ctf10

Question1:

```
 the first 20 numbers in f(pow(10,6)):
 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71
```

try to find the law :

flag defines the number of times the move function execute, the result are as following:

each prime start with a segement, which was generated by this prime multiply before items.

```
2
3 6
5 10 15 30
7 14 21 42 35 70 105 210
11 22 33 66 55 110 165 330 77 154 231 462 385 770 1155 2310
13 26 39 78 65 130 195......
```

we now know the final enc and need to find the position of it in the above list, which indicates the flag.

If the position of a prime is f(x), then f(x)=2*f(x-1)+1 and f(1)=1, so f(x)=2^x-1.

If the number is a composition, then we find the biggest prime smaller then it, it's position was the position of this prime + #remaining items, we find the later recursively.

```
from Crypto.Util.number import long_to_bytes

def f(n):
    q = [True] * (n + 1)
    r = 2
    while r ** 2 <= n:
        if q[r]:
            for i in range(r ** 2, n + 1, r): q[i] = False
        r += 1
    return [p for p in range(2, n + 1) if q[p]]

list=f(1320)

def isPrimes1(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

def find_max_prime(n):
    max=-1
    for i in list:
        if (n%i  ==0):
```

```
            max = i
    return max

def find_position(n):
    if isPrimes1(n)==True:
        index=list.index(n)
        return pow(2,index)
    else:
        max=find_max_prime(n)
        return find_position(n//max)+find_position(max)



if __name__ == '__main__':
    print(list)

a=find_position(311013481418120783358338056057892860742612821878119302285431507313915961977533984577116683231587663543409733366279100721704647040904305965441293568122123756293616331005447102835383096956236545125781223360729147965772360816674239700142672465531108006672678536169705298127382031255161692055319529739782053310)
    print("%d" % a)
    print(long_to_bytes(a))
```

The result is :

421345269367111027067881489430561143269563636946142459110101569256
b'flag{functi0n_h4cking_ftw!}'

```
position:
421345269367111027067881489430561143269563636946142459110101569256
flag:
flag{functi0n_h4cking_ftw!}
```

Question 2:

we can figure out the bit of N one by one, for example, if we want to know the last bit of N, we can make K=2 and l=1, if last bit is 1, then f(n+l,k)=2, else f(n+l,k)=1.In this way, we can figure out the N.

Script is as following:

```
from pwn import *

if __name__ == '__main__':
    string = ''
    c = remote('ali.infury.org', 10007)
    print(c.recv().decode('utf-8'), end=' ')
    k = 2
    l = 1
    for _ in range(600):
        print(c.recv().decode('utf-8'), end=' ')
        c.sendline(k.__str__().encode('utf-8'))
        print(c.recv().decode('utf-8'), end=' ')
        c.sendline(l.__str__().encode('utf-8'))
        res=c.recv().decode('utf-8')
```

```
        print(res)
        match= re.match('f\(N\+l,k\) = ([0-9]+)',res)
        remain =(int)(match.group(1))
        if _ <512:
            if remain % k ==0:
                string ='1'+string
            else:
                string='0'+string
                l+=k//2
            k*=2
        print(string)

    print(c.recv().decode('utf-8'), end=' ')
    number = int(string, 2)
    c.sendline(number.__str__().encode('utf-8'))
    print(c.recv().decode('utf-8'), end=' ')
```

Flag:

```
flag{cfd9117e51f68cebde76afa6cd61e0b7f838a286b6f96e287b83367c304faec2}
```

Question 3:

We should find a faster way to find the private key.

First we should analyse the code:

```
from Crypto.Util.number import getPrime, bytes_to_long
from math import gcd

p = getPrime(20)
print(p)
#产生杨辉三角 p+1
while len(triangle[-1]) <= p:
    r = [1]
    for i in range(len(triangle[-1]) - 1):
        r.append(triangle[-1][i] + triangle[-1][i + 1])
    r.append(1)
    triangle.append(r)
    print(triangle)
```

it will generate yanghui triangle with the last level's length = p+1

```
7  p
[[1], [1, 1]]
[[1], [1, 1], [1, 2, 1]]
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1]]
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10, 10, 5, 1]]
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10, 10, 5, 1], [1, 6, 15, 20, 15, 6, 1]]
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1], [1, 5, 10, 10, 5, 1], [1, 6, 15, 20, 15, 6, 1], [1, 7, 21, 35, 35, 21, 7, 1]]
```

The second part of code:

```
code = ''
for x in triangle[-1]:
    code += str(x % 2)
d = int(code, 2)
print(code,' ',d)
```

It tell us the binary string of d is consist of '0' and '1' which indicate wether the corrosponding number is odd or even.

for example, if triangle[-1] is[1,7,21,35,21,7,1] then d = 11111111

So we only need to know the parity of the number in the last level, that is C(p,i).

By searching, I know that:

to determine if `C(n, p)` is odd, we just need to know whether there is at least one carry when adding p and n in base 2.

So the scrpit is as following:

```
if __name__ == '__main__':
    triangle = [[1]]
    enc =
98206202690728604016658051018812849614213024753824053738887467804674090825750096
33494008131637326951607592072546997831382261451919226781535697132306297667495663
00507269535143095363009975133502019209839772293781215177478623270755538647977446
05291339418486777465812567929605712864182913297802801284193587004 49
    N =
84317137476812805534382776304205215410373527909056058618583365618383741423290821
41027092957431789994586294982948008281108455400926543954030756853794024922738893
51546417798634413012923789758556253253752999802916296089950497422435919015471778
53086110999523167557589597375590016312480342995048934488540440868447
    from Crypto.Util.number import  long_to_bytes
    p = 751921
    code = ''
    for i in range(p + 1):
        if (i & (p - i)):
            code += "0"
        else:
            code += "1"
    d = int(code, 2)
    print(long_to_bytes(pow(enc,d,N)))
```

The flag is:

```
D:\python\python.exe G:/课程内容/大三上/N
b'flag{1ts_ch00se_a11_a10ng??}'
```

```
flag{1ts_ch00se_a11_a10ng??}
```