2023NewStarCTF writeup by Beckoning

WEEK1 Crypto

brainfuck

```
++++++++
[>>+++>++++>++++++>++++++++>++++++++++>++++++++++++>++++++++++++>++++++++++++
+>++++++++++++++++>++++++++++++++++++>++++++++++++++++++++>++++++++++++++++++
++++++++>++++++++++++++++++++++++++>++++++++++++++++++++++++++>++++++++++++++++
++++++++++++++<<<<<<<<<<<<<<<<-]>>>>>>>>>++++++.>----.<-----.>-----.>-----.
<<<-.>>++..<.>.++++++.....------.<.>.<<<<<+++.>>>>+.<<<+++++++.>>>+.<<<------
-.>>>-.<<<+.+++++++.--..>>>>---.-.<<<<-.+++.>>>>.<<<<-------.+.>>>>>++.
```

是brainfuck解密，使用工具解密即可

```
flag{Oiiaioooooiai#b7c0b1866fe58e12}
```

Caesar's Secert

```
kqfl{hf3x4w'x_h1umjw_n5_a4wd_3fed}
```

凯撒加密，在线解密就行，偏移量为5

```
flag{ca3s4r's_c1pher_i5_v4ry_3azy}
```

Fence

```
fa{ereigtepanet668O}lgrodrn_h_litx#8fc3
```

栅栏加密，在线解密即可

```
flag{reordering_the_plaintext#686f8c03}
```

Vigenère

```
pqcq{qc_m1kt4_njn_5slpOb_lkyacx_gcdy1ud4_g3nv5xO}
```

维吉尼亚加密，密钥是kfc，解出：

```
flag{la_c1fr4_del_5ignOr_giovan_batt1st4_b3ll5sO}
```

babyencoding

```
part 1 of flag: ZmxhZ3tkYXp6bGluZ19lbmNvZGluZyM0ZTBhZDQ=
part 2 of flag: MYYGGYJQHBSDCZJRMQYGMMJQMMYGGN3BMZSTIMRSMZSWCNY=
part 3 of flag: =8S4U,3DR8SDY,C`S-F5F-C(S,S<R-C`Q9F8S87T`
```

第一部分用base64解密，得到：

```
flag{dazzling_encoding#4e0ad4
```

第二部分用base32解密，得到：

```
f0ca08d1e1d0f10c0c7afe422fea7
```

第三部分用UUencode解密，得到：

```
c55192c992036ef623372601ff3a}
```

拼接起来是：

```
flag{dazzling_encoding#4e0ad4f0ca08d1e1d0f10c0c7afe422fea7c55192c992036ef623372601ff3a}
```

babyrsa

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

def gen_prime(n):
    res = 1

    for i in range(15):
        res *= getPrime(n)

    return res


if __name__ == '__main__':
    n = gen_prime(32)
    e = 65537
    m = bytes_to_long(flag)
    c = pow(m,e,n)
    print(n)
    print(c)
```

```
#
172900660705949795710096633812142013204595698513585023686512455142135382299699156
580649925581673235868950889339228353538040557726389802513282 61
#
143220384337616554046783935681585378497835894814635210756948026546110488988786051
446637504106557346754233282562131144229299940372407529953635 95
```

先分解n，发现分解为了

```
2217990919<10> · 2338725373<10> · 2370292207<10> · 2463878387<10> ·
2706073949<10> · 2794985117<10> · 2804303069<10> · 2923072267<10> ·
2970591037<10> · 3207148519<10> · 3654864131<10> · 3831680819<10> ·
3939901243<10> · 4093178561<10> · 4278428893<10>
```

然后我们计算phi的值，反向计算m的值就可以了，flag的值为：

```
flag{us4_s1ge_tO_cal_phI}
```

解密代码：

```
from gmpy2 import *
phi=2217990918*2338725372*2370292206*2463878386*2706073948*2794985116*2804303068*
2923072266*2970591036*3207148518*3654864130*3831680818*3939901242*4093178560*4278
428892
e=65537
n=172900660705949795710096633812142013204595698513585023686512455142135382299699
1565806499255816732358689508893392283535380405577263898025132826 1
c=143220384337616554046783935681585378497835894814635210756948026546110488988786
0514466375041065573467542332825621311442292999403724075299536359 5

d=invert(e,phi)
m=pow(c,d,n)
print(m)#642921858775584588910849779736964353704741462609884093106557

from Crypto.Util.number import *
print(long_to_bytes(m))#flag{us4_s1ge_tO_cal_phI}
```

Small d

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

p = getPrime(1024)
q = getPrime(1024)

d = getPrime(32)
e = inverse(d, (p-1)*(q-1))
n = p*q
m = bytes_to_long(flag)

c = pow(m,e,n)
```

```
print(c)
print(e)
print(n)

# c =
675591669677818595230010882488034167372700524951785062842498249986574486415880896
876413563714106893091362609359872892519585959207824267920669052567858469890678202
867196855770127159141998237083958187277956189789670712881566872260928548497830321
686323699702119757633794020475733174970187280844324692777297750057685355953142193
194360018592361032932221959197764457350975548367905995142668617029601879877124313
653065159718198804066858624044909941230145431293706560496122435923503819014585210
847352041390901419860043467903752416552342240136420845063155738020799659798130916
836016065830898274554544275688493114150138795424 8
# e =
861453108713180653607217612660850539648599891219309042009451079259510115824045398
505505365384855632501140992239471112455838361983029001795091235302727040056756862
281624582232442299307469018397109388264077980854647919560474323013711329375289796
833222098964071031199815010831529833381703063417948707542140361779082356088668886
092813311753672497788868373247870862831485731370059652233950958191532345269513687
780281600335385322098649200797018355104130387595875049689286795447751096670893535
853432286740486026718029453823173418417672780528974600499996992373652878343687672
810435178335187934095956818310151529439304865 1825
# n =
198736349834560875201105522774504975292484945819022993272372680307563980577525101
030123364525220301733293217267799358321060301576826722625480768953704434615588515
849516810937878210354889526910342501154404418075575952569847199959831585958434510
375469299187778836750205719455339223215141200754884904790094689432869900027351693
714049732840968698263576590276278158885583915202768661223705511152232826378558942
021704749552741292763566253646631657234312159811849965130233724338620536247921953
612711414518801230901586440952870458622049548299986147176771638413912727541226879
612647239938802394071060303700477941451232929914 33
```

发现d很小，只有32位，我们使用wiener attack来进行攻击

```
#Sage
def factor_rsa_wiener(N, e):
    N = Integer(N)
    e = Integer(e)
    cf = (e / N).continued_fraction().convergents()
    for f in cf:
        k = f.numer()
        d = f.denom()
        if k == 0:
            continue
        phi_N = ((e * d) - 1) / k
        b = -(N - phi_N + 1)
        dis = b ^ 2 - 4 * N
        if dis.sign() == 1:
            dis_sqrt = sqrt(dis)
            p = (-b + dis_sqrt) / 2
            q = (-b - dis_sqrt) / 2
            if p.is_integer() and q.is_integer() and (p * q) % N == 0:
                p = p % N
                q = q % N
```

```
                if p > q:
                    return (p, q)
                else:
                    return (q, p)

e = 
86145310871318065360721761266085053964859989121930904200945107925951011582404539850550536538485563250114099223947111245583836198302900179509123530272704005675686228162458223244229930746901839710938826407798085464791956047432301371132937528979683322209896407103119981501083152983338170306341794870754214036177908235608866888609281331175367249778886837324787086283148573137005965223395095819153234526951368778028160033538532209864920079701835510413038759587504968928679544775109667089353585343228674048602671802945382317341841767278052897460049999699237365287834368767281043517833518793409595681831015152943930486518 25
n = 
19873634983456087520110552277450497529248494581902299327237268030756398057752510103012336452522030173329321726779935832106030157682672262548076895370443461558851584951681093787821035488952691034250115440441807557595256984719995983158595843451037546929918777883675020571945533922321514120075488490479009468943286990002735169371404973284096869826357659027627815888558391520276866122370551115223282637855894202170474955274129276356625364663165723431215981184996513023372433862053624792195361271141451880123090158644095287045862204954829998614717677163841391272754122687961264723993880239407106030370047794145123292991433

factor_rsa_wiener(n,e)
```

输出结果：得到了p和q的值

```
(164887456707385597794671504018267709143220756168005943404439170291961240964917459897851995873318931111003334756829330833030113953921897690894547696970173765400394343219281907593789488243043201193175383927844938693390050315847202652056687832428191702170841515492613755081814436819172309486788028139552841308767,

120528482762181585784791121533263404463364363674762508523888416841598344462609634410166712335775074695194129147081306232127435812293764101753434633378229572294818621955201083222863101440688819614733415001310705111396264686322304770037360220730171417541890902341308693475235102578207842995280367951899591120599)
```

验证一下，将两个数乘起来，发现是对的，然后用RSA解密就可以了

```
flag{learn_some_continued_fraction_technique#dc16885c}
```

解密代码如下：

```
from gmpy2 import *
e=86145310871318065360721761266085053964859891219309042009451079259510115824045398505505365384855632501140992239471112455838361983029001795091235302727040056756862281624582232442299307469018397109388264077980854647919560474323013711329375289796833222098964071031199815010831529833381703063417948707542140361779082356088668860928133117536724977888683732478708628314857313700596522339509581915323452695136877802816003353853220986492007970183551041303875958750496892867954477510966708935358534322867404860267180294538231734184176727805289746004999969923736528783436876728104351783351879340959568183101515294393048651825
p=16488745670738559779467150401826770914322075616800594340443917029196124096491745989785199587331893111100333475682933083303011395392189769089454769697017376540039434321928190759378948824304320119317538392784493869339005031584720265205668783242819170217084151549261375508181443681917230948678802813955284130876 7
q=12052848276218158578479112153326340446336436367476250852388841684159834446260963441016671233577507469519412914708130623212743581229376410175343463337822957229481862195520108322286310144068881961473341500131070511139626468632230477003736022073017141754189090234130869347523510257820784299528036795189959112059 9
n=p*q
phi=(p-1)*(q-1)
c =
67559166967781859523001088248803416737270052495178506284249824998657448641588089687641356371410689309136260935987289251958595920782426792066905256785846989067820286719685577012715914199823708395818727795618978967071288156687226092854849783032168632369970211975763379402047573317497018728084432469277729775005768535595314219319436001859236103293222195919776445735097554836790599514266861702960187987712431365306515971819880406685862404490994123014543129370656049612243592350381901458521084735204139090141986004346790375241655234224013642084506315573802079965979813091683601606583089827455454427568849311415013879542 48
d=2357048593
m=pow(c,d,n)
print(m)#44372835900758329644743961970478115267149538910938782240530006079942280785992929411937603381399200999430346408198915529294325605 09
print(long_to_bytes(m))#b'flag{learn_some_continued_fraction_technique#dc16885c}'
```

babyxor

```
from secret import *

ciphertext = []

for f in flag:
    ciphertext.append(f ^ key)

print(bytes(ciphertext).hex())
# e9e3eee8f4f7bffdd0bebad0fcf6e2e2bcfbfdf6d0eee1ebd0eabbf5f6aeaeaeaeaeaef2
```

这题就是做了一个异或操作，直接恢复就可以了，需要暴力解出我们的key，只需要循环求解就可以了，注意一个字节是256种情况，是两位

```
flag{xOr_15_symm3try_and_e4zy!!!!!!}
```

Affine

```
from flag import flag, key

modulus = 256

ciphertext = []

for f in flag:
    ciphertext.append((key[0]*f + key[1]) % modulus)

print(bytes(ciphertext).hex())

# dd4388ee428bdddd5865cc66aa5887ffcca966109c66edcca920667a88312064
```

这题的话是仿射变换，仿射那一步就用mod运算来解决就可以

```
from itertools import product

c = "dd4388ee428bdddd5865cc66aa5887ffcca966109c66edcca920667a88312064"
c = bytes.fromhex(c)

# 求模逆
def modinv(a, m):
    for i in range(1, m):
        if (a * i) % m == 1:
            return i
    return None

for k0, k1 in product(range(256), repeat=2):
    k0_inv = modinv(k0, 256)
    if k0_inv is None:
        continue

    decrypted = []
    for b in c:
        f = ((b - k1) * k0_inv) % 256
        decrypted.append(f)

    flag = str(bytes(decrypted))
    if "flag" in flag:
        print(flag)
```

输出：`flag{4ff1ne_c1pher_i5_very_3azy}`

babyaes

```
from Crypto.Cipher import AES
import os
from flag import flag
```

```
from Crypto.Util.number import *


def pad(data):
    return data + b"".join([b'\x00' for _ in range(0, 16 - len(data))])


def main():
    flag_ = pad(flag)
    key = os.urandom(16) * 2
    iv = os.urandom(16)
    print(bytes_to_long(key) ^ bytes_to_long(iv) ^ 1)
    aes = AES.new(key, AES.MODE_CBC, iv)
    enc_flag = aes.encrypt(flag_)
    print(enc_flag)


if __name__ == "__main__":
    main()
# 365749176821575063584495806096380512533376138774695461854095848991496457322 9
# b'>]\xc1\xe5\x82/\x02\x7ft\xf1B\x8d\n\xc1\x95i'
```

得到答案：`firsT_cry_Aes`

WEEK2 Crypto

滴啤

```
from Crypto.Util.number import *
import gmpy2
from flag import flag
def gen_prime(number):
    p = getPrime(number//2)
    q = getPrime(number//2)
    return p,q

m = bytes_to_long(flag.encode())
p,q = gen_prime(1024)
print(p*q)
e = 65537
d = gmpy2.invert(e,(p-1)*(q-1))
print(d%(p-1))
print(pow(m,e,p*q))
#
931727884929264383277105925645628542064387123903946361493856083218001349343613537
942066240313969881244558477688837855037955213891788147912130541243610078874963515
040997727571642116667784148006989763357670278687617355331958801829823589372112825
413796977148743138633540976462335752652239783109328414615359369 31
#
307467153394842898333761625034462907680907310539113349710634557900919735848784017
007186630645110812431448648273172817619775466967145608769260573615221635
```

```
#
527777056923275013325284871683401754368321098662185977788222622684170751575678804
094830794529035288830407150971362937651888581871421030816391340559975525432135894
677510375244825780935722443139280303413563599895314517891668154624174848220099370
890583529827396117557176667992782714949333827166335531997392920 89
```

可以看出，我们拥有的信息是d，e，dp，c，我们用dp泄露去解决问题

```
import gmpy2 as gp

e = 65537
n =
931727884929264383277105925645628542064387123903946361493856083218001349343613537
942066240313969881244558477688837855037955213891788147912130541243610078874963515
040997727571642116667784148006989763357670278687617355331958801829823589372112825
413796977148743138633540976462335752652239783109328414615359369 31
dp =
307467153394842898333761625034462907680907310539113349710634557900919735848784017
007186630645110812431448648273172817619775466967145608769260573615221635
c =
527777056923275013325284871683401754368321098662185977788222622684170751575678804
094830794529035288830407150971362937651888581871421030816391340559975525432135894
677510375244825780935722443139280303413563599895314517891668154624174848220099370
890583529827396117557176667992782714949333827166335531997392920 89

for x in range(1, e):
    if(e*dp%x==1):
        p=(e*dp-1)//x+1
        if(n%p!=0):
            continue
        q=n//p
        phin=(p-1)*(q-1)
        d=gp.invert(e, phin)
        m=gp.powmod(c, d, n)
        if(len(hex(m)[2:])%2==1):
            continue
        print('--------------')
        print(m)
        print(hex(m)[2:])
        print(bytes.fromhex(hex(m)[2:]))
```

这样得到结果：`flag{cd5ff82d-989c-4fbf-9543-3f98ab567546}`

不止一个pi

```
from flag import flag
from Crypto.Util.number import *
import gmpy2
p = getPrime(1024)
q = getPrime(1024)
n = p**3*q**2
print("q = ",q)
```

```
print("p = ",p)
m = bytes_to_long(flag.encode())
c = pow(m,65537,n)
print("c = ",c)

# q =
11547886787034752766068032927101285204384586840192836107610277993837027067089749875939184428213714901384595661225753464025999797927561023539570647396597320354492046941628318167766026250948128253646579673140196769468357584318350943001797250675290127088744449090589149095597576252418753405247817396611747114371 3
# p =
17179096037131724408761591304769667077811576520188383552545601620796604865858241784293692514958237830561030450553099783314725183228927612508433961480808535681420223646390038433587876017763050195038491979438661936339416901656048515208389318342091129571244692531839179382237139043965516007721273926087192393521 7
# c =
44591839283243697623976716053176001575127125036943307679384904962256699850500027762534708411931569510876631078667144262302220023996663062876425910779908978831741344048968004822347815315929390435518320497565719870101736670741682823555207119056590130765093535230885833473733589808427076866111570504255845988251513998702680838672699121396349293979575143768261458707521165831853515760517766272088823774134331405774613145047623886175952820851022715107923055606089343535155522015536742879549873233215128521143532663593642826034870989166083029446946002276287877918766009015378881100937036124148366765715624870053302999969088735892280729826411148447619801430479207701145359249597655183656147092722976662314816558572430040720490940785255694602933814795581485063469660649061642093621473133719625670400470845165101350545710806120773332281956081090654752608325801923218539061388111390366584856883201615301312398540039964578716634568501964835202396759813910474523819986203868991018207824216052877087276676630389053781152351637738675082582088673673141087018557090026345923299769122399562124907882623961062301917546808137904254337634273152303304593493204123541890106845251053186101029367152035292224916428073822150234689367555846328493489966665289812692408676120683822438223004188565994182238755224089865969250189755650576962184230364591443926251667615224247212689716760104270963796102666499119391394519892461945255535336998311105681462203476036277454074497617921358981101397434987675435212975258028092548425180021903815089643570012113539970614177107833337 4
```

这题我们知道了$n = p^3 \times q^2$,所以对应的phi也需要变化

$$phi(n) = p^2 \times (1-p) \times q \times (1-q)$$

```
from gmpy2 import *
q =
 11547886787034752766068032927101285204384586840192836107610277993837027067089749
87593918442821371490138459566122575346402599979792756102353957064739659732035449 2
04694162831816776602625094812825364657967314019676946835758431835094300179725067 5
29012708874444909058914909559757625241875340524781739661174711437 13
p =
 17179096037131724408761591304769667077811576520188383552545601620796604865858241
78429369251495823783056103045055309978331472518322892761250843396148080853568142 0
22364639003843358787601776305019503849197943866193633941690165604851520838931834 2
09112957124469253183917938223713904396551600772127392608719239352 17
c =
 44591839283243697623976716053176001575127125036943307679384904962256699850500027
76253470841193156951087663107866714426230222002399666306287642591077990897883174 1
34404896800482234781531592939043551832049756571987010173667074168282355520711905 6
59013076509353523088583347373358980842707686611157050425584598825151399870268083 8
67269912139634929397957514376826145870752116583185351576051776627208882377413433 1
40577461314504762388617595282085102271510792305560608934353515552201553674287954 9
87323321512852114353266359364282603487098916608302944694600227628787791876600901 5
37888110093703612414836676571562487005330299996908873589228072982641114844761980 1
43047920770114535924959765518365614709272297666231481655857243004072049094078525 5
69460293381479558148506346966064906164209362147313371962567040047084516510135054 5
71080612077333228195608109065475260832580192321853906138811139036658485688320161 5
30131239854003996457871663456850196483520239675981391047452381998620386899101820 7
82421605287708727667663038905378115235163773867508258208867367314108701855709002 6
34592329976912239956212490788262396106230191754680813790425433763427315230330459 3
49320412354189010684525105318610102936715203529222491642807382215023468936755584 6
32849348996666528981269240867612068382243822300418856599418223875522408986596925 0
18975565057696218423036459144392625166761522424721268971676010427096379610266649 9
11939139451989246194525553533699831110568146220347603627745407449761792135898110 1
39743498767543521297525802809254842518002190381508964357001211353997061417710783 3
37
n=p**3*q**2
phi=p**2*(p-1)*q*(q-1)
e=65537
d =gmpy2.invert(e, phi)
m=pow(c,d,n)
print(long_to_bytes(m))
```

解出flag：`flag{bu_zhi_yige_p1dsaf}`

halfcandecode

```
from Crypto.Util.number import *
import gmpy2
from flag import flag
import os
from hashlib import md5

def gen_prime(number):
    p = getPrime(number // 2)
    q = gmpy2.next_prime(p)
    return p * q
```

```
def md5_hash(m):
    return md5(m.encode()).hexdigest()
e = 65537
n = gen_prime(1024)
m1 = bytes_to_long(flag[:len(flag) // 2].encode() + os.urandom(8))
c1 = pow(m1, e, n)
m2 = flag[len(flag) // 2:]
with open("out.txt","w") as f:
    f.write(str(n) + '\n')
    f.write(str(c1) + '\n')
    for t in m2:
        f.write(str(md5_hash(t))+'\n')
```

Rotate Xor

```
from secret import flag
from os import urandom
from pwn import xor
from Cryptodome.Util.number import *
k1 = getPrime(64)
k2 = getPrime(64)
ROUND = 12
ciphertext = xor(flag, long_to_bytes(k1))
def round_rotate_left(num, step):
    return ((num) << step | num >> (64-step)) & 0xffffffffffffffff
def encrypt_key(key):

    for _ in range(ROUND):
        key = round_rotate_left(key, 3) ^ k2

    return key
print('ciphertext =', ciphertext)
print('enc_k1 =', encrypt_key(k1))
print('k2 =', k2)

# ciphertext =
b'\x8dSyy\xd2\xce\xe2\xd2\x98\x0fth\x9a\xc6\x8e\xbc\xde`zl\xc0\x85\xe0\xe4\xdfQlc
'
# enc_k1 = 7318833940520128665
# k2 = 9982833494309156947
```

partial decrypt

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

m = bytes_to_long(flag)
```

```
e = 65537
p = getPrime(512)
q = getPrime(512)

n = p*q

c = pow(m,e,n)

dp = inverse(e, (p-1))
dq = inverse(e, (q-1))
m1 = pow(c,dp, p)
m2 = pow(c,dq, q)
q_inv = inverse(q, p)
h = (q_inv*(m1-m2)) % p
print('m2 =', m2)
print('h =', h)
print('q =', q)

# m2 =
481672510709662540833595491298673558464223060451701789089734890181574163266875137
872985175303791716498969848385600411592253857647012777834212149785255488 4
# h = 418072013709044783581624069710063052562457 4275
# q =
732529439982906161428353915785338283162780457179217947784318709700350339890407410
832490098694617565773703577051221353029327711199279933125123122371040693 1
```

broadcast

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

menu = '''
Welcome to RSA Broadcasting system

please select your option:

1. brocast the flag
2. exit
'''
e = 17
def broadcast_the_flag():
    p = getPrime(256)
    q = getPrime(256)
    n=p*q
    m = bytes_to_long(flag)
    c = pow(m,e,n)
    print('n =', n)
    print('c =', c)
    print('e =', e)
while True:
    print(menu)
    opt = input('> ')
    try:
```

```
        opt = int(opt)
        if opt == 1:
            broadcast_the_flag()
        elif opt == 2:
            break
        else:
            print('invalid option')
    except:
        print('oh no, something wrong!')
```

WEEK3 Crypto

Rabin's RSA

```
from Crypto.Util.number import *
from secret import flag
p = getPrime(64)
q = getPrime(64)
assert p % 4 == 3
assert q % 4 == 3

n = p * q

e = 2
m = bytes_to_long(flag)

c = pow(m,e,n)

print('n =', n)
print('c =', c)

# n = 20135409053191838942224151553476153657 3
# c = 204429893813488806300464357511937457 53
```

小明的密码题

```
from Crypto.Util.number import *
from secret import *
flag_part = flag_content + '#' + secret_token
p = getPrime(512)
q = getPrime(512)

m = bytes_to_long(flag_part.encode())

e = 5
n = p*q

c = pow(m,e,n)
```

```
print('n =', n)
print('c =', c)
print('flag_part =', flag_part)
print()
print('--- hint begin ---')
print('flag = "flag{" + flag_part + "}"')
print('type of secret_token is', type(secret_token))
print('length of secret_token is', len(secret_token))

# n =
131889193322687215946601811511407251196213571687093913054335139712633125177496800
529685285401802802683116451016274353008428347997732857844896393358010946452397522
017632024075459908859131965234835870443110233375074265933004741459359128684375786
221535003839961829770182916778717973782408036072622166388614214214899
# c =
111882017573613631415782355648074115830850919333893818878277915513697387171175499
690676603722143662750400556476218178038774954730687675714655218810107078736860363
364755541053144751936763886088128722189437284558416522087118023764530341418832361
426773458805942468799673787705733855223260392064005782603530743 79
# flag_part = sm4ll_rOOts_is_brilliant#■ ■ ■ ■ ■ ■ ■ ■
#
# --- hint begin ---
# flag = "flag{" + flag_part + "}"
# type of secret_token is <class 'str'>
# length of secret_token is 8
```

babyrandom

```
#!/usr/bin/python3
from secret import flag
from Crypto.Util.number import *
from random import randrange

p = 64999433139797068147576269731948390094958654326970231465808792590598519729077

a = randrange(2, p)
b = randrange(2, p)
x = bytes_to_long(flag)
menu = '''

Random as a Service with LCG backend

Enter your option
1. Reset
2. Get
3. Exit
'''

def GetRandom():
    global x
    nx = (a*x + b) % p
    print(nx)
    x = nx
```

```
while True:
    print(menu)
    opt = input('> ')
    try:
        opt = int(opt)
        if opt == 1:
            x = bytes_to_long(flag)
        elif opt == 2:
            GetRandom()
        elif opt == 3:
            break
        else:
            print('invalid option')
    except Exception as e:
        print('oh no, something wrong!')
        print(e)
```

knapsack

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

m = [int(i) for i in bin(int(codecs.encode(flag,  'hex'), 16))[2:]]

# from ASIS Cyber Security Contest Quals 2014
def makeKey(n):
    privKey = [random.randint(1, 4**n)]
    s = privKey[0]
    for i in range(1, n):
        privKey.append(random.randint(s + 1, 4**(n + i)))
        s += privKey[i]
    q = random.randint(privKey[n-1] + 1, 2*privKey[n-1])
    r = random.randint(1, q)
    while gmpy2.gcd(r, q) != 1:
        r = random.randint(1, q)
    pubKey = [ r*w % q for w in privKey ]
    return privKey, q, r, pubKey

priv, q, r, pub = makeKey(len(m))

ciphertext = sum([i*j for i, j in zip(pub, m)])

print(ciphertext)
print(pub)
```

```
#
29203223760570378903144354611216846566258342781745113818615908548394459633489915398356367344167219042668245037506396946137606214702708461240208605666140423038821266902468228849801743444790307989489504464678304461570096021496820646151574770086370780467413532340332267484834294743390741938435153854235219005450 2
```

```
#
[160806598513658995551863687654800274917339819580590856141359456608166096801466789983260809940233030664371430007511149339903101887536789529097880018226258367685398187452447879556216408387672422556291061743472155934881132522847810084034970450282547562530131095998947886100764822065252423213086627413521898456 1,
242666734930292505275312909468160356458154744569442597214905878341097624411340938260965268428567326892658591546512156867410540558324993230385327710602246516229977605488064616466751591463885562165697669549267079399755480623055988779743013096758404582390843526952263547081092959022520173489946912086289785790 6,
342948304892085958822880066405614783520657585913980943772951398961290550696803024773227740517058803694504610314818157173147551596801872835699914606401870581607999033239117172083169717956020844119086940935261003932152466371844874471658242783977391971011627040730720181393430991432545837466217496019024595722 2,
564250109563712564482462432680377816326399227670899728577689320870738623728042609410558470276395688926207490011524564126137762264136489961145118738543908574649400612880495241188542092639217467934763864685433642946847062735766189175949837015309064507907377403736549845944979943983422342242698522999097869 17,
249014840596622149641660527176691794476057468142194777930864317679150897420922686424973896180426775153192051960725303111955265677883261115848242565136022511201488972206792994212941064778954284145191809599312103064514009794091402673741054202969626200835415367051428580468281306875168405522356933129428190511 8,
744335516743470311433495353720410462797590932472861071020740460215850064443048990503206674029768738281613724459334902647142111609386032626419807763430772690331198728393460206599560093644541916308699753208688342851978033928606776623217187488326243128140520766001009008201386016425560302128789112878698236289 ,
381127504985400212467019726644290137041314638071485451804102546415184977553627005726122293636670450471716854082148243252458182359306876611103686781023342418857051791087595413765452221327715323929877526820241226527639196941210160813645003436229759448412768140270375863742299485005636864761548952379486188084 7,
473683666594082604117169413292398893184510141764737433281968386765815121690748313176539282271894450737024228670133241393374209745116297456172536463153415105977394243803878361884246737219582064725385275468166234998338676147995200763548448238246792597526126110011994052595152961202911168268350281186070474410 7,
218428823978489248069535114618152234310305547347205441589205501155707619749334861070578091163906985214268121281397812432059074144407899361767655768671257285249570912585535882641869844445600902993260674857401972042785742343896896913740614716743572831943197701868686231655436196481859353513460552796628309481 2,
588194102339696583350135841824327790124834713596509154434237624805397239340526856351652054792548887203985807293311816082012326594259020613540184041154797618078327175677613074289570022322357182995479906623119955070230707330622384565298411040387807899564480684801466385397273974163223208276864942123556821984 ,
207790646630333103726697231997027921725804393037789780358885025107743748693683943126225963682778049933878600090060773221160780033443743941132058890976141220890311524731632133914406322221514821134136028484079157944927415098422729752585340712975819525811461333692294063760242681358405460351943842940328790180 7,
406069628745817813216696977561118385732171769590959630798352193439137490610546840172466989344023613759775953102640189795426200915800386288971133810933914463474346709624896106966530286356427974391461492900275591398714593815181876855915110868017912056704333406294726041613914852707000541403127833491367627882 5,
464805521218191319924160192264775255914831085906341391561749153102222748222136970632958401987151680712533504989923819329917017894186535195252630326438254062366275197069419667864314880470014031068602992833825811323924636923683981529942141243322812937057276906402502778249955712046748391343955427849890497469 2,
597176878045185163079857438985773795710566287596407799734356772474742131529190202967137742454916982072691342662052310864747435205021028361215252674178315997198269741467302622198726752215953160339956800052041597499968585794961367600105538786381
```

64202615240949076692874682252988681017500851282387274047193766 5,
29100548585273888304039323687796216043633398258517335881866425364709202794599271476430839585720827244659125711994627517275117812064480165907248966314202155623830931341760590688361850075478834673637886415862180972591325329880476804445964813361023440688497058278453731521940828753349895397860612816330880702207,
36962256886595416745144291447478872285484385534213473221793214794417077360854904799256950686828579258209066007168112066686743626702557484414918638261592975635862838567118952749674531115250587419577413881177335772484162278572996417213766483851028436045235482567613239733160134927597067369472555317588614214775,
12649255398177161290145764970351668213999044988432955744780167698818556954490721004386347026851040779009429505317469672368363675419731732527295355878991401455562749614681721061386991573730994817726572342118403568409289343674796977523833296416240889063006641511021902448619167112557012689932860087709674605665,
45710074072181713764241106146198836733405067234971574300343563549463081887912865009306180197687827858087386510093805316643229302860755737328575330022685274833033251258153384377103832767820946974642965565616356501929973769582525191290671604935451515969934698636771686125076788293665045212084692495831561450625,
16072757518623898489315794219085593427093685924625748762749657605261211005107941447860529582499139600250180420871095907459314454937105539760770157328880756127168222135690249443177096961572226583543053847544363385861708669432326113951868038958507061051046339387397005706177562722540189869637656298889517370925,
32314828828215257992962320792689751326282268341888130428346444397622177650535333853269682617000980262120433617470111127212074394711715095171327164086255744482126488850136104544054968236759234326631293245750171773612984124248591204736531688459983185655059044602912192793799499172407145100261682611443287540545,
35207569438691140552477715079835244286033605098921401852150721076321699081642095729711890337662844971931425777142147199376906326208073834529413703892212215788770050207611804206906677013570849628984528189234259652501949387133816368347572199641442876817563209823716336978255016787730726090378344416399462253395644039606888725644940425659046652801371359859808124779961040582333607269549806515562618665508080479052038071359752579556288889257928236329013688483268123958648093660820087930011239471208176432929454211978461854650887983862049559572971608373698718621196087195652862336329334834851078868255879883255664207695315959983412675584879150300480980412474498336983333058244546717323455930859057635392845900347749179949704745226560004600224895066675818550275520877089383803806869595787780164683206194624514937603600847800253240223046758244243173853100476864620378942952204542538466227047478926569550153847751719869006440425,
13590186607787135191826713631240664083952511295922387927025700614583061990666546563276327652063705596280060414775443266794092923996245491484460560024213717799471115120170530249973117820532958878854994999008714251714591047336216572326228876336435351710090467548345068108862085986403133239924025878265107751143143179053950367778166107548875625357080442600180871628766656421101359811522386849734312179769425670257807747763366111882218302544429356682960973863412113938212617992279105368728046281321277793429612514556884875498321471342186005935474830535432685394050401665427753950451870384061138957671000505331584279579,
34240410594818328647422098466328344286784644650537878260561538599720124371096198454481934962221776750210195480283042620544650728047644405358070840609700238954032105344392065606382657066871932610543027980912340695413371634055692390430762188760243439625448881199716525038940765711659790942633904755758931933973394972557115014896582730524811582775165719006608331601536895581299976733846094213905437335060330704042911954267160117024403703576967676979692402911756772597787115284555393956162858666992823855559436532779737873272802348713404077936128613699495330294254713783721192617857365835559822596860279638001145191102,
76798268376333816029001755067721409376910670449146842790587614179766401659340048414849874214553660331148352037063164019425065835951388337241052931755719123818431170670583809524487842283206994989538925012732219486318672947240804298035012330777 4

17486762799947998006528011538313612596799263463241108669469864 2,
34225409249605446128345851237297336166455539001744596385951374997851108334385045 0
98006356850351709807113791111109983429735559987728459946279383689080669449744484 2
16064768428427447084894095681147381917801490711822519249241544083111736676901218 5
50180315075062466088701329497598991799377948260473421526202920 95,
20766281425624114311932604259092201712967056339847430271237222518288992048509329 0
54540981396191843811480827356317057418689530085144077746426013310341803382114967 7
47747912220519683987245134832300812525989696719086965089150914807703017021046733 8
36386715835945229197701748450911529219035769651190909110348774 94,
31890004548947908515086438830034047576869956897281003427042926990518194543706194 9
24355846320469527958280147927242525644374096698613866641284254197492051186398514 2
04168056112659872406505915702888194413313863880968983553002132513141932472034741 7
00040621278301012963106622951369574827637258599118286738700730 10,
29071507687583678865867168564921461411499877208090006273861114216431100941189833 7
14077679299040333719160685362820443125980410473755023946057508269246686214764946 8
12290690747574078953434677112702651468137439249243504671669612468688195440223377 7
76117918073714410634740093145154507646202200736369265216538629 68,
25143850728229389800049225077390396232131977500075795078632862607484356477264475 1
54784217428559734071505980832384083489409486597891661849233253801043580235839680 0
01019867529222810177363364381153347453258084374037566002709892742486987777942997 2
90259756212350938015361258282230109166231832707921678233913143 39,
41291014084269257172705523799398699694623395050837505447977365412597575103201565 5
40694769493664007280417918142464889710331681560323235321081466834452236352466675 2
81632994306261898431501188575601463722153072457841189490727295307578570620918609 1
43865301469662950842377489932977551707881551397016094548876421 89,
59243507079285403646190275563241515093064621982527549499570440521096884104062969 4
14432792809292992290989893401979711306901867696565944894153462791856019245463491 5
26319474467216290674824532335980153903042148636369347974749273916786007679311508 4
69843178428017839590783249629209084826924484556849677612104842 0,
28650817893693068931549910337647354676858717072773772181832233756177799817999738 5
68547867185819007412474305935635382422944814201705483686196938141169861024310747 4
61025305741293851086706748297938854458234883667462119397902553188194706618646024 3
50045404163946837781550910073653144576879435821060863834343508 61,
25139125871426320495665719218523118894316032165213532926026400247643849917975878 5
95485830620390964422666263369424250140316875940643325939740693298925491969121322 7
01515242095219184092576670884556041515201803043121402792049920430795955073181902 2
11663206374936455131654005707367617494841761361602920468220749 8,
65938472301228777898287446910267419472964295076576294733902943829446562417890394 3
23078103500719745304340791708713044069690082157265042568560412496032719508406943 1
36065123995193349831155535839879654945052606234507541019721769720663340954661425 5
59382052202193145153191270454634979475451006908360903140924893 7,
24260151098609636852657465126879377890913779220494780065308051808798556040626515 3
88553519077090572085510106816420614599955700152425561851236535541752288586356842 1
90190152859459798016603517751216968474727347543015514925959868336060311441064717 3
92466475633191181766673080204979019398399889170877461812290656 67,
25258770310786732873680553725413473086205876098194042383609879489372102540034673 9
03514890237758906853194581571992655309620511875246616341434155804454222221188391 3
39136709887919879135504922505015694467837015264860151100503207147401923463079083 5
93459741700830632270171433800757811427975617427571685911704310 05,
73276892858011915609471490547043175595122000169569296725435626642480301917182418 6
10663592004701649373159832296162145613604542605318217428129929817145789294303562 8
93269359300297256981142347293087767592348325032028519913409706362324294224532033 5
61757087600631614228614147773290364251447599839660440387553812 6,
17130753181634825430356078886940287342826947235408667712574893129497317298674518 9
50024150314380721058212819314060906443557229343640125141716943592017236335437901 4
21370931074908265353030146050956598461371063085596155898403352328326367238682152 6

95796480521172572042644240085952057148963888493354587484978225 43,
32060306711289841135308303091778566300611510772727105076543917588601970054260178824467861333537992988311337855980994944404003903204225850182185543312744577795919460195289353810496028676031245484092828169397698036952429979188003823451287026075618285570931746434247311331326063072497094840063968489843717413144587760228269293569620971231795447087714270486485689182824942613560559426802426164614779754528032767938874590876723313310034315230739756568576245367171214580599493394411357805937001414827550457715128744098917203101561106644785600536533224178397433628937563616900943959196816468489090562767050682606764474843628937563616900943959196816468489090562767050682606764474842958064551366884602827928343283996040623387688149214389261766869084234039626285475647395859890881939843316216764314692105754482704231767663500534553705281815281876502871281138420927733063871177070156646087679746494752852321107575404202900852853872770523983693744128391494302411219010701828781777797668279852315183230747468539901572410823712634195405604813627486581938983970506769088605108642213603692757389396301765946984918156979117789903373888118705804351795764852912596312935816764673348499015426746456365834049611680560504798561899893873110027359835176922898163290326344366278162799640783569200922149384205009,270988374038981027936538594043709665656278827060681049869342734907611345058571036212599943415417208392738006204259157980406524038630572568090387668300804141145905911838754547231943630885369625681335708908003199277399950037027422523147106837033328035318735540087396163212670798890785998309939176249480177286819058778038092777460399234081285594602612462533329915581384820993224119222044042566533197877040043922970963857348148886489795574673597682338657570358953162711959392581023242586696793365766793636927553684332842269110876808961827286625045004995839176818984285469889944022282458932576412697109858669087420583512777093303340675091580625532323382682129691864696281736202581450688282358145827305972597165522439567019169702252499397105808228982816274171858619196572325252090716164754428653471484126322475352179404916448695972332203909050954267925410042960886379999636798174822726282635306472726743389957101803486513270453140009906863709227313397296211307218235964345952316434859201483645012218620602612239403083459395937838371876801863090690966217654262147178307651732193084422946004133089508867694551122736657747714192769376985336009408037373813098677681718322124438206390954620854575355633395800984880214324179437877868393253595480232861080887148358949937991373063092649181462448751020036819626579067760501815779519646268230398603103950707525204526097843797674809978090578295201129966765712862260632096215874817242317309284158864004538124514667981606709682660369553173628534354955141758175565938353910452585266775291486924040287287,42065643902741813470800142439120762027748185682551935892124688334023088735150052619041074073507614681511804117855180458967938825552938313548085736320477904719143024069276301747035615255562611531359363557236708432179682007934172495776207590954863762268737028069663916226874086706703320244719462142443605705277695815468172837491077791597401747485231989274100825129159619227667337678138834657132124510897771226911867302729854863407985485725374101983422827889090760074143147431301465944273503372859686430877148689612254254270514434288714205588760010753570183010447576871563608252114372802716375294049628998583455681856572076435197770950017315712210184921961064340013413638913213275784072055670800617280198498410976367292113059360904240872793292322639230272883074388688191241489247501227134113953797523584042177459022347457719985960527177846367,16739815807671419352538404040672833892859847989636048024735355624418411366639825336212957525423704304978852536982359015163811623118451179521421173268565909295430909807738491805673954296276827379672218750122574429221705433379218319609142492822931524994048132229871997103423328619172854380243892900069177742948383909161102296124404708405686547544263621980523729667000508112952109468940409052787442912248402166979623027544965535642155902329447322208510091676218607484329746879331885117246751225023561694366535145121785189289230136389366656500861082418069

1077225750190142248598518041362314450625095641688369586817758181,
3887416604133429389606449254495559458167225845439320898274897424317852791696295373313509870873845629557868181477072374026232142315609706739829084179756122068214642915022078095234288540810633350139229158796821451877239399418934904305674776517367438666855891631003570448701637640176452185800424196171496804818,
3584923335656760081388048204655602518660068883398824707581159203081081015408376551757399963828814151527554461784299143202709377383213906957451546234328728630539121321688315552664038033152917500527208671251533962724714530725358937657015651770518170029333236436184789507378836671360677741780954664934595619255,
6558123352435082018414897249190371711232016589696338869498807325172670838907656455933934347617855075155355155656571162831877847361786683335451330762736361381358339879848098481336115326433672843259043394059119726687562204070731825645063162090550150144694678896796105960733198190578830161162005017508606121583113775244641122181986381191288292039997042853349966089486819525049061751051054873716149031919321626703544417783539774253194197296560794790000355981335530357571600127442879571785843011743587545022436919426522578142377799506957957999574030096426711613548818233897612949883540018947887118158145895368659001913,
3346536951357092794922857686071205592427326377071985417031573600335117248052742314887234312517253459355498430124358526765831884843781046449528488703718774190764032463366296311530766780372073925895846392088965958209670093159081446466328923256632162349085747377661326885434440442540542634700414979519156846378,
2271935487447997615072827715629425090467014316320748569661772071398835781211814130487355070035384524783329644561795490975450269915996630916738702592546834206510451616435749464537109864004197818387853698579320129733065396669474592526208512959974455298956514608692308826341738026708214485942082640212151722824,
7764664921433100725962688021593307895649636008286376275408533992979004892035187210009087933709864616312224990743827574672328645171378716699979311394880270848429420954377622560793504781917498239698203415248448107429487005984950485598833974939314408493959127156525386893997928369476628528201035327175679908251,
2414274645882519519084023913396357272407167832626523088150811990492559442026397060738140352663005223129337187125965764012914780402110158227540931469017964124622910527406899007518667711067230671586565911241474521955001186241607041127655651458938731994650169928913125748912643646524991538893335177389904412170,
1587401214609797602331682440041479743512176522669626586231108195155202838917998415252279686533695314095297559308594308729030682798135639449094642378150127492601244357366175151522933633345524386369722790804782728844770761877138025423706525297203340243728319358418982837426599728607392540185563391682179754091,
1276538048517017724799741818279571647670714435038811528782147972756642391251134331211078877513430205745909112356875543624846934625909319249857420925371596921159773634075408031817229527925246856861069484405721039888258313585591750990749824522485321295837665778185313292953880124411255737622409525193124459735,
1538485199904895231757088486985252668728362373981931817960801712480113335144533151864247552181700949983397258308916404658891775035166276951736469913599712794627172532234260428887502063542242388200715134922033848610833955337511115662975240925681126009991412680235365215325155354418216805045691690371175736694,
3206304631930048482883026701884292735383942399324272782150656595986130488178312338403753056282079561305071127800552734066603684476317468336439123025478022512848222910190232348231941471763205513385207943068182799579334996844368914081954765697685511608500913811387200371518850567627179307512637861874238561003,
4060974136109469828397749836390448767241682974331277126152803469744907898850600258735274223294110211845166146970522493710400206529822968619805354770338935377616260678509327071278654634247560083944287155551698277429669016420247936937963162876469000789003671771730101317302537618538985892325942393698791077443186685901580513856509872460979635165561061950606904814133798621836392578498358298660957761557506177619855027178183087244891177317508831823775875116819379713319584842647833087351777216785095435169291029372340194446221915994696340583825222189633

154046896134028705542642394329177957210294405020319531754308635 4,
313661133428479213336726165089022723293628866560839370314047696027394736399595009
061621684186236467622497599831392749671744215128004651305984423492581348860853886
880616530461515346266692185182836997334882051490094522076853071834051537505915271
98902623796228220924954804722659406968526269660830664490810135 15,
133126988470813640511790539367369002944853686757499176475681011245613379693986616
710575240171065406479984181420857996073984094527455235616253992923820126309476862
843542942478026940561772836166429132692208647072805455784089659486198493233701917
856460476742506830589637036156341530762287585496071595551989136 4,
262968684252855436837121264926620711810429776771472463771572485371006222488336001
865818408105183455292293810368062545776852102070448913828967323619236179721819729
179001189934154755016884719530719467578201672435546183812939169202922887336008049
987690910159824690100973006492709029852333647937755801749324273 8,
142630045088575440133330855307163354919372820653790514845546437388915894662534917
070873989529766940311083546705850221037690663932998535645753828252156730775168959
639181638987602294346382061679749924206559398341355204227113665157821667647735851
764448144520254642495027640933205366570007811089142267673700026 6,
111514358626945704638850886011873129640370274652565279451890140258095310852484655
783625187233046454731618554424297924262649207065568934837518822045634899065672377
535104067086433064992768211216239433538912119680158605721104550607613490245902197
786167606178658655296732536707855569985275795387903852996047882 6,
236701032975904414174045911630233425873807935230902540186518063368196712971804938
176037426553392818576347012697023315339343336214931676242786933854219349061929526
158472118039064244311616212526006354315827709350351939666786875417098973490856364
970263950631018232443296414900182631256516529195986187631610717 4,
236995657118268711338512497054894287672208602891600366860749255479956642650871551
471068458041219667512434830609430251773637719114656003341699822240741070027696803
241501862460314096971532156085508780158700287600091437406524554046016436906702805
568438722968531740849895221811739212014502614193206572273408657 3,
385484135056172759324589109062533853708936357976782203401439808922960635589269934
724526396856484676714674178358449220709385044780590483137780698133500079675572065
024311805275820572595216788913578068566630279294681667775978707597503014326319845
435844400031489453786118480022325620061255072934543461206939273 0,
340158640688687348550777632810114487882216792249387554454448175160028530500107383
748517674742643894216306171901215926297982538087084672286465927876762324901256313
005338047100763370377910102187657120836391391641726751066940057910641615451198448
046997927472484238610436680546443258301134408251054451038743400 7,
178041120474275611333971948565471932351404967165575281845857171966978242921723212
809572946372917510151541788039614721505040157419795248566668352821587823216792753
626561544846147268826149286966974511708849214639181490322750831939282757796230517
690391312758956146477604584028237514130822672266043106626287908 3,
623158759606146964243321604792792604328830973124825909315869354153270589280565795
450154648529086968171949619658615994842143616224671760829560230125417151577087316
903718191831791549792534697044357146454674539729006792403062341153282797354669567
567369833711554020746696510756773687293614969485135144052897743,
287496289887114199498811667825536006651392688026741778171109568230152015511191590
939788223083241531391221903341857505345281951680225987160612875133280289761981293
584979115328933856534647660194457027787908370017101750216469476928585894768364255
635732474529106696017423531444658574335558553149993762950439937 0,
161587310838876902054817805508525530121670436782419768372689028209717276946332755
576019429130935387005082418734717203761176608582038309753585554583731811635335369
434318742422507888364574861260937176217026730717092621772993685984707573802334026
987929577759724971874622740388688737194021266928958736570033699 3,
525108755827176036474671732056255094683730135486249812303654723006821993795522820
737030334467297449372017838930343899416337585019857607592395449704179963379244795
114955118499384308808794611614840974952064962588444759072806960286163050053815 63,

```
0049248251231364052841399285728872519581198866491674190662640 50,
```

48779365289983968825928698781100067214652805986756174570480084 7,
47000013292034498238970467992491677551741573470400319783330056863498255624417795 8,
89812733516763387468285903098107306282047337867493299873031114647095658491337710 0,
44618030473233769541808742175457507268294020629540207557247725801570674902623898 7,
19303951141351748824406325487645023871078099992755895864352245 24,
15143707978191792650057806516536795474287982580416807777078531695508040018185202 2,
96000388960340233669288861177396510714395930496964246047284564260547010667467730 0,
97730207443468590337964712329971926610238111386879950014829507814359102880711632 2,
45272544422994993766541763214020108131339777580307721074194642 23,
12232829010843934803357629704416106241270062820325983037506267010007878262875535 3,
93633573608689263327717590774866536908553160270380475076802020803574288066068980 2,
02953472073735300391810158413025502985788866950522201411707180463350029830570994 2,
18114420902713062583642479323961702458170917369205113739121209 57,
35980887639428918283925624565784838710432792956471230313607763811653108588668738 0,
43020886658501134581966359668973633911582790760577310548446047077503150095432184 3,
66176112142665632026446731777806629830355777445332813978563704247896031512759078 3,
42784056545064526124812742547577682340629375429984023693381973 35,
43366426811175304968615357700647565645816029866147864255454287128265900126343240 9,
46217812553754482238803000185203543677591269533416239401759204374811707345388729 6,
33192411405503043201900054210499026041356474154471538635975820033673507696367815 5,
29137864471531556493254960150647303282121675649804688802533169 39,
12499202884198424473740968035712217097512851984888529776190894108535846321700190 3,
67363056084232928542709576486033442591528616297008017542336748101487132914994183 6,
36653682232894327198144443922171198052825167385293138449559729883702240938615338 6,
31678015183481763417997618721601748376107648834857360437116327 24,
25028102126684529934509752295879082924857844909246034124938082873001540804776422 0,
66307563740435999879047737051770241402956550798107855230705776844389770362396259 7,
98540701650092805132362155899326397159860877582888589824281499090162957693610238 7,
66511612296133035154316806201826998769024363559603883430499712 46,
50932736588134120255433446056436386262390561414568792335984352130496719447833939 8,
84173667687970891513285867898930301055544002030525398401674671848649141114633856 4,
69870953515878890878374674190392734778654841953420820270048777151167931205775683 9,
06960312458556762340711915204052805007089359165376772710813471,
14940511888927774761956354162722220669802391565736020310841178581286714433538528 5,
47573783648862940416833321215166185371379705715371200014258839309036182157337910 8,
45331400743471516936122855848129191879195656636962665627280723681781437710270135 7,
98525207598119059398343007722990931391043258111991736733347239 22,
20834125520023807438974656805154841620335827166766040731249957578610320233094446 3,
17162499093693629496909101919651314687612901443866879679717763925276716361981578 4,
36688460599447133127203422035817515834949546722079227419585504921248901742744280 1,
70309592741457555689596457988043503504139824774113627892704791 10,
46959110194939983259850017204149869300352247203421864159310666153340198679783969 0,
08959792267964184596961399643887845264005391516659995523932676329191922415516384 1,
87579092740178488238993390585613639276940229267775187169460840071339110237535103 9,
95847295744018714685182268625755691580773774663961347547814258 94,
43093860817806712136479948464335448873614243112298321646458982600372784872961454 3,
07120704889563223580887121563671864596109301880434246945429839752323683241158310 5,
90740522656196233437605517591540912150013260822527625891489722864021419530108457 9,
41089177430261607505935950966024224657140149343408776081514654 98,
62233353558075732799532813045151198655321276992498656912477756392272727356304739 6,
22626449527964496646853355184773069131817086435710224974101168110710662287407199 8,
40349786471854445209138196001546754993873618440107897767900976114178113841716706 8,
76168847049470481642690185499789791121983980808141818704294578 4,
41481521369423592035593509308856839700146226971457851847729995855256434293560518 5,
73452179715618695878518108419354846837468017019898741456561419376243609686517744 6,
67337796124516494925108285989169589331324603906633902249797292861393867450447635 6

93548513234903759447167786452464567121516518853912015286296619 25,
20463407301831496264772848299411111188274378982236278935608360919677665281030425 9
59333239887805545144369068859835911839665017686546784227283712012752089233805915 1
24382006942228430432600400916724195399374307758465586585567481183795370023417813 7
77108063273186379986381374392174476885061256285644444597605165 88,
30610667546451590602289169025256065815237393132794602514438880822050679775787559 2
01082584969257103212540286152201212326523313528603017701776995981575505860230789 9
00453482523537205364510832598265930201953530603664562842744910176894241469894705 1
17981672799691811654974795673233592289313887964245939785063588 90,
48139349735682992988036720352173041090713042819943995858656810837880087645710779 6
67480389436568038531200744419309790728258778911754734189206837944552109258476131 4
71452763448312572020594507798973855580518888537416420189119443239579737599510705 8
93256782610587983458822933945822999405649922500933245650821781 10,
26757512737840352453712054680062876759684578550928699615303301228217453069389588 6
82115954256278499486778568786314440192437040071516280578696478329651580235899324 9
99692211687582362760711946690340998487463412263279602103304574180172456466188235 8
92279652122335045938350442655772214537525274778983717992145914 98,
63259501962344463969000073140084198790240429739453089099072658945729457493790705 4
41534027045018570298675600531213606063481685056512622006875338069613752315766405 2
36973802154688142525694440790280287068153644964219269154713103975069384220301956 8
51931599394359404412745797461788150628356985105424893196709246 5,
82434442585234548189490971160613182431792922998179885800456590514932932124184565 1
01501669964457295210486493662264601842260258330085562673375888597776187972235662 0
16411095745252157734864958199898092234222515458842925074282483501252198248045248 2
75890576338835753439485292291586139258672603813924558395474754 7,
46484414590484031930534114117669921809728465256532004656121906288689120322912946 6
48651549714615941827175718508361229145058299081751051297547141181276172872863578 5512
19737637632286303688587754264316489719621869354685789417533694826628850728608835 4
19716228901281781501782735824986855538578424125148188011422990 95,
45961098630756448597420801102995385938312557269606800961954612478688054536610684 2
80473939940567802422489176173040616074030547813322922778335118624213417445440315 3
44528866825173212020684791437239025175546710180180351419359283473644643763131696 5
02425555346888699289328452914082741612667875013407605206252820 57,
46363085370152145986353135941408327978854869971186614370863058595826404791116902 2
36077319014348506526197120906002433850422889840623489418173457145929578611323847 6
08827582441727507805364400728211005781818447447323393984667925384866481484594241 8
94600216898288879044518613557448358476805427377009137505427217 5,
19892286042651798489781727195832770220726938711081996243565049651160260849062033 8
68809661216403754054894208027218715059555491381693147027960151822376829764796402 0
32589886669575172383520607361264674692581460346104819198868733010243076773250299 0
68181876948651109979345152135174371808529128017285812287788013 35,
28946844243350766895555468121757668912807084642375410030769693437831993705694761 6
73774299058851411981098672752756963179474551742159290065856138929315800250881222 2
07761354729155672175056238030433530930281066061669903760681217053499064468538924 9
61356883664786637611142892704631202552666422003938247642734991 25,
14850223268172748923055712798085572860654657812268878709610529014840868530447316 2
37434867684635794514485552876567951120824839664745273711945000768023385665881800 9
83625025115795119015594139447119781502280114941042712698608005611743124243966767 0
51365919519489601989572822502788732213208811013805449709592469 95,
33686292878504299449997719371147035711709498228125981330659571747655555930530180959
71852871205740164650817540569608633036290373189132085305715087679005004710105851 5
23578468724729900493462319103293096552927831151151119965195586311094924247387934 1
39637151500471508802650300042096469745003429305171475914366735 29,
28092310400498557378733003455481155616826303616053975869199126825786278346075847 5
46599684362650356408849462751774880870386990098197498529248654238837072594650353 5
90617584683688252445695642993074122950822042170126084752786676984164364696022446 8

```
90612894547775761896888546707307422411379913224768967195922923 32,
48648187970937226235656776858595439733901128307466240552295339808067767978412829 0
49751311157280268033700482205571635432809928692903097919212284340492511781646372 9
79370401311994634811953816288542384728252075742520679139082486602293855551858535 3
89894801403917676609473066263985535390357461472597676287408673 79,
31617876995782094833988639661715740858006350141286182810695632132064911398963987 2
51384011909241033802581445004004441915787193456072421842787472742584310966165417 4
71306593094324674449644369419329126472274422383828976079309681218864112889765948 5
93632082055008510002812131825023974306224275060165467029939205 98,
29893611145591600521831900840549871761161720682541932876491950763478185086564308 6
54365872311301305933233251503715915203514701212220649035938037251921425346737714 6
78529290746300807075605436348254719771164399634511002845227185052873391511827927 7
30202016197834931511190419731677588734623025662288130683880959 8,
10722123447046316554749543081191629448447834423527020670200937417168561742277942 8
09556297135821242854114281101426152334041737098480261310910943065811528809451006 9
85212112222470149334584927346153272714266319060980366665712665275516435931714198 0
58651962730511291609311495015099251839321646841998819142803771 40,
27046037286781509897396296057672776263930669999889223616673959614845529079537438 1
26093444200073819137059709570022178904787112304855302205520828859493282889334757 8
03544325378077980271859994392781402669351847245791928012111947406376969314075604 2
54969307678761685669784971801007027246067379950268128511751515 10,
47930489639832853650056304598289544887801241239379424097541769355815986721428222 9
49276730593058991455686997314015789804881593889360461578569652650943850662818925 7
42198290633222705222056996052114968018527795183252150309444766907377313480051218 9
44718998441602813493835174099423654496327803469469387316081594 24,
18597159365486464558454194794131787744223129851619503743101302692101487332280446 2
79226487648365932930857153799214779753750331824604255333974855164965513836317026 5
19772052144445287225050578055586207312180846516151199457208897340919370543400482 1
42647363420870833347525697969371280934772948605519488234520708 02,
26486534055190555631261200339493487707569583656528640108940184799497110104136347 7
12712412703836959270728194735719553576133807943194112722685726151273458245845058 1
77479930497639445842430462537884647358025334752897167265070111692006978197185740 6
07312070162857018122977519052257094203881002316760423615270856 31,
90239929852835483191061747049921466468476475837782978939239773148477929288269589 7
69006207031767074651742170246307370975700437349896714669833723709741826073469612 2
21810859617454875937078754911970157512882100217835365000536995372859359428994677 3
39914744365238924031670113680954325605976409874529267737451832 2,
69020266867653879121060366456863179986058892328002255369488298596657844463049978 3
65636555110145027445796942941165112485522520094731328856091901989465339275721587 3
73334150938553980186299644300234064924219130339270888193442312995273585264769305 8
37443162550577349448093810509931564381915006605448773662287114 1,
62238787432522887534200603192360089622062946091585102713674006904641141886029293 0
66518680270392099434924969158754889024745942371154076898632050465027708402577439 3
28499071844298914060955646068460492239891682602476755456621671722394287077314933 3
90101676863980939291156676325336514171930867856508595487692145,
44985818152074292203353362347234111097192721738549904689885220628059559699181534 1
66575326404003845192066278515135415182995218704173809359695877335635422920373679 1
50418453304121140227783351576447923271060506484995466949940024988264883233733645 5
09319228861044260499168342046714489181614341988699041244032789 95,
13791951722207735795872103964853799871074633888861991831000396607365400195155904 3
57995168649264847525516867167892097241823722207323078542035774707912655265991918 9
77957190859275840406176528579868131678677880395157605825489252935396030827748431 6
43227858879817560736492436206446115766312127430234129323455429 5,
96332095168274647494891834022193461963367883805694655280716120376414761733206426 8
09680122090284540831350836896125840521462095088129014005564884571371533264714474 7
22517152309250568249817437699945805907814990986446432259144576911776468441669828 6
```

436252489848360749644537840466325440337825677763722793716864623,
488560295998661465237742453554718911630027796431040109526057219985681826442098591
516839167671508123757198701260814095303706255106332703460123631096071824224345188
644445924621953769734563597104561122444841877394098450351237881778772966832228717
139015030014686378064781027337560175535382556196190852717731118 1,
259000958029088304557341854094131193875375963320423143308386184336450201347241149
894344296197685391314493979078667817369637772196007610164215367920772026019575293
734108996448897296264055619826331829321922139431007213987414053774348976281701109
100910741198279204375115992255699685364624435765300450721459705 8,
163833647912162818621759130429259511998188781350532550157192726517695861475923428
493225253350001598440488618000328390036034648692115912761300720174019751218428156
746177547131559725755176228014284591063084029994781092144247224651830067777013717
722395096651817800146606382353928501249305870714736836433211516 4,
151664387881790724555814259027574240411137773938350886746557734075789654280654538
723343166318081733607828386783108256556266527954943235752677511056137672887885917
772049605854856512443925750297034018053295338397892678572899482548405066955870685
021225587363547767008605881185690741528343051553317248739937827 4,
357157591977484379580998759063463999281509375335098911195974727353341885553891504
242762062416514657649994745219749890230514495290626228012219575980160573030312976
060073868919901316048069098179504188322311187056711289059867195164898374101306713
085601850818766497139238426196789795702632381589686167380021369 9,
489240522413940513838884680908927023591146216432553789148688864222744647334599746
035874314746763347579654572234122798814140943448652386629451866107510656228615341
315223330643044282782963296749231374423267932841613783831591760210415132436826295
706476628660495278840149624091674510977278762898143932353183729 9,
413935086848237535833991006685362402453132714805619074917508256401098874382357477
715201630713042649292701452574806567317849288099342044413379658794091780655470027
323514528439424172367433114821166172614659501999229660955293043978404316667608577
219860982012450203600720592141399569020461681656754391998016081 1,
223038302097928488089190503174383422763453531582315741842288374885153439771233218
651657585697136845491065792551801111866986816304489791951535702651605829482181805
077276621951731334566875990509427832176003781070427512708034547845833019102600300
397927006744568281457308521839606717857797248622515658516690835 7,
600717652933852348785246481199088283108667005937624855304433140679453331083355624
248530716724543658308743710695407144109366744474306911650526299722212721798603058
665309743574106143736804538659126858220138644486645359248829113324069217677121341
197941443376697792885174330761793247161646365337801457957353844,
387476950006811381037577775388426281086619722070522652612399457690600318641499719
236004400483204203645350414894866228069211921908652347416015873957492886969352083
394965073436328247158103149481851115882824147123310169565344444808775008297334342
775045239076807685035378554475163249760320670044977487219219433 8,
207236817693719361305902653268872215573008874118455628029860596785467061028237737
632890135243551031058827057224875904154692920464926073609937230057445661200773884
061502399107710279646049056105331938475305166614099431171514050517042021892665875
016510717070988813021989471792737458608175678398117335094456422 1,
378537608313909830822569031428611548457943580543356336512081994442110836618051052
978815703912947945419661463266639622619125277145875023749528930848421538759608026
094850073799457860493511314285805167245823087072807197718947494394952139097150691
425038172272770214681662154682497835837185669443169676679744415 5,
191853752433169638167307060579104766747028645761850829982991032637817806197053892
067959402753063957912405490006242999983605448187152866211586010220815225430797118
413447401237672907436920505911669975762757878620513419021724874606620522923674764
459461882316187827503621347702574669675905888090164824023300708 6,
181718876722083363127187363981187183847946109276618648384381726136474705149908363
960656757526154177718798595604016326143289500594943884529163645061558413791153578
884560345014114969711393981834401105781871388613824328781288148955749261049622132

35191326081437036248213229650219795413622912729175816900873740 90,
39228324360647548823062860844899509789721390758982559294663377275441662049645186 7,
77386368091803824087865555940174258045082406278763809569702265423415575036899931 2,
52041112218790479030701126411078387972335556787723528667856614020042525727495345 3,
24886242619471271728920068409364772641948512386465636498683724 70,
13691720388530423651904841484401421627897535635974979685545296368135206039061066 8,
17017281673432397598171054749120389811395037304053336815638550637430281408652100 4,
92554217858813042095857307473390320279889802222504015811984059945101284088117109 4,
57655326585382108629666280326978566749916144880491503664289327 46,
28721536582644126546115556872437449860367850210824079403849797525376666878505435 96,
49812688891502408614428145188711346235952099635853636579414857692443020326346499 8,
49458260876763487912307737497820139835678496734553916377797247514489332230368015 4,
89039643910641754561519934275937829922318663416745420908893283 65,
34035815446474234961812514192910069766677932828086573465592378856827547956718329 2,
32772427953114084228139742956940672723506454566501851977483565140942391261663746 0,
11129480852516293109815161177289287861601910131422791973711529494454468943442352 0,
58482607149686699666209322154518783161651888133143617316455728 40,
11435543120606969605270316440980984540092889282193023060652204891442955031900963 1,
24368233906329426897611104275229006555563309558515140007248523549396517329042223 2,
65773923855923729231052772409224166921008371423712038981708846877679722586842310 4,
34300302971997534133706222673542803378321546049381020679946143 5,
46282783825119055687641151259180802411443135227185243057858730074736578519697892 6,
97032106125169238934350025439932245003801434328902343400140356239347995274244445 7,
47798733506072260835763878480146674717233212987998262276016672205332247377622360 2,
80039160379341290025157194334767584363601517542180727822678526 20,
34571579040520485794815014559351220774267456128048316621315814315458105948689898 3,
15087636365192548976538641116083764814414481261484698379867840089674671644636663 1,
43482767052477081044363679274756533298744192665966501473305302308613444707283468 1,
14257640321174014999175657603901529243631733074907630281187176 81,
70848970649458470616248735093618185722307980554550553765309128663316360066031803 9,
02400424712900345433713312448022165347791858370653204556750138993928751327576288 2,
91856219608053546191132433397310792733310024666610354689443973627675166947024797 3,
76714480229848828393147050745596814104456452586607454763601116 4,
16590109453877965916202079818692598090349755159361244766488140128043066446687506 3,
65220420942958694459655470005301748819820022349418753799686788301392809374708682 1,
77460999216229730847415379705188921188654650056410496613703416521561314744565621 6,
06970235695413738959518878288022246608003521857473984178798640 00,
37168588263492345192758107141213708595284894165539423717538904788067586071009072 6,
63351456039365715864960624635326715707061970170030403322719289053773754257008445 5,
40479176074019185271474565743692752361022923567398523381312835265602119737013182 7,
25578123884460338239984691992813658005489158736993854242361234 49,
28635893952677869782778062645393677934867664222109070277507838891607594619556630 9,
71835125493167225402648945071679394756072915067094771682092276665753904410083671 6,
09860990606511624433807793887129867542580847045597831165522874218751657986136489 7,
09172537428836698642153191526107070736773562243501987318155452 90,
23840806676711807682616373457370297987587623415404569568007250107517167824621958 7,
15989449483602729966233258855909108275502572298659215643276112256827269330922316 5,
22492837339242510383398325813335599504153892802433838693750543111758006868617532 7,
43704547305534593635084951463297715021321230300816629921304945 9,
70197552317727866028266168479763438693677854547447328876283741160869192682241149 0,
80578692647555541218548194703606376327316783061442656261069907680122470557402126 1,
44256956154687666973511956188791063861700741764442994015577724689139699026738671 0,
19636029741899353734668091736026785048542276099183277497018150,
45449409943653664866029734158526129588255726719413772897531324481855486192779578 4,
28838610349539829786158369513631155744843028852511047862659361012561144930014802 6,
13698150296779337333468096878141409705282989140499590954251288446183561342359988 0,

562010897482224767866327523140796059092950008675941377927558226 6,
347288239710827917102883942729192380922431937494378558168991141286786705550195350
928713198735231964588141816428846107942013838283761152189327562495268674671484996
676135254580544898480581931351163489522958862679365720420782735743219253703312589
991277947312231417235248175665908350512767051354728591531769958,
670312358589783873754252327861330102026365609268888787802990853581468555369370102
685378771519859186644440920011625687868117015851422916538341850709050565747700425
765556716656791334483823216985400089209070529865962269376378878035935906731300851
739515458756617682957096547265976494259784934442799959692127514,
470141038399567314294230432184801742407496813708937024348707701905606231509200807
990266376939280323838931471809590525130798937988508222516918163365346925447023109
261882152291231496107176404208817516313038127770350056700053839497306612298140411
036090349180351138835156631979100023411485097273280618145905771 7,
250847499349125277450424834079233717275437843576767044610858952826992368701836321
571802710128012112489021041423203303807367521631613504168112662318423988706061333
770607680503733983401610099751093927060910467925122954198689625893232873377160272
597714824522193944594247317034701374061843501001559959133866853 1,
352987520384200935571695656812966814077973691609845705328569337807909580905936 20
944329136843320705626143517905780532631487583970429395279122885309224976041303360
769466345299457865047318211079201060056157440644087953847111553199496094093371443
726703173157600399987220701352305332464497705223695631224794904 7,
514100785207810772232304492110289285660577118135772555763973678868447050637641610
984309242707161367822469534946473981623735841552592701021755166773668597443627 31
190489910612638305143643044049083910703525065196400676431965075860095129559122166
306872278155249131289822509619620288737682252798457501988812974,
133993708544711243572774990331741698576985595124461185824730070786871760671726072
230617216775721223537107972085858261214228662851409616447219118486299038881619212
886863093891285090693604181916734451973939869129737963108833311409754648895097148
428561257369856840731599445843325033959592127416840817665019219 1,
348470060526132003047088974651384090495809343844698823391330822965057915437908467
021031110045419003407316042144019162993570986917026909909539939089597545991944181
679072263177689560499086337514106818901308293481538278856418289912737822685133411
684962509172265406535658069668321272674061082734070058728985275 5,
196497851952368545804021008524587072105204230939367359296489800734862140211647082
174417741529572827640162662964867963525734948529817659967818760677940722582992307
889980985873594834643551276293886067072098610166029610080513709770490044283637372
391498786885404627860793546014000354907471485528352637199608392 0,
144617318288064812802114988562173620196126070528304022717836230994341702070131951
576297956505965745818967165795792697387567773683165878725281515285608257058305999
334358305604528854326359822191065408840542312233291031496650567024079447729682942
554288465359410988022396294834261844395579743887128089031268982 4,
451804090720038176630293743122706567956937021433444867971994442971696596975221816
177776605951252615421687296702763378575019623499701487054103507898201661360652668
392389143462139114475410232820883998020597551417910276879843416875211330988881989
510806672572118967436777014647735003065469886657954805299188484 7,
285431393313626884335605912916219573343988879928563690457394034752731812462447010
398379979028814115341062147839940006471538234695778137511924591966036183009098747
883877194746235042128634499065780433082460899315103475083340787144816655885537048
615781482235596057306439838643576364478845864364344403620766591 0,
277538511706074622366272766336918064151337405197915488456331202625384678792622979
099866728953155567201924249757375864557958797312957323820436951882931131439864605
279755224351047509772951839112387167620963742124586192657409462058341780047441479
776933890024012328236901668520961830653629527625522882593425775 8,
839455001952454520949446940864085043350458061086690925249684484893025062028863023
702735062015092562082059952191937526935456894307000438254101049160111024803524621
244658775227270319018591026722889280194855133382841112050441867401321300789169104

03074184217000894267719329247139963667171620423520918878867723 2,
39080355587208491969552598839666295965561215360674320178857477023931386342547188 6,
67310141663340125728144953748243388416609436943277876901273310387291940316868373 0,
23691343689238481858741778830270678275322780681176624946403293853029672713363397 4,
15328566543978202600064641780239969078723863527153901274498551 24,
11060877927503313344656349359392980525919559521491714528976250110951252977920235 2,
73418512794905510926857031521608458998358682223533597297956351191031601136020461 6,
88377078836261758949942250213006340789398938174817092264988210474829045500664603 1,
07530252795839047698118256237391606082276021521075097560879389 4,
40648859704481515438532362631682318522546636210147765667392152523216209395460470 0,
70979189277460963480322900069667321165840307568592765581173582018918805625722034 7,
32682419086378203507192507321152059783237469127876728142458327264547447683503836 1,
19594443545419996742569174538684149081915325566957340155212638 82,
37263032149913948337951017217260872343994379460677964082607614721442252219291487 7,
61647073694122498208397873248363856158249244932550466419755463228967035912498805 4,
71366757366478609812944880984176506491177519166624325442511042580414369384742109 7,
61428405119384675185202025982020069769019173973281864254975224 12,
38255546583374187775527932266011190701452713993559176458516625120539261718576463 6,
73928503639108133456098853847033741281299219012221408568090985197751644695208214 2,
45310632474022905318031751821010759403262750177546870144208702594890625025968980 6,
90957469146284634217285037023754033837762498823760866953265711 89,
46595356605822738178391642120355479260337678065718602389474397280285702229212791 6,
17324236894330742922079669030034373542249750751344338757281366829261729167396522 4,
29482860490727217662879254322977403815729379192468417376678335130750548966111079 0,
85764755298252942772045332979301845774562105401513090666415512 4,
62222206469257765391782905284397555615948387103546663126883503504463410771301603 6,
83600108107244421589807640362086480078159240618900822320440020129774153430056160 2,
50336644062227276016520203473726302266808312152781304762765509433358659796081957 4,
32997936312302836678336307155699411894234808715780805871276802 2,
47654553193498316484107079974465543432208255901110589385559272282278539976351688 3,
38484573442058838234994967244762218631618250159969284495854927332501631289211267 0,
48958974860849929937416528033311265366671909933591045759814236501708432207979704 3,
98605964327919329145204736961521716840830602748285601365727481 85,
23104697800733934115995118640806188527207338175380381626123895442465608996185165 2,
93143149368843840986816025521011213438904736963370383072236652098322923069527673 5,
51555744541658033395707302665085195905258484713752175817019078047890454293781154 3,
66450987287300474887309134938462581315966612212186198489645601 90,
24180802269254386563344208635004652783656281446897287672827354115511963740874661 5,
09157675028906766782314201206390605955482949222532229208226590580968546235775493 0,
43764097624966745735896135742022751302576224712902700306157165654844442676284445 1,
89437971072900908040559189671921643492865003157269915795192242 88,
10180101215571428148598575799539512953728517360171030987412961340404281228859388 0,
28238747266455132501022227346564770005002829195667604235177500608143615333521949 1,
90680898592650435804974474304270772950268012461036209315320950641673451910034300 9,
41282597181593244937292360882901072808285664203720881163041629 96,
14010304231938703946393263012102435330144671105227212326426474625385159342808636 2,
40762019980266095610276814179289652000501036722673611606303607936735081957280036 7,
36962834961904759282712418448354369354848959933954597060294938408474028010212305 1,
77973271431936703835378688359723176795544860344232875231671 1829,
14900413834122151985062748679342997815845670395086454110534957920459334281649901 5,
52624764950117592550015115853788151266365119844416882886245309710185319903661588 15,
00948182414007493815114422932995518266527173092152815200297939538746678852590461 5,
98798699464956767585496287230176207304042803974558991353077731 32,
44759751818116855532546633972494087397700679155617313232053115112399166079397663 0,
03944551118555108661395872828456138622567283333981643680738328558714680628792122 3,
75028233256171684554137239482234389880047349349431197721084856949314284834631634 1,

47108149336508218634915474617139168423082495744361283485681332 64,
44922106124486359967655053634889342240998714790324773393253617089569903590880248 6
35988823134802262556476627909124639609267023834794149646335859618988378597680739 9
79826196739448867195278943118116885197312406412469919568312808526252218734678437 0
41343022726973292936177953204339436794748543451322705243384995 93,
15558252678035118050932898438222155108110132984819125686844400994712677053530222 6
43268204510150512562625889487835322200942375130235985213308805867973527998505828 4
91068751591845205535926979905856685901183017961405716985781612459196261659406308 9
47998810504276286596752746572825095487396856834291917314194998,
16359426084259684828272957593601714331579651521901028426447583384605513455675171 1
89092706745618452149077374796675657447646856536527833393397709975313895895429331 8
70447371084088079766030298139298083351041609421272923780396268487093073553143420 9
44146553896070828420880336842224940082380732795909905507874280 53,
29750702538972642701648289045506405732234078934355737915295192026949144374032610 1
54066657252843650532752698332228504464559400866485852773055840151899104765967003 6
15331679441744697203661555888879697977398975859117728362095099101320858573002607 1
86678296213610128270467041718713955113969884429596431447306551 21,
47414942360389015537505154826365780948018971455220573367283261264310267248556081 6
55331628125909340233214579784324875625793107480112054606445943402838582704329629 5
80238105591146676129651208948295578138547063154840572315218753845731851629361636 2
76437835010548178774154230163703393242964193543467786591945427 52,
32781712720623534542863227950355621965342907435929838931045641452740342693732937 5
07563093104323168915281181483541496658539559417136068489297718330064602563155290 7
98045493211825760232652995319368008293676614826848136751586327920629905287026284 3
00783952194328785900543827254080936670051204630357056705182766 29,
20955971707393342024703143226504778514969463841607311258261029960035423519019570 4
60177608855295265269982289663607069740908308446504915300913171373550314288365676 9
04117314957563147373435246911553735817485526639430023940631099275645644167170890 1
40862390038813528246588222257828804884716726687962215829232574 46,
19854307994720729556401885473547869544555155859926002806551510961574500503905232 9
51626825516997302301390085891211507908510019730396597965511089543644042621339106 2
41446357608610203943334040925566598923861081120954741681731341509705471784636037 0
66989997840012090808033558417915023367371390980926262487240503 24,
18297024811017652790043441458070861824168905624566136884946827282270342410251450 8
69823635102575001360831334096021057636544049390554811914149120010308800225123349 1
72649165346806775109407706609868561023371213512267914578393277937556084636247972 1
55518931946850837207132570797333708558252314556238392226200829 55,
43722866839999140201887189508438356598617010190287653267292148667950520738083839 0
79135737366533518807812624911764047143673868178035948764145917902841693090858466 2
12684437904198851861404785749765674984725594592863144464701779622531966125096714 2
42180504874830104289046721639977394514659227486536724227309832 53,
13028527733381280332655459229909233723020395230502671088624367869404787389046915 8
89639753313551170780129826125200874419630941306258226209032896917836084955732188 9
14866643879152160340452208502381105450093497774987337230206684286275358213338808 8
68971767227213367712964387968830733105261280851925106219740312 84,
27978584761102687234099317852659304522624785285085277249280544165670154992073144 9
02846579210047747424542135554906359631724884941070857239161722104515828334777192 5
17135498956323678828139265361021380431469136620361432690239074277223532564596656 5
73916080863658414732317832966617984642120356629225950142829193 96,
31604519125171388005329934102551641887376403472347123619459698790869840709031030 4
87094687819829788017377427431551626776387505303319389914469025140990374858756991 7
98195792086969784779524084746193627529659490524882515941545288190147851939996334 9
31086612064496208918248218274592432899877596529572348266471872 25,
23437088636932946296863753214950833413742000565839266604720769957226347928490521 7
91730810348906309608869332066292681236349175652945047776713394471765716059143254 3
06646547272541158273827842857309583804637923106634888028097623793921385866991624 5

37882116595500255214486338918427724799214189519402225924786202 15,
34895517996862880143345066989136047049412127385961200029372180557086525650981880 1
26646809085476299823419823992177598780846290003482613303990774972188994288806313 1
59475181120544690629194186054054422588028216738992131993017152504053468584639430 5
00134100093187888990663419630524087361316794612788385229299281 57,
31199284897281042929862027534749261307507691935258507854198057226700091381160993 5
71928214993426119911821482014426779510247927782777337015133829953863102933990231 2
41389147667162145567349924077651527022854079179815698791383782726505091360271134 7
84878223180339902686877972559538568271351915515332807514215725 0,
42143473309479022573566681206700327619606578553155984190520377470462707306870990 9
99958401383448575649248833546708642844360459461444581574071501899853446089280134 2
01855329873754402716502656307957611713324057037958049353473857312773989544022898 5
49902649378631839369404494564188434216410540497837562330693615 48,
29408592663964841770469502977470350755815282614217801447455108688339409474368928 8
43179454130090621951312824273534218435994355408706085334270823103719914482636689 4
34464670122342354232246213068533224205302733109155149111992419400740566794890706 1
51327144753760519920219263597670404335255380156814489998401304 75,
54868954584335264470659481779191679893592616903994961204570447545074840950872918 5
77076111983834678132362134938986623607108156139924340424471415514154464830306345 6
31967129910525757828920025677047082258481353679341284971623963957671848984057211 2
59432655841500029652217805756839508603832530797794744443972658 4,
30106165971944078433570114937682263119967799360204476805993964644334602441082899 7
64952513540169710389218697239109164257678097083755349654494765332498365561483358 2
42476039065281204717109419526962388350586531126300486724173884783525332771972692 1
68430205685743500816448292593923235425584332119926230685671205 87,
30196106238036703781501241096144570841365114637294005114686097427727031834211660 8
84117889646931210542696845820336585383943878638011629358211735142118074164948478 7
16397628833809563545853771415540578746937367827292840831098970255085010012219853 8
83130114926222693087867785809546704671792376287230907724121126 87,
34568924421535528257913513041040385742527900186883054043963772688190346450160228 8
30660379144063793125850859693370362975932800705257289887050716141507522822450874 3
22217968073826067330437853666637256286568545862231329785871702625236953397066481 0
67988696528922755864475451229972513054830294238283305315939328 62,
41938912649042050185086184163393525846712601777298071273001655818399259461774916 5
42336811483898629862081223373111245427053438644520232257168896722532828374184176 4
66144186282734455405604333306228776606014269761135516058534460062429522726637788 4
32285859731282068255501952339138985394236009131830645777588739 54,
24070922717512159720480435028152953521855467735162415475401319543939628814424069 7
61926136959847105500476281043376710950962885623989631651751051749017695436813592 2
18194697709198398380346205236814069072158589410195676151241444286206831698451504 4
07181908068221281205505720371417341045500258164288799981613951 63,
30394935504767784255435428098257494480251086355883780369555292858696037658626519
75803006389846183293647495744271970256643407731904568903026532373009952052107990 3
71050366627539097797371569389532740437804962986737966969438469646299868749222718 1
06384481978164942640417083726695039145504219749467110312899262 93,
27256393947117130995917015377163499696093613567327173178873349596034469238113694 5
91694256878604931219825703550489632352872474479512386146158440118847043804778493 4
12255002958020891085450857965020812764534528152210378395517945877089314574041488 4
31639044858975789235837711394603828067501177967009485893133401 8,
10093064844535683673511354452838644454214007474169278753481136190414349054269950 5
74297630039997146258339447900651101562361934337624968364490279115339183824960529 2
18419066448456033045885586491471513650765605893158348951324276214489109976593665 3
03837836728252070597494037044804780420781066490578109149472661 77,
45429757250900928752653298189791137907891785319337975175005451463370690059445581 0
11177486769282309136798060694998480904634102233179197081212807222480128712114792 2
20127076931028969146460706815152559206543539543436739558656388949510483938035180 3

731727931318845388904585094744453502423143565044528816781054764 9,
4541184573914494358056310873718792156873915165017364649446765919137270473233297899008780718262557622801252575143138363870973098933309381350581741631290071906188187227622769537538376634847041994461786471554562165473594675322217352891801359308585450444547118810706738694625771019573945192976805457085516828719,
1635702691731010191983942236732369682553721178949755326872380020576168419895627594750721768640028730725526766600288974800547044582100453085620162260228694559885129457259603249847460965670797863491108618898897657515386424594055752464965242098978185854850317501460275816296862145418447280844274646388414104618,
8564746115019355453166208713664121566170513688834112727950038857249713718921291756815122024818193391798653452226837521288301347104408503812815093696565455971140851363726552596227501338928036581146133807719813913358706419049503859123205772834913280680673875610113990715806519852833617669284820587525172592303,
1184851695633196590859530896073053906108307787616752524429376482465869626210628976885827250056034383829298058194689386758516503610156407878349969493521298971808429815854303741756018364323387421074696755820552095597773436027425294236122929051268931281237544341614342480229020965356208060914322588066116834577,
4364906495590796738335955322543931098956945210026244667393966899719680694397416682350670916217739200553947579206761725867187591625622231548462432928821232527495734590690944852505592543951471197775480073107818603718660413692269427883077524264382099276042094197115647712590727443405374653920204523425809596964388537304133772715863925316641169075310249482820183896348626469192505828920208159671411483068230837953341786867798126172090049058711846929085459995896179765106731427761260844243991010221357209110209314936043522650394328902532384864309990120987592122633321068101325301231904078972595722775277294596713340213,
2393101447609058423972930395701691663427846813221239320934606678459413504768644926181460017799519382449397718313025961069893544988479646385919424405779326117178588032903929941862027150270119711598802163480952922120187094718686898939410748317800523138265797423665663221828132875519135258017417196723394991589,
2076619377477031848639856175879543470343734230329455509726069209675934720306012788659428906143908031714533138217029168088704862709255685909509412415865310492415529179862803114166904885039563512848807162958978688888862433736068791165778446148630009314317367780254227290580067976686185051291022979661750915754,
3491077294883860264163831470480731798100952316266751074114141651012550304283058960434530180849262526809044918444772385895477880956597558643610336430554267189540135235980259742427410947649578494040773591269499992232386779146605217149697181339334540183232972164203652901657790093101500810462049416615463177546,
1900282698989463723132580002213591610224600569598698293743548176893816262621986651465369623867470701906655323671861637049755140533957531464521427859781669543880016267139703210772613428562790216713087985891753909255158869760207735542707780506499364324239566426940858870670235658504146842251118347853718573687,
8458354961893766538861500625381362205299495123168378435526830196394782391388230634951309852043746924683383722572633012714625021774446565616147152141846134226564255477822614609368397663890423339219440456160454866429696167357148613518141531149628027621274166843630238217831476949356523887009620688092636433891692948566193147198415868174132947385960361457234775960996688163030768614452245060590761872803325778389788629611003585774682395160900493537852600395064146263629935684962163363737792411465123735249475890890275593935918443912139384104086401671117297441852839961645132964273067219203682142732326663844844075005,
3901970947731392475716050740298719400950276818247119306832129595796167756242376176561993909746155516755870152152426215315380852798393154894808746740712267603715980983656834079622454549206763317947354722296743984624987127611816894580092963610536272846303456628936054806658299294477036779787773094630293784171577043788886187395642609069164097976028680286900550453860267666151053881614552629192152797547831199391087342194362363394388982584111597031153132134424331287494091263926113657497887818333781008531372953107929672293402747435497312779887642688 71,

43780209984137897508037725402676779345811488082954135252161368 03,
32760021685634014692801879296116473051975434299763504271307380492817074383621845 1,
11601084321248387339572153239760506729881587154073429814304588424507086466781512 6,
97727505344041614504650113184536844491499087348235282469926847977818000551956814 3,
36780357683696104201023978840046283563476723525893301682484856 54,
44110091770169807254720224637866382906356589626584054303904462823579055173693541 5,
37495563178802878916197285515713652917722280082508079561649744554023623843084943 4,
70276234778431751534096038825419413377588366717841724884964887549021317906494580 4,
79422145479607908756382341056937075980464160614337226022774179 13,
72630721026082104937314499215440994834563763536887709281952301562292297375799079 8,
66595212293722092809092891558035487356684084133471032175155867351002372555792099 4,
74424155899316160700559374696225928061020418514015093221825626297114962505042408 1,
86489390308012549219425780628902188942072596008735217660547640 1,
36935475969000437411009317931245220477049995109245691213826606700340747649081004 2,
11564138994669919147427391961081342696945881651140353711290935709344645835585517 5,
75482952165037676968942036928516809490735784615759402020692737898084338352473678 3,
36487637427707512754541837623873805125314531332821611368914819 29,
36651674829521870961604062602847633625106468046871237481163927845433983982796411 7,
16040170299737181305183685906223978711781129563921325853542869950996396343467241 5,
90395759804225148387006632556514345517976666222491886606222022118774235602160797 3,
45840703357031759274755940168586896494223450267266885181927764 09,
31772258799399503956516189938050082224125321187882735861563680762668151872334027 8,
39690562139425656600083995693356499087510569246451596554122608223193461406715227 6,
03033869126748534268189896558045831187350504888076303422540220389412073639684634 8,
63498059928738185223162151088979542296704339337237279537455699 23,
37790289502092144474331611678666721772703179110400161535656843628684728979209541 2,
86675482988759725097132299804532128688667098063503828795488593044417138626170686 7,
33598301660233967996236275801646060853460855861987503117740066430943093837719536 2,
06898449768321179882968670043027139995318519059395055613911641 91,
15654042811146571563158378167586580262330331730608003170057708935679631943026588 0,
14846233896629923056723823980238188500685258648287756254300861721138345907619046 3,
77836662237471949485789772915630291193313597721269042994505087638266640554728438 4,
16140233748849399782675401667310119794586753266350197916955652 62,
45906127067329382579806847334183221548169885492923128788091242572364580540009976 5,
58048943357592273399620039882279698948590991641860234733438384748965217505854813 6,
83803099873452432495277175146503695789650677124486235097947510794466627402477875 2,
28021571469799335617375695814626464012102026111372777197763258 74,
46561340447668574505975602405581446767524109108114351677491590911616968948345860 8,
76157749897575136670929548564684077600175605421363167156706937777836795581191314 0,
10555372862855193624713101018024890731985811361264479048657199772684015457324674 3,
57908815842912556252289479567119680190160451262327789250030043 30,
13024420604642878441409586998968192743750200087293515972300408913624450674212124 6,
29574009587226427917681926570083636755871232666891476077591024729705531212773997 4,
29389124214264765629871725116000837042554302382841875519520361803078990918189532 0,
71319420206891970310408787985742982410424454893556236413045099 01,
17133718778702247036300493147629464747789204359552838347391262070787134646689040 2,
65946055133052713840386316550873428985880844513491878685886848018740908654040018 6,
67048992448656552094707927180010150375777638253730107851372827062661737290731671 7,
60236604110022994533170751134165386137820929693121143635706145 17,
36377268196891451192988593242226898293488286543792422845225141865343749071054499 5,
17895777044478557004846566936116496508492925076216311548758536571025268847253846 6,
82063460338157300752519732023789466556755422059551424138742968685452633858841258 7,
77825639796577006953344666623629494441887257745792303101520882 75,
21581022785302310874950344718223263585706766365627302922692879717804570354154393 9,
81546643936114009399345459178495582240303714218679596403932911895262567691273442 1,
31910719390080796034661361763333801101837585436533576943874702367687422327316225 6,

```
689287204347408482943739836470117231008853275004877026928410144 0,
135791751151726600275349499517453448386331146232149691244436591053297575744000903
910942666942546045071816501686971911343193308115626688999597144997359881599156481
592735050993842792310710155277975142273319489733804457885489094536821693091143746
211683555445690323246247222216134818195829384092824740655767497 4,
225101806029479820854640560914928907098838273115735702163237078656772741524773197
008316948382055390201664541112958856362759299698429428833779259252093687710897414
800878719431782856169827272874504834073109653199055847323351819739171038084660133
244658369152973071681295343197586716963771518511334922175226893 3,
148206895468982288684202593624544291881166097924864451086133951865886180761770851
178728192924309675104440143666708725936549742121370245291989408199455825925136404
485715395831863177089053356302066706977255391013269936562986700561263792218597098
711718036066480719983453626407395097830831402823594603045717845 0]
```

eazy_crt

```python
from Crypto.Util.number import *
from secret import flag, p, q
from hashlib import *
import random

assert 'flag{' + md5(str(p).encode()).hexdigest() + '}' == flag
assert isPrime(p), isPrime(q)
assert p.bit_length() == 1024, q.bit_length() == 1024

m = random.getrandbits(400)
e = 65537
d = inverse(e, (p - 1) * (q - 1))
r1, r2, r3 = getPrime(20), getPrime(20), getPrime(10)

dp, dq = d % (p - 1), d % (q - 1)
Sp, Sq = pow(m + getPrime(10), dp, r1 * p), pow(m, dq, r2 * q)
s1, s2 = pow(m, dp % (r1 - 1), r1), pow(m, dq % (r2 - 1), r2)
S_ = Sq + (q * r2) * (int(inverse(q * r2, p * r1)) * (Sp - Sq) % (p * r1))
c1, c2 = (S_ - s1 + 1) % r1, (S_ - s2 + 1) % r2
gamma = (r3 * c1 + (2 ** 10 - r3) * c2) // (2 ** 10)
S = pow(S_, gamma, p * q)

print(m)
print(p * q)
print(S_)
print(S)

'''
m=218024051213898288993573375877602528949284854207299990541190389830242749681433647543655223092032668180974577847058322698 7
```

```
n=25505131259827344749407187081729819350996141100990518281765117676936124636084125400315049858697199427401342785804654120926568235761577895862889807660442415521870277729420875825744007886870384790308986342360349597392841568418588521694478184632631896474390291958350681472768485356865513284619086754437723630874827593280089682939629265210875169009057935264259019861755270570945614034505771690412042781423771110441028258110022746603974882162934979726300741541857444013708508946471384525030286343828680432038605288717842755346907256658746733811881247992925881684393431852248253701825024590345480994598867741811599162649467
S_=55100865618422501389088753425332941083319516596126714666958013436869729194434021634015210404576406027567779100816391917534361221717561747305313859138659518268699959847871024396791706844227178087712602175414398786777505080657030640813754738454059166743279327981531005745559334485706187328423657957381204915323980814673120172039334132967790706110241249657727875022424990168845372330289478652880377180743524487737593632421110805406303609023885406618319927767076001332533297790037079380650201216455307191409545548009867717633431913982101003259715730698123816930893842214417352787368896735002182746731963338062222662488443791276523669911980057632167934442505201156347300527580178727186167189831852303341564238851204703565099104795331960134691219446212231336688812610009363596947669687140388368794661757583706169481366988378222100670170448793850088695234700363162632612715408178701669285662856120038694168375639773410069852046419924981123801314689935239045350013266684060658576030672389465493307709437581066616846483575660737799895967513230597172110966164423161342632267535097337343413868608602326591088350951457555442950221421746005952161962569375093811742783265479235580880332
'''
```

Door

```python
from secret import flag
import string
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
from os import urandom
import re

key = urandom(16)

menu = '''
[+] WatchDog Security System
[+] Copyright (c) 1010 by School of Natural Philosophy

please select your option:
1. Unlock Secret Entry
2. Help
3. Exit
'''

valid_code = [1033,3329,4431,5052]

auth_context_pattern = re.compile(r'^SoNP#[0-9]{4}$')

def auth_context_checker(ctx : bytes):
    for c in ctx:
```

```
        if chr(c) not in string.printable:
            return False
    if auth_context_pattern.match(ctx.decode()) : return True

    return False

def unlock():
    token = bytes.fromhex(input('Enter your token > '))
    auth_code = bytes.fromhex(input('Enter your authentication code > '))

    cipher = AES.new(key, AES.MODE_CBC,token)

    check = cipher.decrypt(auth_code)
    try:

        msg = unpad(check, 16)
        if auth_context_checker(msg) and int(msg[5:].decode()) in valid_code:
            print('door unlocked, here is your reward')
            print(flag)
        else:
            print('get out')

    except Exception as e:
        print('oops, something wrong')
        print(e)
def help():
    print('To unlock the door, please enter your token and authentication code.')
while True:
    print(menu)
    opt = input('> ')
    try:
        opt = int(opt)
        if opt == 1:
            unlock()
        elif opt == 2:
            help()
        elif opt == 3:
            break
        else:
            print('invalid option')
    except:
        print('oh no, something wrong!')
```

WEEK4 Crypto

RSA Variation II

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

p = getPrime(1024)
q = getPrime(1024)
```

```
N = p*p*q

d= inverse(N, (p-1)*(q-1)//GCD(p-1, q-1))

m = bytes_to_long(flag)

c = pow(m, N, N)

print('c =', c)
print('N =', N)
print('d =', d)

# c =
165339662711354953576051650366845511139236990540441984733618718005193935051440851
809536985241171855334015650524637203781103291908042688504254972312559874278377841
364222156361635838669969764581422585508945404598444309644716674088269322804350596
001133261674078597674315062411465359463177942704405572918539285496178632321514631
858816413942392540077268022686169999033242024644718063141752318119663118854032377
948785845371944480751563802577158627596957920180690979944881311203486708986651386
497141474237051624465325934726723143613185087134610631600795825674901659975859954
918090726009308050046939447314200314764317277007809271391220011004321443507827712
584411281626096749008603835866978800618283327235152679622853613563807167082920674
683534678499743704470795058008706766645922291604090203857415757788188002739142576
350369318426410493269398583398018298681666437701850748769776986653010392737592657
856994707663392387319310014775146 3
# N =
176842744715813185651403488945639742402793779661782975630352570531615231476912905
088889974266798653234661122915720777848706519451372200551661196975419748131033014
972105485568964613372160083819474112329041038431598033951694725717298100248041425
402325326909853996252783417478135665777998876175458234309633239176356092149141452
070711285289678297012301826350542644712619564537194111639565936915265436811856951
648225144251319289262622257641974704834394294757001604501612791757827281981276063
278834332174258335334015800932479462600673105726760380370166325670659790478904706
097842757336103517100882246712014822769889323877330532021576941059497436057372715
012203666698771893416662278542146464794608416289508424835264372180844437030725441
750185226457298590855083993386256300118647702131323611369079384389364019037813137
321410404446563348395361640268085377648071259966913257290709615166491611818548673
746325355909353731103651746174943 9
# d =
206506469331185442250955445523730074559285744801758016581681052270379501056422489
486457624888812195761741316245932934873253297039193131566597000022343924006364746
101430327451134738426758573237745669452291486649696597977791464884025889377623914
709716171634964330085018589075856834286526379588449029097968490807991419994902318
773788632440939003632514159728341460314909289239622710540532780563471812549367505
362806383212115451675209358702208297864906868260621424157550637246391105685119690
411750198980319904559115259410367270919610832011239107612909989682403382178952754
140724757019094975186161122363803898519843770 79
```

babyNTRU

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

```
q = getPrime(2048)

f = getPrime(1024)
g = getPrime(768)

h = (inverse(f, q) * g) % q

m = bytes_to_long(flag)

e = (getPrime(32) * h + m) % q

print((h, q))
print(e)

#
(89164527228214184632487268257212570217441942868747069158324446317715966161164917
75091473142798867278598586482678387668986764461265131119164500473719939894343163 4
96325556340181429675937641495981353857724627081847304246987074303722642172988864 1
38967404024201246050387152854001746763104417773214408906879366958729744259612772
57542351501592019483745621824894790096639205771421560295175631528776677200383961
54571697861326821483170835238092879747297506606983322890706220824261581533324824 8
58599082611886026668788577757970984892292609271082176311433507931993672945925883 9
85629311514143607457603297458439759594085898425992,
31985842636498685945330905726539498901443694955736332073639744466389039373143618 9
20511122288844282849407290205804991634167816417468703459229138891348115191921395 2
78336695684210437130681337971686008048054340499654721317721241239990701099685207 2
53476642931586563363638141636011941268962999641130263828151538489139254625099330 1
99557503153680089387538863574480134898211311252227463870838947777479309928195791 2
41005127445821671684607237706849308372923372795573732000365072815112119533702614 6
20325238183899266147682193892866330678076925199674555690181031642282787421517788
32319406135513140669049734660019551179692615505961)
#
20041713613876382007969284056698149007154248857420752520496829246324512197188211 0
29665990713599667984019715503486507126224558092176392282486689347953069815123212 7
79090783909545244160318938357529307482025697769394114967028564546355310883670462 1
97528011181768588878447856875173263800885048676190978206851268887445527785387532 1
67370943745180538168965461612097037041570912365648125449804109299630958840398397 7
21916860876687808474004391843869813396858468730877627733234832744328768443830669 4
69345926766882446378765847334421595034470639171397587395341977453536859946410431 2
52287203312913117023084978959318406160721042580688)
```

Smart

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

p = 75206427479775622966537995406541077245842499523456803092204668034148875719001
a = 40399280641537685263236367744605671534251002649301968428998107181223348036480
b = 34830673418515139976377184302022321848201537906033092355749226925568830384464

E = EllipticCurve(GF(p), [a, b])
```

```
d = bytes_to_long(flag)

G = E.random_element()

P = d * G

print(G)
print(P)

# (631992919767290175851167314221815736630763115132401584121088784602347640258 98
: 119779599288543097006112171029171865872421053431373839793646796069778242285 58 :
1)
# (750172753784385432462149542873623491769080421274391177343187007697685126244 29
: 395214832760097381154747142816268943611238048377831177256532438184982593519 84 :
1)
```

signin

```
from Crypto.Util.number import isPrime,bytes_to_long, sieve_base
from random import choice
from secret import flag

m=bytes_to_long(flag)
def uniPrime(bits):
    while True:
        n = 2
        while n.bit_length() < bits:
            n *= choice(sieve_base)
        if isPrime(n + 1):
            return n + 1


p=uniPrime(512)
q=uniPrime(512)
n=p*q
e= 196608
c=pow(m,e,n)

print("n=",n)
print("c=",c)

'''
n=
33267160053211754748663119153974012541119508087055762939323456905332631084148838775302943392942749148374245806183753465095556275787348833576529960058177663708048421616030276363937760791130357454955088397490067734837206980669435774459775512680932477483136913922653329709925004404229511738894193777791359525370887 33
c=
27093363160756501770793762447961881325612504597511521846770227455519145448845173248876523684506359956440192128785437454758859068642655591393799030492217651598529226414074083953836614741153324211691589279267273632187969495605158639959420629368575057363310735410978492122908806312440407384055702674705691051421824 6
```

```
'''
```

error

```
from sage.all import *
from secret import flag
import random
data = [ord(x) for x in flag]

mod = 0x42
n = 200
p = 5
q = 2**20

def E():
  return vector(ZZ, [1 - random.randint(0,p) for _ in range(n)])

def creatematrix():
  return matrix(ZZ, [[q//2 - random.randint(0,q) for _ in range(n)] for _ in
range(mod)])

A, B, C= creatematrix(), creatematrix(), creatematrix()
x = vector(ZZ, data[0:mod])
y = vector(ZZ, data[mod:2*mod])
z = vector(ZZ, data[2*mod:3*mod])
e = E()
b = x*B+y*A+z*C + e
res = ""
res += "A=" + str(A) +'\n'
res += "B=" + str(B) +'\n'
res += "C=" + str(C) +'\n'
res += "b=" + str(b) +'\n'

with open("enc.out","w") as f:
  f.write(res)
```

WEEK5 Crypto

last_signin

```
from Crypto.Util.number import *
flag = b'?'

e = 65537
p, q = getPrime(1024), getPrime(1024)
N = p * q
gift = p&(2**923-2**101)
m = bytes_to_long(flag)
c = pow(m, e, N)
```

```
print("N = ",N)
print("gift = ",gift)
print("c = ",c)

"""
N =
120559684715230533948513940380070911228093673924676912136515209440388617960110639
654604562850880117548952604288143585995920328652360067338798434931644119070322920
515397545205743952522989973790202688689721602978938712617132631960923804168766974
721601049800155548347989491559172921892788889140038467586872155599585061163593947
431352119505750602018870250326948250841047920592715843518891348115430884049529771
378096738806029469747985975067219067518350198550634624606860365675788354772499090
616758451574436799477305858803921104823017508278022138776436496590699451873539877
137171457091887904275725826893396436286595150177 49
p0 =
705611679085645433556303476203333501226071897723532788606747864066635645565571776
609541350107481893021042881559392692045594211985952622770646014837703310172827013
543821904726615834447749202973678899593125170096827406316739408405976512199561420
535753288113507709198527253383741 44
c =
247559234968979055141895126346799450343095930331773426633338258660820877583769643
613983044394289090033387320603184414678218471238195275371884810966318824510122653
804310179088128527092779507589368061558605368007745590133486108534997222268032206
795281136536628202675673718526310562169514605069538562665663830957708793345756650
157930895473954332136774146353241379071241987973321701782109991686649092847637277
254225492945921825930160841381196976300150424571763723119884819634865687861178884
338011549374412552008093006831847960646462389624028938160171190875945067251922886
448715310314121856755108314717138592069332587601 8
"""
```

School of CRC32

```
import secrets
from secret import flag
import zlib

ROUND = 100

LENGTH = 20

print('Extreme hard CRC32 challenge')
print('ARE YOU READY')

for i in range(ROUND):
    print('ROUND', i, '!'*int(i/75 + 1))

    target = secrets.randbits(32)

    print('Here is my CRC32 value: ', hex(target))

    dat = input('Show me some data > ')
    raw = bytes.fromhex(dat)
```

```
    if zlib.crc32(raw) == target and len(raw) == LENGTH:
        print("GREAT")
    else:
        print("OH NO")
        exit()

print("Congratulation! Here is your flag")
print(flag)
```

PseudoHell_EASY

challenge:

```
import os
def xor(a:bytes,b:bytes):
    assert len(a) == len(b)
    return bytes([i ^ j for i,j in zip(a,b)])

class PRP():
    def __init__(self, n):
        self.domain_cache = {}
        self.range_cache = {}
        self.n = n

    def tran(self, m, inverse = False):
        if not inverse:
            x = m
            if x in self.domain_cache:
                return self.domain_cache[x]
            while True:
                y = os.urandom(self.n)
                if y in self.range_cache:continue
                self.domain_cache[x] = y
                self.range_cache[y] = x
                return y
        else:
            y = m
            if y in self.range_cache:
                return self.range_cache[y]
            while True:
                x = os.urandom(self.n)
                if x in self.domain_cache:continue
                self.domain_cache[x] = y
                self.range_cache[y] = x
                return x

class PRF():
    def __init__(self, n):
        self.domain_cache = {}
        self.range_cache = {}
        self.n = n
```

```
    def tran(self, x):
        if x not in self.domain_cache:
            self.domain_cache[x] = os.urandom(self.n)
        return self.domain_cache[x]

class Challenge1:
    def __init__(self):
        self.coin = os.urandom(1)[0] & 1
        self.block_size = 8
        self.input_size = 2 * self.block_size
        self.RO1 = PRF(self.block_size)
        self.RO2 = PRF(self.input_size)

    def function1(self, M):
        L, R = M[:self.block_size], M[self.block_size:]
        L, R = R, xor(L, self.RO1.tran(R))
        return L+R

    def function2(self, M):
        return self.RO2.tran(M)

    def roll(self, M, inverse):
        assert inverse == False, "In Challenge1, inverse is not allowed"
        return [self.function1,self.function2][self.coin](M)

class Challenge2:
    def __init__(self):
        self.coin = os.urandom(1)[0] & 1
        self.block_size = 8
        self.input_size = 2 * self.block_size
        self.RO1 = PRF(self.block_size)
        self.RO2 = PRF(self.input_size)

    def function1(self, M):
        L, R = M[:self.block_size], M[self.block_size:]
        L, R = R, xor(L, self.RO1.tran(R))
        L, R = R, xor(L, self.RO1.tran(R))
        return L+R

    def function2(self, M):
        return self.RO2.tran(M)

    def roll(self, M, inverse):
        assert inverse == False, "In Challenge2, inverse is not allowed"
        return [self.function1,self.function2][self.coin](M)

class Challenge3:
    def __init__(self):
        self.coin = os.urandom(1)[0] & 1
        self.block_size = 8
        self.input_size = 2 * self.block_size
        self.RO1 = PRF(self.block_size)
        self.RO2 = PRF(self.input_size)

    def function1(self, M, inverse):
        if not inverse:
```

```
            L, R = M[:self.block_size], M[self.block_size:]
            L, R = R, xor(L, self.RO1.tran(R))
            L, R = R, xor(L, self.RO1.tran(R))
            L, R = R, xor(L, self.RO1.tran(R))
        else:
            L, R = M[:self.block_size], M[self.block_size:]
            L, R = xor(R, self.RO1.tran(L)), L
            L, R = xor(R, self.RO1.tran(L)), L
            L, R = xor(R, self.RO1.tran(L)), L
        return L+R

    def function2(self, M, inverse):
        if inverse or not inverse:
            return self.RO2.tran(M)

    def roll(self, M, inverse):
        return [self.function1,self.function2][self.coin](M,inverse)

class Challenge4:
    def __init__(self):
        self.coin = os.urandom(1)[0] & 1
        self.block_size = 8
        self.input_size = 2 * self.block_size
        self.PRP1 = PRP(self.block_size)
        self.PRP2 = PRP(self.input_size)

    def function1(self, M, inverse):
        X, T = M[:self.block_size], M[self.block_size:]
        X = xor(X, T)
        for _ in range(2):
            X = xor(self.PRP1.tran(X, inverse),T)
        return X

    def function2(self, M, inverse):
        return self.PRP2.tran(M, inverse)[:self.block_size]

    def roll(self, M, inverse):
        return [self.function1,self.function2][self.coin](M,inverse)

class Challenge5:
    def __init__(self):
        self.coin = os.urandom(1)[0] & 1
        self.block_size = 1
        self.input_size = self.block_size
        self.PRP = PRP(self.block_size)
        self.PRF = PRF(self.input_size)

    def function1(self, M):
        return xor(bytes(1),self.PRP.tran(M))

    def function2(self, M):
        return xor(self.PRF.tran(M),bytes(1))

    def roll(self, M, inverse):
        assert inverse == False, "In Challenge 5, inverse is not allowed"
        return [self.function1,self.function2][self.coin](M)
```

problems:

```
from challenge import Challenge1,Challenge2,Challenge3,Challenge4,Challenge5
from secret import flag_easy,flag_hard
roll_left = 56

def guess_coin(level, query_num):
    Ch = level()
    for _ in range(query_num):
        msg = bytes.fromhex(input("msg? > "))
        inverse = input("inverse? > ") == 'y'
        assert len(msg) == Ch.input_size
        print(Ch.roll(msg, inverse).hex())
    assert input("coin? > ") == str(Ch.coin)

def roll_challenge(challenge_level, challenge):
    global roll_left
    print(f"[+] Challenge Level: {challenge_level}")
    roll_num = int(input(f"How many times are required to roll for solving
{challenge_level}? > "))
    roll_left -= roll_num
    [guess_coin(challenge, roll_num) for _ in range(33)]

roll_challenge("1", Challenge1)
roll_challenge("2", Challenge2)
roll_challenge("3", Challenge3)

if roll_left <= 0:
    print("You passed all challenges for EASY but times limit exceeded. Try
harder :(")
    exit(-1)

print("flag for easy is ", flag_easy)

roll_challenge("4", Challenge4)
roll_challenge("5", Challenge5)

if roll_left <= 0:
    print("You passed all challenges for HARD but times limit exceeded. Try
harder :(")
    exit(-1)

print("flag for hard is ", flag_hard)
```

PseudoHell_HARD

challenge:

```
import os
def xor(a:bytes,b:bytes):
    assert len(a) == len(b)
```

```
    return bytes([i ^ j for i,j in zip(a,b)])

class PRP():
    def __init__(self, n):
        self.domain_cache = {}
        self.range_cache = {}
        self.n = n

    def tran(self, m, inverse = False):
        if not inverse:
            x = m
            if x in self.domain_cache:
                return self.domain_cache[x]
            while True:
                y = os.urandom(self.n)
                if y in self.range_cache:continue
                self.domain_cache[x] = y
                self.range_cache[y] = x
                return y
        else:
            y = m
            if y in self.range_cache:
                return self.range_cache[y]
            while True:
                x = os.urandom(self.n)
                if x in self.domain_cache:continue
                self.domain_cache[x] = y
                self.range_cache[y] = x
                return x

class PRF():
    def __init__(self, n):
        self.domain_cache = {}
        self.range_cache = {}
        self.n = n

    def tran(self, x):
        if x not in self.domain_cache:
            self.domain_cache[x] = os.urandom(self.n)
        return self.domain_cache[x]

class Challenge1:
    def __init__(self):
        self.coin = os.urandom(1)[0] & 1
        self.block_size = 8
        self.input_size = 2 * self.block_size
        self.RO1 = PRF(self.block_size)
        self.RO2 = PRF(self.input_size)

    def function1(self, M):
        L, R = M[:self.block_size], M[self.block_size:]
        L, R = R, xor(L, self.RO1.tran(R))
        return L+R

    def function2(self, M):
        return self.RO2.tran(M)
```

```
    def roll(self, M, inverse):
        assert inverse == False, "In Challenge1, inverse is not allowed"
        return [self.function1,self.function2][self.coin](M)

class Challenge2:
    def __init__(self):
        self.coin = os.urandom(1)[0] & 1
        self.block_size = 8
        self.input_size = 2 * self.block_size
        self.RO1 = PRF(self.block_size)
        self.RO2 = PRF(self.input_size)

    def function1(self, M):
        L, R = M[:self.block_size], M[self.block_size:]
        L, R = R, xor(L, self.RO1.tran(R))
        L, R = R, xor(L, self.RO1.tran(R))
        return L+R

    def function2(self, M):
        return self.RO2.tran(M)

    def roll(self, M, inverse):
        assert inverse == False, "In Challenge2, inverse is not allowed"
        return [self.function1,self.function2][self.coin](M)

class Challenge3:
    def __init__(self):
        self.coin = os.urandom(1)[0] & 1
        self.block_size = 8
        self.input_size = 2 * self.block_size
        self.RO1 = PRF(self.block_size)
        self.RO2 = PRF(self.input_size)

    def function1(self, M, inverse):
        if not inverse:
            L, R = M[:self.block_size], M[self.block_size:]
            L, R = R, xor(L, self.RO1.tran(R))
            L, R = R, xor(L, self.RO1.tran(R))
            L, R = R, xor(L, self.RO1.tran(R))
        else:
            L, R = M[:self.block_size], M[self.block_size:]
            L, R = xor(R, self.RO1.tran(L)), L
            L, R = xor(R, self.RO1.tran(L)), L
            L, R = xor(R, self.RO1.tran(L)), L
        return L+R

    def function2(self, M, inverse):
        if inverse or not inverse:
            return self.RO2.tran(M)

    def roll(self, M, inverse):
        return [self.function1,self.function2][self.coin](M,inverse)

class Challenge4:
    def __init__(self):
```

```
        self.coin = os.urandom(1)[0] & 1
        self.block_size = 8
        self.input_size = 2 * self.block_size
        self.PRP1 = PRP(self.block_size)
        self.PRP2 = PRP(self.input_size)

    def function1(self, M, inverse):
        X, T = M[:self.block_size], M[self.block_size:]
        X = xor(X, T)
        for _ in range(2):
            X = xor(self.PRP1.tran(X, inverse),T)
        return X

    def function2(self, M, inverse):
        return self.PRP2.tran(M, inverse)[:self.block_size]

    def roll(self, M, inverse):
        return [self.function1,self.function2][self.coin](M,inverse)

class Challenge5:
    def __init__(self):
        self.coin = os.urandom(1)[0] & 1
        self.block_size = 1
        self.input_size = self.block_size
        self.PRP = PRP(self.block_size)
        self.PRF = PRF(self.input_size)

    def function1(self, M):
        return xor(bytes(1),self.PRP.tran(M))

    def function2(self, M):
        return xor(self.PRF.tran(M),bytes(1))

    def roll(self, M, inverse):
        assert inverse == False, "In Challenge 5, inverse is not allowed"
        return [self.function1,self.function2][self.coin](M)
```

problems:

```
from challenge import Challenge1,Challenge2,Challenge3,Challenge4,Challenge5
from secret import flag_easy,flag_hard
roll_left = 56

def guess_coin(level, query_num):
    Ch = level()
    for _ in range(query_num):
        msg = bytes.fromhex(input("msg? > "))
        inverse = input("inverse? > ") == 'y'
        assert len(msg) == Ch.input_size
        print(Ch.roll(msg, inverse).hex())
    assert input("coin? > ") == str(Ch.coin)

def roll_challenge(challenge_level, challenge):
    global roll_left
    print(f"[+] Challenge Level: {challenge_level}")
```

```
    roll_num = int(input(f"How many times are required to roll for solving
{challenge_level}? > "))
    roll_left -= roll_num
    [guess_coin(challenge, roll_num) for _ in range(33)]

roll_challenge("1", Challenge1)
roll_challenge("2", Challenge2)
roll_challenge("3", Challenge3)

if roll_left <= 0:
    print("You passed all challenges for EASY but times limit exceeded. Try
harder :(")
    exit(-1)

print("flag for easy is ", flag_easy)

roll_challenge("4", Challenge4)
roll_challenge("5", Challenge5)

if roll_left <= 0:
    print("You passed all challenges for HARD but times limit exceeded. Try
harder :(")
    exit(-1)

print("flag for hard is ", flag_hard)
```