H&NCTF WriteUp By Beckoning

Crypto

BabyPQ

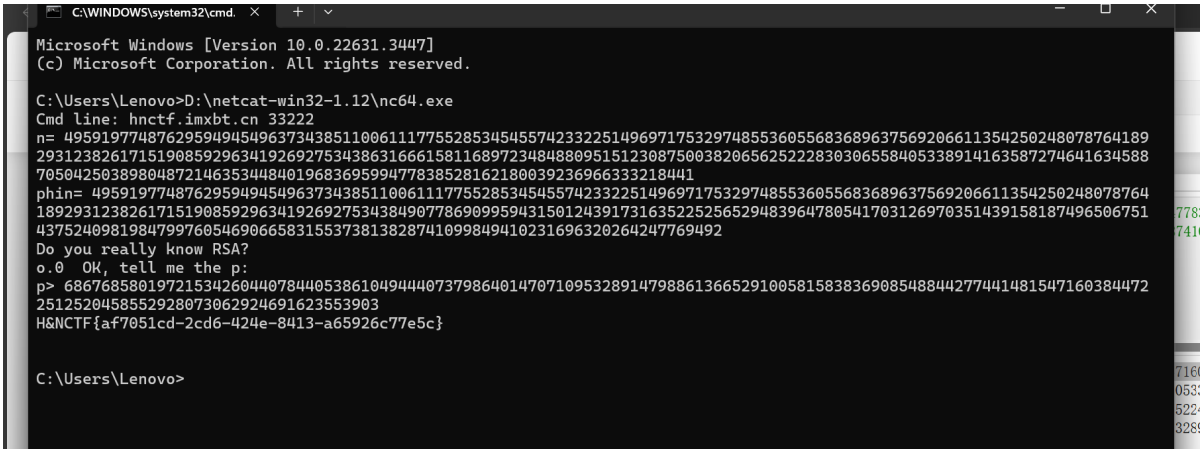
进入端口，得到我们的n和phi

我们已知$n = p \times q, phi = (p-1) \times (q-1)$

```
n=
49591977487629594945496373438511006111775528534545574233225149697175329748553605
68368963756920661135425024807876418929312382617151908592963419269275343863166615
81168972348488095151230875003820656252228303065584053389141635872746416345887050425038980487214635344840196836959947783852816218003923696633321844 1
phi=4959197748762959494549637343851100611177552853454557423322514969717532974855360556836896375692066113542502480787641895293123826171519085929634192692753438631666158116897234848809515123087500382065625222830306558405338914163587274641634588705042503898048721463534484019683695994778385281621800392369663332184410
from sympy import *
x,y= symbols('x,y')
print(solve([x*y-n,(x-1)*(y-1)-phi],[x,y]))
```

通过解方程，我们就可以得到我们的p和q

然后输入我们的p就可以了，得到$flag$



$flag$的值为：`H&NCTF{af7051cd-2cd6-424e-8413-a65926c77e5c}`

f=(?*?)

```
ve9MPTSrRrq89z+I5EMXZg1uBvHoFWBGuzxhSpIwu9XMxE4H2f2O3l+VBt4wR+MmPJlS9axvH9dCn1KqF
UgOIzf4gbMqOMPtRRp+PvfUZWGrJLpxcTjsdml2SS5+My4NIY/VbvqgeH2qVA==
```

先看我们的第一个文件，应该是某种加密方式，最后的hint提示我们e=65537，那么这应该是一个RSA的题

Is this ISO

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

def nextPrime(p):
    while(not isPrime(p)):
        p += 1
    return p


#part1 gen Fp and init supersingular curve
while(1):
    p = 2^randint(150,200)*3^randint(100,150)*5^randint(50,100)-1
    if(isPrime(p)):
        break

F.<i> = GF(p^2, modulus = x^2 + 1)
E = EllipticCurve(j=F(1728))
assert E.is_supersingular()


#part2 find a random supersingular E
ways = [2,3,5]
for i in range(20):
    P = E(0).division_points(choice(ways))[1:]
    shuffle(P)
    phi = E.isogeny(P[0])
    E = phi.codomain()


#part3 gen E1 E2 E3
E1 = E

deg1 = 2
P = E1(0).division_points(deg1)[1:]
shuffle(P)
phi1 = E1.isogeny(P[0])
E2 = phi1.codomain()

deg2 = choice(ways)
P = E2(0).division_points(deg2)[1:]
shuffle(P)
phi2 = E2.isogeny(P[0])
E3 = phi2.codomain()


#part4 leak
j1 = E1.j_invariant()
j2 = E2.j_invariant()
j3 = E3.j_invariant()
```

```
m = bytes_to_long(flag)
n = getPrime(int(j3[0]).bit_length())*nextPrime(int(j3[0]))

print("p =",p)
print("deg1 =",deg1)
print("deg2 =",deg2)
print("leak1 =",j1[0] >> 400 << 400)
print("leak2 =",j1[1] >> 5 << 5)
print("leak3 =",j2[0] >> 5 << 5)
print("leak4 =",j2[1] >> 400 << 400)
print("n =",n)
print("cipher =",pow(m,65537,n))
```

output:

```
p =
680201537161531317827869565786140240595567913096417274637134403255116055511280864
892266374758399999999999999999999999999999999999999999999999999999999999999999999999
999
deg1 = 2
deg2 = 5
leak1 =
846243825149573244267941674169800841612974494600451648078423117633758302748754008
095886353431951741356916130554534930355166966303572547636243946742754925135506964
48
leak2 =
569334021319485756763137861791243638993859221789562816154577308542976649646202822
945301844698450579193326962681107954744030417590603938838204734858601313640848501
344
leak3 =
325720000771917646719671745106544502680895911477018701616420509369836768451047103
170212051953041518446572892754419417720965474892312189833039602545011787135282170
400
leak4 =
607188653779811312711900086497209011406043384341389739547214249680956969386970129
072753246609362372591781044704784040165947962309186896993435085198595013761765998
592
n =
336631348442872227475735277623305104458216242371136122513108670525622121689052346
348620824105820633714562546800934735845952338946486424402720093748392510487705996
344393513464710255615634248631016778273746776059762111364616763243266453211367983
670687473800622632626192654424964609849049774336197739675356042226277067017714668
077407
cipher =
981061494156122429841474191981020211649018630546036250145025388886041926443910410
285625419501928838649549917030871572929742350310704315153379328333472134922063511
471108919837229977468012201791441719683803557984059535585338232823952825639526236
984615792970145973819040810958950113933055041192448122321649098522876888158799842
63470
```

EZmath

```
#sage9.3
from Crypto.Util.number import *
flag = b'Kicky_Mu{KFC_v_me_50!!!}'
p = getPrime(256)
q = getPrime(256)
n = p*q^3
e = # what is usually used ?
N = pow(p, 2) + pow(q, 2)
m = bytes_to_long(flag)
c = pow(m,e,n)

print(c)
print(N)

# c =
34992437145329058006346797890363070594973075282993832268508442432592383794878795192132088668900695623924153165395583430068203662437982480669703879475321408183026259569199414707773374072930515794134567251046302713509056391105776219609788157691337060835717732824405538669820477381441348146561989805141829340641
# N =
14131431108308143454435007577716000559419205062698618708133959457011972529354493686093109431184291126255192573090925119389094648901918393503865225710648658
```

BabyAES

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
from secret import flag
import time
import random

flag = pad(flag,16)
assert b"H&NCTF" in flag

seed = int(time.time())
random.seed(seed)
key = random.randbytes(16)
iv = random.randbytes(16)
aes = AES.new(key,AES.MODE_CBC,iv)
cipher = aes.encrypt(flag)

print(f"cipher = {cipher}")

"""
cipher =
b'\x96H_hz\xe7)\x0c\x15\x91c\x9bt\xa4\xe5\xacwch\x92e\xd1\x0c\x9f\x8fH\x05\x9f\x1d\x92\x81\xcc\xe0\x98\x8b\xda\x89\xcf\x92\x01a\xe1B\xfb\x97\xdc\x0cG'
"""
```

ez_Classic

```
玚辬歈涷鵃敽蒳璋篕虋疱胦曡扻搓叜愡翱蒗羛琡骫敽篬璋穃甗叆肥齵蓥搓朕塖貓暷嚅愡瘃肥孳搽壂谿朘斸嶹嬲瀼
敽聜縏豬蒿彀瘖骫於箓璋緼膉斸黝於裞袨欳磚笉蒴彀肞曰胜栊睾剣廱敽畱穈胅蘄忩勷娍貉敽脉諍愮舋臬蝱綣盬甹
篔穓瘔册姲忼朸劅肐屎於骔滁肻忢廱辬彀積蚀滁
```

Is this ISO2

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

def nextPrime(p):
    while(not isPrime(p)):
        p += 1
    return p


#part1 gen Fp and init supersingular curve
while(1):
    p = 2^randint(250,300)*3^randint(200,250)*5^randint(150,200)-1
    if(isPrime(p)):
        break

F.<i> = GF(p^2, modulus = x^2 + 1)
E = EllipticCurve(j=F(1728))
assert E.is_supersingular()


#part2 find a random supersingular E
ways = [2,3,5]
for i in range(20):
    P = E(0).division_points(choice(ways))[1:]
    shuffle(P)
    phi = E.isogeny(P[0])
    E = phi.codomain()


#part3 gen E1 E2 E3
E1 = E

deg1 = 2
P = E1(0).division_points(deg1)[1:]
shuffle(P)
phi1 = E1.isogeny(P[0])
E2 = phi1.codomain()

deg2 = 5
P = E2(0).division_points(deg2)[1:]
shuffle(P)
phi2 = E2.isogeny(P[0])
E3 = phi2.codomain()
```

```
#part4 leak
unknown = 48
j1 = E1.j_invariant()
j2 = E2.j_invariant()
j3 = E3.j_invariant()

m = bytes_to_long(flag)
n = getPrime(int(j3[0]).bit_length())*nextPrime(int(j3[0]))

print("p =",p)
print("leak1 =",j1[0] >> unknown << unknown)
print("leak2 =",j1[1] >> unknown << unknown)
print("leak3 =",j2[0] >> unknown << unknown)
print("leak4 =",j2[1] >> unknown << unknown)
print("n =",n)
print("cipher =",pow(m,65537,n))
```

MatrixRSA

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

flag = b"H&NCTF{??????????????}" + os.urandom(73)

p = getPrime(56)
q = getPrime(56)
n = p * q

part = [bytes_to_long(flag[13*i:13*(i+1)]) for i in range(9)]

M = Matrix(Zmod(n),[
    [part[3*i+j] for j in range(3)] for i in range(3)
])

e = 65537
C = M ** e
print(f"n = {n}")
print(f"C = {list(C)}")

"""
n = 39234907755759700827296884608902O3
C = [(14197459043254607210198994758701 91, 213451483756822569182900190728 9833,
333208165435748303886136733249733 5), (325463172914139575900236249192 6143,
325020885796084151389919682030227 4, 143405115863064715809863649571 1534),
(281920091466834458073657744435569 7, 252167465901951879537209308626 3363,
285062395941017570536792781753401O)]
"""
```

文献题，查阅关键词Matrix和RSA就可以找到对应的论文

我们只需要将我们的欧拉函数进行替换就可以了

exp:

```
from gmpy2 import *
from Crypto.Util.number import *
import numpy
p=56891773340056609
q=6896411458514866 7
n=p*q
e=65537
phi=(p**2-1)*(p**2-p)*(q**2-1)*(q**2-q)
d=invert(e,phi)
C = [(14197459043254607210198994758701 91, 213451483756822569182900190728 9833,
333208165435748303886136733249733 5), (325463172914139575900236249192 6143,
325020885796084151389919682030227 4, 143405115863064715809863649571 1534),
(281920091466834458073657744435569 7, 252167465901951879537209308626 3363,
285062395941017570536792781753401 0)]
#C = Matrix(Zmod(n),[
 #   [(14197459043254607210198994758701 91, 213451483756822569182900190728 9833,
333208165435748303886136733249733 5), (325463172914139575900236249192 6143,
325020885796084151389919682030227 4, 143405115863064715809863649571 1534),
(281920091466834458073657744435569 7, 252167465901951879537209308626 3363,
285062395941017570536792781753401 0)]
#])
#print(numpy.linalg.matrix_power(C, d) )

print(d)
```

```
import numpy as np
def mod_matrix_multiply(A, B, n):
    # 在模 n 的有限域上计算矩阵乘法
    C = np.dot(A, B) % n
    return C

def mod_matrix_power(A, power, n):
    # 计算矩阵 A 的幂
    result = np.eye(len(A), dtype=int)  # 创建单位矩阵
    while power > 0:
        if power % 2 == 1:
            result = mod_matrix_multiply(result, A, n)
        A = mod_matrix_multiply(A, A, n)
        power //= 2
    return result

C = [(14197459043254607210198994758701 91, 213451483756822569182900190728 9833,
333208165435748303886136733249733 5), (325463172914139575900236249192 6143,
325020885796084151389919682030227 4, 143405115863064715809863649571 1534),
(281920091466834458073657744435569 7, 252167465901951879537209308626 3363,
285062395941017570536792781753401 0)]
n=392349077557597008272968846089020 3
d=629437863600457837298726314164523054437264145529454215952583017642710566035350 2
08107291780058432378770434942883572208005012615834501 13
print(mod_matrix_power(C,d,n))
```

这样我们就可以求出原来的矩阵了，然后我们进行字符转化就可以得到我们的flag了

```
H&NCTF{58bff5c1-4d5f-4010-a84c-8fbe0c0f50e8}
```

HappyDance

```python
#!/usr/bin/env python3
from chacha20 import *
from Crypto.Util.number import long_to_bytes, bytes_to_long
from secret import flag, mykey, mynonce

def chacha_init(key, nonce, counter):
    assert len(key) == 32
    assert len(nonce) == 8

    state = [0 for _ in range(16)]
    state[0] = bytes_to_long(b"expa"[::-1])
    state[1] = bytes_to_long(b"nd 3"[::-1])
    state[2] = bytes_to_long(b"2-by"[::-1])
    state[3] = bytes_to_long(b"te k"[::-1])

    key = bytes_to_long(key)
    nonce = bytes_to_long(nonce)

    for i in range(8):
        state[i+4] = key & 0xffffffff
        key >>= 32

    state[12] = (counter >> 32) & 0xffffffff
    state[13] = counter & 0xffffffff
    state[14] = (nonce >> 32) & 0xffffffff
    state[15] = nonce & 0xffffffff

    return state


def encrypt(data):
    global state
    state = chacha_block(state)
    buffer = b"".join(long_to_bytes(x).rjust(4, b"\x00") for x in state)
    output = []
    for b in data:
        output.append(b ^ buffer[0])
        buffer = buffer[1:]

    return bytes(output)

FLAG = b"H&NCTF{****FAKE****}"
MYNONCE =b"sdf*h*o*"
assert len(flag) == 64
assert len(mynonce) == 8

if __name__ == "__main__":
    while True:
```

```
        print("""==============================
Enjoy the happiness of dancing
1. Dance on My Stage
2. Dance on Your Stage
3. Encrypt flag
""")
        choice = input("> ")

        if choice == '1':
            state = chacha_init(mykey, mynonce, 0)
            print(encrypt(input("input what you want to dance on my stage >
").encode()).hex())

        elif choice == '2':
            yournonce = input("build your own stage > ")
            assert len(yournonce) == 8
            state = chacha_init(mykey, yournonce.encode(), 0)
            yourIn = bytes.fromhex(input("then, have a hex dance > "))
            print(encrypt(yourIn).hex())

        elif choice == '3':
            state = chacha_init(mykey, mynonce[::-1], 0)
            print(encrypt(flag).hex())

        else:
            print("Let's dance together next time~")
            exit()
```

Misc

签到

直接发送公众号，获得我们的第一个$flag$

$flag$=`H&NCTF{W3lcOme_4o_H&NCTF2024!}`