907469802591861912921281833890613186317787813611372838066924894691892444503039545946728621696590087591246339208248647926966446848123290344911662916758039134817404720512465817867255277476717353439505243247568126193361558042940352204093381260402400739429050280526212446967632582771424597203000629197487733610187359662268583
m2 = 347051559622463144539669950096658163425646411435797691973701513725701575100810446175849424000000075855070430240507732735393411493866540572679626172742301366146501862670272443070970511943485865887494229487420503750457974262802053722093905126235340380261828593508455621667309946361705530667957484731929151875527489478449361198648310684702574627199321092927111137398333029697068474762820818553

x1 = 305345133911395218573790903508296238659147802274031796643017539011648802808763162902335644195648525375518941848430114497150082025133000033835083076541927530829557051524161069423494451667848236452337271862085346869364976989047180532167560796470067549915390773271207901537847213882479997325575278672917648417868759077150999044891099206133296336190476413164240995177077671480352739572539631359
m1 = 347051559622463144539669950096658163425646411435797691973701513725701575100810446175849424000000075855070430240507732735393411493866540572679626172742301366146501862670272443070970511943485865887494229487420503750457974262802053722093905126235340380261828593508455621667309946361705530667957484731929151875527489478449361198648310684702574627199321092927111137398333029697068474762820813413
c=[x1,x2,x4]
n=[m1,m2,m4]
a=crt(c,n)
#通过这一步来去除掉kn1n2n3的干扰，实际上这题题目中给你的就是n1*n2*n3*n4*n5.
r=a%z
print(long_to_bytes(r))
#b"A treasure map is a map that marks the location of buried treasure, a lost mine, a valuable secret or a hidden locale. So flag is afctf{1sn't_s0_int3Resting}."
```

117.[V&N2020 公开赛]Fast

查看题目

```
p = getPrime(1024)
q = getPrime(1024)
N = p * q
g, r1, r2 = [getRandomRange(1, N) for _ in range(3)]
g1 = pow(g, r1 * (p-1), N)
g2 = pow(g, r2 * (q-1), N)
def encrypt(m):
    s1, s2 = [getRandomRange(1, N) for _ in range(2)]
    c1 = (m * pow(g1, s1, N)) % N
    c2 = (m * pow(g2, s2, N)) % N
    return (c1, c2)
def decrypt(c1, c2):
    xp = c1 % p
    xq = c2 % q
    # Chinese Remainder Theorem
    m = (xp*inverse(q, p)*q + xq*inverse(p, q)*p) % N
    return m
```

注意到后面给了一个解密方程，所以只需要求出p、q，放进去就可以解密。
考虑encrypt中给了c1/c2，那按逻辑来说，就是通过g1/g2的值，求p/q。带入解密，得到flag:

$$g^{r1(p-1)} \equiv g1 \bmod N$$
$$g1 = kN + g^{r1(p-1)}$$
两边都 $\bmod p$
$$g1 \bmod p = g^{r1(p-1)} \bmod p$$
费马小定理有：$a^{p-1} \equiv 1 \bmod p$
$$故 g1 \bmod p = g^{r1(p-1)} \bmod p = \left(g^{(p-1)} \bmod p\right)^{r1} \bmod p = 1^{r1} \bmod p = 1$$
$$g1 \bmod p = 1$$
同理$g2 \bmod q = 1$

所以

```
g1- 1 = x*p
gcd(n,g1-1) == p
g2 - 1 = y*q
gcd(n,g2-1) == q
#事实上好像gcd(n,g1-1)并不一定等于p，有可能是k*p。所以先求出来，至少公因数会是n的因子
#本题正好求出是p和q
```

所以通过最大公约数可以求得pq。
再看encrypt:

```
def encrypt(m):
    s1, s2 = [getRandomRange(1, N) for _ in range(2)]
    c1 = (m * pow(g1, s1, N)) % N
    c2 = (m * pow(g2, s2, N)) % N
    return (c1, c2)
```

直接套用decrypt可以得到flag:

```
from Crypto.Util.number import *
p=gcd((g1-1),N)
q=gcd((g2-1),N)
phi=(p-1)*(q-1)
xp = c1 % p
xq = c2 % q
m = (xp*inverse_mod(q, p)*q + xq*inverse_mod(p, q)*p) % N
long_to_bytes(m)
#b'flag{1CE9514E-12AF-49BE-B002-6A3D7E6078FA}\x00\x0bob\xf8
```

118.[BJDCTF2020]伏羲六十四卦

查看题目

```
from secret import flag

def encrpyt5():
    enc=''
    for i in flag:
        enc+=chr((a*(ord(i)-97)+b)%26+97)
    return(enc)

def encrypt4():
    temp=''
    offset=5
    for i in range(len(enc)):
        temp+=chr(ord(enc[i])-offset-i)
    return(temp)
```

```
这是什么，怎么看起来像是再算64卦！！！

密文:升随临损巽睽颐萃小过讼艮颐小过震蛊屯未济中孚艮困恒晋升损蛊萃蛊未济巽解艮贲未济观豫损蛊晋噬嗑晋旅解大畜困未济随蒙升解睽未济井困未济旅萃未济震蒙未济师涣归妹大有

嗯？为什么还有个b呢？
b=7


flag：请按照格式BJD{}
```

脚本如下

```
def decrypt4(enc):
    temp=''
    offset=5
    for i in range(len(enc)):
        temp+=chr(ord(enc[i])+offset+i)
    return temp
def decrypt5(flag):
    for a in range(1,200):
        enc = ''
        for i in flag:
            for k in range(200):
                if (ord(i) - 97 - 7+26*k)%a==0:
                    enc+= chr((ord(i) - 97 - 7 + 26 * k) // a + 97)
                    break
        print(enc)

s='升随临损巽睽颐萃小过讼艮颐小过震蛊屯未济中孚艮困恒晋升损蛊萃蛊未济巽解艮贲未济观豫损蛊晋噬嗑晋旅解大畜困未济随蒙升解睽未济井困未济旅萃未济震蒙未济师涣归妹大有'
dic={'坤': '000000', '剥': '000001', '比': '000010', '观': '000011', '豫': '000100', '晋': '000101', '萃': '000110', '否': '000111', '谦': '001000', '艮': '001001', '蹇': '001010', '渐': '001011', '小过': '001100', '旅': '001101', '咸': '001110', '遁': '001111', '师': '010000', '蒙': '010001', '坎': '010010', '涣': '010011', '解': '010100', '未济': '010101', '困': '010110', '讼': '010111', '升': '011000', '蛊': '011001', '井': '011010', '巽': '011011', '恒': '011100', '鼎': '011101', '大过': '011110', '姤': '011111', '复': '100000', '颐': '100001', '屯': '100010', '益': '100011', '震': '100100', '噬嗑': '100101', '随': '100110', '无妄': '100111', '明夷': '101000', '贲': '101001', '既济': '101010', '家人': '101011', '丰': '101100', '离': '101101', '革': '101110', '同人': '101111', '临': '110000', '损': '110001', '节': '110010', '中孚': '110011', '归妹': '110100', '睽': '110101', '兑': '110110', '履': '110111', '泰': '111000', '大畜': '111001', '需': '111010', '小畜': '111011', '大壮': '111100', '大有': '111101', '夬': '111110', '乾': '111111'}
li=[]
k=0
for i in range(len(s)):
    if k ==1:
        k=0
        continue
    try:
        li.append(dic[s[i]])
    except:
        t=''
        t=t+s[i]+s[i+1]
        li.append(dic[t])
        k=1
ss=''.join(li)
print(ss)
enc=''
for i in range(0,len(ss),8):
    enc+=chr(eval('0b'+ss[i:i+8]))
import base64
print(enc)
x=base64.b64decode(enc).decode()
print(x)
x=decrypt4(x)
x=decrypt5(x)
```

```
bjdcongratulationsongettingtheflag
```

运行得到

flag包裹提交即可

119.[ACTF新生赛2020]crypto-aes

查看题目`91144196586662942563895769614300232343026691029427747065707381728622849079757 b'\x8c-\xcd\xde\xa7\xe9\x7f.b\x8aKs\xf1\xba\xc75\xc4d\x13\x07\xac\xa4&\xd6\x91\xfe\xf3\x14\x10|\xf8p'`

```
from Cryptodome.Cipher import AES
import os
import gmpy2
from flag import FLAG
from Cryptodome.Util.number import *

def main():
    key=os.urandom(2)*16
    iv=os.urandom(16)
    print(bytes_to_long(key)^bytes_to_long(iv))
    aes=AES.new(key,AES.MODE_CBC,iv)
    enc_flag = aes.encrypt(FLAG)
    print(enc_flag)
if __name__=="__main__":
    main()
```

key是两个字节不断重复得到的，因此结合输出的与iv向量的异或值很容易的到key和iv
脚本如下

```
key_iv=91144196586662942563895769614300232343026691029427747065707381728622849079757
flag_encrypt=b'\x8c-\xcd\xde\xa7\xe9\x7f.b\x8aKs\xf1\xba\xc75\xc4d\x13\x07\xac\xa4&\xd6\x91\xfe\xf3\x14\x10|\xf8
p'
#print(hex(key_iv))
key=hex(key_iv)[2:6]*16
iv=key_iv^eval('0x'+key)
import Crypto.Util.number
iv=Crypto.Util.number.long_to_bytes(iv)
key=Crypto.Util.number.long_to_bytes(eval('0x'+key))
import Crypto.Cipher.AES
decrypt=Crypto.Cipher.AES.new(key,Crypto.Cipher.AES.MODE_CBC,iv)
print(decrypt.decrypt(flag_encrypt))
```

运行得到`b'actf{W0W_y0u_can_so1v3_AES_now!}'`

120.[NCTF2019]Sore

查看题目

```
91144196586662942563895769614300232343026691029427747065707381728622849079757
b'\x8c-\xcd\xde\xa7\xe9\x7f.b\x8aKs\xf1\xba\xc75\xc4d\x13\x07\xac\xa4&\xd6\x91\xfe\xf3\x14\x10|\xf8p'
```

```
from Cryptodome.Cipher import AES
import os
import gmpy2
from flag import FLAG
from Cryptodome.Util.number import *

def main():
    key=os.urandom(2)*16
    iv=os.urandom(16)
    print(bytes_to_long(key)^bytes_to_long(iv))
    aes=AES.new(key,AES.MODE_CBC,iv)
    enc_flag = aes.encrypt(FLAG)
    print(enc_flag)
if __name__=="__main__":
    main()
```

加密脚本看起来很简单,就是一个普通的移位替换加密(有点类型维吉尼亚密码),但是题目的难点是没有key(key就是要求的flag).
在没有key的情况下要破解密文要经过3个阶段(通过概率分析明文)

Kasiski 实验
重合指数攻击
字母频率分析
其中Kasiski实验通过查找相同的子串(3个字符以上)来猜测可能的key的长度.
比如两个相同的子串的距离相差8位,那么key的长度应该为8的因子,这样当我们有足够多的数据时就可以基本确定key的长度了.
重合指数攻击
重合指数CI

```
key_iv=91144196586662942563895769614300232343026691029427747065707381728622849079757
flag_encrypt=b'\x8c-\xcd\xde\xa7\xe9\x7f.b\x8aKs\xf1\xba\xc75\xc4d\x13\x07\xac\xa4&\xd6\x91\xfe\xf3\x14\x10|\xf8
p'
#print(hex(key_iv))
#key_iv= 0xc981c981c981c981c981c981c981c9814eed98e380b1356763849930c850b9cd
key=hex(key_iv)[2:6]*16
#print(key)
#key=c981c981c981c981c981c981c981c981c981c981c981c981c981c981c981c981
iv=key_iv^eval('0x'+key)
#print(iv)
#iv=180008198153874854856760470039650463820
import Crypto.Util.number
iv=Crypto.Util.number.long_to_bytes(iv)
#print(iv)
#iv=\x87lQbI0\xfc\xe6\xaa\x05P\xb1\x01\xd1pL
key=Crypto.Util.number.long_to_bytes(eval('0x'+key))
#print(key)
#key=\xc9\x81\xc9\x81\xc9\x81\xc9\x81\xc9\x81\xc9\x81\xc9\x81\xc9\x81\xc9\x81\xc9\x81\xc9\x81\xc9\x81\xc9\x81\xc
9\x81\xc9\x81\xc9\x81'
import Crypto.Cipher.AES
decrypt=Crypto.Cipher.AES.new(key,Crypto.Cipher.AES.MODE_CBC,iv)
print(decrypt.decrypt(flag_encrypt))
#运行得到b'actf{W0W_y0u_can_so1v3_AES_now!}‘
```