[buuctf] crypto全解——121-146（不建议直接抄flag）

原创

121.[INSHack2017]rsa16m

查看题目
给了nec都很大
所以当m^e 严重小于n的时候。c很可能就是
m^e
所以对c开e次方就能得到m

```
import gmpy2
from Crypto.Util.number import *
#读取
data = open('rsa_16m.txt', 'r').read().split('\n')
#print(data)
m = gmpy2.iroot(int(data[1][4:], 16), int(data[2][4:], 16))[0]
#print(m)
#30156321943599743278455918182580886589695285093075236154009535613
print(long_to_bytes(m))

#b'INSA{(I)NSA_W0uld_bE_pr0uD}'
```

122.[SUCTF2019]MT

```python
from Crypto.Random import random
from Crypto.Util import number
from flag import flag

def convert(m):
    m = m ^ m >> 13
    m = m ^ m << 9 & 2029229568
    m = m ^ m << 17 & 2245263360
    m = m ^ m >> 19
    return m

def transform(message):
    assert len(message) % 4 == 0
    new_message = ''
    for i in range(len(message) / 4):
        block = message[i * 4 : i * 4 +4]
        block = number.bytes_to_long(block)
        block = convert(block)
        block = number.long_to_bytes(block, 4)
        new_message += block
    return new_message

transformed_flag = transform(flag[5:-1].decode('hex')).encode('hex')
print 'transformed_flag:', transformed_flag
# transformed_flag: 641460a9e3953b1aaa21f3a2
```

脚本

```python
#python2
from Crypto.Random import random
from Crypto.Util import number

def convert(m):
    m = m ^ m >> 13
    m = m ^ m << 9 & 2029229568
    m = m ^ m << 17 & 2245263360
    m = m ^ m >> 19
    return m

def transform(message):
    assert len(message) % 4 == 0
    new_message = ''
    for i in range(len(message) / 4):
        block = message[i * 4 : i * 4 +4]
        block = number.bytes_to_long(block)
        block = convert(block)
        block = number.long_to_bytes(block, 4)
        new_message += block
    return new_message
def circle(m):
    t=m
    while True:
        x=t
        t=transform(t)
        if t==m:
            return x
transformed_flag='641460a9e3953b1aaa21f3a2'
flag = circle(transformed_flag.decode('hex')).encode('hex')
print('transformed_flag:', flag)
#84b45f89af22ce7e67275bdc
```

123-[De1CTF2019]xorz

查看题目

```python
from itertools import *
from data import flag,plain

key=flag.strip("de1ctf{").strip("}")
assert(len(key)<38)
salt="WeAreDe1taTeam"
ki=cycle(key)
si=cycle(salt)
cipher = ''.join([hex(ord(p) ^ ord(next(ki)) ^ ord(next(si)))[2:].zfill(2) for p in plain])
print cipher
# output:
# 49380d773440222d1b421b3060380c3f403c3844791b202651306721135b6229294a3c3222357e766b2f15561b35305e3c3b670e49382c
295c6c170553577d3a2b791470406318315d753f03637f2b614a4f2e1c4f21027e227a4122757b446037786a7b0e37635024246d60136f78
02543e4d36265c3e035a725c6322700d626b345d1d6464283a016f35714d434124281b607d315f66212d671428026a4f4f79657e34153f34
67097e4e135f187a21767f02125b375563517a3742597b6c394e78742c4a725069606576777c314429264f6e330d7530453f22537f5e3034
560d22146831456b1b72725f30676d0d5c71617d48753e26667e2f7a334c731c22630a242c7140457a42324629064441036c7e646208630e
745531436b7c51743a36674c4f352a5575407b767a5c747176016c0676386e403a2b42356a727a04662b4446375f36265f3f124b724c6e34
654470627764102506342001662922 5b43432428036f29341a2338627c47650b264c477c653a67043e6766152a485c7f3361726478065653
7e5468143f305f4537722352303c3d4379043d69797e6f3922527b24536e310d653d4c33696c635474637d0326516f745e610d7733403066
21105a7361654e3e392970687c2e335f3015677d4b3a724a4659767c2f5b7c16055a126820306c14315d6b59224a27311f747f336f4d5974
321a22507b22705a226c6d446a37375761423a2b5c29247163046d7e4703224437750830075172712632 6f117f7a38670c2b23203d4f2704
6a5c5e1532601126292f577776606f0c6d0126474b2a73737a41316362146e581d7c1228717664091c
```

脚本如下

```
import string
from binascii import unhexlify, hexlify
from itertools import *

def bxor(a, b):     # xor two byte strings of different lengths
    if len(a) > len(b):
        return bytes([x ^ y for x, y in zip(a[:len(b)], b)])
    else:
        return bytes([x ^ y for x, y in zip(a, b[:len(a)])])

def hamming_distance(b1, b2):
    differing_bits = 0
    for byte in bxor(b1, b2):
        differing_bits += bin(byte).count("1")
    return differing_bits

def break_single_key_xor(text):
    key = 0
    possible_space = 0
    max_possible = 0
    letters = string.ascii_letters.encode('ascii')
    for a in range(0, len(text)):
        maxpossible = 0
        for b in range(0, len(text)):
            if(a == b):
                continue
            c = text[a] ^ text[b]
            if c not in letters and c != 0:
                continue
            maxpossible += 1
        if maxpossible > max_possible:
            max_possible = maxpossible
            possible_space = a
    key = text[possible_space] ^ 0x20
    return chr(key)

salt = "WeAreDe1taTeam"
si = cycle(salt)
b = unhexlify(b'49380d773440222d1b421b3060380c3f403c3844791b202651306721135b6229294a3c3222357e766b2f15561b35305e
3c3b670e49382c295c6c170553577d3a2b791470406318315d753f03637f2b614a4f2e1c4f21027e227a4122757b446037786a7b0e376350
24246d60136f7802543e4d36265c3e035a725c6322700d626b345d1d6464283a016f35714d434124281b607d315f66212d671428026a4f4f
79657e34153f3467097e4e135f187a21767f02125b375563517a3742597b6c394e78742c4a725069606576777c314429264f6e330d753045
3f22537f5e3034560d22146831456b1b72725f30676d0d5c71617d48753e26667e2f7a334c731c22630a242c7140457a4232462906444103
6c7e646208630e745531436b7c51743a36674c4f352a5575407b767a5c747176016c0676386e403a2b42356a727a04662b4446375f36265f
3f124b724c6e34654470627764102506342001662922 5b43432428036f29341a2338627c47650b264c477c653a67043e6766152a485c7f33
617264780656537e5468143f305f4537722352303c3d4379043d69797e6f3922527b24536e310d653d4c33696c635474637d0326516f745e
610d773340306621105a7361654e3e392970687c2e335f3015677d4b3a724a4659767c2f5b7c16055a126820306c14315d6b59224a27311f
747f336f4d5974321a22507b22705a226c6d446a37375761423a2b5c29247163046d7e4703224437750830075172712632 6f117f7a38670c
2b23203d4f27046a5c5e1532601126292f577776606f0c6d0126474b2a73737a41316362146e581d7c1228717664091c')
plain = ''.join([hex(ord(c) ^ ord(next(si)))[2:].zfill(2) for c in b.decode()])
b = unhexlify(plain)
print(plain)

normalized_distances = []

for KEYSIZE in range(2, 40):
    # 我们取其中前6段计算平局汉明距离
    b1 = b[: KEYSIZE]
```

```
    b2 = b[KEYSIZE: KEYSIZE * 2]
    b3 = b[KEYSIZE * 2: KEYSIZE * 3]
    b4 = b[KEYSIZE * 3: KEYSIZE * 4]
    b5 = b[KEYSIZE * 4: KEYSIZE * 5]
    b6 = b[KEYSIZE * 5: KEYSIZE * 6]

    normalized_distance = float(
        hamming_distance(b1, b2) +
        hamming_distance(b2, b3) +
        hamming_distance(b3, b4) +
        hamming_distance(b4, b5) +
        hamming_distance(b5, b6)
    ) / (KEYSIZE * 5)
    normalized_distances.append(
        (KEYSIZE, normalized_distance)
    )
normalized_distances = sorted(normalized_distances, key=lambda x: x[1])

for KEYSIZE, _ in normalized_distances[:5]:
    block_bytes = [[] for _ in range(KEYSIZE)]
    for i, byte in enumerate(b):
        block_bytes[i % KEYSIZE].append(byte)
    keys = ''
    try:
        for bbytes in block_bytes:
            keys += break_single_key_xor(bbytes)
        key = bytearray(keys * len(b), "utf-8")
        plaintext = bxor(b, key)
        print("keysize:", KEYSIZE)
        print("key is:", keys, "n")
        s = bytes.decode(plaintext)
        print(s)
    except Exception:
        continue
```

运行得到

```
key is: W3lc0m3tOjo1nu55un1ojOt3m0cl3W r
```

124.[INSHack2019]Yet Another RSA Challenge - Part 1

查看题目

```
719579745653303119025873098043848913976880838286635817351790189702008424828505522253331968992725441130409959387942238566082746772468987336980704680915524591881919460709921709513741059003955050088052599067720107149755856317364317707629467090624585752920523062378696431510814381603360130752588995217840721808871896469275562085215852034302374902524921137398710508865248881286824902780186249148613287250056380811479959269915786545911048030947364841177976623684660771594747297272818410589981294227084173316280447729440036251406684111603371364957690353449585185893322538541593242187738587675489180722498945337715511212885934126635221601469699184812336984707723198731876940991485904637481371763302337637617744175461566445514603405016576604569057507997291470369704260553992902776099599438704680775883984720946337235834374667842758010444010254965664863296455406931885650448386682827401907759661117637294
```

```
0xDCC5A0BD3A1FC0BEB0DA1C2E8CF6B474481B7C12849B76E03C4C946724DB577D2825D6AA193DB559BC9DBABE1DDE8B5E7805E48749EF002F622F7CDBD7853B200E2A027E87E331AFCFD066ED9900F1E5F5E5196A451A6F9E329EB889D773F08E5FBF45AACB818FD186DD74626180294DCC31805A88D1B71DE5BFEF3ED01F12678D906A833A78EDCE9BDAF22BBE45C0BFB7A82AFE42C1C3B8581C83BF43DFE31BFD81527E507686956458905CC9A660604552A060109DC81D01F229A264AB67C6D7168721AB36DE769CEAFB97F238050193EC942078DDF5329A387F46253A4411A9C8BB71F9AEB11AC9623E41C14FCD2739D76E69283E57DDB11FC531B4611EE3
```

```
596380963583874022971492302071822444225514552231574984926542429117396590795270181084030717066220888052607057994262255729890598322976783889090993129161030148064314476199052180347747135088933481343974996843632511300255010825580875930722684714290535684951679115573751200980708359500292172387447570080875531002842462002727646367063816531958020271149645805755077133231395881833164790825731218786554806777097126212126561056170733032553159740167058242065879953688453169613384659653035659118823444582576657499974059388261153064772228570460351169216103620379299362366574826080703907036316546232196313193923841110510170689800892941998845140534954264505413254429240789223724066502818922164419890197058252325607667959185100118251170368909192832882776642565026481260424714348087206462283972676596101498123547647078981435969530082351104111747783346230914935599764345176602456069568419879060577771404946743580809330315332836749661503035076868102720709045692483171306425207758972682717326821412843569770615848397477633761506670219845039890098105484693890695897858251238713238301401843678654564558196040100908796513657968507381392735855990706254646471937809011610992016368630851454275478216664521360246605400986428230407975530880206404171034278692756
```

```
import subprocess
p = subprocess.check_output('openssl prime -generate -bits 2048 -hex')
q = subprocess.check_output('openssl prime -generate -bits 2048 -hex')
flag = int('INSA{REDACTED}'.encode('hex'), 16)

N = int(p,16) * int(q,16)
print N
print '0x'+p.replace('9F','FC')
print pow(flag,65537,N)
```

这道题告诉你了rsa的n的因子其中的p，但是其中的字母有所替换，我们就可以发现，这串字符中的‘FC’可能是’FC‘也可能是’9F’于是话不多说直接爆破即可，下面给出本题脚本

```
import gmpy2
from Crypto.Util.number import *
n=719579745653303119025873098043848913976880838286635817351790189702008424828505522253331968992725441130409959387942238566082746772468987336980704680915524591881919460709921709513741059003955050088052599067720107149755856317364317707629467090624585752920523062378696431510814381603360130752588995217840721808871896469275562085215852034302374902524921137398710508865248881286824902780186249148613287250056380811479959269915786545911048030947364841177976623684660771594747297272818410589981294227084173316280447729440036251406684111603371364957690353449585185893322538541593242187738587675489180722498945337715511212885934126635221601469699184812336984707723198731876940991485904637481371763302337637617744175461566445514603405016576604569057507997291470369704260553992902776099599438704680775883984720946337235834374667842758010444010254965664863296455406931885650448386682827401907759661117637294
```

```
838753325610213809162253020362015045242003388829769019579522792182295457962911430276020610658073659629786668639126004851910536565721128484604554703970965744790413684836096724064390486888113608024265771815004188203124405817878645103282802994701531113849607969243815078720289912255827700390198089699808626116357304202660642601149742427766381
#p="DCC5A0BD3A1"+a+"0BEB0DA1C2E8CF6B474481B7C12849B76E03C4C946724DB577D2825D6AA193DB559BC9DBABE1DDE8B5E7805E48749EF002F622F7CDBD7853B200E2A027E87E331A"+b+"FD066ED9900F1E5F5E5196A451A6F9E329EB889D773F08E5FBF45AACB818FD186DD74626180294DCC31805A88D1B71DE5BFEF3ED01F12678D906A833A78EDCE9BDAF22BBE45C0BFB7A82AFE42C1C3B8581C83BF43DFE31BFD81527E507686956458905CC9A660604552A060109DC81D01F229A264AB67C6D7168721AB36DE769CEAFB97F238050193EC942078DDF5329A387F46253A4411A9C8BB71F9AEB11AC9623E41C14"+c+"D2739D76E69283E57DDB11"+d+"531B4611EE3"
cipher=5963809635838740229714923020718224442255145522315749849265424291173965907952701810840307170662208880526070579942622557298905983229767838890909931291610301480643144761990521803477471350889334813439749968436325113002550108255808759307226847142905356849516791155737512009807083595002921723874475700808755310028424620027276463670638165319580202711496458057550771332313958818331647908257312187865548067770971262121265610561707330325531597401670582420658799536884531696133846596530356591188234445825766574999740593882611530647722285704603511692161036203792993623665748260807039070363165462321963131939238411105101706898008929419988451405349542645054132544292407892237240665028189221644198901970582523256076679591851001182511703689091928328827766425650264812604247143480872064622839726765961014981235476470789814359695300823511041117477833462309149355997643451766024560695684198790605777714049467435808093303153328367496615030350768681027207090456924831713064252077589726827173268214128435697706158483974776337615066702198450398900981054846938906958978582512387132383014018436786545645581960401009087965136579685073813927358559907062546464719378090116109920163686308514542754782166645213602466054009864282304079755308802064041710342786927 56
plain=['9F','FC']
for a in plain:
 for b in plain:
  for c in plain:
   for d in plain:
    p="DCC5A0BD3A1"+a+"0BEB0DA1C2E8CF6B474481B7C12849B76E03C4C946724DB577D2825D6AA193DB559BC9DBABE1DDE8B5E7805E48749EF002F622F7CDBD7853B200E2A027E87E331A"+b+"FD066ED9900F1E5F5E5196A451A6F9E329EB889D773F08E5FBF45AACB818FD186DD74626180294DCC31805A88D1B71DE5BFEF3ED01F12678D906A833A78EDCE9BDAF22BBE45C0BFB7A82AFE42C1C3B8581C83BF43DFE31BFD81527E507686956458905CC9A660604552A060109DC81D01F229A264AB67C6D7168721AB36DE769CEAFB97F238050193EC942078DDF5329A387F46253A4411A9C8BB71F9AEB11AC9623E41C14"+c+"D2739D76E69283E57DDB11"+d+"531B4611EE3"
    p1=int(p,16)
    if(n%p1==0):
     print p1
     #27869881035956015184979178092922248885674897320108269064145135676677416930908750101386898785101159450077433625380803555071301130739332256486285289470097290409044426739584302074834857801721989648648799253740641480496433764509396039330395579654527851232078667173592401475356727873045602595552393666889257027478385213547302885118341490346766830846876201911076530008127691612594913799272782226366932754058372641521481522494577124999360890113778202218378165756595787931498460866236502220175258385407478826827807650036729385244897815805427164434537088709092238894902485613707990645011133078730017425033369999448757627854563
e=65537
p=27869881035956015184979178092922248885674897320108269064145135676677416930908750101386898785101159450077433625380803555071301130739332256486285289470097290409044426739584302074834857801721989648648799253740641480496433764509396039330395579654527851232078667173592401475356727873045602595552393666889257027478385213547302885118341490346766830846876201911076530008127691612594913799272782226366932754058372641521481522494577124999360890113778202218378165756595787931498460866236502220175258385407478826827807650036729385244897815805427164434537088709092238902485613707990645011133078730017425033369999448757627854563
q=n//p
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
m=pow(cipher,d,n)
print long_to_bytes(m)
#INSA{I_w1ll_us3_OTp_n3xT_T1M3}
```

125.[GUET-CTF2019]NO SOS

查看题目`..-.-.-.-……-.-..-…-..-…-.-.-….-..-.-.-.-.-.-.-.-.--`
像不像摩斯密码，NO SOS说明不是摩斯密码，.-转成二进制
用培根密码解密即可
flag{guetkkp}

126.[UTCTF2020]OTP

传说中的读题就行

- **(Optional) Tools Required / Used:**

Steps

Step 1

From the prompt and problem title, you should hopefully figure out that this problem exploiting key reuse in a One Time Pad scheme. As the name of the scheme suggests, should only be used once or else you can recover the key.

Step 2

You can then use an online tool like this (or any other crib dragging tool) to guess for w the plaintexts. Since the prompt mentions that they are arguing about CTF categories, eventually guess that these are the two plaintext messages:

```
THE BEST CTF CATEGORY IS CRYPTOGRAPHY!
NO THE BEST ONE IS BINARY EXPLOITATION
```

Step 3

From there, you can get the key (which is the flag) by XORing the plaintext with the cor ciphertext (you might have to do a little trial and error here). Then you can get the key `utflag{tw0_tim3_p4ds}`.

127.[QCTF2018]Xman-RSA

查看题目

encryption.encrypted - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
gqhb jbkl2 pbkhqw pt_kqpbd
gqhb ht pbkhqw zqreahb
pbkhqw urtd64

adg ulwdt_wh_ezb(u):
        qdwzqe pew(u.dexhad('mdi'), 16)

adg ezb_wh_ulwdt(e):
        u = mdi(e)[2:-1]
        u = '0' + u pg yde(u)%2 == 1 dytd u
        qdwzqe u.adxhad('mdi')

adg jdw_r_kqpbd(y):
        qreahb_tdda = zqreahb(y)

        ezb = ulwdt_wh_ezb(qreahb_tdda)

        fmpyd Tqzd:
                pg pt kqpbd(ezb):
```


文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
2639c28e3609a4a8c953cca9c326e8e062756305ae8aee6efcd346458aade3ee8c2106ab9dfe5f4708
04f366af738aa493fd2dc26cb249a922e121287f3eddec0ed8dea89747dc57aed7cd2089d75c23a69b
f601f490a64f73f6a583081ae3a7ed52238c13a95d3322065adba9053ee5b12f1de1873dbad9fbf4a5
0a2f58088df0fddfe2ed8ca1118c81268c8c0fd5572494276f4e48b5eb424f116e6f5e9d66da1b6b3a
8f102539b690c1636e82906a46f3c5434d5b04ed7938861f8d453908970eccef07bf13f723d6fdd26a
61be8b9462d0ddfbedc91886df194ea022e56c1780aa6c76b9f1c7d5ea743dc75cec3c805324e90ea5
77fa396a1effdafa3090
42ff1157363d9cd10da64eb4382b6457ebb740dbef40ade9b24a174d0145adaa0115d86aa2fc2a4125
7f2b62486eaebb655925dac78dd8d13ab405aef5b8b8f9830094c712193500db49fb801e1368c73f88
f6d8533c99c8e7259f8b9d1c926c47215ed327114f235ba8c873af7a0052aa2d32c52880db55c5615e
5a1793b690c37efdd5e503f717bb8de716303e4d6c4116f62d81be852c5d36ef282a958d8c82cf3b45
8dcc8191dcc7b490f227d1562b1d57fbcf7bf4b78a5d90cd385fd79c8ca4688e7d62b3204aeaf9692b
a4d4e44875eaa63642775846434f9ce51d138ca702d907849823b1e86896e4ea6223f93fae68b026cf
e5fa5a665569a9e3948a
```


n2&n3 - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
PVNHb2BfGAnmxLrbKhgsYXRwWIL9eOj6KOs3IOsIKHCTXTAUtZh3TOr
+RoSIhp03+77AY8P7WETYz2Jzuv5FV/mMODoFrM5fMyQsNt90VynR6J3Jv
+fnPJPsm2hJ1Fqt7EKaVRwCbt6a4BdcRoHJsYN/+eh7k/X+FL5XM7viyvQxyFawQrhSV79FIoX6xfjtGW
+uAeVF7DScRcI49dlwODhFD7SeLqzoYDJPIQS+VSb3YtvrDgdV
+EhuS1bfWvkkXRijlJEpLrgWYmMdfsYX8u/
+YIf5xcBGn3hv1YhQrBCg77AHuUF2w/gJ/ADHFiMcH3ux3nqOsuwnbGSr7jA6Cw==
TmNVbWUhCXR1od3gBpM
+HGMKK/4ErfIKITxomQ/QmNCZIzmmsNyPXQBiMEeUB8ud07IWjQTYGjD6k21xjThHTNDG4z6C2cNNPz73V
IaNTGz0hrh6CmqDowFbyrk+rv53QSkVKPa8EZnFKwGz9B3zXimm1D
+01cov7V/ZDfrHrEjsDkgK4ZIrQxPpZAPI+yqGIRK8soBKhY/PF3/GjbquRYeYKbagpUmW0hLnF4/
+DP33ve/EpaSAPirZXzf8hyatL4/5tAZ0uNq9W6T4GoMG
+N7aS2GeyUA2sLJMHymW4cFK5I5kUvjsIRdXOHTmz5eHxqIV6TmSBQRgovUijINamQ==
```


首先看到encryption.encrypted这个文件感觉就很像是被加密的加密脚本

词频分析后得到大致的结果，但是发现还有一些错误，然后手动修改还原得到原加密脚本。

得到原脚本

```python
from gmpy2 import is_prime
from os import urandom
import base64

def bytes_to_num(b):
    return int(b.encode('hex'), 16)
def num_to_bytes(n):
    b = hex(n)[2:-1]
    b = '0' + b if len(b) % 2 == 1 else b
    return b.decode('hex')
def get_a_prime(l):
    random_Teed = urandom(l)
    num = bytes_to_num(random_Teed)
    while True:
          if is_prime(num):
                   break
          num+= 1
    return num

def encrypt(s, e, n):
    p = bytes_to_num(s)
    p = pow(p, e, n)
    return num_to_bytes(p).encode('hex')
def separate(n):
    p = n % 4
    t = (p * p) % 4
    return t == 1
f = open('flag.txt', 'r')
flag = f.read()
msg1 = ""
msg2 = ""
for i in range(len(flag)):
    if  separate(i):
             msg2 += flag[i]
    else:
           msg1 += flag[i]
p1 = get_a_prime(128)
p2 = get_a_prime(128)
p3 = get_a_prime(128)
n1 = p1 * p2
n2 = p1 * p3
e = 0x1001
c1 = encrypt(msg1, e, n1)
c2 = encrypt(msg2, e, n2)
print(c1)
print(c2)
e1 = 0x1001
e2 = 0x101
p4 = get_a_prime(128)
p5 = get_a_prime(128)
n3 = p4 * p5
c1 = num_to_bytes(pow(n1, e1, n3)).encode('hex')
c2 = num_to_bytes(pow(n1, e2, n3)).encode('hex')
print(c1)
print(c2)
print(base64.b64encode(num_to_bytes(n2)))
print(base64.b64encode(num_to_bytes(n3)))
```

再从另外三个文件里面找出给的条件

```
e1 = 0x1001
e2 = 0x101
e = 0x1001
n2='PVNHb2BfGAnmxLrbKhgsYXRwWIL9eOj6K0s3I0slKHCTXTAUtZh3T0r+RoSlhpO3+77AY8P7WETYz2Jzuv5FV/mMODoFrM5fMyQsNt90VynR
6J3Jv+fnPJPsm2hJ1Fqt7EKaVRwCbt6a4BdcRoHJsYN/+eh7k/X+FL5XM7viyvQxyFawQrhSV79FIoX6xfjtGW+uAeVF7DScRcl49dlwODhFD7Se
LqzoYDJPIQS+VSb3YtvrDgdV+EhuS1bfWvkkXRijlJEpLrgWYmMdfsYX8u/+Ylf5xcBGn3hv1YhQrBCg77AHuUF2w/gJ/ADHFiMcH3ux3nqOsuwn
bGSr7jA6Cw=='
n3='TmNVbWUhCXR1od3gBpM+HGMKK/4ErfIKITxomQ/QmNCZlzmmsNyPXQBiMEeUB8udO7lWjQTYGjD6k21xjThHTNDG4z6C2cNNPz73VIaNTGz0
hrh6CmqDowFbyrk+rv53QSkVKPa8EZnFKwGz9B3zXimm1D+01cov7V/ZDfrHrEjsDkgK4ZlrQxPpZAPl+yqGlRK8soBKhY/PF3/GjbquRYeYKbag
pUmWOhLnF4/+DP33ve/EpaSAPirZXzf8hyatL4/5tAZ0uNq9W6T4GoMG+N7aS2GeyUA2sLJMHymW4cFK5l5kUvjslRdXOHTmz5eHxqIV6TmSBQRg
ovUijlNamQ=='
c11=0x2639c28e3609a4a8c953cca9c326e8e062756305ae8aee6efcd346458aade3ee8c2106ab9dfe5f470804f366af738aa493fd2dc26c
b249a922e121287f3eddec0ed8dea89747dc57aed7cd2089d75c23a69bf601f490a64f73f6a583081ae3a7ed52238c13a95d3322065adba9
053ee5b12f1de1873dbad9fbf4a50a2f58088df0fddfe2ed8ca1118c81268c8c0fd5572494276f4e48b5eb424f116e6f5e9d66da1b6b3a8f
102539b690c1636e82906a46f3c5434d5b04ed7938861f8d453908970eccef07bf13f723d6fdd26a61be8b9462d0ddfbedc91886df194ea0
22e56c1780aa6c76b9f1c7d5ea743dc75cec3c805324e90ea577fa396a1effdafa3090
c22=0x42ff1157363d9cd10da64eb4382b6457ebb740dbef40ade9b24a174d0145adaa0115d86aa2fc2a41257f2b62486eaebb655925dac7
8dd8d13ab405aef5b8b8f9830094c712193500db49fb801e1368c73f88f6d8533c99c8e7259f8b9d1c926c47215ed327114f235ba8c873af
7a0052aa2d32c52880db55c5615e5a1793b690c37efdd5e503f717bb8de716303e4d6c4116f62d81be852c5d36ef282a958d8c82cf3b458d
cc8191dcc7b490f227d1562b1d57fbcf7bf4b78a5d90cd385fd79c8ca4688e7d62b3204aeaf9692ba4d4e44875eaa63642775846434f9ce5
1d138ca702d907849823b1e86896e4ea6223f93fae68b026cfe5fa5a665569a9e3948a
c1=0x1240198b148089290e375b999569f0d53c32d356b2e95f5acee070f016b3bef243d0b5e46d9ad7aa7dfe2f21bda920d0ac7ce7b1e48
f22b2de410c6f391ce7c4347c65ffc9704ecb3068005e9f35cbbb7b27e0f7a18f4f42ae572d77aaa3ee189418d6a07bab7d93beaa365c983
49d8599eb68d21313795f380f05f5b3dfdc6272635ede1f83d308c0fdb2baf444b9ee138132d0d532c3c7e60efb25b9bf9cb62dba9833aa3
706344229bd6045f0877661a073b6deef2763452d0ad7ab3404ba494b93fd6dfdf4c28e4fe83a72884a99ddf15ca030ace978f2da87b79b4
f504f1d15b5b96c654f6cd5179b72ed5f84d3a16a8f0d5bf6774e7fd98d27bf3c9839
c2=0x129d5d4ab3f9e8017d4e6761702467bbeb1b884b6c4f8ff397d078a8c41186a3d52977fa2307d5b6a0ad01fedfc3ba7b70f776ba379
0a43444fb954e5afd64b1a3abeb6507cf70a5eb44678a886adf81cb4848a35afb4db7cd7818f566c7e6e2911f5ababdbdd2d4ff9825827e5
8d48d5466e021a64599b3e867840c07e29582961f81643df07f678a61a9f9027ebd34094e272dfbdc4619fa0ac60f0189af785df77e7ec78
4e086cf692a7bf7113a7fb8446a65efa8b431c6f72c14bcfa49c9b491fb1d87f2570059e0f13166a85bb555b40549f45f04bc5dbd09d8b85
8a5382be6497d88197ffb863810857563 65bd757ec3cdfa8a77ba1728ec2de596c5ab
```

通过分析代码，第一步先通过共模攻击得到n1,然后通过最大公因数得到p1,顺势分解n1,n2,然后逆过去就可以得到flag
解密脚本

```
import base64
import Crypto.Util.number
import gmpy2

e1 = 0x1001
e2 = 0x101
e = 0x1001
n2='PVNHb2BfGAnmxLrbKhgsYXRwWIL9eOj6K0s3I0slKHCTXTAUtZh3T0r+RoSlhpO3+77AY8P7WETYz2Jzuv5FV/mMODoFrM5fMyQsNt90VynR
6J3Jv+fnPJPsm2hJ1Fqt7EKaVRwCbt6a4BdcRoHJsYN/+eh7k/X+FL5XM7viyvQxyFawQrhSV79FIoX6xfjtGW+uAeVF7DScRcl49dlwODhFD7Se
LqzoYDJPIQS+VSb3YtvrDgdV+EhuS1bfWvkkXRijlJEpLrgWYmMdfsYX8u/+Ylf5xcBGn3hv1YhQrBCg77AHuUF2w/gJ/ADHFiMcH3ux3nqOsuwn
bGSr7jA6Cw=='
n3='TmNVbWUhCXR1od3gBpM+HGMKK/4ErfIKITxomQ/QmNCZlzmmsNyPXQBiMEeUB8udO7lWjQTYGjD6k21xjThHTNDG4z6C2cNNPz73VIaNTGz0
hrh6CmqDowFbyrk+rv53QSkVKPa8EZnFKwGz9B3zXimm1D+01cov7V/ZDfrHrEjsDkgK4ZlrQxPpZAPl+yqGlRK8soBKhY/PF3/GjbquRYeYKbag
pUmWOhLnF4/+DP33ve/EpaSAPirZXzf8hyatL4/5tAZ0uNq9W6T4GoMG+N7aS2GeyUA2sLJMHymW4cFK5l5kUvjslRdXOHTmz5eHxqIV6TmSBQRg
ovUijlNamQ=='
c11=0x2639c28e3609a4a8c953cca9c326e8e062756305ae8aee6efcd346458aade3ee8c2106ab9dfe5f470804f366af738aa493fd2dc26c
b249a922e121287f3eddec0ed8dea89747dc57aed7cd2089d75c23a69bf601f490a64f73f6a583081ae3a7ed52238c13a95d3322065adba9
053ee5b12f1de1873dbad9fbf4a50a2f58088df0fddfe2ed8ca1118c81268c8c0fd5572494276f4e48b5eb424f116e6f5e9d66da1b6b3a8f
102539b690c1636e82906a46f3c5434d5b04ed7938861f8d453908970eccef07bf13f723d6fdd26a61be8b9462d0ddfbedc91886df194ea0
22e56c1780aa6c76b9f1c7d5ea743dc75cec3c805324e90ea577fa396a1effdafa3090
c22=0x42ff1157363d9cd10da64eb4382b6457ebb740dbef40ade9b24a174d0145adaa0115d86aa2fc2a41257f2b62486eaebb655925dac7
```

```
8dd8d13ab405aef5b8b8f9830094c712193500db49fb801e1368c73f88f6d8533c99c8e7259f8b9d1c926c47215ed327114f235ba8c873af
7a0052aa2d32c52880db55c5615e5a1793b690c37efdd5e503f717bb8de716303e4d6c4116f62d81be852c5d36ef282a958d8c82cf3b458d
cc8191dcc7b490f227d1562b1d57fbcf7bf4b78a5d90cd385fd79c8ca4688e7d62b3204aeaf9692ba4d4e44875eaa63642775846434f9ce5
1d138ca702d907849823b1e86896e4ea6223f93fae68b026cfe5fa5a665569a9e3948a
c1=0x1240198b148089290e375b999569f0d53c32d356b2e95f5acee070f016b3bef243d0b5e46d9ad7aa7dfe2f21bda920d0ac7ce7b1e48
f22b2de410c6f391ce7c4347c65ffc9704ecb3068005e9f35cbbb7b27e0f7a18f4f42ae572d77aaa3ee189418d6a07bab7d93beaa365c983
49d8599eb68d21313795f380f05f5b3dfdc6272635ede1f83d308c0fdb2baf444b9ee138132d0d532c3c7e60efb25b9bf9cb62dba9833aa3
706344229bd6045f0877661a073b6deef2763452d0ad7ab3404ba494b93fd6dfdf4c28e4fe83a72884a99ddf15ca030ace978f2da87b79b4
f504f1d15b5b96c654f6cd5179b72ed5f84d3a16a8f0d5bf6774e7fd98d27bf3c9839
c2=0x129d5d4ab3f9e8017d4e6761702467bbeb1b884b6c4f8ff397d078a8c41186a3d52977fa2307d5b6a0ad01fedfc3ba7b70f776ba379
0a43444fb954e5afd64b1a3abeb6507cf70a5eb44678a886adf81cb4848a35afb4db7cd7818f566c7e6e2911f5ababdbdd2d4ff9825827e5
8d48d5466e021a64599b3e867840c07e29582961f81643df07f678a61a9f9027ebd34094e272dfbdc4619fa0ac60f0189af785df77e7ec78
4e086cf692a7bf7113a7fb8446a65efa8b431c6f72c14bcfa49c9b491fb1d87f2570059e0f13166a85bb555b40549f45f04bc5dbd09d8b85
8a5382be6497d88197ffb86381085756365bd757ec3cdfa8a77ba1728ec2de596c5ab

n2=Crypto.Util.number.bytes_to_long(base64.b64decode(n2))
n3=Crypto.Util.number.bytes_to_long(base64.b64decode(n3))

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)#扩展欧几里得算法
def separate(n):
    p = n % 4
    t = (p * p) % 4
    return t == 1
ans=exgcd(e1,e2,0,0)
s1=ans[1]
s2=ans[2]
n1=(gmpy2.powmod(c11,s1,n3)*gmpy2.powmod(c22,s2,n3))%n3#powmod()函数真香,分数取模也可直接算,一开始不知道还去找了很多的
算法知识
p1=gmpy2.gcd(n1,n2)
p2=n1//p1
p3=n2//p1
phi1=(p1-1)*(p2-1)
phi2=(p1-1)*(p3-1)
d1=gmpy2.invert(e,phi1)
d2=gmpy2.invert(e,phi2)
m1=gmpy2.powmod(c1,d1,n1)
```

```
m2=gmpy2.powmod(c2,d2,n2)
m1=Crypto.Util.number.long_to_bytes(m1).decode()
m2=Crypto.Util.number.long_to_bytes(m2).decode()
length=len(m1)+len(m2)
flag=''
temp1=0
temp2=0
for i in range(length):
    if separate(i):
        flag+=m2[temp2]
        temp2=temp2+1
    else:
        flag+=m1[temp1]
        temp1=temp1+1
print(flag)
```

运行得到flag
XMAN{CRYPT0_l5_50_Interestingvim rsa.py}

128.[ACTF新生赛2020]crypto-des

两个文件一个压缩包

```
import struct
import binascii

s=[72065910510177138000000000000000.000000,71863209670811371000000.000000,184896826254127600000000000000000.000000,72723257588050687000000.000000,46746591674697662000000000.000000,190616988374992920000000000000000000000.000000]
a=''
b=''
for i in s:
    i=float(i)
    tmp=struct.pack('<f', i).hex()#小端
    a+=tmp

for j in s:
    j=float(j)
    tmp=struct.pack('>f', j).hex()#大端
    b+=tmp

print (binascii.a2b_hex(a))
print (binascii.a2b_hex(b))
```

运行得到压缩包密码

```
import pyDes
import base64
from FLAG import flag
deskey = "********"
DES = pyDes.des(deskey)
DES.setMode('ECB')
DES.Kn = [
   [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0],
   [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1,
0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0],
   [0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0],
   [1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0,
1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1],
   [0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0,
0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
   [0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0,
0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0],
   [0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0],
   [0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0],
   [1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0],
   [0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0],
   [0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1,
1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1],
   [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1,
0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0],
   [1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0],
   [1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1],
   [1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0,
1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1],
   [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1]
  ]
cipher_list = base64.b64encode(DES.encrypt(flag))
#b'vrkgBqeK7+h7mPyWujP8r5FqH5yyVlqv0CXudqoNHVAVdNO8ML4lM4zgez7weQXo'
```

发现是一个des加密.

但是这个加密的16轮子密钥都告诉了我们,于是可以直接decrypt

```
b'actf{breaking_DES_is_just_a_small_piece_of_cake}'
```

129.[watevrCTF 2019]Swedish RSA

查看题目

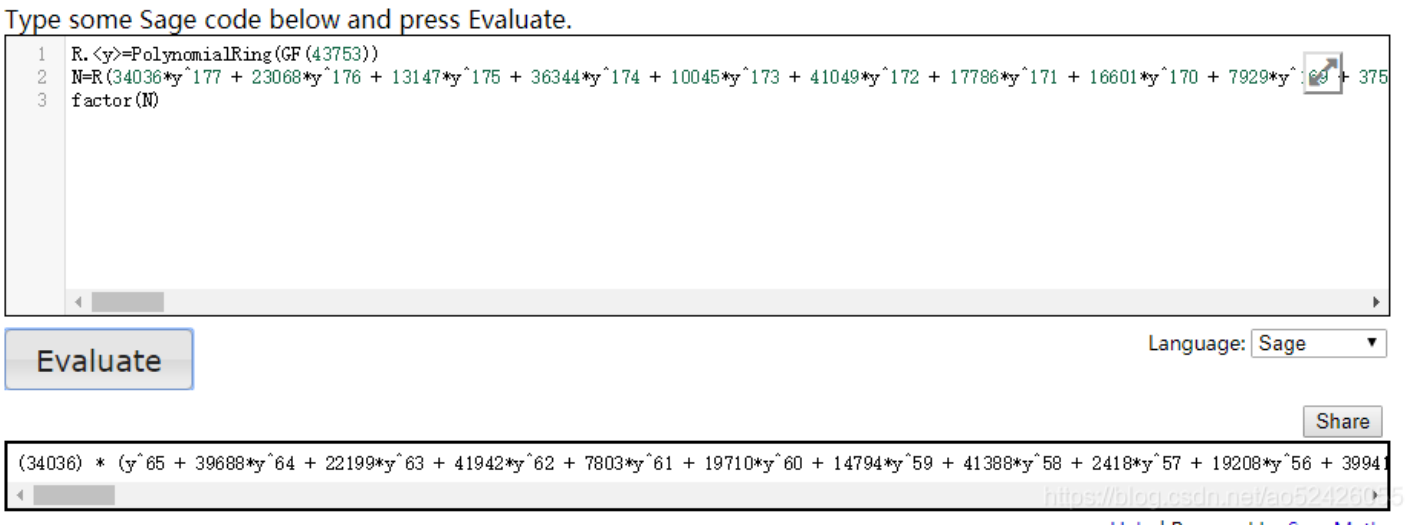
```
flag = bytearray(raw_input())
flag = list(flag)
length = len(flag)
bits = 16

## Prime for Finite Field.
p = random_prime(2^bits-1, False, 2^(bits-1))

file_out = open("downloads/polynomial_rsa.txt", "w")
file_out.write("Prime: " + str(p) + "\n")

## Univariate Polynomial Ring in y over Finite Field of size p
R.<y> = PolynomialRing(GF(p))

## Analogous to the primes in Z
def gen_irreducable_poly(deg):
    while True:
        out = R.random_element(degree=deg)
        if out.is_irreducible():
            return out#生成一个“素数”多项式


## Polynomial "primes"
P = gen_irreducable_poly(ZZ.random_element(length, 2*length))
Q = gen_irreducable_poly(ZZ.random_element(length, 2*length))

## Public exponent key
e = 65537

## Modulus
N = P*Q
file_out.write("Modulus: " + str(N) + "\n")

## Univariate Quotient Polynomial Ring in x over Finite Field of size 659 with modulus N(x)
S.<x> = R.quotient(N)

## Encrypt
m = S(flag)
c = m^e

file_out.write("Ciphertext: " + str(c))
file_out.close()
```

Prime: 43753

Modulus: 34036*y^177 + 23068*y^176 + 13147*y^175 + 36344*y^174 + 10045*y^173 + 41049*y^172 + 17786*y^171 + 16601*y^170 + 7929*y^169 + 37570*y^168 + 990*y^167 + 9622*y^166 + 39273*y^165 + 35284*y^164 + 15632*y^163 + 18850*y^162 + 8800*y^161 + 33148*y^160 + 12147*y^159 + 40487*y^158 + 6407*y^157 + 34111*y^156 + 8446*y^155 + 21908*y^154 + 16812*y^153 + 40624*y^152 + 43506*y^151 + 39116*y^150 + 33011*y^149 + 23914*y^148 + 2210*y^147 + 23196*y^146 + 43359*y^145 + 34455*y^144 + 17684*y^143 + 25262*y^142 + 982*y^141 + 24015*y^140 + 27968*y^139 + 37463*y^138 + 10667*y^137 + 39519*y^136 + 31176*y^135 + 27520*y^134 + 32118*y^133 + 8333*y^132 + 38945*y^131 + 34713*y^130 + 1107*y^129 + 43604*y^128 + 4433*y^127 + 18110*y^126 + 17658*y^125 + 32354*y^124 + 3219*y^123 + 40238*y^122 + 10439*y^121 + 3669*y^120 + 8713*y^119 + 21027*y^118 + 29480*y^117 + 5477*y^116 + 24332*y^115 + 43480*y^114 + 33406*y^113 + 43121*y^112 + 1114*y^111 + 17198*y^110 + 22829*y^109 + 24424*y^108 + 16523*y^107 + 20424*y^106 + 36206*y^105 + 41849*y^104 + 3584*y^103 + 26500*y^102 + 31897*y^101 + 34640*y^100 + 27449*y^99 + 30962*y^98 + 41434*y^97 + 22125*y^96 + 24314*y^95 + 3944*y^94 + 18400*y^93 + 38476*y^92 + 28904*y^91 + 27936*y^90 + 41867*y^89 + 25573*y^88 + 25659*y^87 + 33443*y^86 + 18435*y^85 + 5934*y^84 + 38030*y^83 + 17563*y^82 + 24086*y^81 + 36782*y^80 + 20922*y^79 + 38933*y^78 + 23448*y^77 + 10599*y^76 + 7156*y^75 + 29044*y^74 + 23605*y^73 + 7657*y^72 + 28200*y^71 + 2431*y^70 + 3860*y^69 + 23259*y^68 + 14590*y^67 + 33631*y^66 + 15673*y^65 + 36049*y^64 + 29728*y^63 + 22413*y^62 + 18602*y^61 + 18557*y^60 + 23505*y^59 + 17642*y^58 + 12595*y^57 + 17255*y^56 + 15316*y^55 + 8948*y^54 + 38*y^53 + 40329*y^52 + 9823*y^51 + 5798*y^50 + 6379*y^49 + 8662*y^48 + 34640*y^47 + 38321*y^46 + 18760*y^45 + 13135*y^44 + 15926*y^43 + 34952*y^42 + 28940*y^41 + 13558*y^40 + 42579*y^39 + 38015*y^38 + 33788*y^37 + 12381*y^36 + 195*y^35 + 13709*y^34 + 31500*y^33 + 32994*y^32 + 30486*y^31 + 40414*y^30 + 2578*y^29 + 30525*y^28 + 43067*y^27 + 6195*y^26 + 36288*y^25 + 23236*y^24 + 21493*y^23 + 15808*y^22 + 34500*y^21 + 6390*y^20 + 42994*y^19 + 42151*y^18 + 19248*y^17 + 19291*y^16 + 8124*y^15 + 40161*y^14 + 24726*y^13 + 31874*y^12 + 30272*y^11 + 30761*y^10 + 2296*y^9 + 11017*y^8 + 16559*y^7 + 28949*y^6 + 40499*y^5 + 22377*y^4 + 33628*y^3 + 30598*y^2 + 4386*y + 23814

Ciphertext: 5209*x^176 + 10881*x^175 + 31096*x^174 + 23354*x^173 + 28337*x^172 + 15982*x^171 + 13515*x^170 + 21641*x^169 + 10254*x^168 + 34588*x^167 + 27434*x^166 + 29552*x^165 + 7105*x^164 + 22604*x^163 + 41253*x^162 + 42675*x^161 + 21153*x^160 + 32838*x^159 + 34391*x^158 + 832*x^157 + 720*x^156 + 22883*x^155 + 19236*x^154 + 33772*x^153 + 5020*x^152 + 17943*x^151 + 26967*x^150 + 30847*x^149 + 10306*x^148 + 33966*x^147 + 43255*x^146 + 20342*x^145 + 4474*x^144 + 3490*x^143 + 38033*x^142 + 11224*x^141 + 30565*x^140 + 31967*x^139 + 32382*x^138 + 9759*x^137 + 1030*x^136 + 32122*x^135 + 42614*x^134 + 14280*x^133 + 16533*x^132 + 32676*x^131 + 43070*x^130 + 36009*x^129 + 28497*x^128 + 2940*x^127 + 9747*x^126 + 22758*x^125 + 16615*x^124 + 14086*x^123 + 13038*x^122 + 39603*x^121 + 36260*x^120 + 32502*x^119 + 17619*x^118 + 17700*x^117 + 15083*x^116 + 11311*x^115 + 36496*x^114 + 1300*x^113 + 13601*x^112 + 43425*x^111 + 10376*x^110 + 11551*x^109 + 13684*x^108 + 14955*x^107 + 6661*x^106 + 12674*x^105 + 21534*x^104 + 32132*x^103 + 34135*x^102 + 43684*x^101 + 837*x^100 + 29311*x^99 + 4849*x^98 + 26632*x^97 + 26662*x^96 + 10159*x^95 + 32657*x^94 + 12149*x^93 + 17858*x^92 + 35805*x^91 + 19391*x^90 + 30884*x^89 + 42039*x^88 + 17292*x^87 + 4694*x^86 + 1497*x^85 + 1744*x^84 + 31071*x^83 + 26246*x^82 + 24402*x^81 + 22068*x^80 + 39263*x^79 + 23703*x^78 + 21484*x^77 + 12241*x^76 + 28821*x^75 + 32886*x^74 + 43075*x^73 + 35741*x^72 + 19936*x^71 + 37219*x^70 + 33411*x^69 + 8301*x^68 + 12949*x^67 + 28611*x^66 + 42654*x^65 + 6910*x^64 + 18523*x^63 + 31144*x^62 + 21398*x^61 + 36298*x^60 + 27158*x^59 + 918*x^58 + 38601*x^57 + 4269*x^56 + 5699*x^55 + 36444*x^54 + 34791*x^53 + 37978*x^52 + 32481*x^51 + 8039*x^50 + 11012*x^49 + 11454*x^48 + 30450*x^47 + 1381*x^46 + 32403*x^45 + 8202*x^44 + 8404*x^43 + 37648*x^42 + 43696*x^41 + 34237*x^40 + 36490*x^39 + 41423*x^38 + 35792*x^37 + 36950*x^36 + 31086*x^35 + 38970*x^34 + 12439*x^33 + 7963*x^32 + 16150*x^31 + 11382*x^30 + 3038*x^29 + 20157*x^28 + 23531*x^27 + 32866*x^26 + 5428*x^25 + 21132*x^24 + 13443*x^23 + 28909*x^22 + 42716*x^21 + 6567*x^20 + 24744*x^19 + 8727*x^18 + 14895*x^17 + 28172*x^16 + 30903*x^15 + 26608*x^14 + 27314*x^13 + 42224*x^12 + 42551*x^11 + 37726*x^10 + 11203*x^9 + 36816*x^8 + 5537*x^7 + 20301*x^6 + 17591*x^5 + 41279*x^4 + 7999*x^3 + 33753*x^2 + 34551*x + 9659

输出给了我们三个量，一个是多项式素数最大值，一个是模数多项式，一个是密文多项式。
所以首先分解Modulus



分解N后得到两个多项式。
于是要开始求d
回顾rsa的加密过程，d是e关于φ(n)的唯一逆元，于是转化为求φ(n)，而φ(n)在整数中是指与n互质且小于n的数的个数。于是在多项式中应该是与n没有公共多项式的且不高于其幂级的多项是的个数，这样我们很快就可以确认φ(n)的值为(pow(43753,65)-1)*(pow(43753,112)-1)
接着就可以用以下脚本求出m

```
from sage.all import *
R.<y> = PolynomialRing(GF(43753))
N = R("34036*y^177 + 23068*y^176 + 13147*y^175 + 36344*y^174 + 10045*y^173 + 41049*y^172 + 17786*y^171 + 16601*y
^170 + 7929*y^169 + 37570*y^168 + 990*y^167 + 9622*y^166 + 39273*y^165 + 35284*y^164 + 15632*y^163 + 18850*y^162
 + 8800*y^161 + 33148*y^160 + 12147*y^159 + 40487*y^158 + 6407*y^157 + 34111*y^156 + 8446*y^155 + 21908*y^154 +
16812*y^153 + 40624*y^152 + 43506*y^151 + 39116*y^150 + 33011*y^149 + 23914*y^148 + 2210*y^147 + 23196*y^146 + 4
3359*y^145 + 34455*y^144 + 17684*y^143 + 25262*y^142 + 982*y^141 + 24015*y^140 + 27968*y^139 + 37463*y^138 + 106
67*y^137 + 39519*y^136 + 31176*y^135 + 27520*y^134 + 32118*y^133 + 8333*y^132 + 38945*y^131 + 34713*y^130 + 1107
*y^129 + 43604*y^128 + 4433*y^127 + 18110*y^126 + 17658*y^125 + 32354*y^124 + 3219*y^123 + 40238*y^122 + 10439*y
^121 + 3669*y^120 + 8713*y^119 + 21027*y^118 + 29480*y^117 + 5477*y^116 + 24332*y^115 + 43480*y^114 + 33406*y^11
3 + 43121*y^112 + 1114*y^111 + 17198*y^110 + 22829*y^109 + 24424*y^108 + 16523*y^107 + 20424*y^106 + 36206*y^105
 + 41849*y^104 + 3584*y^103 + 26500*y^102 + 31897*y^101 + 34640*y^100 + 27449*y^99 + 30962*y^98 + 41434*y^97 + 2
2125*y^96 + 24314*y^95 + 3944*y^94 + 18400*y^93 + 38476*y^92 + 28904*y^91 + 27936*y^90 + 41867*y^89 + 25573*y^88
 + 25659*y^87 + 33443*y^86 + 18435*y^85 + 5934*y^84 + 38030*y^83 + 17563*y^82 + 24086*y^81 + 36782*y^80 + 20922*
y^79 + 38933*y^78 + 23448*y^77 + 10599*y^76 + 7156*y^75 + 29044*y^74 + 23605*y^73 + 7657*y^72 + 28200*y^71 + 243
1*y^70 + 3860*y^69 + 23259*y^68 + 14590*y^67 + 33631*y^66 + 15673*y^65 + 36049*y^64 + 29728*y^63 + 22413*y^62 +
18602*y^61 + 18557*y^60 + 23505*y^59 + 17642*y^58 + 12595*y^57 + 17255*y^56 + 15316*y^55 + 8948*y^54 + 38*y^53 +
 40329*y^52 + 9823*y^51 + 5798*y^50 + 6379*y^49 + 8662*y^48 + 34640*y^47 + 38321*y^46 + 18760*y^45 + 13135*y^44
+ 15926*y^43 + 34952*y^42 + 28940*y^41 + 13558*y^40 + 42579*y^39 + 38015*y^38 + 33788*y^37 + 12381*y^36 + 195*y^
35 + 13709*y^34 + 31500*y^33 + 32994*y^32 + 30486*y^31 + 40414*y^30 + 2578*y^29 + 30525*y^28 + 43067*y^27 + 6195
*y^26 + 36288*y^25 + 23236*y^24 + 21493*y^23 + 15808*y^22 + 34500*y^21 + 6390*y^20 + 42994*y^19 + 42151*y^18 + 1
9248*y^17 + 19291*y^16 + 8124*y^15 + 40161*y^14 + 24726*y^13 + 31874*y^12 + 30272*y^11 + 30761*y^10 + 2296*y^9 +
 11017*y^8 + 16559*y^7 + 28949*y^6 + 40499*y^5 + 22377*y^4 + 33628*y^3 + 30598*y^2 + 4386*y + 23814")
C = R("5209*y^176 + 10881*y^175 + 31096*y^174 + 23354*y^173 + 28337*y^172 + 15982*y^171 + 13515*y^170 + 21641*y^
169 + 10254*y^168 + 34588*y^167 + 27434*y^166 + 29552*y^165 + 7105*y^164 + 22604*y^163 + 41253*y^162 + 42675*y^1
61 + 21153*y^160 + 32838*y^159 + 34391*y^158 + 832*y^157 + 720*y^156 + 22883*y^155 + 19236*y^154 + 33772*y^153 +
 5020*y^152 + 17943*y^151 + 26967*y^150 + 30847*y^149 + 10306*y^148 + 33966*y^147 + 43255*y^146 + 20342*y^145 +
4474*y^144 + 3490*y^143 + 38033*y^142 + 11224*y^141 + 30565*y^140 + 31967*y^139 + 32382*y^138 + 9759*y^137 + 103
0*y^136 + 32122*y^135 + 42614*y^134 + 14280*y^133 + 16533*y^132 + 32676*y^131 + 43070*y^130 + 36009*y^129 + 2849
7*y^128 + 2940*y^127 + 9747*y^126 + 22758*y^125 + 16615*y^124 + 14086*y^123 + 13038*y^122 + 39603*y^121 + 36260*
y^120 + 32502*y^119 + 17619*y^118 + 17700*y^117 + 15083*y^116 + 11311*y^115 + 36496*y^114 + 1300*y^113 + 13601*y
^112 + 43425*y^111 + 10376*y^110 + 11551*y^109 + 13684*y^108 + 14955*y^107 + 6661*y^106 + 12674*y^105 + 21534*y^
104 + 32132*y^103 + 34135*y^102 + 43684*y^101 + 837*y^100 + 29311*y^99 + 4849*y^98 + 26632*y^97 + 26662*y^96 + 1
0159*y^95 + 32657*y^94 + 12149*y^93 + 17858*y^92 + 35805*y^91 + 19391*y^90 + 30884*y^89 + 42039*y^88 + 17292*y^8
7 + 4694*y^86 + 1497*y^85 + 1744*y^84 + 31071*y^83 + 26246*y^82 + 24402*y^81 + 22068*y^80 + 39263*y^79 + 23703*y
^78 + 21484*y^77 + 12241*y^76 + 28821*y^75 + 32886*y^74 + 43075*y^73 + 35741*y^72 + 19936*y^71 + 37219*y^70 + 33
411*y^69 + 8301*y^68 + 12949*y^67 + 28611*y^66 + 42654*y^65 + 6910*y^64 + 18523*y^63 + 31144*y^62 + 21398*y^61 +
 36298*y^60 + 27158*y^59 + 918*y^58 + 38601*y^57 + 4269*y^56 + 5699*y^55 + 36444*y^54 + 34791*y^53 + 37978*y^52
+ 32481*y^51 + 8039*y^50 + 11012*y^49 + 11454*y^48 + 30450*y^47 + 1381*y^46 + 32403*y^45 + 8202*y^44 + 8404*y^43
 + 37648*y^42 + 43696*y^41 + 34237*y^40 + 36490*y^39 + 41423*y^38 + 35792*y^37 + 36950*y^36 + 31086*y^35 + 38970
*y^34 + 12439*y^33 + 7963*y^32 + 16150*y^31 + 11382*y^30 + 3038*y^29 + 20157*y^28 + 23531*y^27 + 32866*y^26 + 54
28*y^25 + 21132*y^24 + 13443*y^23 + 28909*y^22 + 42716*y^21 + 6567*y^20 + 24744*y^19 + 8727*y^18 + 14895*y^17 +
28172*y^16 + 30903*y^15 + 26608*y^14 + 27314*y^13 + 42224*y^12 + 42551*y^11 + 37726*y^10 + 11203*y^9 + 36816*y^8
 + 5537*y^7 + 20301*y^6 + 17591*y^5 + 41279*y^4 + 7999*y^3 + 33753*y^2 + 34551*y + 9659")
e = 65537
phi = (43753^65-1)*(43753^112-1)
d = inverse_mod(e, phi)
ans = R("1")
temp= C
while(True):
 if(d % 2 == 1):
  ans = (ans * temp) % N
  d = d - 1
 d = d / 2
 temp = (temp * temp) % N
 if(d == 0):
  break
#快速幂
print (ans)
```

Type some Sage code below and press Evaluate.

```
    if(d % 2 == 1):
        ans = (ans * temp) % N
        d = d - 1
    d = d / 2
    temp = (temp * temp) % N
    if(d == 0):
        break
#快速幂
print (ans)
```

Evaluate

Language: Sage

Share

```
125*y^62 + 111*y^61 + 114*y^60 + 117*y^59 + 53*y^58 + 51*y^57 + 51*y^56 + 100*y^55 + 106*y^54 + 110*y^53 + 102*y^52 + 106*y^51 + 100*y^50
```


130.[NPUCTF2020]认清形势，建立信心

查看题目

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

p = getPrime(25)
e = # Hidden
q = getPrime(25)
n = p * q
m = bytes_to_long(flag.strip(b"npuctf{").strip(b"}"))

c = pow(m, e, n)
print(c)
print(pow(2, e, n))
print(pow(4, e, n))
print(pow(8, e, n))

'''
1691699126541 78
1285091601792 02
5188187424143 40
3585530020644 50
'''
```

首先，这里展开一个公式：
((a mod x) ^ b) mod x = (a ^ b) mod x
作为小白，接下来是推导证明:
设 a = kx + d
(a mod x) ^ b = d ^ b
a ^ b = (kx + d) ^ b,此处二项式展开得知共b+1项 前b项都有x这个因数 最后一个为d ^ b.
则根据题目，
2 ^ e mod n = a
4 ^ e mod n = 2 ^ 2e mod n = (2 ^ e mod n) ^ 2 mod n = a ^ 2 mod n = b
a ^ 2 - b = kn(k=1,k=2…) 同理，a ^ 3 - c = kn(k=1,k=2…). n为两式的公因数，由此根据gcd（ a ^ 2 - b，a ^ 3 - c）求出n

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

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

a = 128509160179202
b = 518818742414340
c = 358553002064450
n = GCD(a**2-b, a**3-c)
print(n)
```

```
[NPUCTF2020]认清形势，建立信心
D:\python38\python.exe D:/pycharm/venv/mima/RSA/[NPUCTF2020]认清形势，建立信心.py
1054494004042394
```


把N分解

1054494004042394

Result:

status (?)	digits	number
FF	16 (show)	1054494004042394<16> = 2 · 18195301 · 28977097

把2去掉 这就是pq了

然后就是求e

```cpp
#include<iostream>
#include<math.h>
#define LONG long long
using namespace std;

int main(){
    LONG e=1,c=2,n=527247002021197,a=128509160179202;
    while(e<100000000000){
        e++;
        c=c*2%n;
        if(c==a){
            cout<<e<<endl;
            return 0;
        }
    }
    return 0;
}
```

run (ctrl+x) 输入 Copy 分享当前代码 意见反馈

文本方式显示 html方式显示

808723997

duisanfenli.py
Ersa.py
giftrsa.py
kaisa.py
nc.py
output.py
pqr.py
PSA3.py
qingshu.py
RSA.py
RSA0.py
rsa1(xiao).py
rsa1(xiao2).py
RSA1.py
RSA2.py
RSA4.py
RSA4 (2).py
RSA5(1).py
RSA5(2).PY
rsagongmogongji.py

```python
c = pow(m, e, n)
print(c)
print(pow(2, e, n))
print(pow(4, e, n))
print(pow(8, e, n))

'''
169169912654178
128509160179202
518818742414340
358553002064450
'''
```

```
Run: [NPUCTF2020]认清形势，建立信心
D:\python38\python.exe D:/pycharm/venv/mima/RSA/[NPUCTF202
1054494004042394

进程完成，退出码 0
```


这样epqc就都有了就正常写入就行了

```
'''from Crypto.Util.number import *
from gmpy2 import *
from secret import flag

p = getPrime(25)
e = # Hidden
q = getPrime(25)
n = p * q
m = bytes_to_long(flag.strip(b"npuctf{").strip(b"}"))

c = pow(m, e, n)
print(c)
print(pow(2, e, n))
print(pow(4, e, n))
print(pow(8, e, n))

169169912654178
128509160179202
518818742414340
358553002064450
'''
'''from Crypto.Util.number import *

a = 128509160179202
b = 518818742414340
c = 358553002064450
n = GCD(a**2-b, a**3-c)
print(n)
'''
from Crypto.Util.number import *
from gmpy2 import *

e = 808723997
p = 18195301
q = 28977097
c = 169169912654178
m = pow(c,invert(e,(p-1)*(q-1)),p*q)
flag = long_to_bytes(m)
print (flag)
```

当然用以前写的脚本也是可以的

```
print(d)

m = pow(c,d,n)

'''pow(c,d,n)
c的d次方，在对结果进行取模，其结果等效于pow(c,d)%n
'''
print(m)
print(hex(m))#转换十六进制
print(bytes.fromhex(hex(m)[2:]))#ascll码
```

```
rsa1(xiao2)
D:\python38\python.exe D:/pycharm/venv/mima/RSA/rsa1(xiao2).py
315420901534133
219919251745
0x3334357921
b'345y!'

进程完成，退出码 0
```


131.[AFCTF2018]MagicNum

查看题目

where_is_flag.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
720659105101771380000000000000000.000000
71863209670811371000000.000000
184896826254127600000000000000000.000000
72723257588050687000000.000000
46746591674697662000000000.000000
190616988374992920000000000000000000000000.000000
```

132.[AFCTF2018]Tiny LFSR

这题看起来特别的麻烦，我们先来一步步分析。题目用同一个加密脚本加密了两份文件，一份是plain加密得到的cipher，另一个是flag加密得到的flagencode，再看看加密的方式，前一部分是通过lfsr的密钥key与plain前一部分按位异或得到的，后一部分是通过，lfsr生成的密钥流与plain的后一部分按位异或得到的，感觉就是特别的繁琐了。于是，我们的思路是先通过cipher与plain按位异或得到key值先，然后我们可以知道LFSR中的key与mask位数是相同的，看了一下mask的位数是二进制64位，那么key的位数就是16进制16位，也就是8位ASCII字符，于是我们设置异或的长度为8个字符，当然也可以设置更多

```
cipher="72472201E3C0AC877A27C18729749FDA185C1DF902500AEB425C5B6A53574B4A00508546094A90A2F1547780FD401E8C2983A70F
22931F0BCC0EBE6EC83B1284BF2023AEBE59B1CBD2D9C395E9C76D42DF65C470C23C92E65F66504F3025B5F660E772096A172CDD"
c=cipher.decode('hex')
#print c
plain="sdgfjkahblskdjxbvfskljdfbguisldfbvghkljsdfbghsjkldhbgjklsdbgvlkjsdgbkljb sdkljfhwelo;sdfghioeurthgbnjl k"
a=""
for i in range(0, 8):
 a+=chr(ord(c[i])^ord(plain[i]))
print a
```

通过这样即可得到密钥key，也可以带入原脚本验证

```
cipher="72472201E3C0AC877A27C18729749FDA185C1DF902500AEB425C5B6A53574B4A00508546094A90A2F1547780FD401E8C2983A70F
22931F0BCC0EBE6EC83B1284BF2023AEBE59B1CBD2D9C395E9C76D42DF65C470C23C92E65F66504F3025B5F660E772096A172CDD"
c=cipher.decode('hex')
#print c
plain="sdgfjkahblskdjxbvfskljdfbguisldfbvghkljsdfbghsjkldhbgjklsdbgvlkjsdgbkljb sdkljfhwelo;sdfghioeurthgbnjl k"
a=""
for i in range(0, 8):
 a+=chr(ord(c[i])^ord(plain[i]))
print a
```

可以发现与给的cipher的后一部分相同，可以认为得到的key是正确的，于是我们可以生成lfsr产生的密钥流，生成位数设置位flagencode的位数

```
R = bytes_to_long(a)
tmptext=""
#for i in range(len(a), len(plain)):
for i in range(len(a), 1213):
    tmp=0
    for j in range(8):
        (R,out)=lfsr(R,mask)
        tmp=(tmp << 1)^out
    tmptext+=chr(tmp)
print 2,tmptext
```

这里要注意，一定要初始话R的值
于是我们可以开始求解flag，先将前一部分与key按位异或

```
flagencode1="484D6504E6C6BD9A6C22DC8B613E8698025300F70F4843EB455E4A614158440C114E8B48160688B4F25B6693F04154CF09A
19C3F6CD91D149F0BFC7AD63E1F9AEC3621ABBC46AFCC808AD096EADE6E0AC16ED86A8D6F94E94C2E50537531E7EE61EE7506725B6AC3BBE
2C1886B8B9B6FECB4464F1778D57F0729924FBED5B7D9E581120E95BC75564B40DF37C57DFD152D7163FD61E12DD86347FD55EB3FB994818
E61AF3845FB59D2A1633105606FF861F4809934AC6994B4A09EA57629C816C4F3A7C159BCCEF293534D3EFFFFF4AA9CB87E5E37D7292FE20
C15F9A282859C93F0190F9DA409A125D47BCD80E39EE103DDA17498B5EF7EED50A064F2A5A78C8BDB69CDE2855F75752460AE0346E171627
4BE3D2A733107927ACE7B3B3CD86AB5F4FB3AA52A3B5ADAE1A603B4C97C04067EA64785C243634382B6C66066F09A58FC4F381FB656E8C42
3041D338738E52CE9E816BCDFFD7BB2F02F90ACDDB76DF2E3E9861B40520DA20670A341A88CD9D1AA37B17F03672D424A8FB3BF01E2CAEE3
2F6A9F871AA0CA0BA999F4659FC0A4506FB64C910610F0DBEFFCBCA04AD81A96608ED9BDE070A2BF48982D82BDFE096F8FFAA2824F66B3E5
DA04DAC0F985CF571EBDB14B3A99EEC5F861129D62EA28310D0F7F1FC5F09D5BAB601961E38BD9F0151175F71421BBFFCDD19FAFA4047CA2
7EF6C74C351CDF73F02457E682AE6370C1BC2E3DC9AD8EC0C45D2B011657ABEBDEE68BEA83453CAF5B13D999BB706A44A1D69EA1E43049D1
219CBF433A4668E0750ECEC29874A904525DAA518F0247B13AF5571DB7E97F5CECD0E0B082F6EEF736A96ECC30F69B12B94289FF597E643D
C5DBF54E1393E9DBF70981867845FA58351B07F1A891FDEFA69CB9CF50F803E7961B9E8F0D89C5F698269B28C32588F5E0229CB75E8135AE
3E4A2F59C8F314A1041D629A4FD6CC2D1603AE7B866258FED3740EBC8305203972AF3BB89E64D34248162A36CCB1288AD7380914507C9589
4842B8A4780659D5A31992E6BDCCDD6D2C60D3A1B0B90A382B7BEC2F4F2E5C1A1A2B03D45E0A0F2B091C37D5869F0E7483C7575A12F55126
1D6B326E6FB4B59B604234D36BC35050A608D738C31AD5F18EC82364B9C41000F511274443F5A8F02DCF901CC0874C06567296598CA6BD5D
AF357F4C5F01A8C555C4DE796315FB9B5C5C43DB2C0E4D4E32A5B26DA45E28EABDF5575B4F16445197C7A9E93C89B4C7DDBA31E117D8ADFE
342CE6518B32F9E24F974829B8DAF0F07C1BAE2DB64D390DB5DBBC765D075270198B3F788A2DA30CCAED2D6658108C7593ACFE65B9A98FB9
E156C2E6921B7E9A7555DFF69744433EBACA2C02BC8BF3C9DE7DC5BCB04C3504D25F285A70D0B0ED17A0AFFB776ACA2958B8B1009DC4ECDF
74159E3C192041CB85ED364381A21881579A4DEBC4C1585C9803B117D6D8C16C984743FBC9A220D3C15014407D716068DC6520096FC4FA73
4E65EBACB00EFE7A737400CB815EE2BD6948C8186651EDA7D5D5397D27E0F1851CCED80E0D752F9BFF4D7DC3CA1441611BEA3297FB6FAFFE
BD18B15D6456A408D5E6217E31F0D375CA6CE5F799DE9C177800EFC622BE9060982D23DC6857B79A1178FC03CD6FC2EC7CB91D8F3277ABA8
6F5F02AFD1428A6D4B595D0359A28C74EBD4EEEE7DD6C9C9BE9C71420E9D4277EED4F474A21D39C02734581"
flagencode2=flagencode1.decode('hex')
print len(flagencode2)
#print flagencode2
flag=""
for i in range(0, 8):
 flag+=chr(ord(a[i])^ord(flagencode2[i]))
print flag
#In compu
```

后半部分通过将flagencrypt与密钥流按位异或即可，整题脚本给出

```
import os,random,sys,string
from hashlib import sha256
import gmpy2
from Crypto.Util.number import *
import base64
mask = 0b1101100000000000000000000000000000000000000000000000000000000000
cipher="72472201E3C0AC877A27C18729749FDA185C1DF902500AEB425C5B6A53574B4A00508546094A90A2F1547780FD401E8C2983A70F
22931F0BCC0EBE6EC83B1284BF2023AEBE59B1CBD2D9C395E9C76D42DF65C470C23C92E65F66504F3025B5F660E772096A172CDD"
c=cipher.decode('hex')
#print c
plain="sdgfjkahblskdjxbvfskljdfbguisldfbvghkljsdfbghsjkldhbgjklsdbgvlkjsdgbkljb sdkljfhwelo;sdfghioeurthgbnjl k"
a=""
for i in range(0, 8):
 a+=chr(ord(c[i])^ord(plain[i]))
print a

def lfsr(R, mask):
 output = (R << 1) & 0xffffffffffffffff
 i=(R&mask)&0xffffffffffffffff
 lastbit=0
 while i!=0:
  lastbit^=(i&1)
  i=i>>1
 output^=lastbit
 return (output,lastbit)
```

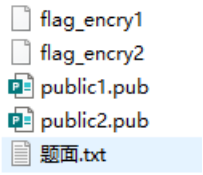
```
R = bytes_to_long(a)
t=""
for i in range(len(a), len(plain)):
    tmp=0
    for j in range(8):
        (R,out)=lfsr(R,mask)
        tmp=(tmp << 1)^out
    t+=long_to_bytes((tmp^ord(plain[i])))
print 1,t

R = bytes_to_long(a)
tmptext=""
#for i in range(len(a), len(plain)):
for i in range(len(a), 1213):
    tmp=0
    for j in range(8):
        (R,out)=lfsr(R,mask)
        tmp=(tmp << 1)^out
    tmptext+=chr(tmp)
print 2,tmptext


flagencode1="484D6504E6C6BD9A6C22DC8B613E8698025300F70F4843EB455E4A614158440C114E8B48160688B4F25B6693F04154CF09A
19C3F6CD91D149F0BFC7AD63E1F9AEC3621ABBC46AFCC808AD096EADE6E0AC16ED86A8D6F94E94C2E50537531E7EE61EE7506725B6AC3BBE
2C1886B8B9B6FECB4464F1778D57F0729924FBED5B7D9E581120E95BC75564B40DF37C57DFD152D7163FD61E12DD86347FD55EB3FB994818
E61AF3845FB59D2A1633105606FF861F4809934AC6994B4A09EA57629C816C4F3A7C159BCCEF293534D3EFFFFF4AA9CB87E5E37D7292FE20
C15F9A282859C93F0190F9DA409A125D47BCD80E39EE103DDA17498B5EF7EED50A064F2A5A78C8BDB69CDE2855F75752460AE0346E171627
4BE3D2A733107927ACE7B3B3CD86AB5F4FB3AA52A3B5ADAE1A603B4C97C04067EA64785C243634382B6C66066F09A58FC4F381FB656E8C42
3041D338738E52CE9E816BCDFFD7BB2F02F90ACDDB76DF2E3E9861B40520DA20670A341A88CD9D1AA37B17F03672D424A8FB3BF01E2CAEE3
2F6A9F871AA0CA0BA999F4659FC0A4506FB64C910610F0DBEFFCBCA04AD81A96608ED9BDE070A2BF48982D82BDFE096F8FFAA2824F66B3E5
DA04DAC0F985CF571EBDB14B3A99EEC5F861129D62EA28310D0F7F1FC5F09D5BAB601961E38BD9F0151175F71421BBFFCDD19FAFA4047CA2
7EF6C74C351CDF73F02457E682AE6370C1BC2E3DC9AD8EC0C45D2B011657ABEBDEE68BEA83453CAF5B13D999BB706A44A1D69EA1E43049D1
219CBF433A4668E0750ECEC29874A904525DAA518F0247B13AF5571DB7E97F5CECD0E0B082F6EEF736A96ECC30F69B12B94289FF597E643D
C5DBF54E1393E9DBF70981867845FA58351B07F1A891FDEFA69CB9CF50F803E7961B9E8F0D89C5F698269B28C32588F5E0229CB75E8135AE
3E4A2F59C8F314A1041D629A4FD6CC2D1603AE7B866258FED3740EBC8305203972AF3BB89E64D34248162A36CCB1288AD7380914507C9589
4842B8A4780659D5A31992E6BDCCDD6D2C60D3A1B0B90A382B7BEC2F4F2E5C1A1A2B03D45E0A0F2B091C37D5869F0E7483C7575A12F55126
1D6B326E6FB4B59B604234D36BC35050A608D738C31AD5F18EC82364B9C41000F511274443F5A8F02DCF901CC0874C06567296598CA6BD5D
AF357F4C5F01A8C555C4DE796315FB9B5C5C43DB2C0E4D4E32A5B26DA45E28EABDF5575B4F16445197C7A9E93C89B4C7DDBA31E117D8ADFE
342CE6518B32F9E24F974829B8DAF0F07C1BAE2DB64D390DB5DBBC765D075270198B3F788A2DA30CCAED2D6658108C7593ACFE65B9A98FB9
E156C2E6921B7E9A7555DFF69744433EBACA2C02BC8BF3C9DE7DC5BCB04C3504D25F285A70D0B0ED17A0AFFB776ACA2958B8B1009DC4ECDF
74159E3C192041CB85ED364381A21881579A4DEBC4C1585C9803B117D6D8C16C984743FBC9A220D3C15014407D716068DC6520096FC4FA73
4E65EBACB00EFE7A737400CB815EE2BD6948C8186651EDA7D5D5397D27E0F1851CCED80E0D752F9BFF4D7DC3CA1441611BEA3297FB6FAFFE
BD18B15D6456A408D5E6217E31F0D375CA6CE5F799DE9C177800EFC622BE9060982D23DC6857B79A1178FC03CD6FC2EC7CB91D8F3277ABA8
6F5F02AFD1428A6D4B595D0359A28C74EBD4EEEE7DD6C9C9BE9C71420E9D4277EED4F474A21D39C02734581"
flagencode2=flagencode1.decode('hex')
print len(flagencode2)
#print flagencode2
flag=""
for i in range(0, 8):
 flag+=chr(ord(a[i])^ord(flagencode2[i]))
print flag
#In compu

for i in range(len(a), 1213):
    flag+=long_to_bytes(ord(tmptext[i-len(a)])^ord(flagencode2[i]))
print 3,flag
#3 In computing, a linear-feedback shift register (LFSR) is a shift register whose input bit is a linear functio
n of its previous state.
```

```
#The most commonly used linear function of single bits is exclusive-or (XOR). Thus, an LFSR is most often a shif
t register whose input bit is driven by the XOR of some bits of the overall shift register value.

#The initial value of the LFSR is called the seed, and because the operation of the register is deterministic, t
he stream of values produced by the register is completely determined by its current (or previous) state. Likewi
se, because the register has a finite number of possible states, it must eventually enter a repeating cycle. How
ever, an LFSR with a well-chosen feedback function can produce a sequence of bits that appears random and has a
very long cycle.

#Applications of LFSRs include generating pseudo-random numbers, pseudo-noise sequences, fast digital counters,
and whitening sequences. Both hardware and software implementations of LFSRs are common.

#The mathematics of a cyclic redundancy check, used to provide a quick check against transmission errors, are cl
osely related to those of an LFSR.

#Congratulations! flag is afctf{read_is_hard_but_worthy}
```

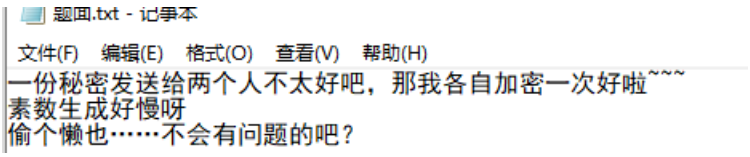
这道题是看的大佬的wp

133.[AFCTF2018]One Secret, Two encryption

查看题目



两个pub后缀的用010打开是两个公钥加密
得到两个n，e



从这个猜一手，有公共素数gmpy2.gcd(n1,n2)求得一个公共素数就很简单了

```
import gmpy2
from Crypto.Util.number import *
n1=48502971381622234688264816230824402495791368767983126527352046986896139690086325452209766991703084540823908347425707182478042020609294935716420746794285651684058771106815181056673017856535176976844909823750789898860404511150821209289825883809146092730081539779079505324986054862258839736431415160240583153605729887446071341102544894215160269372491634939826813366287260334891247056572177682290584871558652650804274880289218796083388989335408255648890121661813461772766398283463763621689342088224672956737618769658645731645293368852505773577673142565810194741306514121008978396064911894243739592440236956696532134983293
n2=23675367686720009596681811717872952718987892883976729971348434189324059599467396373680444203198617978567714905734430035201371493240802179718367805705222586614190344815148830680927521667529678794970955647325056147515323304086750562852753542501579553214575790063603932183271648049513842900419565518553344927967199018181657889025475845634557479415172968756972418411772196350244613955961175841942261347770788745436991177618936996343035714211069178942150789388859999635805868244970400732410558903287943100258790142940512305907165629425380318839653173977282715897597183760734146320268018065608629066919890932984787525802770
e1=16666266329603682390011594080477659912702500422062441574471711881956573029330195019321017779995100012357363388431077098717859067493930042576141298020610811558614337223801450015371811426135152901388357652360028116899864722807624081571764375030217530615887465204337207346089536391115585569304907215175799944930885510130508356900197726007443173982181838834021920604809799794564699378632577813625211845781421294441224288321067217254093091139759864362416621078790853610146507164390428560132034402428348786485062444283677067084311211097145059817285298188746218686247542850696933687794953166006012990372770039947903965892990
e2=65537
p=gmpy2.gcd(n1,n2)
q=n2//p
assert(p*q==n2)
phi=(p-1)*(q-1)
d=gmpy2.invert(e2,phi)
c_bytes=open("flag_encry2","rb").read()
c=bytes_to_long(c_bytes)
m=pow(c,d,n2)
flag=long_to_bytes(m)
print(flag)
```

用这个也可以，或者直接用

运行得到 `b'\x02~\x83\xa7\xed\xd5\xde\xf1\xb8\xc6\x17%5:\xf7wZn\xc2WbDn)\x070\xfc\x98}\xa5\x96@\x90?Y\xba\xe5\xca\xdam\xbaGF\tz\xe7W\xcd\x94\x1c#\xecti\x8b\x89\x18hH\xc5\xbf\x10\xe983C}3Cz>HbX\xbe\x98<JG\x86??\xe2?\xfc\xaf\xb60\xbe\xec\xe3h\x07\xda\xea\xefSw\xb1t\x9bp\x03y\x12\xf2<\x99\xae\xf6\xde\x9b\xdf\xdd\xb4\xf7\x88t\xe0\xff%\x11"B\xa7r\xf2}\xf3\xaf\xca\x9dfI.\x08\xd2\xdb1\x18E\xed\xb38\xe5\xcc\xc3#\x1eT*\xec\xc4a\x95\xd9\xd9\xe7\xd4\x88O\xa3\'\xae9S\xa3\xd1\x7f~+\xefqa\xe1y\x82\x19l\x8a\x8b\xc5\xe8\xc1\xbd\t\x88\xbf+Th\xba\x8f\xfd\xb0\x89\n\x92\x00OpenSSL is widely used\r\nflag is afctf{You_Know_0p3u55I}'`

134.[NCTF2019]easyRSA

查看题目

```
from flag import flag

e = 0x1337
p = 19913867782374383733992752015760782002974657455774654909492148829287722650919831501601891938525978123814840283331603363496816327619899927932782790187942642966467435884408449183054327162514728095027393440587934143842917145300245383889745810212883669038560415032497290798196062676767915312573567741739707819605983388974581021288366903856041503249729079819606267676791531257356774173970781960599
q = 11221369590547214241522144451532653232035242947834168335281118350326967655543460122901367931942387823894495683024438665367441341165869675117384444339460824671605308622691058140052816784830611917987911580977879309361138176493978905752457534950116368945281014828062522654160938316634787983213449544470669712474199
n = p * q

assert(flag.startswith('NCTF'))
m = int.from_bytes(flag.encode(), 'big')
assert(m.bit_length() > 1337)

c = pow(m, e, n)
print(c)
# 10562302690541901187975815594605242014385201583329309191736952454310803387032252007244962585846519762051885640856082157060593829013572592812958261432327975138581784360302599265408134332094134880789013207382277849503344042487389850373487656200657856862096900860792273206447552132458430989534820256156021128891296387414689693952047302604774923411425863612316726417214819110981605912408620996068520823370069362751149060142640529571400977787330956486849449005402750224992048562898004309319577192693315658275912449198365737965570035264841782399978307388920681068646219895287752359564029778568376881425070363592696751183359
```

眨眼一看pqec都有了还在想这么简单的题，结果啪啪打脸,没办法逆元
然后我就想用c=pow(m,e,n)来解嗯事实好像能解，当时我没运行出来，后面我就去找大佬wp
正常情况下的RSA都要求e和phi(n)要互素，不过也有一些e和phi(n)有很小的公约数的题目，这些题目基本都能通过计算e对phi(n)的逆元d来求解。

然而本题则为e和p-1(或q-1)的最大公约数就是e本身，也就是说e | p-1，只有对c开e次方根才行。
可以将同余方程
$$m^e \equiv c \quad (\text{mod}\ n)$$
化成
$$\begin{aligned}
m^e &\equiv c \quad (\text{mod}\ p)\newline
m^e &\equiv c \quad (\text{mod}\ q)
\end{aligned}$$
然后分别在GF§和GF(q)上对c开e=0x1337次方根，再用CRT组合一下即可得到在mod n下的解。
问题是，如何在有限域内开根？

这里e与p-1和q-1都不互素，不能简单地求个逆元就完事。

这种情况下，开平方根可以用Tonelli–Shanks algorithm，wiki说这个算法可以扩展到开n次方根。

在这篇paper里给出了具体的算法：Adleman-Manders-Miller rth Root Extraction Method

Table 4: Adleman-Manders-Miller rth root extraction algorithm in F_q

Input: F_q and a rth residue δ, $r|q-1$.

Output: A rth root of δ.

Step 1: Choose ρ uniformly at random from F_q^*.

Step 2: **if** $\rho^{\frac{q-1}{r}} = 1$, **go to** Step 1.

Step 3: Compute t, s such that $q-1 = r^t s$, where $(r, s) = 1$.

Compute the least nonnegative integer α such that $s|r\alpha - 1$.

Compute $a \leftarrow \rho^{r^{t-1}s}, b \leftarrow \delta^{r\alpha-1}, c \leftarrow \rho^s, h \leftarrow 1$

Step 4: **for** $i = 1$ **to** $t-1$

compute $d = b^{r^{t-1-i}}$

if $d = 1$, $j \leftarrow 0$,

else $j \leftarrow -\log_a d$ (compute the discrete logarithm)

$b \leftarrow b\,(c^r)^j$, $h \leftarrow h\,c^j$

$c \leftarrow c^r$

end for

Step 5: **return** $\delta^\alpha \cdot h$

这个算法只能开出一个根，实际上开0x1337次方，最多会有0x1337个根（这题的情况下有0x1337个根）。如何找到所有的primitve 0x1337th root of 1?

In a finite field of size q, the multiplicative subgroup has order $q-1$ (i.e. all elements are invertible except 0). If n is relatively prime to $q-1$, then there is only one n-th root of unity, i.e. 1 itself. If n divides $q-1$, then there are n roots of unity. In the remainder of this answer, I assume that you are in that case, i.e. n divides $q-1$.

Everything I write below uses computations *in the finite field* (i.e. modulo q, if q is prime).

To get an n-th root of unity, you generate a random non-zero x in the field. Then:

$$(x^{(q-1)/n})^n = x^{q-1} = 1$$

Therefore $x^{(q-1)/n}$ is an n-th root of unity. Note that you can end up with any of the n n-th roots of unity (including 1 itself), each with probability $1/n$.

Now you may want to have a *primitive* n-th root of unity, i.e. one value g such that all n-th roots of unity can be obtained with values g^j for integers j ranging from 0 to $n-1$. If g is an n-th root of unity, then it is primitive if and only if $g^j \neq 1$ for any $j > 0$ that divides n, except n itself. In your case this is easy: since n is a power of 2, any j that divides n is also a power of 2. In practice, this yields the following:

先用Adleman-Manders-Miller rth Root Extraction Method在GF§和GF(q)上对c开e次根，分别得到一个解。大概不到10秒。
然后去找到所有的0x1336个primitive nth root of 1，乘以上面那个解，得到所有的0x1337个解。大概1分钟。
再用CRT对GF§和GF(q)上的两组0x1337个解组合成mod n下的解，可以得到0x1337**2==24196561个mod n的解。最后能通过check的即为flag。大概十几分钟。
exp.sage如下

```
import random
import time

# About 3 seconds to run
def AMM(o, r, q):
    start = time.time()
    print('\n----------------------------------------------------------------------------------------')
    print('Start to run Adleman-Manders-Miller Root Extraction Method')
    print('Try to find one {:#x}th root of {} modulo {}'.format(r, o, q))
    g = GF(q)
    o = g(o)
    p = g(random.randint(1, q))
```

```
    while p ^ ((q-1) // r) == 1:
        p = g(random.randint(1, q))
    print('[+] Find p:{}'.format(p))
    t = 0
    s = q - 1
    while s % r == 0:
        t += 1
        s = s // r
    print('[+] Find s:{}, t:{}'.format(s, t))
    k = 1
    while (k * s + 1) % r != 0:
        k += 1
    alp = (k * s + 1) // r
    print('[+] Find alp:{}'.format(alp))
    a = p ^ (r**(t-1) * s)
    b = o ^ (r*alp - 1)
    c = p ^ s
    h = 1
    for i in range(1, t):
        d = b ^ (r^(t-1-i))
        if d == 1:
            j = 0
        else:
            print('[+] Calculating DLP...')
            j = - dicreat_log(a, d)
            print('[+] Finish DLP...')
        b = b * (c^r)^j
        h = h * c^j
        c = c ^ r
    result = o^alp * h
    end = time.time()
    print("Finished in {} seconds.".format(end - start))
    print('Find one solution: {}'.format(result))
    return result

def findAllPRoot(p, e):
    print("Start to find all the Primitive {:#x}th root of 1 modulo {}.".format(e, p))
    start = time.time()
    proot = set()
    while len(proot) < e:
        proot.add(pow(random.randint(2, p-1), (p-1)//e, p))
    end = time.time()
    print("Finished in {} seconds.".format(end - start))
    return proot

def findAllSolutions(mp, proot, cp, p):
    print("Start to find all the {:#x}th root of {} modulo {}.".format(e, cp, p))
    start = time.time()
    all_mp = set()
    for root in proot:
        mp2 = mp * root % p
        assert(pow(mp2, e, p) == cp)
        all_mp.add(mp2)
    end = time.time()
    print("Finished in {} seconds.".format(end - start))
    return all_mp


c = 10562302690541901187975815594605242014385201583329309191736952454310803387032252007244962585846519762051885640856082157060593829013572592812958261432327975138581784360302599265408134332094134880789013207382277849503344044
```

```
248738985037348765620065785686209690086079227320644755213245843098953482025615602112889129638741468969395204730260477492341142586361231672641721481911098160591240862099606852082337006936275114906014264052957140097778733095648684944900540275022499204856289800430931957719269331565827591244919836573796557003526484178239997830738892068106864621989528775235956402977856837688142507036359269675118335986
p = 199138677823743837339927520157607820029746574557746549094921488292877226509198315016018919385259781238148402833316033634968163276198999279327827901879426429664674358844084491830543271625147280950273934405879341438429171453002453838897458102128836690385604150324972907981960626767679153125735677417397078196059
q = 112213695905472142415221444515326532320352429478341683352811183503269676555434601229013679319423878238944956830244386653674413411658696751173844443394608246716053086226910581400528167848306119179879115809778793093611381764939789057524575349501163689452810148280625226541609383166347879832134495444706697124741
e = 0x1337
cp = c % p
cq = c % q
mp = AMM(cp, e, p)
mq = AMM(cq, e, q)
p_proot = findAllPRoot(p, e)
q_proot = findAllPRoot(q, e)
mps = findAllSolutions(mp, p_proot, cp, p)
mqs = findAllSolutions(mq, q_proot, cq, q)
print mps, mqs

def check(m):
    h = m.hex()
    if len(h) & 1:
        return False
    if h.decode('hex').startswith('NCTF'):
        print(h.decode('hex'))
        return True
    else:
        return False


# About 16 mins to run 0x1337^2 == 24196561 times CRT
start = time.time()
print('Start CRT...')
for mpp in mps:
    for mqq in mqs:
        solution = CRT_list([int(mpp), int(mqq)], [p, q])
        if check(solution):
            print(solution)
    print(time.time() - start)

end = time.time()
print("Finished in {} seconds.".format(end - start))
```

好像少了一个括号，这是我改之前的
运行的到flag{T4k31ng_Ox1337_r00t_1s_n0t_th4t_34sy}

135.[NPUCTF2020]共 模 攻 击

查看题目

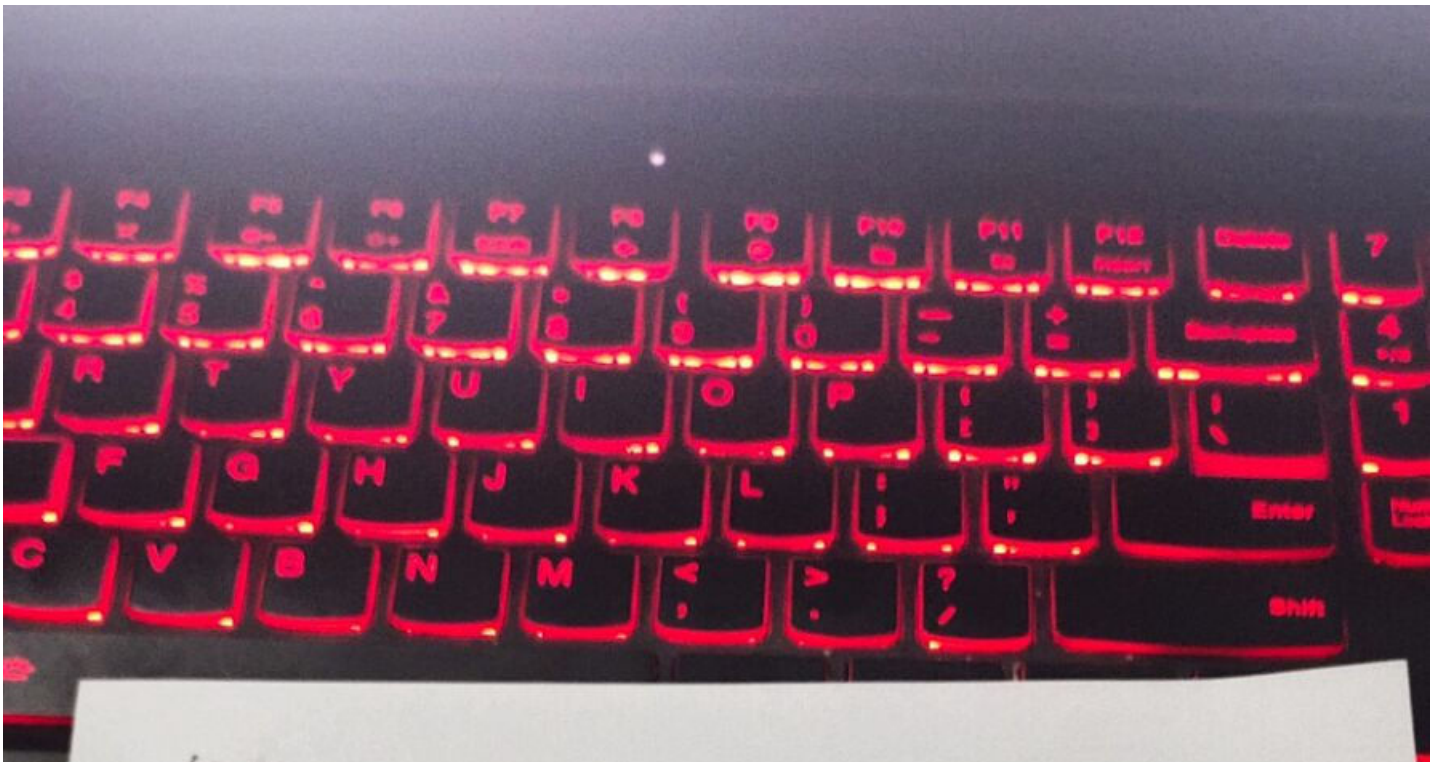
```
from gmpy2 import *
from Crypto.Util.number import *
from secret import flag

flag = flag.strip(b"npuctf{").strip(b"}")
m = bytes_to_long(flag)

p, q = getPrime(512), getPrime(512)
n = p * q
e1, e2 = p, q
c1, c2 = pow(m, e1, n), pow(m, e2, n)

print(n)
print(c1)
print(c2)
```

```
128205304743751985889679351195836799434324346996129753896234917982647254577214018524580290192396070591032007818847697193260130051396080104704981594190602854241936777324431673564677900773992273463534717009587530152480725448774018550562603894883079711995434332008363470321069097619786793617099517770260029108149
968606542352752022173681301950898396080375583888845227375006111212715713351239815888079940438004685290021475706555976106396809777807794948803306694663897884970467103192133762283911380219763889251713077600300584569348987715894358362613172837439516145051368403646387069144244335667820449261116399556124121341989566853166416448316408476072940703716510748416699965603380497338943730666656667456274146023583837768495637484138572090891246105018219222267465595710692705776272469703739932909158740030049375350999465338363044226512016686534246611049299981674236577960786526527933966681954486377462298197949323271904405241585
```

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

m = bytes_to_long(hint)
p = getPrime(256)
c = pow(m, 256, p)
print(p)

p, q = getPrime(256), getPrime(256)
n = p * q
e1, e2 = getPrime(32), getPrime(32)
c1, c2 = pow(c, e1, n), pow(c, e2, n)
print(n)
print(e1, c1)
print(e2, c2)
```

```
107316975771284342108362954945096489708900302633734520943905283655283318535709
680749200621993533523372223202480978443429329317231728281497868893171142393962968222437487023358796996071363831006878441547453503378077276617132046128157923034139611754421169036191391717309256938035960912941109206872374826444526733030696056821731708193270151759843780894750696642659795452787547355043345348714129217723
2622163991 161345401595155528971114836697729761362454402593755937178473605944845443765263384711127261924812661350002899281373284204101858870720145839872670082884424
```

我写的笔记有点乱



hint:

m=pow(c,d,n)　　qq随机生成

n=q·p　　$e_1 e_2$随机生成

c_1 = pow(c,e1,n)

c_2 = pow(c,e2,n)　　P e_1 c_1 e_2 c_2 n 已知

s=gcdext(e1,e2)

c=pow(c1,s[1],n)*pow(c2,s[2],n)%n

m=hthroot_mod(c,256,p)

task: flag=flag.strip()
m=bytes_to_long(flag)

pq 随机生成.

e_1, e_2 = P, q　　求m. m=flag

c_1, c_2 = pow(m,e1,n), pow(m,e2,n) = pow(m,p,n), pow(m,q,n)

已知n, c_1, c_2

$c_1 = m^p \bmod n = m^p \bmod p$ 同理 $c_2 = m^q \bmod p$

pq为素数 由费马定理

$m^p = m \bmod p$ 同理　　$m^q = m \bmod q$

import libnum　libnum.n2s(n)

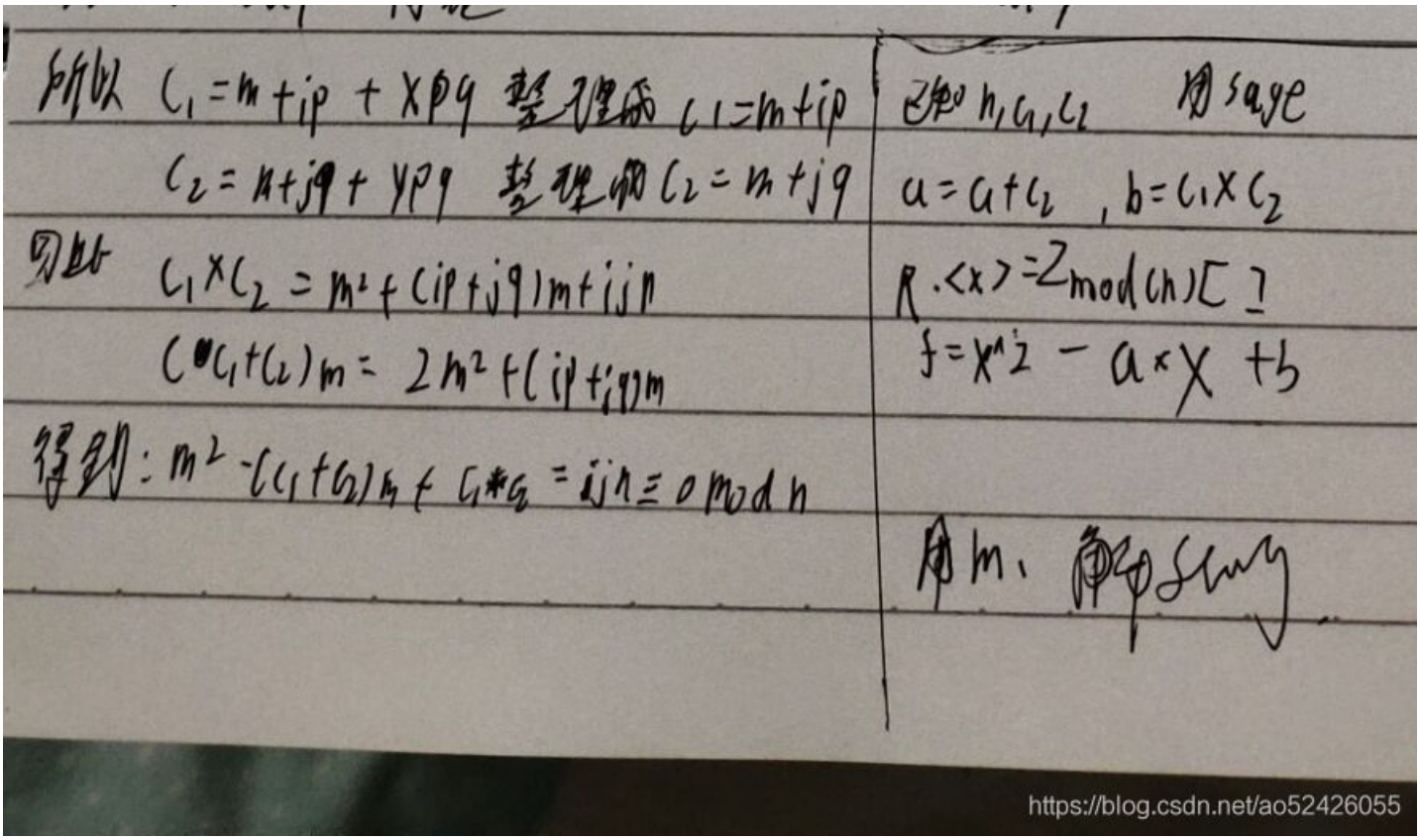
n = invert(m,phi)　求mod phi之逆元

pow(m,e,n)　求 $a^d \bmod n$

gmpy2.is_prime(n)

gmpy2.gcd(a,b)　最大公约数

gmpy.iroot(x,y)　开n次方



解密脚本

```
from gmpy2 import*
from libnum import*
from sympy import*

p = 10731697577128434210836295494509648970890030263373452094390528365528331853570 9

e1 = 2303413961
c1 = 1754421169036191391717309256938035960912941109206872374826444526733030696056821731708193270151759843780894750696642659795452787547355043345348714129217723

e2 = 2622163991
c2 = 1613454015951555289711148366977297613624544025937559371784736059448454437652633847111272619248126613500028992813732842041018588707201458398726700828844249
n =  6807492006219935335233722232024809784434293293172317282814978688931711423939629682224374870233587969960713638310068784415474535033780772766171320461281579

s = gcdext(e1,e2)
c = pow(c1,s[1],n)*pow(c2,s[2],n) % n
print(c)
#c = 193840023587257596791989176867633103490509882236276250960508003697604842375 57
m = nthroot_mod(c,256,p)
print(n2s(m))
```

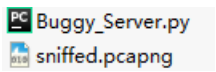
hiti得到一个`m.bit_length() < 400`

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

n = 12820530474375198588967935119583679943432434699612975389623491798264725457721401852458029019239607059103200781884769719326013005139608010470498159419060285424193677732443167356467790077399227346353471700958753015248072544877401855056260389488307971199543433200836347032106909761978679361709951777026002910814 9
c1 = 96860654235275202217368130195089839608037558388884522737500611121271571335123981588807994043800468529002147570655597610639680977780779494880330669466389788497046710319213376228391138021976388925171307760030058456934898771589435836261317283743951614505136840364638706914424433566782044926111639955612412134198
c2 = 95668531664164483164084760729407037165107484166999656033804973389437306666566674562741460235838377684956374841385720908912461050182192222674655957106927057762724697037399329091587400300493753509994653383630442265120166865342466110492999816742365779607865265279339666819544863774622981979493232719044052415 85

a = c1+c2
b = c1*c2

print(a)
print(b)
#中间 Sage 求 m  的代码:
#a = 10642750740169165053377660626803054332454830680558448834088110846021530200178064904508214006738430629749778505479416970153092708279899871714779626517708249427331918002295330913754987805202576427617077309839310268344691545812368244731061726541818819246592336689257267359637891994424434312406096322751681737578 3
#b = 92665165667191133359702240196887040934394240049288125514237795175917663149491501694199150412381026532986224659286114571921367550279537805356490481876537702509648360103602501226710326070278755561221675518852191340530586145112581414940950842560223167029213142227326872862978263335449864802185961422362712348931889920562778542640259799631944019821877403839080940328195270273088330600722679763238926738691270785709355633584626995427057292036134701961436540274402653371344244991655542567818440638016761401113170241849307375981631089005628191731011003445300721041524270792414169774981890738324854517911859451192763022383 0
#n = 12820530474375198588967935119583679943432434699612975389623491798264725457721401852458029019239607059103200781884769719326013005139608010470498159419060285424193677732443167356467790077399227346353471700958753015248072544877401855056260389488307971199543433200836347032106909761978679361709951777026002910814 9
#R.<x>=Zmod(n)[]
#f = x^2 - a*x +b
#f.small_roots(X=2^400)   #根的绝对边界，根就是flag
m=42428390430197820007881188873721328073715682794724994779987584662240029054422271565377881105203356523858 55
print(hex(m))#转换十六进制
print(bytes.fromhex(hex(m)[2:]))#ascll码
```

那个sage代码也给了m使用sgae求出来的
最终flag就是`verrrrrrry_345yyyyyyy_rsaaaaaaa_rightttttttt?`

136.[XNUCA2018]Warmup

查看题目



打开流量包。
通过分析一共有6个流。发现Alice和Dave的n相同，于是可以采用共模攻击。

```
n=25118186052801903419891574512806521370646053661385577314262283167479853375867074736882903917202574957661470179
14888253836156078436274020764962053674686088339511044393077813234364229524774979704144960196743469028075427958896
91669366595486824752597992245067619256368446164574344449914827664991591873150416287647528776014468498025993455581
97670042137263891600360771709739948484807394990524813865392934259830936447999603225814377345600010180258230478777
93210521636296183895996437133328740707108025097942148921016548590840401992739305332580906178756029448991147597783
42741920115134298253806238766543518220987363050115050813263
e1=7669
c1=22917655888781915689291442748409371798632133107968171254672911561608350738343707972881819762532175014157796940
21207377735136231438507478540075810259434835557827508062626913754313622502257932110719960285629025469622796643624
24461844135056466787287919626907443375181163243722813947072320384800680385686823770640186843632122565612649170175
05346889662805787719960214596204727314067283796282864052149964611648924867341706625565187820438817599183946745174
09304629842710180023814702447187081112856416034885511215626693534876901484105593275741829434329109239483368867
51838452295517680733243754057868886707756972854851387684147 1
e2=6947
c2=20494665879116666159961016125949070097530413770391893858215547229071116025581822729798313796823204861624912909
03097545074212280277587919444523206436777103601102136612339391735413484991167530787732410383487128851327445794103
64534770347986471821064226195043450552595436757299833078690637683033540333961090354725596512719631511333130051264
10469332270081014014160268092568132214806046620121015428464790528321287882790317278807506424993290417803724055
67816904384164559191879422615238580181357183882111249939492668328771614509476229785062819586796660370798030562805
22470449757044684413165003007500490121614189342014014056 8
import  gmpy2
import  binascii
import  rsa
import math
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)#扩展欧几里得算法
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{g00d_Luck_&_Hav3_Fun}'`

137.[AFCTF2018]MyOwnCBC

查看题目

```
#!/usr/bin/python2.7
# -*- coding: utf-8 -*-

from Crypto.Cipher import AES
from Crypto.Random import random
from Crypto.Util.number import long_to_bytes

def MyOwnCBC(key, plain):
 if len(key)!=32:
  return "error!"
 cipher_txt = b""
 cipher_arr = []
 cipher = AES.new(key, AES.MODE_ECB, "")
 plain = [plain[i:i+32] for i in range(0, len(plain), 32)]
 print plain
 cipher_arr.append(cipher.encrypt(plain[0]))
 cipher_txt += cipher_arr[0]
 for i in range(1, len(plain)):
  cipher = AES.new(cipher_arr[i-1], AES.MODE_ECB, "")
  cipher_arr.append(cipher.encrypt(plain[i]))
  cipher_txt += cipher_arr[i]
 return cipher_txt

key = random.getrandbits(256)
key = long_to_bytes(key)

s = ""
with open("flag.txt","r") as f:
 s = f.read()
 f.close()

with open("flag_cipher","wb") as f:
 f.write(MyOwnCBC(key, s))
 f.close()
```

简单的对MyOwnCBC.py脚本进行分析，可以发现，其实就是ECB模式下的AES，然后将每个分组单独进行了AES ECB加密，可以看到的是每一组AES加密的key为上一组加密结果的密文。
出题人设计这个题目的意图应该是想带大家了解一下AES CBC模式，用ECB模式进行一个大概的模拟（当然不是这个原理）。主要是想表达CBC模式下会把上一轮的加密影响扩散到下一轮的意思吧。

那解密也很简单，因为key其实就是每组加密后的密文嘛，直接AES ECB模式下解密即可。
脚本如下

```
from Crypto.Cipher import AES
f=open("flag_cipher","rb")
#<_io.BufferedReader name='flag_cipher'>
st=f.read()
#b'\xe5\xdf\x94sJ\xc2\xcd\x04\xeb\xb7\xcf\x05(\xbe\x98\\\xe9\xc3^\x1f!\xfb\xea6\xdac\x1f\xfe\x901\xbb\x13[+\xb5|
\xb3_\x06\x08\xa71JL\xe4\xf2\x05\x88\x1d\xe1c\xd99\xf6\xb9\'>c\xbbi\x84\x80l\x18st\xdf\tY\x91\xf5kb\xf55\xefd[hC
\x844\tx\x95\xdc\x14\xbb@\xb5u\x12\xfc\nHC)\xccv\xe8\x86\xe1\xc2\xd7\xd6\xb5\xc3\xf4(g\xd6\x99\x8e\x95*M\xaf^\xd
e\x07=\x16\xc5G\xe7\xfa\xd4\xfcn8\xd66\xe2\x1e\xa7\xb1\xf5{\xdb\x90?\x97`z\x02S\'/\xfdH\xd6\xc2M\xc7\xf6\xdb\x0c
u=|\xe7\x85\x10\x18\xa9k\xa9\x05-R\x8e\x8bS\xd8\x07\xc7\xf3N,a\xcd\x98AD\x91\x0e\x08\xc2\xb60\xbe ~@\xd3/S\xe8\x
8878&\x001C\x8c\x05B\x0b\xad\x9b\\\x19\x85|\x02X\x0e\x9f\xb43;\xabw\x8c\xb2a\xdc\x88\x9e\xac\x17\xf82\xbb\x9f\xb
0z\xbc\x13\xab[X\xe1\x8a\xf9\xed\xd8\x9e\xf4:\x08\xd3:\x0cH\x9a\xdal\x0c\xfb\xb9\xd0\x01\xe4m\x06\xf9e\xae\xfc\x
be\x1d\x18\x89+Y\x00\t\xf8\x17\xb6\xe88\xc4\xb6w\xba\x06\x9a\x93\x13\xbf\xb2\x1a\xf7c\x03\x02\xd2}\xb1\xc7\x9f\x
e4\x01\xa6\x0c\x87\x7f\xaf\n\xbcb\xd9\xa6\xb2\x06\xc8u\xc4\xf39k)k\xf72\xeb\xc5\x06\x91\x86~\x9ftWK\xaa\xe31\xa7
\x93\xc4J\xd1\xf9\x12\x15C>E~\x8bo\xc7\xc8\xcd\x8d\xe9l\xe5\n\xd6\xc7\xe1\x8b\x98f\x9d\xbc\xae\xd6\xcf\xdc\x1a\x
```

```
ad\xd1\xd0\x0f\x94\xa2D\x92\xc1\xae\xc6/\xe4\xcb\xd1\xc7\xcaG4-\x8a=\xd5O\x8f\xc8\xa4\x80x\x0e\x94o\xae\xf1\xeb\
xa2N8UY\x1e\x96\xf2_\xc9\x95I)\x1d^\xfaN\x95\x9a\x89\xfdk8T)\xd7v\xbe\x84\x88\x87\xb3\xea<D0\xd1\x81\x11n&U\xbb\
x9aO\xf0\xecY\x15\xf1m\xe2\xa8\x7f\xf7\xe9\x9f\xa1\xd9\xa0\xe8gH\x8b\xaf\xb9r\x8a\xdf2\xbe\x87\xbc\xab\xa6YYw\xa
4j^\x10U_\x1e\xc7\x7f\xb6W\xa4\xc6\xa4\x1fd\xa2\xee\xfc\xe6\xa7\x1e\xd3\xc9\x12_\x1d\xcd\xe2\x98\x9eg\xb0\x0b\x8
3o\xaa\x07\xd7jJ\x0e\x96\'\xed\x00jI}[dRp9\x82h\xc6\xf2g\xcec\x1e\x95\x03\xc6\xeb/\r8\x02\xcb0K\x1c\x8e\xf7>@\x1
9\xa03\x9e;@\x9f\xd4\xfb\x8e\xb4\xcchT\xf94Ds*\xbbX2\xf1:\xdb\xf7\xe3\x85y\xa7\xcfm\x89\xbf_Y\x9eGo\xa2c\x80\x9c
0MI\\Z\xa8\xa4#\xb3k[\x99\x99h-\x99\xf7\xae\xd4\xeb\xc6g\xa8;4M:3\xd2\x80\x85\r\r\xd9\x82\xa0\xc6\xae\x8c\x13edZ
\xacRG\xd3;\x95t3\x9e\xe6\xdf\x85\xe8\xab2b\xb3@\x1f\td\xfe\xd1c\xe2?\xb3\xdc\x85&\x82\x14\x97\x84\t\x00\xde\x12
\x1a\xb1\xf3\x8e\xe6N~\xff\x7fi^\n`P\x0e\''\xd4\xa3\x16\x11vp\xcb\xc5\x0f\x94\xc0\xf7H\xf1\x97\xf3\xae\x02\xde\xa
f?\xee"\xb2yIq\xa6M\xf3F\xfa<{\\\xfdd\xbf\x90\tV\x9b\xe7+\xc4\x14\x11\xe8\xff.\xdf\xed\xce\x14\xb8\x87DJ\x18\x00
\x90\x95\xc1WP\xd4\x18\x1b\x9d\xfc\xa2\xfe\xa1$5\x17\xa0\x98\xe0m_u\xc0\xe5\xef\x88\xbd\x82Q\x81-\xe7\x99\x07\x9
0Ej\x8d\x93j\xa6\x8f\xc6\xd2\xeen\xc5`\xb9\xc1\x91rI\x16\xcd\xef\xb3\x00\xc2\x96F\x8b#T\xc7E\xeb\x1e\x99%AF\xc9\
x90>\xcd\x01\x861A\x9b5\x18\xa7Z\x93\x0b5\xb0E\x83=\xb2\x14\x0b\xa2\x99\xe2\xbd\xb2A\xd1YR\xa9y\xdf\xebq\xa0\xb6
\x05rO\x08\x97\xcdsr\x9a\xfd\x04\xb1\x94\xf2\x1as\xfc\x0e\xaa\xdb\xce\x9580lC\xa8\x97_\x93\xb0\xe7\x91;W\xa8v\xb
e\xa75\\\x06Y\x05\x17\x8f\xdbih\x89\xdaa\x9e\xb6}\xc4\x89\'/\x94\xdc\x07\xa7\x9b\xa5\x87\xd8*\t\x84\xcev6\x86\xb
3\x8a\x1b\xfa\xebu\xa4\x06\x93:\xbfbh\x17\xe7\x1c[\xe6\x88\xa3\x82\xb6\xc5\xf5\x1d\x85M\x10k\x10f\x11\x17\xfba\x
a2\x11\xf7\xda\xd7Fs0\x9d\x1bF\xfc\x878\xdd\x12>"Ou\x93\xaee\xd3w\xdbJ\xeaM\xe2\xc4\xaa\xeb\x01\xb0<\xfe\xbe\x16
\xfe\xd9 9\x04+\x04Bg\xd6Y\x9b\xe2rI\xb0&S\x9a\x1c\xa2\xf5`\x84s\x90\xd1\x85a\xde\xf3\xc8r\xacY[Y}\xbd\x9a\xef\x
b9\x82\x8bK?\xec\xf7\xcdI\xa0\x04SK\x0b\x95\xc9\xa4\xe26\x0f@\xa3j\x0c8\xc6Q\xf8\xcb]\xa3W\xf9#7UQ\x11\x9f\xc5t\
x07m\x8a6O^\x88g\xf39f\xd8\xd9\x9c\x18[\xe3\x1dn\xfe9\xdd\xe1:\x9b\xc9t\x9d\x87\x04\xc5\x93T\xb3\x1e\xbb@wC\rS\x
e3\xb7/\xd1>\\\xb3H\xb7R\xce) \x0eR\x8c\x8e\xa6k\xe0\xc7 W\xaa\xe9\xf2\xea\x0fK\xc5\xb5\xb3\x08\\\xd6"b\xb6{\x9a
\x16\x1aIO\x1a\xf0\xfd\xde/(\xb3\x8bB!Hk\x8e\xe6\x10\x18@\xd2y6\xf9u\xdfX\xebO;k\x90V.]b]u\x1c\xd1\x91\x15\xab\x
1a!H\xf7>\x81pS\xcb\x11 a\xdah\x93\xe8\xb6b\x0b\xbc%e\xa1\x9e\xd8\x81\x98\t\xb2\xcc\x133:\x99\xf1TY\x9cl\tB\x8aX
z\xb9k\xa68e\xc3p\x00\xd5j|\x93Yz\xcd\x17"JT\xb6(b"t\xe7p6\x03\xaf\xcc\xf9n\xdb%\xee@:\xa9\xea\x0cV\xb8\x93\xe5\
xf8\xc9\x17\x1f\x91\x8d\xadF\x10\xdb4\xb2\xc0\x7f\xe2\xad\xbfwZ\xa1\xc4\xa7\xf1\x93\x0e\xe0\xed\xcc\xe5K|Kd"\xd4
\x00\xbf8\x7ff9\xb2\xecm\xeb\xa5X\x03HO\xc2\x97\x01\x86\xff\xd1\xc1\xa5E\xd4Z\xd7\xe4#H\xd4\x02c\x8e\xe2\x94\x93
s\xbb\xee;<gT\\\xc4\xea\x92\xc5\xd4)\xf3\xd9v"G\x98\xd7gx"]\xac\xec\xe3\xef\xecy\xb1\xbav\x9c\xd4\x1d\xee\x91\xd
5\x90\xb1\xeb\xd3$\xe5\x96\x91w[eO\xf4liB\nM\x96\x9b\x92\xc9\x03\xafs\xd8\x83\xd6!\xa1\x90/\xcba\xdf\xbb\xa6\x15
\\\xa3t\xd8\xb3j\xb2\xd3\xd5\x19\xf6\xa0\x16\xefE\xca\xbc\xbaB\xf3#\xa0\x0c\xec\x04wp\x15\x06~\x0c\x9c\x17\x8b\x
a6\x04qr\xe0\x1f\xf49\x84YD\xd7\xcf\\1v \xe5$\xcc\xf9d\xdb\xb2\x12e\x0f8\x08\xf5\xa8\x8d\xd3\xc74H\xfd\xec\xc8\x
a4\xc1\x9b\xa0R\xf4\x130\xc3\xe4\xe7\xc5\xe9\\fU\t\x0fgK\xe7\xfd]\n\x17\x1de\x95\xcd9%\xd6}&c\x0e\xa10\xdcy.O\x8
b\xc6\x91\xe5\x88]T?$\x14~\x97\xcf\x13\xb0\xcc\x88\xdbN\xd3\x13\xe9\xcd\x9c\xe2\x80\x03%\xcf;\xfdm\x91;\xc2\xc1\
xb1T\xe5\xb7\xb1k5\x9a\x1c\x90\xc0\x92\xf5\x88U\xaaQ\xa03gs\x0fX"U:\xf6C<\xe0\xedS\xdfE\xcb\xf3\x06\x03\x83\xcf8
\x0c\x85\x03\xfc\xad-Ku=\xa8\xd6lY\x91su%\n(\x15\xaa\xba)\'=1+\x18F\x88\x94c\x9e'

print(len(st))
#1696
def MyOwnCBC(key, plain):
  cipher_txt = b""
  cipher = AES.new(key, AES.MODE_ECB)
  cipher_txt=cipher.decrypt(plain)
#b'_be_fooled_by_yourself}~~~~~~~~~~'
#b"s.\n\nAh you found it~ afctf{Don't"
#b'as authenticated encryption mode'
#b'an efficient way, and are known '
#b'identiality and authenticity in '
#b' modes of operation combine conf'
#b' cryptographic goal. Some modern'
#b'otection as an entirely separate'
#b'evelopment regarded integrity pr'
  return cipher_txt
#for i in range(len(st)//32):
flag=""
for i in range(1,10):
  plain=MyOwnCBC(st[(52-i-1)*32:(52-i)*32],st[(52-i)*32:(53-i)*32])
  flag=plain.decode()+flag
print(flag)
```

其实我个人的话就写到cipher_txt就行但是嘛为了美观嘛，运行得到

```
evelopment regarded integrity protection as an entirely separate cryptographic goal. Some modern modes of operat
ion combine confidentiality and authenticity in an efficient way, and are known as authenticated encryption mode
s.

Ah you found it~ afctf{Don't_be_fooled_by_yourself}~~~~~~~~~
```

138.LeftOrRight

查看题目jpg文件，用010打开可以发现开头和结尾有十六进制，我们在线转文本得到

```
f09e54c1bad2x38mvyg7wzlsuhkijnop
905e4c1fax328mdyvg7wbsuhklijznop
```

题目提示`Left?Middle?No, I want right! (flag is right? ! )`
字母相同...顺序不同，再分析一下题目，left，mid都不要，只要right，然后这个地方又是一个树，二叉树1
前序遍历，中序遍历，后序遍历，
前尝试把文件尾部的字符串交上去，不对.
那就是第一个是前序遍历，第二个是中序，求后序遍历：

```python
def get_after_deep(pre, mid, a):#已知前中，求后，a就是后序
    if len(pre) == 1:
        a.append(pre[0])
        return
    if len(pre) == 0:
        return
    root = pre[0]
    root_index = mid.index(root)
    get_after_deep(pre[1:root_index+1], mid[:root_index], a)
    get_after_deep(pre[root_index+1:], mid[root_index+1:], a)
    a.append(root)
    return a


def get_hou():
    pre=input("请依次输入前序遍历、中序遍历的结果，以换行分割:\n")
    mid=input()
    pre_list=list(pre)
    mid_list=list(mid)
    a=[]
    res_list=get_after_deep(pre,mid,a)
    res="".join(res_list)
    print("后序遍历为:",res)
get_hou()
```

运行得到`951c4e03xm82yw7gvdakhusjilponzbf`

139.[watevrCTF 2019]ECC-RSA

查看题目

```python
from fastecdsa.curve import P521 as Curve
from fastecdsa.point import Point
from Crypto.Util.number import bytes_to_long, isPrime
from os import urandom
from random import getrandbits

def gen_rsa_primes(G):
 urand = bytes_to_long(urandom(521//8))
 while True:
  s = getrandbits(521) ^ urand

  Q = s*G
  if isPrime(Q.x) and isPrime(Q.y):
   print("ECC Private key:", hex(s))
   print("RSA primes:", hex(Q.x), hex(Q.y))
   print("Modulo:", hex(Q.x * Q.y))
   return (Q.x, Q.y)


flag = int.from_bytes(input(), byteorder="big")

ecc_p = Curve.p
a = Curve.a
b = Curve.b

Gx = Curve.gx
Gy = Curve.gy
G = Point(Gx, Gy, curve=Curve)


e = 0x10001
p, q = gen_rsa_primes(G)
n = p*q


file_out = open("downloads/ecc-rsa.txt", "w")

file_out.write("ECC Curve Prime: " + hex(ecc_p) + "\n")
file_out.write("Curve a: " + hex(a) + "\n")
file_out.write("Curve b: " + hex(b) + "\n")
file_out.write("Gx: " + hex(Gx) + "\n")
file_out.write("Gy: " + hex(Gy) + "\n")

file_out.write("e: " + hex(e) + "\n")
file_out.write("p * q: " + hex(n) + "\n")

c = pow(flag, e, n)
file_out.write("ciphertext: " + hex(c) + "\n")
```

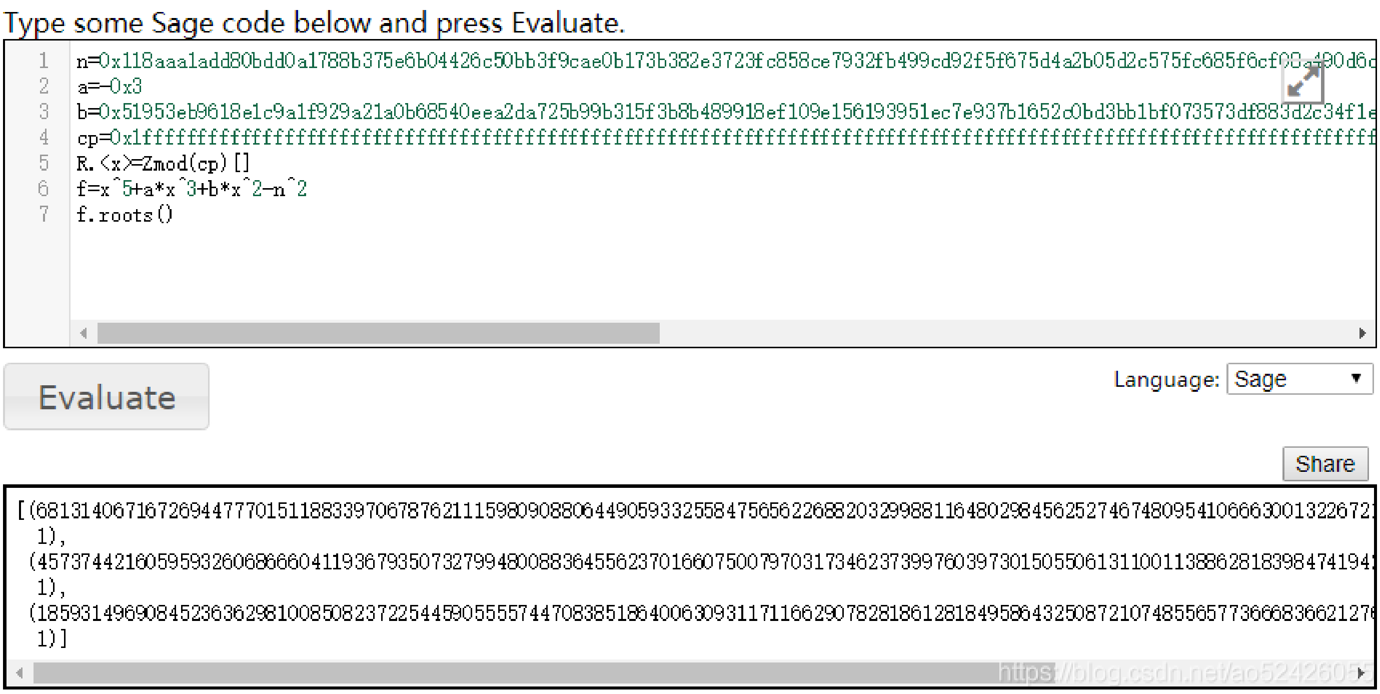
```
ECC Curve Prime: 0x1fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
Curve a: -0x3
Curve b: 0x51953eb9618e1c9a1f929a21a0b68540eea2da725b99b315f3b8b489918ef109e156193951ec7e937b1652c0bd3bb1bf073573df883d2c34f1ef451fd46b503f00
Gx: 0xc6858e06b70404e9cd9e3ecb662395b4429c648139053fb521f828af606b4d3dbaa14b5e77efe75928fe1dc127a2ffa8de3348b3c1856a429bf97e7e31c2e5bd66
Gy: 0x11839296a789a3bc0045c8a5fb42c7d1bd998f54449579b446817afbd17273e662c97ee72995ef42640c550b9013fad0761353c7086a272c24088be94769fd16650
e: 0x10001
p * q: 0x118aaa1add80bdd0a1788b375e6b04426c50bb3f9cae0b173b382e3723fc858ce7932fb499cd92f5f675d4a2b05d2c575fc685f6cf08a490d6c6a8a6741e8be4572adfcba233da791ccc0aee033677b72788d57004a776909f6d699a0164af514728431b5aed704b289719f09d591f5c1f9d2ed36a58448a9d57567bd232702e9b28f
ciphertext: 0x3862c872480bdd067c0c68cfee4527a063166620c97cca4c99baff6eb0cf5d42421b8f8d8300df5f8c7663adb5d21b47c8cb4ca5aab892006d7d44a1c5b5f5242d88c6e325064adf9b969c7dfc52a034495fe67b5424e1678ca4332d59225855b7a9cb42db2b1db95a90ab6834395397e305078c5baff78c4b7252d7966365afed9e
```

通过分析代码，我们可以知道
这个题是 ECC 和 RSA 的混合加密，尽管用了两种加密方式，但加密并不复杂。

我们只需要求出，p,q的值就能得到 flag 了。点（p，q）是椭圆曲线上的两个点。代入椭圆曲线的方程就能解出 p，q了。
q ^ 2 = p ^ 3 + a * p + b

又因为 n = p * q，将方程左右两边同时乘以 p ^ 2

所以有：n ^ 2 = p ^ 5 + a * p ^ 3 + b * p ^2
ECC加密学习
用sage计算p



得到

```
[(68131406716726944770151188339706787621115980908806449059332558475656226882032998811648029845625274674809541066630013226721309443190963022963143497241622588 5),
 (45737442160595932606866604119367935073279948008836455623701660750079703173462373997603973015055061311001138862818398474194254829189324361390808372469147365577 3666836621276225304019796277975316319238677306 0),
 (18593149690845236362981008508237225445905555744708385186400630931171166290782818612818495864325087210748556577366683662127622530401979627797531631923867730600 5)]
```

我是一个一个试的，第二个符合条件

```
n=0x118aaa1add80bdd0a1788b375e6b04426c50bb3f9cae0b173b382e3723fc858ce7932fb499cd92f5f675d4a2b05d2c575fc685f6cf08a490d6c6a8a6741e8be4572adfcba233da791ccc0aee033677b72788d57004a776909f6d699a0164af514728431b5aed704b289719f09d591f5c1f9d2ed36a58448a9d57567bd232702e9b28f
p=45737442160595932606866604119367935073279948008836455623701660750079703173462373997603973015055061311001138862818398474194254829189324361390808372469147365577
import gmpy2
q=n//p
ciphertext=0x3862c872480bdd067c0c68cfee4527a063166620c97cca4c99baff6eb0cf5d42421b8f8d8300df5f8c7663adb5d21b47c8cb4ca5aab892006d7d44a1c5b5f5242d88c6e325064adf9b969c7dfc52a034495fe67b5424e1678ca4332d59225855b7a9cb42db2b1db95a90ab6834395397e305078c5baff78c4b7252d7966365afed9e
e=0x10001
d=gmpy2.invert(e,(p-1)*(q-1))
import Crypto.Util.number
print(Crypto.Util.number.long_to_bytes(gmpy2.powmod(ciphertext,d,n)))

#b'watevr{factoring_polynomials_over_finite_fields_is_too_ez}'
```

140.[V&N2020 公开赛]Backtrace

查看题目

```
# !/usr/bin/env/python3
import random


flag = "flag{" + ''.join(str(random.getrandbits(32)) for _ in range(4)) + "}"

with open('output.txt', 'w') as f:
    for i in range(1000):
        f.write(str(random.getrandbits(32)) + "\n")

print(flag)
```

和一个数字文件

2.分析

MT19937随机数破解的破解，恕我这个数学白痴理解不能啊。。。

这里只能简单的说一下我理解的“结论”。

（1）MT19937算法生产随机数的过程
1.利用seed初始化624的状态
2.对状态进行旋转
3.根据状态提取伪随机数
（2）逆向 extract_number
extract_number函数，可以发现输出的伪随机数是对state[i]进行了异或，位运算后的结果。预测随机数的题型就是基于对extract_number 函数的逆向，我们可以根据题目输出的随机数逆向extract_number得到对应的n个state（n>624），实际上只需要前624个随机数恢复前624个state，就可以预测此后生成的随机数。

（3）逆向twist
在已知连续624个随机数时(state没有进行twist)，可以还原state，预测后续的随机数。那么如何获得624个随机数之前的随机数呢？此时可以考虑，逆向twist获得前一组的state，进而获得前一组的624个随机数。
此题就是一个逆向twist。

3.解题思路
题目要求恢复前四个随机数（就是flag）。
这里要提一点的是不要纠结那个‘_’，其实跟li没有区别。就是连续取4个随机数的意思。
前面说过。逆向twist我们需要连续的624个随机数才能获得上一组随机数的状态，这里只有1000个随机数，这里的前4位随机数丢失，而前4位随机数产生的新状态对应第624，625，626，627位随机数的状态，而他们的状态是可逆的。所以不需要完整的连续624个随机数，也可以求解完整的state。
说实话还是不大理解，但是大概可以知道就是我们可以用后面产生的1000个随机数，利用随机数状态可逆的问题，反向（backtrace）得到之前产生的随机数。
直接抄袭了大神的脚本：

```python
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/10/25 21:59
# @Author : A.James
# @FileName: exp2.py
from random import Random

# right shift inverse
def inverse_right(res,shift,bits=32):
    tmp = res
    for i in range(bits//shift):
        tmp = res ^ tmp >> shift
    return tmp

# right shift with mask inverse
def inverse_right_values(res,shift,mask,bits=32):
    tmp = res
    for i in range(bits//shift):
        tmp = res ^ tmp>>shift & mask
    return tmp

# left shift inverse
def inverse_left(res,shift,bits=32):
    tmp = res
    for i in range(bits//shift):
        tmp = res ^ tmp << shift
    return tmp

# left shift with mask inverse
def inverse_left_values(res,shift,mask,bits=32):
    tmp = res
    for i in range(bits//shift):
        tmp = res ^ tmp << shift & mask
```

```
        tmp = res ^ tmp << shift & mask
    return tmp

def backtrace(cur):
    high = 0x80000000
    low = 0x7fffffff
    mask = 0x9908b0df
    state = cur
    for i in range(3,-1,-1):
        tmp = state[i+624]^state[i+397]
        # recover Y,tmp = Y
        if tmp & high == high:
            tmp ^= mask
            tmp <<= 1
            tmp |= 1
        else:
            tmp <<=1
        # recover highest bit
        res = tmp&high
        # recover other 31 bits,when i =0,it just use the method again it so beautiful!!!!
        tmp = state[i-1+624]^state[i+396]
        # recover Y,tmp = Y
        if tmp & high == high:
            tmp ^= mask
            tmp <<= 1
            tmp |= 1
        else:
            tmp <<=1
        res |= (tmp)&low
        state[i] = res
    return state

def recover_state(out):
    state = []
    for i in out:
        i = inverse_right(i,18)
        i = inverse_left_values(i,15,0xefc60000)
        i = inverse_left_values(i,7,0x9d2c5680)
        i = inverse_right(i,11)
        state.append(i)
    return state

f = open("output.txt","r").readlines()
c = []
for i in range(1000):
    c.append(int(f[i].strip()))

partS = recover_state(c)
state = backtrace([0]*4+partS)[:624]
# print(state)
prng = Random()
prng.setstate((3,tuple(state+[0]),None))
flag = "flag{" + ''.join(str(prng.getrandbits(32)) for _ in range(4)) + "}"
print(flag)
#flag{188673746538768657392417575392387977135O}
```

这道题没看懂 大佬脚本也一样这题引用wp

141.[GUET-CTF2019]Uncle Sam

查看题目

```
from Crypto.Util.number import *

def generkey(k):
 p, q = getPrime(k), getPrime(k)
 pubkey = p**2 * q
 n = pubkey
 l = (p-1)*(q-1) / gcd(p-1, q-1)
 privkey = inverse(n, l)
 return pubkey, privkey
def encrypt(m, pubkey):
 return pow(bytes_to_long(m), pubkey, pubkey)

# pubkey =  21889679777493782742235156893635998013206982479389971359479655501966818365432754297675816330443544121953522291757647845035629890452680754312068767262659683686052108242322072904107739796066626898662656127971038539820141984554333802666718563555642731961511360253196246368056595052339752085704099140549169550975948737023958120445062059436714042037743606565533509874915584911769620188427084760095781273035668345349146051098599956495551227518916470404489807188827558554203244824665592237480650375207881596544362934314701640574903508412098724895384600602160151968751369275021620275625463165603424649682379576928735887966406195304552683671362433134225798578235295921671012607793826658323800546907273581976465128966121609067703339520919600724959439451513031504176098829200993067519274901022859215604715902909540602181288425881088922561754440479986390398275896120818704297204781935825686634675833727747301606837520631983731722252359 7
# privkey = 14303757900655747216021961969296511745726744290407255356982172073018811616952965195670512462901995519822863278319856490375848851371345806259825556344092255511217123768495790153209472797162044247165662227213387352566488731645104292069911416486468693781413122531359978519088620309905760041735145565413173951069243700195742168945604478173196696901405447282773020437831638880378366752904683207232157596939035698782934754473707666824777264532627710048727493352579535074691099664481266341016040295060060385276129174180167837117298007193872983988483700797427901260473291823498998242583550032001736125671917478516692207666 03
# enc = 14914213913648717673579316397103946223994510198245723622884584311862992316644599577554224744335208890843518412980560661002164408534093460066577230865019218163812262925264901958109034594833182759313264330524688638506907936594053679025939993950606069721001699250740059924785830352260268292144430089416317712922913052264702164307350509442855435423544591624743465213276499345125112024700990206682351152458196347620673384329160126644520356964228656510023054457117784760720047082562008722264753464483604912488238436882681263410946129813087914994347709363606760874903039517285634826863071648770003000827423163685979582972170613751406962723981403100439426372877639463059610195186397454263708211245599395975594753627693827963867200303433058897016161942790581395168119412627472987616463173831124709232955436357547472882593247455836894400619564780837776639964873895532384817591039085880042193906625784463130044047848352635430830883271 98
```

上脚本

```
from Crypto.Util.number import *
'''
def generkey(k):
 p, q = getPrime(k), getPrime(k)
 pubkey = p**2 * q
 n = pubkey
 l = (p-1)*(q-1) / gcd(p-1, q-1)
 privkey = inverse(n, l)
 return pubkey, privkey
def encrypt(m, pubkey):
 return pow(bytes_to_long(m), pubkey, pubkey)

# pubkey =  21889679777493782742235156893635998013206982479389971359479655501966818365432754297675816330443544121953522291757647845035629890452680754312068767262659683686052108242322072904107739796066626898662656127971038539820141984554333802666718563555642731961511360253196246368056595052339752085704099140549169550975948737023958120445062059436714042037743606565533509874915584911769620188427084760095781273035668345349146051098599956495551227518916470404489807188827558554203244824665592237480650375207881596544362934314701640574903508412098724895384600602160151968751369275021620275625463165603424649682379576928735887966406195304552683671362433134225798578235295921
```

```
710126077938266583238005469072735819764651289666121609067703339520919600724959439451513031504176098829200993067519274901022859215604715902909540602181288425881088922561754440479986390398275896120818704297204781935825686634675833727747301606837520631983731722252359 7
# privkey = 14303757900655747216021961969296511745726744290407255356982172073018811616952965195670512462901995519822863278319856490375848851371345806259825556344092255511217123768495790153209472797162044247165662227213387352566488731645104292069911416486468693781413122531359978519088620309905760041735145565413173951069243700195742168945604478173196696901405447282773020437831638880378366752904683207232157596939035698782934754473707666824777264532627710048727493352579535074691099664481266341016040295060060385276129174180167837117298007193872983988483700797427901260473291823498998242583550032001736125671917478516692207666037
# enc = 14914213913648717673579316397103946223994510198245723622884584311862992316644599577554224744335208890843518412980560661002164408534093460066577230865019218163812262925264901958109034594833182759313264330524688638506907936594053679025939993950606069721001699250740059924785830352260268292144430089416317712922913052264702164307350509442855435423544591624743465213276499345125112024700990206682351152458196347620673384329160126644520356964228656510023054457117784760720047082562008722264753464483604912488238436882681263410946129813087914994347709363606760874903039517285634826863071648770003000827423163685979582972170613751406962723981403100439426372877639463059610195186397454263708211245599395975594753627693827963867200303433058897016161942790581395168119412627472987616463173831124709232955436357547472882593247455836894400619564780837776639964873895532384817591039085880042193906625784463130044047848352635430830883271987
'''
def gcd(a, b):
    if a < b:
        a, b = b, a

    while b != 0:
        temp = a % b
        a = b
        b = temp

    return a


def generkey(k):
    p, q = getPrime(k), getPrime(k)
    pubkey = p ** 2 * q
    n = pubkey
    l = (p - 1) * (q - 1) / gcd(p - 1, q - 1)
    privkey = inverse(n, l)
    return pubkey, privkey


def encrypt(m, pubkey):
    return pow(bytes_to_long(m), pubkey, pubkey)


pubkey = 2188967977749378274223515689363599801320698247938997135947965550196681836543275429767581633044354412195352229175764784503562989045268075431206876726265968368605210824232207290410773979606662689866265612797103853982014198455433380266671856355564273196151136025319624636805659505233975208570409914054916955097594873702395812044506205943671404203774360656553350987491558491176962018842708476009578127303566834534914605109859995649555122751891647040448980718882755855420324482466559223748065037520788159654436293431470164057490350841209872489538460060216015196875136927502162027562546316560342464968237957692873588796640619530455268367136243313422579857823529592167101260779382665832380054690727358197646512896661216090677033395209196007249594394515130315041760988292009930675192749010228592156047159029095406021812884258810889225617544404799863903982758961208187042972047819358256866346758337277473016068375206319837317222523597
privkey = 14303757900655747216021961969296511745726744290407255356982172073018811616952965195670512462901995519822863278319856490375848851371345806259825556344092255511217123768495790153209472797162044247165662227213387352566488731645104292069911416486468693781413122531359978519088620309905760041735145565413173951069243700195742168945604478173196696901405447282773020437831638880378366752904683207232157596939035698782934754473707666824777264532627710048727493352579535074691099664481266341016040295060060385276129174180167837117298007193872983988483700797427901260473291823498998242583550032001736125671917478516692207666037
```

```
enc = 14914213913648717673579316397103946223994510198245723622884584311862992316644599577554224744335208890843518412980560661002164408534093460066577230865019218163812262925264901958109034594833182759313264330524688638506907936594053679025939993950606069721001699250740059924785830352260268292144430089416317712922913052264702164307350509442855435423544591624743465213276499345125112024700990206682351152458196347620673384329160126644520356964228656510023054457117784760720047082562008722264753464483604912488238436882681263410946129813087914994347709363606760874903039517285634826863071648770003000827423163685979582972170613751406962723981403100439426372877639463059610195186397454263708211245599395975594753627693827963867200303433058897016161942790581395168119412627472987616463173831124709232955436357547472882593247455836894400619564780837776639964873895532384817591039085880042193906625784463130044047848352635430830883271980
n = pubkey
d = privkey

pow2 = pow(2, n * d, n)
assert pow2 != 2

g = gcd(pow2 - 2, n)
assert g != 1 and g != n
assert n % g == 0

pq = g
m=pow(enc, d, pq)
flag=long_to_bytes(m)
print(flag)
```

运行得到`flag{61e19444-7afb-11e9-b704-4ccc6adfc6f0}`

142.[b01lers2020]safety_in_numbers

查看题目

pubke.pem储存了n和e。

flag.enc是加密结果

enc.py是加密脚本

用脚本读pubke.pem里面的ne,n太大了，写不进来就截个图

```
884593748129874276229881884365121245481928659765321188101796249130196981035957525093896541510262073244610370293203368204421327536466899599597908657864231075018594871559095421993034513569358323371713561795500485752365110614671835806325629900793622189018217520315987732399429713588744163993719079523004669487454879641388227546931660279196017224749697856398497225841109618285669585955254150228691548915616553
```

n很大，e很小。那么m就是c直接开e次方即可。

```
import gmpy2

e = 65537
with open('flag.enc','rb') as f:
    cipher = f.read()
c = int.from_bytes(cipher, byteorder='little')
m = gmpy2.iroot(c,e)[0]
print(m)
print(hex(m))#转换十六进制
print(bytes.fromhex(hex(m)[2:])[::-1])#ascll码
```

运行得到`pctf{!fUtuR3_pR00f}`

143.[AFCTF2018]一道有趣的题目

```
#加密代码
def encrypt(plainText):
    space = 10
    cipherText = ""
    for i in range(len(plainText)):
        if i + space < len(plainText) - 1:
            cipherText += chr(ord(plainText[i]) ^ ord(plainText[i + space]))
        else:
            cipherText += chr(ord(plainText[i]) ^ ord(plainText[space]))
        if ord(plainText[i]) % 2 == 0:
            space += 1
        else:
            space -= 1
    return cipherText
```

首先要处理space部分,space每一次变化都和每个明文字符的最后一位有关,明文的长度和密文的长度是一样的,因此可以先求出密文的长度,发现是28,于是尝试爆破每个明文字符的最后一位.有228种可能.

处理完space部分后,发现每一次加密都是一个方程,一共有28个方程,也一共有28个未知数,但是前几个未知数我们可以猜测一下为‘a’ ‘f’ ‘c’ ‘t’ ‘f’ ‘{’
这样我们很快就可以将所有的未知数给求解出来了.

```
def crackit():
    '''
    暴力求出每个明文字符的最后一位
    :return: 每个明文字符最后一位组成的字符串
    '''
    tag=b'\x15\x12\r\x1a\n\x08\x10\x01\n\x03\x1d>1\x00\r\x1d\x17\r\x17;\r\x17;\x0c\x07\x06\x02\x06'
    for x in range(268435456):
        temp=bin(x)[2:].zfill(28)
        space = 10
        flag=0
        for i in range(len(temp)):
            if i + space < len(temp) - 1:
                tempx= int(temp[i]) ^ int(temp[i + space])
            else:
                tempx= int(temp[i]) ^ int(temp[space])
            if tempx!=(tag[i]%2):
                flag=1
                break
            elif int(temp[i]) % 2 == 0:
                space += 1
            else:
                space -= 1
        if flag!=1:
            print(temp)

#temp ='1010011010010101111111101001'

def crackit_2():
    '''
    解出每个方程中的明文字符的序号.
    :return: 每个方程中的明文字符的序号组成的列表.
    '''
    tag=b'\x15\x12\r\x1a\n\x08\x10\x01\n\x03\x1d>1\x00\r\x1d\x17\r\x17;\r\x17;\x0c\x07\x06\x02\x06'
    temp ='1010011010010101111111101001'
```

```
    space = 10
    g=[]
    for i in range(len(temp)):
        if i + space < len(temp) - 1:
            tempx = int(temp[i]) ^ int(temp[i + space])
            g.append((i,i+space))
        else:
            tempx = int(temp[i]) ^ int(temp[space])
            g.append((i,space))
        if int(temp[i]) % 2 == 0:
            space += 1
        else:
            space -= 1
    return g
def solve(g,m):
    '''
    解明文方程组
    :param g: 明文方程组
    :param m :密文
    :return: 明文
    '''
    plain=[ord('a'),ord('f'),ord('c'),ord('t'),ord('f'),ord('{')]+[-1]*22#明文未知数
    while -1 in plain:
        for i in range(28):
            if plain[i] != -1:
                for j in range(len(g)):
                    if g[j][0] == i or g[j][1] == i:
                        if g[j][0] == i:
                            plain[g[j][1]] = m[j] ^ plain[i]
                        if g[j][1] == i:
                            plain[g[j][0]] = m[j] ^ plain[i]
    return plain

#[97, 102, 99, 116, 102, 123, 99, 114, 121, 112, 116, 97, 110, 97, 108, 121, 115, 105, 115, 95, 105, 115, 95, 10
4, 97, 114, 100, 125]

tag=b'\x15\x12\r\x1a\n\x08\x10\x01\n\x03\x1d>1\x00\r\x1d\x17\r\x17;\r\x17;\x0c\x07\x06\x02\x06'#密文
g=crackit_2()
print(g)
plain=solve(g,tag)
print(plain)
flag=''
for i in plain:
    flag+=chr(i)
print(flag)


#afctf{cryptanalysis_is_hard}
```

144.[NCTF2019]Reverse

查看题目

```
import os
import pyDes


flag = "NCTF{****************************************}"
key = os.urandom(8)

d = pyDes.des(key)
cipher = d.encrypt(flag.encode())

with open('cipher', 'wb') as f:
    f.write(cipher)

# Leak: d.Kn[10] == [0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1
, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1]
```

知道一个子密钥,尝试爆破所有可能的子密钥的情况,一共有28 种.
需要注意的细节:
** 每轮循环左移的位数不一定相同**
** d.Kn[10] 已知,说明已经进行了11轮了(每一轮生成一个子密钥),因此要做sum(movnum[:11])次逆运算**
这题我也是看了大佬的wp

```
import copy
import pyDes
key='********'
d=pyDes.des(key)
key10=[0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1]
PC1=[56, 48, 40, 32, 24, 16, 8, 0, 57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18, 10, 2, 59, 51, 43, 35,
62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 60, 52, 44, 36, 28, 20, 12, 4, 27, 19, 11, 3]
PC2=[13, 16, 10, 23, 0, 4, 2, 27, 14, 5, 20, 9, 22, 18, 11, 3, 25, 7, 15, 6, 26, 19, 12, 1, 40, 51, 30, 36, 46,
54, 29, 39, 50, 44, 32, 47, 43, 48, 38, 55, 33, 52, 45, 41, 49, 35, 28, 31]
movnum = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]#对应16轮中每一轮的循环左移位数
def gen_key(C1,D1,k):
    tempc=C1
    tempd=D1
    for i in range(k):
        tempc = tempc[1:] + tempc[:1]
        tempd = tempd[1:] + tempd[:1]
    tempCD1=tempc+tempd
    tempkey=[]
    for i in range(len(PC2)):
        tempkey.append(tempCD1[PC2[i]])
    return (tempkey,tempCD1)#轮运算得到下一轮子密钥
def re_gen_key(C1,D1):
    tempc=C1[-1:]+C1[:-1]
    tempd=D1[-1:]+D1[:-1]
    tempCD1=tempc+tempd
    return tempCD1 #轮运算得到上一轮CD
def get_key(CD):
    tempkey=[]
    for i in range(len(PC2)):
        tempkey.append(CD[PC2[i]])
    return tempkey
def RE_pc2():
    CD1=['*']*56
    for i in range(len(PC2)):
```

```
        CD1[PC2[i]]=key10[i]#初步还原CD1
    results=[]
    for i in range(256):
        temp=bin(i)[2:].zfill(8)
        tempi=copy.deepcopy(CD1)
        d=0
        for j in range(len(tempi)):
            if tempi[j]=='*':
                tempi[j]=eval(temp[d])
                d=d+1
        results.append(tempi)
    return results
f=open('cipher','rb')
flag_enc=f.read()
results=RE_pc2()
for i in range(len(results)):
    temp=results[i]
    for j in range(sum(movnum[:11])):
        temp=re_gen_key(temp[:28],temp[28:])
    tempK=[]
    Z=temp
    for j in range(16):
        tempx=gen_key(Z[:28],Z[28:],movnum[j])
        tempK.append(tempx[0])
        Z=tempx[1]
    d.Kn=tempK
    print(d.decrypt(flag_enc))
#b'NCTF{1t_7urn3d_0u7_7h47_u_2_g00d_@_r3v3rs3_1snt}'
```

145.[INSHack2018]Crypt0r part 1

查看题目

```
# Crypt0r part 1
Our IDS detected an abnormal behavior from one of our user. We extracted this pcap, could you have a look at it?

<a href="http://crypt0r.challenge-by.ovh/ids_alert_24032018.pcap">http://crypt0r.challenge-by.ovh/ids_alert_2403
2018.pcap</a>
```

下载后是个cap。打开后流量很简单，直接跟踪TCP流，得到

```
CRYPT0R_SEED:58
CRYPT0R:PMSFADNIJKBXQCGYWETOVHRULZSELYO0E_PSB
SELYO0E:PXX_NGGFSELYO0E:NAO_HJSOJQ_JF>{A2FS3118-0399-48S7-857S-43D9528DD98F}
SELYO0E:HJSOJQ_JF_JT>....SELYO0E:NAO_DJCPX_QTN
SELYO0E:DJCPX_QTN_JT>!!! PXX LGVE DJXAT IPHA MAAC ACSELYOAF !!!

Selyo0e toegba mpsb pcf lgv ngo dvsb*f mvffl. Lgv spccgo faselyo lgve fpop ausayo jd lgv ypl qa $500. #TIGRQAOIA
QGCAL pcf J rjxx njha lgv mpsb lgve fpop.

Dgxxgr oiata jctoevsojgct:
- Jctopxx oia oge megrtae, pcf ng og gve yplqaco yxpodgeq: iooy://bu4ifi2zg5etosvk.gcjgc (YSJ-FTT pyyeghaf gds m
eg).
- Acoae lgve yaetgcpx bal: JCTP{mW9CLVlPjpUtbZFdccPioVV01jdaUeGv}

Oipcbt dge vtjcn ql epctgqrpea.

Rjoi xgha,
Selyo0qpc
```

```
string1 = "PMSFADNIJKBXQCGYWETOVHRULZpmsfadnijkbxqcgywetovhrulz"
string2 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
str1 = """
CRYPT0R_SEED:58
CRYPT0R:PMSFADNIJKBXQCGYWETOVHRULZSELYO0E_PSB
SELYO0E:PXX_NGGFSELYO0E:NAO_HJSOJQ_JF>{A2FS3118-0399-48S7-857S-43D9528DD98F}
SELYO0E:HJSOJQ_JF_JT>....SELYO0E:NAO_DJCPX_QTN
SELYO0E:DJCPX_QTN_JT>!!! PXX LGVE DJXAT IPHA MAAC ACSELYOAF !!!

Selyo0e toegba mpsb pcf lgv ngo dvsb*f mvffl. Lgv spccgo faselyo lgve fpop ausayo jd lgv ypl qa $500. #TIGRQAOIA
QGCAL pcf J rjxx njha lgv mpsb lgve fpop.

Dgxxgr oiata jctoevsojgct:
- Jctopxx oia oge megrtae, pcf ng og gve yplqaco yxpodgeq: iooy://bu4ifi2zg5etosvk.gcjgc (YSJ-FTT pyyeghaf gds m
eg).
- Acoae lgve yaetgcpx bal: JCTP{mW9CLVlPjpUtbZFdccPioVV01jdaUeGv}

Oipcbt dge vtjcn ql epctgqrpea.

Rjoi xgha,
Selyo0qpc
"""
print (str1.translate(str.maketrans(string1,string2)))
```

运行得到

```
NWPAS0W_CRRF:58
NWPAS0W:ABCDEFGHIJKLMNOPQRSTUVWXYZCRYPT0R_ACK
CRYPT0R:ALL_GOODCRYPT0R:GET_VICTIM_ID>{E2DC3118-0399-48C7-857C-43F9528FF98D}
CRYPT0R:VICTIM_ID_IS>....CRYPT0R:GET_FINAL_MSG
CRYPT0R:FINAL_MSG_IS>!!! ALL YOUR FILES HAVE BEEN ENCRYPTED !!!

Crypt0r stroke back and you got fuck*d buddy. You cannot decrypt your data except if you pay me $500. #SHOWMETHE
MONEY and I will give you back your data.

Follow these instructions:
- Install the tor browser, and go to our payment platform: http://kx4hdh2zo5rstcuj.onion (PCI-DSS approved ofc b
ro).
- Enter your personal key: INSA{bQ9NYUyAiaXskZDfnnAhtUU01ifeXrOu}

Thanks for using my ransomware.

With love,
Crypt0man


进程完成，退出码 0
```

key就是flag

146.[XNUCA2018]baby_crypto

查看题目

```
The 26 letters a, b, c, ..., y, z correspond to the integers 0, 1, 2, ..., 25
len(key_a) = m
len(key_k) = n
c[i] = (p[i] * key_a[i % m] + key_k[i % n]) % 26

p is plain text, only lowercase letters are refered to.
c is encrypted text

I have appended the flag at the end of plain text, the format of which is like 'flagis......'
Now you have the encrypted text, Good luck!
```

看了两个大佬的wp1，wp2总结得到下面的脚本

```
#重合指数的应用:
import gmpy2
c=open('encrypted_message.txt','r').read()
best_index=0.065
sum=0
dic_index={'a': 0.08167,'b': 0.01492,'c': 0.02782,'d':0.04253,'e': 0.12702,'f':0.02228,'g': 0.02015,'h':0.06094,
'i':0.06966,'j':0.00153,'k':0.00772,'l':0.04025,'m':0.02406,'n':0.06749,'o':0.07507,'p':0.01929,'q':0.00095,'r':
0.05987,'s':0.06327,'t':0.09056,'u':0.02758,'v':0.00978,'w':0.02360,'x':0.00150,'y':0.01974,'z':0.00074}
def index_of_coincidence(s):
    '''
    计算字符串的重合指数(所有字母出现频率的平方和)
    :param s: 给定字符串
    :return: 重合指数
    '''
    alpha='abcdefghijklmnopqrstuvwxyz'#给定字母表
    freq={}#统计字母频率(frequency)
    for i in alpha:
```

```
    for i in alpha:
        freq[i]=0
    #先全部初始化为0
    for i in s:
        freq[i]=freq[i]+1
    #统计频率
    index=0
    for i in alpha:
        index = index + (freq[i] * (freq[i] - 1)) / (len(s) * (len(s) - 1))
    return index
def index_of_coincidence_m(s):
    '''
    计算明文s中的各字母的频率与英文字母中的频率的吻合程度.
    :param s:明文s
    :return:吻合程度
    '''
    alpha = 'abcdefghijklmnopqrstuvwxyz'  # 给定字母表
    freq = {}  # 统计字母频率(frequency)
    for i in alpha:
        freq[i] = 0
    # 先全部初始化为0
    for i in s:
        freq[i] = freq[i] + 1
    # 统计频率
    index = 0
    for i in alpha:
        index = index + freq[i] / len(s) * dic_index[i]
    return index
def get_cycle(c):
    '''
    求出最符合统计学的m,n的最小公共周期,方法为通过爆破足够大的周期样本,观察成倍出现的周期.
    计算方法为解出每一个子密文段的重合指数和然后求平均值 再与最佳重合指数相减 误差在0.01以内.
    :param c: 密文
    :return: 公共周期列表
    '''
    cycle=[]
    for i in range(1,100):
        average_index=0#平均重合指数初始化为0
        for j in range(i):
            s = ''.join(c[j+i*x] for x in range(0,len(c)//i))
            index=index_of_coincidence(s)
            average_index+=index
        average_index=average_index/i-best_index
        if abs(average_index)<0.01:
            cycle.append(i)
    return cycle
cycle=get_cycle(c)
print(cycle)#[6, 12, 18, 24, 30, 36, 42, 48, 54, 60, 66, 72, 78, 84, 90, 96]

#通过计算得到cycle都是6的倍数,因此cycle最小很有可能为6

cycle=6

#开始爆破keys

def decrypt(c,i,j):
    '''
    通过i,j解出与之相对应的密文段
    :param c: 密文段
    :param i:与明文相乘的key
```

```
    :param j: 位移j(维吉尼亚密码)
    :return: 明文段
    '''
    alpha = 'abcdefghijklmnopqrstuvwxyz'
    m=''
    for x in c:
        m+=alpha[((alpha.index(x)-j)*gmpy2.invert(i,26))%26]
    return m
def get_key(c):
    '''
    得到某一密文段的单个字符key i j
    方法为暴力枚举所有的可能性,找到最符合统计学规律的 i,j 即该密文段的重合指数与最佳重合指数误差小于0.01
    :param c: 密文段
    :return: i,j
    '''
    for i in range(26):
        if  gmpy2.gcd(i,26)!=1:#i对26的逆元不只一个,造成明文不唯一,因此不符合条件.
            continue
        for j in range(26):
            m=decrypt(c,i,j)
            index=index_of_coincidence_m(m)
            if abs(index-0.065)<0.01:
                return (i,j)
def get_all_key(s,cycle):
    '''
    得到一个周期内的所有的密文段的key
    :param s: 原密文
    :param cycle: 周期
    :return: 无
    '''
    for i in range(cycle):
        temps=''.join([s[i+x*cycle] for x in range(0,len(s)//cycle)])
        print(get_key(temps))
get_all_key(c,6)
# (19, 10)
# (7, 9)
# (23, 3)
# (19, 24)
# (7, 14)
# (23, 15)
#此时我们大致可以推测出:keya=[19,7,23],keyb=[10,9,3,24,14,15],因此根据题目给的式子,我们就可以还原出明文了.
plaintext=''
keya=[19,7,23]
keyb=[10,9,3,24,14,15]
len_a=len(keya)
len_b=len(keyb)
alpha='abcdefghijklmnopqrstuvwxyz'
for i in range(len(c)):
    plaintext+=alpha[((alpha.index(c[i])-keyb[i%len_b])*gmpy2.invert(keya[i%len_a],26))%26]
print(plaintext)
#sandandfoambykahlilgibraniamforeverwalkinguponthese
```

```
```

canweexpresstheancientofdaysinuswithonlythesoundsofouryesterdaysthesphinxspokeonlyonceandthesphinxsaidagrainofsa
ndisadesertandadesertisagrainofsandandnowletusallbesilentagainiheardthesphinxbutididnotunderstandLongdidilieinth
edustofegyptsilentandunawareoftheseasonsthenthesungavemebirthandiroseandwalkeduponthebanksofthenilesingingwiththe
edaysanddreamingwiththenightsandnowthesunthreadsuponmewithathousandfeetthatimaylieagaininthedustofegyptbutbehold
amarvelandariddletheverysunthatgatheredmecannotscattermestillerectamiandsureoffootdoiwalkuponthebanksofthenilere
membranceisaformofmeetingforgetfulnessisaformoffreedomwemeasuretimeaccordingtothemovementofcountlesssunsandtheym
easuretimebylittlemachinesintheirlittlepocketsnowtellmehowcouldweevermeetatthesameplaceandthesametimespaceisnots
pacebetweentheearthandthesuntoonewholooksdownfromthewindowsofthemilkywayhumanityisariveroflightrunningfromtheexe
ternitytoeternitydonotthespiritswhodwellintheetherenvymanhispainonmywaytotheholycityimetanotherpilgrimandiaskedh
imisthisindeedthewaytotheholycityandhesaidfollowmeandyouwillreachtheholycityinadayandanightandifollowedhimandwew
alkedmanydaysandmanynightsyetwedidnotreachtheholycityandwhatwastomysurprisehebecameangrywithmebecausehehadmisled
memakemeohgodthepreyofthelionereyoumaketherabbitmypreyonemaynotreachthedawnsavebythepathofthenightmyhousesaystom
edonotleavemeforheredwellsyourpastandtheroadsaystomecomeandfollowmeforiamyourfutureandisaytobothmyhouseandtheroa
dihavenopastnorhaveiafutureifistayherethereisagoinginmystayingandifigothereisastayinginmygoingonlyloveanddeathwi
llchangeallthingshowcanilosefaithinthejusticeoflifewhenthedreamsofthosewhosleepuponfeathersarenotmorebeautifulth
anthedreamsofthosewhosleepupontheearthstrangethedesireforcertainpleasuresisapartofmypainseventimeshaveidespisedm
ysoulthefirsttimewhenisawherbeingmeekthatshemightattainheightthesecondtimewhenisawherlimpingbeforethecrippledthe
thirdtimewhenshewasgiventochoosebetweenthehardandtheeasyandshechosetheeasythefourthtimewhenshecommittedawrongand
comfortedherselfthatothersalsocommitwrongthefifthtimewhensheforboreforweaknessandattributedherpatiencetostrength
thesixthtimewhenshedespisedtheuglinessofafaceandknewnotthatitwasoneofherownmasksandtheseventhtimewhenshesangason
gofpraiseanddeemeditavirtueiamignorantofabsolutetruthbutiamhumblebeforemyignoranceandthereinliesmyhonorandmyrewa
rdthereisaspacebetweenmansimaginationandmansattainmentthatmayonlybetraversedbyhislongingparadiseistherebehindtha
tdoorinthenextroombutihavelostthekeyperhapsihaveonlymislaidityouareblindandiamdeafanddumbsoletustouchhandsandund
erstandthesignificanceofmanisnotinwhatheattainsbutratherinwhathelongstoattainsomeofusarelikeinkandsomelikepapera
ndifitwerenotfortheblacknessofsomeofussomeofuswouldbedumbandifitwerenotforthewhitenessofsomeofussomeofuswouldbeb
lindgivemeanearandiwillgiveyouavoiceourmindisaspongeourheartisastreamisitnotstrangethatmostofuschoosesuckingrath
erthanrunningwhenyoulongforblessingsthatyoumaynotnameandwhenyougrieveknowingnotthecausethenindeedyouaregrowingwi
thallthingsthatgrowandrisingtowardyourgreaterselfwhenoneisdrunkwithavisionhedeemshisfaintexpressionofittheverywi
neyoudrinkwinethatyoumaybeintoxicatedandidrinkthatitmaysobermefromthatotherwinewhenmycupisemptyiresignmyselftoit
semptinessbutwhenitishalffullliresentitshalffulnesstherealityoftheotherpersonisnotinwhatherevealstoyoubutinwhathe
cannotrevealtoyouthereforeifyouwouldunderstandhimlistennottowhathesaysbutrathertowhathedoesnotsayhalfofwhatisayi
smeaninglessbutisayitsothattheotherhalfmayreachyouasenseofhumourisasenseofproportionmyloneliesswasbornwhenmenpr
aisedmytalkativefaultsandblamedmysilentvirtueswhenlifedoesnotfindasingertosingherheartsheproducesaphilosophertos
peakhermindatruthistobeknownalwaystobeutteredsometimestherealinusissilenttheacquiredistalkativethevoiceoflifeinm
ecannotreachtheearoflifeinyoubutletustalkthatwemaynotfeellonelywhentwowomentalktheysaynothingwhenonewomanspeaksss
herevealsalloflifefrogsmaybellowlouderthanbullsbuttheycannotdragtheploughinthefieldnotturnthewheelofthewinepress
andoftheirskinsyoucannotmakeshoesonlythedumbenvythetalkativeifwintershouldsayspringisinmyheartwhowouldbelievewin
tereveryseedisalongingshouldyoureallyopenyoureyesandseeyouwouldbeholdyourimageinallimagesandshouldyouopenyourear
sandlistenyouwouldhearyourownvoiceinallvoicesittakestwoofustodiscovertruthonetoutteritandonetounderstanditthough
thewaveofwordsisforeveruponusyetourdepthisforeversilentmanyadoctrineislikeawindowpaneweseetruththroughitbutitdiv
idesusfromtruthnowletusplayhideandseekshouldyouhideinmyheartitwouldnotbedifficulttofindyoubutshouldyouhidebehind
yourownshellthenitwouldbeuselessforanyonetoseekyouawomanmayveilherfacewithasmilehownobleisthesadheartwhowouldsin
gajoyoussongwithjoyousheartshewhowouldunderstandawomanordissectgeniusorsolvethemysteryofsilenceistheverymanwhowo
uldwakefromabeautifuldreamtositatabreakfasttableiwouldwalkwithallthosewhowalkiwouldnotstandstilltowatchtheproces
sionpassingbyyouowemorethangoldtohimwhoservesyougivehimofyourheartorservehimnaywehavenotlivedinvainhavetheynotbu
ilttowersofourbonesletusnotbeparticularandsectionalthepoetsmindandthescorpionstailriseingloryfromthesameeartheve
rydragongivesbirthtoastgeorgewhoslaysittreesarepoemsthattheearthwritesupontheskywefellthemdownandturnthemintopap
erthatwemayrecordouremptinessshouldyoucaretowriteandonlythesaintsknowwhyyoushouldyoumustneedshaveknowledgeandart
andmusictheknowledgeofthemusicofwordstheartofbeingartlessandthemagicoflovingyourreaderstheydiptheirpensinourhear
tsandthinktheyareinspiredshouldatreewriteitsautobiographyitwouldnotbeunlikethehistoryofaraceifiweretochoosebetwe
enthepowerofwritingapoemandtheecstasyofapoemunwritteniwouldchoosetheecstasyitisbetterpoetrybutyouandallmyneighbo
rsagreethatialwayschoosebadlypoetryisnotanopinionexpresseditisasongthatrisesfromableedingwoundorasmilingmouthwor
dsaretimelessyoushouldutterthemorwritethemwithaknowledgeoftheirtimelessnessapoetisadethronedkingsittingamongthea
shesofhispalacetryingtofashionanimageoutoftheashespoetryisadealofjoyandpainandwonderwithadashofthedictionaryinva
inshallapoetseekthemotherofthesongsofhisheartonceisaidtoapoetweshallnotknowyourworthuntilyoudieandheansweredsayi
ngyesdeathisalwaystherevealerandifindeedyouwouldknowmyworthitisthatihavemoreinmyheartthanuponmytongueandmoreinmy
desirethaninmyhandifyousingofbeautythoughaloneintheheartofthedesertyouwillhaveanaudiencepoetryiswisdomthatenchan
tstheheartwisdomispoetrythatsingsinthemindifwecouldenchantmansheartandatthesametimesinginhismindthenintruthhewou
ldliveintheshadowofgodinspirationwillalwayssinginspirationwillneverexplainweoftensinglullabiestoourchildrenthatw
eourselvesmaysleepallourwordsarebutcrumbsthatfalldownfromthefeastofthemindthinkingisalwaysthestumblingstonetopoe

tryagreatsingerishewhosingsoursilenceshowcanyousingifyourmouthbefilledwithfoodhowshallyourhandberaisedinblessing
ifitisfilledwithgoldtheysaythenightingalepierceshisbosomwithathornwhenhesingshislovesongsodoweallhowelseshouldwe
singgeniusisbutarobinssongatthebeginningofaslowspringeventhemostwingedspiritcannotescapephysicalnecessityamadman
isnotlessamusicianthanyouormyselfonlytheinstrumentonwhichheplaysisalittleoutoftunethesongthatliessilentinthehear
tofamothersingsuponthelipsofherchildnolongingremainsunfulfilledihaveneveragreedwithmyotherselfwhollythetruthofth
ematterseemstoliebetweenusyourotherselfisalwayssorryforyoubutyourotherselfgrowsonsorrowsoalliswellthereisnostrug
gleofsoulandbodysaveinthemindsofthosewhosesoulsareasleepandwhosebodiesareoutoftunewhenyoureachtheheartoflifeyous
hallfindbeautyinallthingsevenintheeyesthatareblindtobeautyweliveonlytodiscoverbeautyallelseisaformofwaitingsowas
eedandtheearthwillyieldyouaflowerdreamyourdreamtotheskyanditwillbringyouyourbelovedthedevildiedtheverydayyouwere
bornnowyoudonothavetogothroughhelltomeetanangelmanyawomanborrowsamansheartveryfewcouldpossessitifyouwouldpossess
youmustnotclaimwhenamanshandtouchesthehandofawomantheybothtouchtheheartofeternityloveistheveilbetweenloverandlov
ereverymanlovestwowomentheoneisthecreationofhisimaginationandtheotherisnotyetbornmenwhodonotforgivewomentheirlit
tlefaultswillneverenjoytheirgreatvirtueslovethatdoesnotrenewitselfeverydaybecomesahabitandinturnaslaveryloversem
bracethatwhichisbetweenthemratherthaneachotherloveanddoubthaveneverbeenonspeakingtermsloveisawordoflightwrittenb
yahandoflightuponapageoflightfriendshipisalwaysasweetresponsibilityneveranopportunityifyoudonotunderstandyourfri
endunderallconditionsyouwillneverunderstandhimyourmostradiantgarmentisoftheotherpersonsweavingyoumostsavorymeali
sthatwhichyoueatattheotherpersonstableyourmostcomfortablebedisintheotherpersonshousenowtellmehowcanyouseparateyo
urselffromtheotherpersonyourmindandmyheartwillneveragreeuntilyourmindceasestoliveinnumbersandmyheartinthemistwes
hallneverunderstandoneanotheruntilwereducethelanguagetosevenwordshowshallmyheartbeunsealedunlessitbebrokenonlygr
eatsorroworgreatjoycanrevealyourtruthifyouwouldberevealedyoumusteitherdancenakedinthesunorcarryyourcrossshouldna
tureheedwhatwesayofcontentmentnoriverwouldseekttheseaandnowinterwouldturntospringshouldsheheedallwesayofthrifthow
manyofuswouldbebreathingthisairyouseebutyourshadowwhenyouturnyourbacktothesunyouarefreebeforethesunofthedayandfr
eebeforethestarsofthenightandyouarefreewhenthereisnosunandnomoonandnostaryouareevenfreewhenyoucloseyoureyesupona
llthereisbutyouareaslavetohimwhomyoulovebecauseyoulovehimandaslavetohimwholovesyoubecausehelovesyouweareallbegga
rsatthegateofthetempleandeachoneofusreceiveshisshareofthebountyofthekingwhenheentersthetempleandwhenhegoesoutbut
wearealljealousofoneanotherwhichisanotherwayofbelittlingthekingyoucannotconsumebeyondyourappetitetheotherhalfoft
heloafbelongstotheotherpersonandthereshouldremainalittlebreadforthechanceguestifitwerenotforyourguestsallhousesw
ouldbegravessaidagraciouswolftoasimplesheepwillyounothonorourhousewithavisitandthesheepansweredwewouldhavebeenho
noredtovisityourhouseifitwerenotinyourstomachistoppedmyguestonthethresholdandsaidnaywipenotyourfeetasyouenterbut
asyougooutgenerosityisnotingivingmethatwhichineedmorethanyoudobutitisingivingmethatwhichyouneedmorethanidoyouare
indeedcharitablewhenyougiveandwhilegivingturnyourfaceawaysothatyoumaynotseetheshynessofthereceiverthedifferenceb
etweentherichestmanandthepoorestisbutadayofhungerandanhourofthirstweoftenborrowfromourtomorrowstopayourdebtstoou
ryesterdaysitooamvisitedbyangelsanddevilsbutigetridofthemwhenitisanangeliprayanoldprayerandheisboredwhenitisadev
ilicommitanoldsinandhepassesmebyafterallthisisnotabadprisonbutidonotlikethiswallbetweenmycellandthenextprisoners
cellyetiassureyouthatidonotwishtoreproachthewardernotthebuilderoftheprisonthosewhogiveyouaserpentwhenyouaskforaf
ishmayhavenothingbutserpentstogiveitisthengenerosityontheirparttrickerysucceedssometimesbutitalwayscommitssuicid
eyouaretrulyaforgiverwhenyouforgivemurdererswhoneverspillbloodthieveswhoneverstealandliarswhoutternofalsehoodhew
hocanputhisfingeruponthatwhichdividesgoodfromevilishewhocantouchtheveryhemofthegarmentofgodifyourheartisavolcano
howshallyouexpectflowerstobloominyourhandsastrangeformofselfindulgencetherearetimeswheniwouldbewrongedandcheated
thatimaylaughattheexpenseofthosewhothinkidonotknowiambeingwrongedandcheatedwhatshallisayofhimwhoisthepursuerplay
ingthepartofthepursuedlethimwhowipeshissoiledhandswithyourgarmenttakeyourgarmenthemayneeditagainsurelyyouwouldno
titisapitythatmoneychangerscannotbegoodgardenerspleasedonotwhitewashyourinherentfaultswithyouracquiredvirtuesiwo
uldhavethefaultstheyarelikemineownhowoftenhaveiattributedtomyselfcrimesihavenevercommittedsothattheotherpersonma
yfeelcomfortableinmypresenceeventhemasksoflifearemasksofdeepermysteryyoumayjudgeothersonlyaccordingtoyourknowled
geofyourselftellmenowwhoamongusisguiltyandwhoisunguiltythetrulyjustishewhofeelshalfguiltyofyourmisdeedsonlyanidi
otandageniusbreakmanmadelawsandtheyarethenearesttotheheartofgoditisonlywhenyouarepursuedthatyoubecomeswiftihaven
oenemiesogodbutifiamtohaveanenemylethisstrengthbeequaltominethattruthalonemaybethevictoryouwillbequitefriendlywi
thyourenemywhenyoubothdieperhapsamanmaycommitsuicideinselfdefenselongagotherelivedamanwhowascrucifiedforbeingtoo
lovingandtoolovableandstrangetorelateimethimthriceyesterdaythefirsttimehewasaskingapolicemannottotakeaprostitute
toprisonthesecondtimehewasdrinkingwinewithanoutcastandthethirdtimehewashavingafistfightwithapromoterinsideachurc
hifalltheysayofgoodandevilweretruethenmylifeisbutonelongcrimepityisbuthalfjusticetheonlyonewhohasbeenunjusttomei
stheonetowhosebrotherihavebeenunjustwhenyouseeamanledtoprisonsayinyourheartmayhapheisescapingfromanarrowerprison
andwhenyouseeamandrunkensayinyourheartmayhaphesoughtescapefromsomethingstillmoreunbeautifuloftentimesihavehatedi
nselfdefensebutifiwerestrongeriwouldnothaveusedsuchaweaponhowstupidishewhowouldpatchthehatredinhiseyeswiththesmi
leofhislipsonlythosebeneathmecanenvyorhatemeihaveneverbeenenviednorhatediamabovenooneonlythoseabovemecanpraiseor
belittlemeihaveneverbeenpraisednorbelittlediambelownooneyoursayingtomeidonotunderstandyouispraisebeyondmyworthan
daninsultyoudonotdeservehowmeanamiwhenlifegivesmegoldandigiveyousilverandyetideemmyselfgenerouswhenyoureachthehe
artoflifeyouwillfindyourselfnothigherthanthefelonandnotlowerthantheprophetstrangethatyoushouldpitytheslowfooteda
ndnottheslowmindedandtheblindeyedratherthantheblindhearteditiswiserforthelamenottobreakhiscrutchesuponthehead ofh

isenemyhowblindishewhogivesyououtofhispocketthathemaytakeoutofyourheartlifeisaprocessiontheslowoffootfindsittoos
wiftandhestepsoutandtheswiftoffootfindsittooslowandhetoostepsoutifthereissuchathingassinsomeofuscommititbackward
followingourforefathersfootstepsandsomeofuscommititforwardbyoverrulingourchildrenthetrulygoodishewhoisonewithall
thosewhoaredeemedbadweareallprisonersbutsomeofusareincellswithwindowsandsomewithoutstrangethatwealldefendourwron
gswithmorevigorthanwedoourrightsshouldweallconfessoursinstooneanotherwewouldalllaughatoneanotherforourlackoforig
inalityshouldweallrevealourvirtueswewouldalsolaughforthesamecauseanindividualisabovemanmadelawsuntilhecommitsacr
imeagainstmanmadeconventionsafterthatheisneitheraboveanyonenorlowerthananyonegovernmentisanagreementbetweenyouan
dmyselfyouandmyselfareoftenwrongcrimeiseitheranothernameofneedoranaspectofadiseaseisthereagreaterfaultthanbeingc
onsciousoftheotherpersonsfaultsiftheotherpersonlaughsatyouyoucanpityhimbutifyoulaughathimyoumayneverforgiveyours
elfiftheotherpersoninjuresyouyoumayforgettheinjurybutifyouinjurehimyouwillalwaysrememberintruththeotherpersonisy
ourmostsensitiveselfgivenanotherbodyhowheedlessyouarewhenyouwouldhavemenflywithyourwingsandyoucannotevengivethem
afeatheronceamansatatmyboardandatemybreadanddrankmywineandwentawaylaughingatmethenhecameagainforbreadandwineandi
spurnedhimandtheangelslaughedatmehateisadeadthingwhoofyouwouldbeatombitisthehonorofthemurderedthatheisnotthemurd
ererthetribuneofhumanityisinitssilentheartneveritstalkativemindtheydeemmemadbecauseiwillnotsellmydaysforgoldandi
deemthemmadbecausetheythinkmydayshaveapricetheyspreadbeforeustheirrichesofgoldandsilverofivoryandebonyandwesprea
dbeforethemourheartsandourspiritsandyettheydeemthemselvesthehostsandustheguestsiwouldnotbetheleastamongmenwithdr
eamsandthedesiretofulfillthemratherthanthegreatestwithnodreamsandnodesiresthemostpitifulamongmenishewhoturnshisd
reamsintosilverandgoldweareallclimbingtowardthesummitofourheartsdesireshouldtheotherclimberstealyoursackandyourp
urseandwaxfatontheoneandheavyontheotheryoushouldpityhimtheclimbingwillbeharderforhisfleshandtheburdenwillmakehis
waylongerandshouldyouinyourleannessseehisfleshpuffingupwardhelphimastepitwilladdtoyourswiftnessyoucannotjudgeany
manbeyondyourknowledgeofhimandhowsmallisyourknowledgeiwouldnotlistentoaconquerorpreachingtotheconqueredthetrulyf
reemanishewhobearstheloadofthebondslavepatientlyathousandyearsagomyneighborsaidtomeihatelifeforitisnaughtbutathi
ngofpainandyesterdayipassedbyacemeteryandsawlifedancinguponhisgravestrifeinnatureisbutdisorderlongingforordersol
itudeisasilentstormthatbreaksdownallourdeadbranchesyetitsendsourlivingrootsdeeperintothelivingheartofthelivingea
rthonceispokeoftheseatoabrookandthebrookthoughtmebutanimaginativeexaggeratorandonceispokeofabrooktotheseaandthes
eathoughtmebutadepreciativedefamerhownarrowisthevisionthatexaltsthebusynessoftheantabovethesingingofthegrasshopp
erthehighestvirtueheremaybetheleastinanotherworldthedeepandthehighgotothedepthortotheheightinastraightlineonlyth
espaciouscanmoveincirclesifitwerenotforourconceptionofweightsandmeasureswewouldstandinaweofthefireflyaswedobefor
ethesunascientistwithoutimaginationisabutcherwithdullknivesandoutwornscalesbutwhatwouldyousincewearenotallvegeta
rianswhenyousingthehungryhearsyouwithhisstomachdeathisnotnearertotheagedthantothenewbornneitherislifeifindeedyou
mustbecandidbecandidbeautifullyotherwisekeepsilentforthereisamaninourneighborhoodwhoisdyingmayhapafuneralamongme
nisaweddingfeastamongtheangelsaforgottenrealitymaydieandleaveinitswillseventhousandactualitiesandfactstobespenti
nitsfuneralandthebuildingofatombintruthwetalkonlytoourselvesbutsometimeswetalkloudenoughthatothersmayhearustheob
viousisthatwhichisneverseenuntilsomeoneexpressesitsimplyifthemilkywaywerenotwithinmehowshouldihaveseenitorknowni
tunlessiamaphysicianamongphysicianstheywouldnotbelievethatiamanastronomerperhapstheseasdefinitionofashellisthepe
arlperhapstimesdefinitionofcoalisthediamondfameistheshadowofpassionstandinginthelightarootisaflowerthatdisdainsf
amethereisneitherreligionnorsciencebeyondbeautyeverygreatmanihaveknownhadsomethingsmallinhismakeupanditwasthatsm
allsomethingwhichpreventedinactivityormadnessorsuicidethetrulygreatmanishewhowouldmasternooneandwhowouldbemaster
edbynoneiwouldnotbelievethatamanismediocresimplybecausehekillsthecriminalsandtheprophetstoleranceislovesickwitht
hesicknessofhaughtinesswormswillturnbutisitnotstrangethatevenelephantswillyieldadisagreementmaybetheshortestcutb
etweentwomindsiamtheflameandiamthedrybushandonepartofmeconsumestheotherpartweareallseekingthesummitoftheholymout
ainbutshallnotourroadbeshorterifweconsiderthepastachartandnotaguidewisdomceasestobewisdomwhenitbecomestooproudto
weeptoogravetolaughandtooselffultoseekotherthanitselfhadifilledmyselfwithallthatyouknowwhatroomshouldihaveforall
thatyoudonotknowihavelearnedsilencefromthetalkativetolerationfromtheintolerantandkindnessfromtheunkindyetstrange
iamungratefultotheseteachersabigotisastoneleaforatorthesilenceoftheenviousistoonoisywhenyoureachtheendofwhatyous
houldknowyouwillbeatthebeginningofwhatyoushouldsenseanexaggerationisatruththathaslostitstemperifyoucanseeonlywha
tlightrevealsandhearonlywhatsoundannouncesthenintruthyoudonotseenordoyouhearafactisatruthunsexedyoucannotlaughan
dbeunkindatthesametimethenearesttomyheartareakingwithoutakingdomandapoormanwhodoesnotknowhowtobegashyfailureisno
blerthananimmodestsuccessdiganywhereintheearthandyouwillfindatreasureonlyyoumustdigwiththefaithofapeasantsaidahu
ntedfoxfollowedbytwentyhorsemenandapackoftwentyhoundsofcoursetheywillkillmebuthowpoorandhowstupidtheymustbesurel
yitwouldnotbeworthwhilefortwentyfoxesridingontwentyassesandaccompaniedbytwentywolvestochaseandkillonemanitisthem
indinusthatyieldstothelawsmadebyusbutneverthespiritinusatraveleramiandanavigatorandeverydayidiscoveranewregionwi
thinmysoulawomanprotestedsayingofcourseitwasarighteouswarmysonfellinitisaidtolifeiwouldheardeathspeakandliferais
edhervoicealittlehigherandsaidyouhearhimnowwhenyouhavesolvedallthemysteriesoflifeyoulongfordeathforitisbutanothe
rmysteryoflifebirthanddeatharethetwonoblestexpressionsofbraverymyfriendyouandishallremainstrangersuntolifeandunt
ooneanotherandeachuntohimselfuntilthedaywhenyoushallspeakandishalllistendeemingyourvoicemyownvoiceandwhenishalls
tandbeforeyouthinkingmyselfstandingbeforeamirrortheysaytomeshouldyouknowyourselfyouwouldknowallmenandisayonlywhe
niseekallmenshalliknowmyselfmanistwomenoneisawakeindarknesstheotherisasleepinlightahermitisonewhorenouncesthewor
ldoffragmentsthathemayenjoytheworldwhollyandwithoutinterruptiontherelies agreenfieldbetweenthescholarandthepoetsh
ouldthescholarcrossitbecomesawisemanshouldthepoetcrossithebecomesaprophetyesterdayisawphilosophersinthemarketp

lacecarryingtheirheadsinbasketsandcryingaloudwisdomwisdomforsalepoorphilosophersthey mustneedssell theirheadstofee
dtheirheartssaidaphilosophertoastreetsweeperipityyouyoursisahardanddirtytaskandthestreetsweepersaidthankyousirbu
ttellmewhatisyourtaskandthephilosopheransweredsayingistudymansmindhisdeedsandhisdesiresthenthestreetsweeperwento
nwithhissweepingandsaidwithasmileipityyoutoohewholistenstotruthisnotlessthanhewhouttersstruthnomancandrawthelineb
etweennecessitiesandluxuriesonlytheangelscandothatandtheangelsarewiseandwistfulperhapstheangelsareourbetterthoug
htinspaceheisthetrueprincewhofindshisthroneintheheartofthedervishgenerosityisgivingmorethanyoucanandprideistakin
glessthanyouneedintruthyouowenaughttoanymanyouowealltoallmenallthosewhohavelivedinthepastlivewithusnowsurelynone
ofuswouldbeanungracioushostthewholongsthemostlivesthelongesttheysaytomeabirdinthehandisworthteninthebushbutisayab
irdandafeatherinthebushisworthmorethantenbirdsinthehandyourseekingafterthatfeatherislifewithwingedfeetnayitislif
eitselfthereareonlytwoelementsherebeautyandtruthbeautyintheheartsofloversandtruthinthearmsofthetillersofthesoilg
reatbeautycapturesmebutabeautystillgreaterfreesmeevenfromitselfbeautyshinesbrighterintheheartofhimwholongsforitt
hanintheeyesofhimwhoseesitiadmirehimwhorevealshismindtomeihonorhimwhounveilshisdreamsbutwhyamishyandevenalittlea
shamedbeforehimwhoservesmethegiftedwereonceproudinservingprincesnowtheyclaimhonorinservingpauperstheangelsknowth
attoomanypracticalmeneattheirbreadwiththesweatofthedreamersbrowwitisoftenamaskifyoucouldtearityouwouldfindeither
ageniusirritatedorclevernessjugglingtheunderstandingattributestomeunderstandingandthedulldullnessithinktheyarebo
thrightonlythosewithsecretsintheirheartscoulddivinethesecretsinourheartshewhowouldshareyourpleasurebutnotyourpai
nshalllosethekeytooneofthesevengatesofparadiseyesthereisanirvanahitisinleadingyoursheeptoagreenpastureandinputti
ngyourchildtosleepandinwritingthelastlineofyourpoemwechooseourjoysandoursorrowslongbeforeweexperiencethemsadness
isbutawallbetweentwogardenswheneitheryourjoyoryoursorrowbecomesgreattheworldbecomessmalldesireishalfoflifeidiffe
renceishalfofdeaththebitterestthinginourtodayssorrowisthememoryofouryesterdaysjoytheysaytomeyoumustneedschoosebe
tweenthepleasuresofthisworldandthepeaceofthenextworldandisaytothemihavechosenboththedelightsofthisworldandthepea
ceofthenextforiknowinmyheartthatthesupremepoetwrotebutonepoemanditscansperfectlyanditalsorhymesperfectlyfaithisa
noasisintheheartwhichwillneverbereachedbythecaravanofthinkingwhenyoureachyourheightyoushalldesirebutonlyfordesir
eandyoushallhungerforhungerandyoushallthirstforgreaterthirstifyourevealyoursecretstothewindyoushouldnotblamethew
indforrevealingthemtothetreestheflowersofspringarewintersdreamsrelatedatthebreakfasttableoftheangelssaidaskunkto
atuberoseseehowswiftlyirunwhileyoucannotwalknorevencreepsaidthetuberosetotheskunkohmostnobleswiftrunnerpleaserun
swiftlyturtlescantellmoreaboutroadsthanharesstrangethatcreatureswithoutbackboneshavethehardestshellsthemosttalka
tiveistheleastintelligentandthereishardlyadifferencebetweenanoratorandanauctioneerbegratefulthatyoudonothavetoli
vedowntherenownofafathernorthewealthofanunclebutaboveallbegratefulthatnoonewillhavetolivedowneitheryourrenownory
ourwealthonlywhenajugglermissescatchinghisballdoesheappealtometheenviouspraisesmeunknowinglylongwereyouadreaminy
ourmotherssleepandthenshewoketogiveyoubirththegermoftheraceisinyourmotherslongingmyfatherandmotherdesiredachilda
ndtheybegotmeandiwantedamotherandafatherandibegotnightandtheseasomeofourchildrenareourjustificationsandsomearebu
tourregretswhennightcomesandyoutooaredarkliedownandbedarkwithawillandwhenmorningcomesandyouarestilldarkstandupan
dsaytothedaywithawilliamstilldarkitisstupidtoplayarolewiththenightandthedaytheywouldbothlaughatyouthemountainvei
ledinmistisnotahillanoaktreeintherainisnotaweepingwillowbeholdhereisaparadoxthedeepandhigharenearertooneanothert
hanthemidleveltoeitherwhenistoodaclearmirrorbeforeyouyougazedintomeandsawyourimagethenyousaidiloveyoubutintruthy
oulovedyourselfinmewhenyouenjoylovingyourneighboritceasestobeavirtuelovewhichisnotalwaysspringingisalwaysdyingyo
ucannothaveyouthandtheknowledgeofitatthesametimeforyouthistoobusylivingtoknowandknowledgeistoobusyseekingitselft
oliveyoumaysitatyourwindowwatchingthepassersbyandwatchingyoumayseeanunwalkingtowardyourrighthandandaprostituteto
wardyourlefthandandyoumaysayinyourinnocencehownobleistheoneandhowignobleistheotherbutshouldyoucloseyoureyesandli
stenawhileyouwouldhearavoicewhisperingintheetheroneseeksmeinprayerandtheotherinpainandinthespiritofeachthereisab
owerformyspiritonceeveryhundredyearsjesusofnazarethmeetsjesusofthechristianinagardenamongthehillsoflebanonandthe
ytalklongandeachtimejesusofnazarethgoesawaysayingtojesusofthechristianmyfriendifearweshallneverneveragreemaygodf
eedtheoverabundantagreatmanhastwoheartsonebleedsandtheotherforbearsshouldonetellaliewhichdoesnothurtyounoranyone
elsewhynotsayinyourheartthatthehouseofhisfactsistoosmallforhisfanciesandhehadtoleaveitforlargerspacebehindeveryc
loseddoorisamysterysealedwithsevensealswaitingisthehoofsoftimewhatiftroubleshouldbeanewwindowintheeasternwallofy
ourhouseyoumayforgettheonewithwhomyouhavelaughedbutnevertheonewithwhomyouhaveweptthere mustbesomethingstrangelysa
credinsaltitisinourtearsandintheseaourgodinhisgraciousthirstwilldrinkusallthedewdropandthetearyouarebutafragment
ofyourgiantselfamouththatseeksbreadandablindhandthatholdsthecupforathirstymouthifyouwouldrisebutacubitaboveracea
ndcountryandselfyouwouldindeedbecomegodlikeifiwereyouiwouldnotfindfaultwiththeseaatlowtideitisagoodshipandourcap
tainisableitisonlyyourstomachthatisindisordershouldyousituponacloudyouwouldnotseetheboundarylinebetweenonecountr
yandanothernortheboundarystonebetweenafarmandafarmitisapityyoucannotsituponacloudsevencenturiesagosevenwhitedove
srosefromadeepvalleyflyingtothesnowwhitesummitofthemountainoneofthesevenmenwhowatchedtheflightsaidiseeablackspot
onthewingoftheseventhdovetodaythepeopleinthatvalleytellofsevenblackdoveswhoflewtothesummitofthesnowymountaininth
eautumnigatheredallmysorrowsandburiedtheminmygardenandwhenaprilreturnedandspringcametowedtheearththeregrewinmyga
rdenbeautifulflowersunlikeallotherflowersandmyneighborscametobeholdthemandtheyallsaidtomewhenautumncomesagainats
eedingtimewillyounotgiveusoftheseedsoftheseflowersthatwemayhavetheminourgardensitisindeedmiseryifistretchanempty
handtomenandreceivenothingbutitishopelessnessifistretchafullhandandfindnonetoreceiveilongforeternitybecausethere
ishallmeetmyunwrittenpoemsandmyunpaintedpicturesartisastepfromnaturetowardtheinfiniteaworkofartisamistcarvedinto

```
animageeventhehandsthatmakecrownsofthornsarebetterthanidlehandsourmostsacredtearsneverseekoureyeseverymanisthede
scendantofeverykingandeveryslavethateverlivedifthegreatgrandfatherofjesushadknownwhatwashiddenwithinhimwouldheno
thavestoodinaweofhimselfwastheloveofjudasmotherofhersonlessthantheloveofmaryforjesustherearethreemiraclesofourbr
otherjesusnotyetrecordedinthebookthefirstthathewasamanlikeyouandmethesecondthathehadasenseofhumourandthethirdtha
theknewhewasaconquerorthoughconqueredcrucifiedoneyouarecrucifieduponmyheartandthenailsthatpierceyourhandspiercet
hewallsofmyheartandtomorrowwhenastrangerpassesbythisgolgothahewillnotknowthattwobledherehewilldeemitthebloodofon
emanyoumayhaveheardoftheblessedmountainitisthehighestmountaininourworldshouldyoureachthesummityouwouldhaveonlyon
edesireandthattodescendandbewiththosewhodwellinthedeepestvalleythatiswhyitiscalledtheblessedmountaineverythought
ihaveimprisonedinexpressionimustfreebymydeedsflagishelloxnucagoodluck
#看结尾
#flag{helloxnucagoodluck}
```