Crypto

从零开始的勇士之路

密码学每年都不变的签到题，亘古不变的仿射密码，刚开始用114514做参数，结果和26不互质，加密过程不是满射，题目锅了，在此给大家（特别是猫猫）表示歉意

修好了之后题目很简单，首先要获取affine的两个参数a和b，我们已知flag格式为 `NUAACTF{*}` ，所以直接爆破找a和b即可，如下：

```
import string

cipher = 'SNFFLKU{D_4p_ka3_aRe0_0u_Ka3_LezykV_tvEmo}'
flag = 'NUAACTF'

def check(a, b):
    table = string.ascii_uppercase
    for i in range(7):
        if (table.index(flag[i]) * a + b) % 26 != table.index(cipher[i]):
            return False
    return True

for a in range(26):
    for b in range(26):
        if check(a, b):
            print(a, b)
            break
```

即可得出 $a = 3$ ， $b = 5$ ，有了affine的参数直接对其解密即可

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

def dec(m, a, b):
    if m not in string.ascii_lowercase + string.ascii_uppercase:
        return m
    table = string.ascii_lowercase if m in string.ascii_lowercase else
string.ascii_uppercase
    return table[((table.index(m) - b) * inverse(a, 26)) % 26]

cipher = 'SNFFLKU{D_4p_ka3_aRe0_0u_Ka3_LezykV_tvEmo}'
```

```
    flag = ''
    for i in cipher:
        flag += dec(i, 3, 5)

    print(flag)
```

走一步，再走亿步

首先拿到题目

```
    from hashlib import md5
    from tqdm import tqdm

    aim = 1919810
    sequence = [0, 1]
    for i in tqdm(range(2, aim)):
        sequence.append(233 * 114514 ** i + 1919810 * sequence[i-1] + sequence[i-2])

    print(len(sequence))
    print(sequence[-1])
    flag = md5(str(sequence[-1]).encode()).hexdigest()
    flag = 'NUAACTF{' + flag + '}'
    print(flag)
```

其实就是给力一个递推式， $a_x = 233 \times 114514^x + 1919810a_{x-1} + a_{x-2}$ ，给定 $a_0 = 0$ 与 $a_1 = 1$ ，要求 $a_{1919809}$，由于没有模数，也没有算法简化，直接运行1919810次显然是不现实的~~（可能跑到你挂子都跑不完）~~，所以我们的目的很明确，要么直接求出通项公式，要么简化算法。求通项依然不现实，因为 114514^i 将会是一个巨大无比的数，且该项和之前的两项都有关，导致你在计算通项公式的过程中异常艰难，那么我们的目标就是简化该算法了。

因为递推式比较复杂，简单的快速幂已经不足够满足我们的实现，所以我们选择矩阵快速幂，根据递推式即可很容易的设计出矩阵，我们有：

$$\begin{bmatrix} 114514^x \\ a_{x-1} \\ a_{x-2} \end{bmatrix} \begin{bmatrix} 114514 & 233 & 0 \\ 0 & 1919810 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 114514^x \times 114514 \\ 233 \times 114514^x + 1919810a_{x-1} + a_{x-2} \\ a_{x-1} \end{bmatrix} = \begin{bmatrix} 114514^{x+1} \\ a_x \\ a_{x-1} \end{bmatrix}$$

这便是递推式的矩阵表现形式，当然这个结果并不唯一，只要你能表示出来即可。这时我们把后面这个变化矩阵记为 M ，则上式可以得到更加简化，即

$$\begin{bmatrix} 114514^x \\ a_{x-1} \\ a_{x-2} \end{bmatrix} M^k = \begin{bmatrix} 114514^{x+k} \\ a_{x-1+k} \\ a_{x-2+k} \end{bmatrix}$$

那么显然我们最后要处理的式子即为：

$$\begin{bmatrix} 114514^2 \\ a_1 \\ a_0 \end{bmatrix} M^{1919809} = \begin{bmatrix} 114514^{1919811} \\ a_{1919810} \\ a_{1919809} \end{bmatrix}$$

所以问题就被转化为要求一个固定矩阵的1919809次幂，这一过程可以通过矩阵快速幂来实现，当然也可以利用sage直接计算

```
from hashlib import md5
from sage.all import *

A = [114514**2, 1, 0]
M = [
    [114514, 233, 0],
    [0, 1919810, 1],
    [0, 1, 0]
]
A = matrix(A)
M = matrix(M)
ans = A*(M**1919809)
ans = ans[0][2]

flag = md5(str(ans).encode()).hexdigest()
flag = 'NUAACTF{' + flag + '}'
print(flag)
```

这是？异或的密件？

题目给出了RSA的n和 $p \bigoplus q$，即p和q的乘积和异或值，所以如果我们拥有其中一个因子，就能够利用异或值表示出另一个因子，由于因子只有512位，直接暴搜就行了

具体做法为：在p的每一位上枚举尝试0或者1，并且根据异或值表示出q，接着利用 $p \times q = n$ 这一点进行剪枝（比如一部分的pq乘积应该小于n等等），最后即可得到RSA的真正因子

```
import math
import sys

def check_cong(k, p, q, n, xored=None):
    kmask = (1 << k) - 1
    p &= kmask
    q &= kmask
```

```
    n &= kmask
    pqm = (p*q) & kmask
    return pqm == n and (xored is None or (p^q) == (xored & kmask))

def extend(k, a):
    kbit = 1 << (k-1)
    assert a < kbit
    yield a
    yield a | kbit

def factor(n, p_xor_q):
    tracked = set([(p, q) for p in [0, 1] for q in [0, 1]
                   if check_cong(1, p, q, n, p_xor_q)])

    PRIME_BITS = int(math.ceil(math.log(n, 2)/2))

    maxtracked = len(tracked)
    for k in range(2, PRIME_BITS+1):
        newset = set()
        for tp, tq in tracked:
            for newp_ in extend(k, tp):
                for newq_ in extend(k, tq):
                    newp, newq = sorted([newp_, newq_])
                    if check_cong(k, newp, newq, n, p_xor_q):
                        newset.add((newp, newq))

        tracked = newset
        if len(tracked) > maxtracked:
            maxtracked = len(tracked)
    print('Tracked set size: {} (max={})'.format(len(tracked), maxtracked))

    for p, q in tracked:
        if p != 1 and p*q == n:
            return p, q

n =
9556286220142782333606758294601566459232209416559556473463354468898104685255501141977
5655956374062008786746746891813021004709042123448569672651054374497117781385077761604
7980080405597869659452834821875687238608431027615500956754621130344908341063911462061
27719571788775471987423072874061061356320902761206747
```

```
p_xor_q =
22269850879776772111039148429865621582206046953973410665736989766977682254864130120246018633163338384593583114676712943981240820821500389619193641655309 8
p, q = factor(n, p_xor_q)
print(p)
print(q)
```

薛定谔的宝藏

这个题看起来很吓人，但其实很简单，我们需要通过必定有一次错误的15次问答来确定11个值，也可以理解成我们可以获取任意的15个方程结果，由于我们并不知道出错的位置，所以在解方程的过程中也充满了很多未知。（这里因为我降低了题目难度，直接暴力枚举哪个方程是错误的也可以）

那么我们换一种思路，现在有11个0或1的值，将他们之间拼接起来即可得到一个11bit的编码，而我们现在可以将它变成15bit的编码同时具有1bit的纠错能力，那么自然而然可以联想到纠错码，由于降低了题目难度，1bit纠错能力的编码有很多，这里选择汉明吗实现即可。我们每一轮从服务器获取一个15bit的汉明码，都一定可以将其恢复，那么11bit的结果也就可以得到了，连续如此100次即可得到flag。

```
from pwn import *
from tqdm import tqdm

def talk(io, payload):
    io.recvuntil("Ask for the god:\n")
    io.sendline(payload)
    return 1 if b'True' in io.recvline() else 0

def makexor(a, b):
    return f'( ( not ( {a} ) ) and ( {b} ) ) or ( ( {a} ) and ( not ( {b} ) ) )'

def gen_payloads():
    payloads = []
    count = 0
    for i in range(1, 16):
        if bin(i).count('1') == 1:
            payloads.append('')
        else:
            payloads.append(f'C{count}')
            count += 1

    for i in range(len(payloads)):
        if payloads[i] == '':
            tmp, res = i+1, '0'
```

```
            for j in range(1, len(payloads)+1):
                if tmp & j != 0 and tmp != j:
                    res = makexor(res, payloads[j-1])
            payloads[i] = res
    return payloads

io = remote('121.4.118.92', 9367)
# context.log_level = 'debug'

payloads = gen_payloads()
for _ in tqdm(range(100)):
    code = [talk(io, i) for i in payloads]
    tmp = 1
    lie = ''
    while tmp <= len(code):
        res = 0
        for i in range(len(code)):
            if (i+1) & tmp != 0:
                res = res ^ code[i]
        lie += str(res)
        tmp <<= 1
    lie = int(lie[::-1], 2)-1
    code[lie] = 1 ^ code[lie]
    ans = ''
    for i in range(len(code)):
        if bin(i+1).count('1') != 1:
            ans += str(code[i]) + ' '
    io.recvuntil("Now open the chests:\n")
    io.sendline(ans)

io.recvuntil("You've found all the keys!\n ")
flag = io.recvline(False)
print(flag)
io.close()
```