中间的那个人

考点:

- DH算法
- 简单的爆破

DH算法具体细节和作用，望新师傅们利用搜索引擎了解一下，这里不细谈。这道题比较简单，有些时候面对未知密钥的时候，需要考虑爆破求解，那么这时候就要进行一定的计算量估计，这题不管是求A还是B都是可以出的来的，两者的位数比32位低，也没啥限制，还是可以求的。

exp:

```python
from Crypto.Util.number import *
from Crypto.Cipher import AES
from hashlib import sha256
g=2
p=250858685680234165065801734515633434653
Bob=33067794433420687511728239091450927373
Alice=235866450680721760403251513646370485539

x = 3992780394
key = pow(Bob,x,p)
key = sha256(long_to_bytes(key)).digest()
iv = b"0xGame0xGameGAME"
aes = AES.new(key, AES.MODE_CBC, iv)
enc=b's\x04\xbc\x8bT6\x846\xd9\xd6\x83 y\xaah\xde@\xc9\x17\xdc\x04v\x18\xef\xcf\xef\xc5\xfd|\x0e\xca\n\xbd#\x94{\x8e[.\xe8\xe1GU\xfa?\xda\x11w'

flag = aes.decrypt(enc)
print(flag)
#b'0xGame{51393fe1fd5fc2df1bf018d06f0fa11d}\x08\x08\x08\x08\x08\x08\x08\x08'
```

What's CRT?

考点:

- 中国剩余定理
- 对逆元的理解运用
- 解方程

小彩蛋：公钥260792700是一个QQ号码，因为出题人的Q号拿来做公钥不怎么合适就用小号来玩了。

这题可能是我弄得有点复杂了，因为考虑到第一周逆向题里面大家已经做过解方程的题了，就不直接放出q,p了（本意就是想直接送出q,p,q_,p_的）。

思路：首先解方程

$$q + p = gift$$
$$q * p = N$$

用高中知识还是直接用函数库解都可以，我选择用sagemath自带的函数：

```
mygift=[15925416640901708561793293991573474917595642805739825596593339102414328214313430010166125066639132916608736569443045051644173933089503934675628814467277922,18342424676996843423829480445042578097182127446865571536445030052846412665700132683433441858073625594933132038175200824257774638419166516796318527302903098]
n=63329068473206068067147844002844348796575899624395867391964805451897110448983910133293450006821779608031734813916287079551030950968978400757306879502402868643716591624454744334316879241573399993026873598478532467624301968439714860262226444947188860653891307141363434638142890135810927320308703076377909166479
n_=84078907800136966150486965612788894868587998005459927216462899940718213455112139441858657865215211843183780436155474431592540465189966648565764225210091190218976417210291521208716206733270743675534820816685370480170120230334766919110311980614082807421812749491464201740954627794429460268010183163151688591417
var('q p q_ p_')
solve([q+p==gift[0],q*p==n,q_+p_==gift[1],q_*p_==n_],q,p,q_,p_)
```

拿到因子之后考虑直接求私钥解题，但是可以发现求不出来，原因是：

$$gcd(e, phi) == 4$$

在上周的题中，我们已经知道只有在e,phi互质的时候才能求出逆元，所以考虑退而求次，求解另外一个逆元：

$$\begin{gathered}
令 e' = \frac{e}{4}, 求 d' \equiv e'^{-1}(mod \phi(n)) \\
得 c^{d'} \equiv m^{e*d'} \equiv m^{4*\frac{e}{4}*d'} \equiv m^4(mod n) \\
但是发现：m^4 > n，m^4 = c^{d'} + k*n \\
我们并不能直接开根
\end{gathered}$$

此时就需要利用中国剩余定理去将模数进行变换，用来求解以下情况：

$$\begin{cases}
x = a_1(mod m_1) \\
x = a_2(mod m_2) \\
x = a_3(mod m_3) \\
x = a_4(mod m_4)
\end{cases}$$

证明：

> 假设整数 $m_1, m_2, \ldots, m_n$ 其中**任两数互质**，则对任意的整数：$a_1, a_2, \ldots, a_n$，方程组 (S) 有解，并且通解可以用如下方式构造得到：
>
> 1. 设 $M = m_1 \times m_2 \times \cdots \times m_n = \prod_{i=1}^{n} m_i$ 是整数 $m_1, m_2, \ldots, m_n$ 的乘积，并设 $M_i = \frac{M}{m_i}, \forall i \in \{1, 2, \ldots, n\}$，即 M_i 是除了 m_i 以为的 $n-1$ 个整数的乘积。
> 2. 设 $t_i = M_i^{-1}$ 为 M_i 模 m_i 的数论倒数：$t_i M_i \equiv 1 \pmod{m_i}, \forall i \in \{1, 2, \ldots, n\}$。
> 3. 方程组 (S) 的通解形式为：
>
> $$x = a_1 t_1 M_1 + a_2 t_2 M_2 + \cdots + a_n t_n M_n + kM = kM + \sum_{i=1}^{n} a_i t_i M_i, \quad k \in \mathbb{Z}$$
>
> 在模 M 的意义下，方程组 (S) 只有一个解：$x = \sum_{i=1}^{n} a_i t_i M_i$。

那么知道公式的情况下，直接按照公式去构造就可以了，需要注意的是不同模数之间必须**互质**。

exp:

```python
import gmpy2
from Crypto.Util.number import *
p_=899169086989724632190750998342530743736528841786145773272131457216577388089870110506581828124837367675840502115770319013251121938470465008656534588572777 7
q_=935073380709959710192197046161727065981683902900411380372371548068063878480143157836762357682525191817472701701749763412526341903446186670975318141717 5321
q = 76876531925742836898424657632996115920079098138011768435771893414094096929757530374022534966324103645946556113371563376690835824004430423484582681613310 43
p = 82377634483274248719508282282738633255877329919386487530161497610049185213376769727638715700067225520140809581058887139750903506890608923271705463059468 79
e = 260792700
mygift=[15925416640901708561793293991573474917595642805739825596593339102414328214313430010166125066639132916608736569443045051644173933089503934675628814467277922,18342424676996843423829480445042578097182127446865571536445030052846412665700132683433441858073625594933132038175200824257774638419166516796318527302903098]
mq_=62296150987887226643923691467122911699484859513711330861540288328057505516550729461703323354581864795652633719855346010355592294033573965645686672188171 97
mp_=75145984493611914867994802250879389139450617158451280060692968764578145283473713154936440460293768301669836455700921003205661962272105028970682060730437 18
n=63329068473206068067147844002844348796575899624395867391964805451897110448983910133293450006821779608031734813916287079551030950968978400757306879502402868643716591624454744334316879241573399993026873598478532467624301968439714860262264449471888606538913071413634346381428901358109273203087030763779091664797
n_=84078907800136966150486965612788894868587998005459927216462899940718213455112139441858657865215211843183780436155474431592540465189966648565764225210091190218976417210291521208716206733270743675534820816685370480170120230334766919110311980614082807421812749491464201740954627794429460268010183163151688591417
c=12623780002384219022772693100787925315981488689172490837413686188416255911213044332780064192900824150269364486747430892667624289724721692959334462348218416297309304391635919115701692314532111050955120844126517392040880404049818026059951326039894605004852370344012563287210613795011783419126458214779488303552
def CRT(r,d):
    M = 1
    l = len(r)
    for i in range(0,l):
        M = d[i] * M
    x = 0
    for i in range(0,l):
        md = M//d[i]
        x = (x + gmpy2.invert(md, d[i])  * md *r[i] )%M
    return int(M+x% M)%M

phi = (q-1)*(p-1)
d = gmpy2.invert(e//4,phi)
m2 = pow(c,d,n)
mq = m2%q
mp = m2%p

m = CRT([mq,mp,mq_,mp_],[q,p,q_,p_])
m = (gmpy2.iroot(m,4))[0]
print(long_to_bytes(m))
#b'0xGame{7881ed67088e9f72b860f8c376599785}'
```

题后总结，基本解法需要使用到的函数、数学知识在上周都了解过了，剩下的就是一个中国剩余定理的考点，望周知。

以上是预期的解法，下面放出一些非预期解，用SageMath自带的有限域开方的函数nthroot_mod()，也是可以的，还有师傅说能直接phi//4后直接求逆元进行求解也是可以，总之解法蛮多。

EzRSA

考点:

- 费马小定理
- 光滑数分解
- 连分数分解

hint已经给出所有考点，题目的交互脚本也给了，本意就是上点送分题的，，

因为交互也不算太复杂，直接连上去之后把数据拖到本地解就可以了，详细的概念和知识点这里就不拓展了，在这里，都有详细的介绍。

exp:

```
#数据从服务器上面拖下来解就好，这里就只做分解，不做解密了

#challenge1
gift =
N =
q = GCD(gift-1,N)
p = N//q
print(f'p={p}\nq={q}')

#challenge2
N =
a = 2
n = 2
while True:
    a = gmpy2.powmod(a, n, N)
    res = gmpy2.gcd(a-1, N)
    if res != 1 and res != N:
        q = N // res
        print("p=",res)
        print("q=",q)
        break
    n += 1

#challenge3
def transform(x,y):     #使用辗转相除将分数x/y转为连分数的形式
    res=[]
    while y:
        res.append(x//y)
        x,y=y,x%y
    return res
def continued_fraction(sub_res):
    numerator,denominator=1,0
    for i in sub_res[::-1]:    #从sublist的后面往前循环
        denominator,numerator=numerator,i*numerator+denominator
```

```python
    return denominator,numerator   #得到渐进分数的分母和分子，并返回
#求解每个渐进分数
def sub_fraction(x,y):
    res=transform(x,y)
    res=list(map(continued_fraction,(res[0:i] for i in range(1,len(res)))))  #将连分数的结果逐一截取以求渐进分数
    return res

def wienerAttack(n1,n2):
    for (q2,q1) in sub_fraction(n1,n2):  #用一个for循环来注意试探n1/n2的连续函数的渐进分数，直到找到一个满足条件的渐进分数
        if q1==0:
            continue
        if n1%q1==0 and q1!=1:
            return (q1,q2)
    print("该方法不适用")

N1=
N2=
print(wienerAttack(N1,N2))
```

EzLFSR

考点:

- 矩阵的理解应用
- 线性反馈移位寄存器
- SageMath的应用(直接解方程不用也行，这里还是推荐使用)

这题是出题人从网上摁抄的，可能直接搜索都能搜得出原题，重点是想让大家了解**异或**和**按位**与这两个操作其实就是mod2下的加法和乘法，以及矩阵的构造(新师傅都是大一的话，这个时间点应该也学到了矩阵怎么构了)。

从网上搜索就能知道LFSR的概念，这个就是反复的：按照状态位计算，生成新的状态位，直接解了这个128元方程组就好，结合矩阵的知识，用逆矩阵去求解题设的增广矩阵就好了。

exp:

```python
#SageMath
from Crypto.Util.number import *
def string2bits(s):
    return [int(b) for b in s]

initState = [0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0]
outputState = string2bits('1101111111011101100001000011111101001000111000110100010011110111010011100110100100111001101010110110101110000011110101000110010010000011111111001111000110111001100111101110010100100001101001111111000101000010011110101101110001000000010000010000010011101011 0')
states = initState + outputState
```

```

ms = MatrixSpace(GF(2), 128, 128)
mv = []
for i in range(128):
    mv += states[i : i + 128]
m= ms(mv)

vs = MatrixSpace(GF(2), 128, 1)
vv = outputState[0:128]
v = vs(vv)

secret = m.inverse() * v
M=secret.str().replace('\n','').replace('[','').replace(']','')
print(long_to_bytes(eval('0b'+M)))
```

Fault!Fault!

考点:

- 数学推导、数据处理
- 远程交互脚本的编写

关于推导的过程和原理直接看这篇文章就好，如果没推过的话直接百度“RSA Fault”都能出的来这篇文章，所以我觉得理解这部分应该不会太复杂。

脚本编写部分：首先因为服务器有时间限制、交互要验证等一系列的原因，一般不推荐写好脚本了再去和服务器交互、调试脚本，容易浪费时间、搞自己心态，我的建议是在自己本地上部署一遍题目，或者是模拟题目的条件再调试数据，看看符不符合自己的预期、设定。

其次就是在这种时间限定要拿很多数据的情况下，就不推荐一边拖数据一边做数据处理了，影响速度，建议是写好数据处理的脚本之后，直接从服务器拖下我们需要的1024组数据，本地运算一遍交上去就行，方便故障排查、也不会浪费很多时间。

exp:

```
#part1 拖数据:
from pwn import *
import itertools
import hashlib
import string

def proof(io):
    io.recvuntil(b"XXXX+")
    suffix = io.recv(16).decode("utf8")
    io.recvuntil(b"== ")
    cipher = io.recvline().strip().decode("utf8")
    for i in itertools.product(string.ascii_letters+string.digits, repeat=4):
        x = f"{i[0]}{i[1]}{i[2]}{i[3]}"
        proof=hashlib.sha256((x+suffix).encode()).hexdigest()
        if proof == cipher: break
    print(x)
    io.sendlineafter(b"XXXX:",x.encode())

```

```python
f = open('data.txt','a')
io = remote('43.139.107.237',10005)
proof(io)
io.recvuntil(b'>')
io.sendline(b'S')
io.recvuntil(b'>')
io.sendline(b'test')
n = int(io.recvline())
e = int(io.recvline())
c = int(io.recvline())
for i in range(1023):
    io.recvuntil(b'>')
    io.sendline(b'F')
    io.recvuntil(b'>')
    io.sendline(f'{c}'.encode())
    io.recvuntil(b'>')
    io.sendline(f'{i}'.encode())
    io.recvline()
    m_ = int(io.recvline())
    f.write(str(m_)+'\n')

io.close()
f.close()
print(f'n={n}')
print(f'c={c}')

#part2 处理数据:
from Crypto.Util.number import *

m = b'test'
m = bytes_to_long(m)
n=97914749446436063122542581376873112820400732267124998400088179058780712855378248201542023213009277089224170180542304344638059090556781844777641757174279080863658472878763702075705376304717343862101956239090701126225622317784075619757963099253602226642056966461019740454740445226152574361794251236011891077789
c=73133825445675329950286077126832004352164006658709453405485979363609175208129785294437379266100324978770868694885347204515053234232666436453863941132493106687387106354265743735994029551983269772204386433432638435914078485461320417024614354676213557287698752845797412775400104995650602775848941301838035593870

e = 65537

dbin = ''
with open('data.txt','r') as f:
    for i in range(1023):#私钥位数不同可能要判断的数量不同
        m_ = int(f.readline())
        h = (inverse(m,n)*m_)%n
        test = pow(c,2**i,n)
        if h == test:
            dbin += '0'
        elif h == inverse(test,n):
            dbin += '1'

d = eval('0b'+dbin[::-1])#校验
if (pow(c,d,n)==m):
    print(d)
#最后把私钥交上去就拿到flag了
```

小思考：虽然时间是有一定的限制，但这道题的私钥是固定的，所以我们可以多次交互拿数据推导出这个私钥。

但是如果这个私钥不是固定的呢？我们只有一次交互的机会，多次交互拿数据推导出这个私钥这个思路恐怕不太行，不管怎么优化可能时间都是不太够的，那么我们要如何处理这种情况？哈哈，我们第四周再见。