🐈‍⬛

ctf_task WP——komiko

RSA

题目：

```
import gmpy2,libnum
from Crypto.Util.number import getPrime
from secret import flag



p = getPrime(1024)
q = gmpy2.next_prime(p)
n = p * q
e = 0x10001
print("n =",n)
m = libnum.s2n(flag)
c = pow(m,e,n)
print("c =", c)

# n = 14249698794788660632082649745239538576694780729238433273
# c = 11958770070010179741596317998031086988694478903332118030
```

分析：pq为相邻素数问题，因为p，q非常接近，故将n开方的下一个素数就是p，然后用n整除p得到q，即成功分解了n

exp：

```
from Crypto.Util.number import *
from gmpy2 import *
n = 1424969879478866063208264974523953857669478072923843327397
c = 119587700700101797415963179980310869886944789033321180306
e = 65537

sn = isqrt(n)
```

```
q = next_prime(sn)
p = n//q
phi = (p-1)*(q-1)
d = invert(e,phi)
m = pow(c,d,n)
print(long_to_bytes(m))

#flag = b'SquareRootOfNIsNotSafeWhenYouAreLearned'
```

CRT

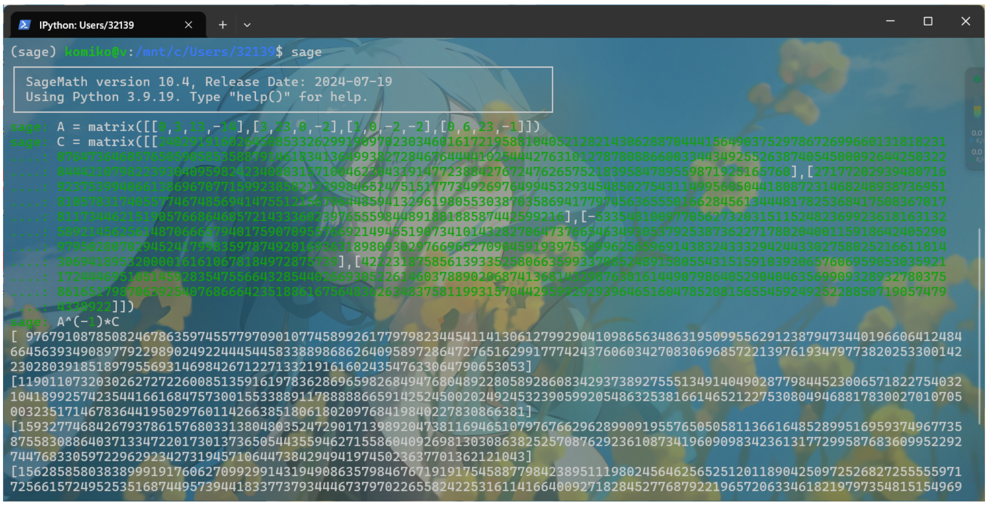
题目：

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

p = [getPrime(1024) for i in range(4)]
A = random_matrix(ZZ,4,4)
x = vector(p)
C = A*x
m = bytes_to_long(flag)
C_list = [m^2 % p[i] for i in p]

print(A.list())
print(C.list())
print(C_list)
#x = inverse A*C
#x_1,x_2,x_3,x_4





# [0, 3, 13, -14, 3, 23, 0, -2, 1, 0, -2, -2, 0, 6, 23, -1]

# [240291916026450853326299190970230346016172195881040521282l
#  2717720293948071692375399406613869670771599238582123998465
#  -5335481009770562732031511524823699236181631325892145625614
#  4222318758561393352580663599337085248915805543151591039306
```

```
# [14809604703296380436806887270382397389304757930982186596831
```

分析:

C = A*x ，所以x = inverse(A)*C,对C矩阵左乘A矩阵的逆得到p，发现C_list = [m^2 % p[i] for i in p]就是同余方程组，且p[i]互质，用中国剩余定理求解得m^2,最后开方得到的就是m



假设我们有明文块 $P_1, P_2, \ldots, P_n$，IV 为 IV，加密密钥为 K，AES-CBC 加密流程如下：

1. $C_1 = E_K(P_1 \oplus IV)$
2. $C_2 = E_K(P_2 \oplus C_1)$
3. $C_3 = E_K(P_3 \oplus C_2)$
4. ...
5. 最终密文为 $C = C_1||C_2||\ldots||C_n$

这里 E_K 表示使用密钥 K 的 AES 加密操作，$\oplus$ 表示异或操作，$||$ 表示块的拼接。

exp:

```
from Crypto.Util.number import long_to_bytes
from sympy.ntheory.modular import crt
from gmpy2 import*
x = [ 976791087850824678635974557797090107745899261779798234
c = [1480960470329638043680688727038239738930475793098218659б
m_2, _ = crt(x, c)
m = isqrt(m_2)
```

```
flag = long_to_bytes(m)
print(flag)

#flag = 'MaybeLinearAlgebraStillNeedYourEffort'
```

AES

题目：

```
from Crypto.Cipher import AES
import os
from Crypto.Util.number import *
from secret import flag



def pad(text):
    if len(text) % 16:
       add = 16 - (len(text) % 16)
    else:
       add = 0
    text = text + (b'0' * add)
    return text

def enc():
    key=os.urandom(2)*16
    print(key)

    iv=os.urandom(16)
    print(iv)

    aes=AES.new(key,AES.MODE_CBC,iv)
    enc_flag = aes.encrypt(flag)
    print(enc_flag)

flag = pad(flag)
enc()
```

```
'''
b'\xb8r\xb8r\xb8r\xb8r\xb8r\xb8r\xb8r\xb8r\xb8r\xb8r\xb8r\xb8
b"\t\x17\x87\xeb\xe0'P#2\xa9\x83\xdf\xad\x04\xa1\xa9"
b"\xc4']\xab!\x02/\x1d\x04\xef,\xf4^mb\x0f\x9b?\xd6\xa6k*\xd7
'''
```

从题目里提取到AES_CBC加密关键词，现学一下：

CBC是AES对称加密算法的常用工作模式之一，在CBC模式下，每个明文块在加密前与前一个明文块进行XOR操作，再进行加密，直至所有明文块加密完成。（第一个明文块通过初始化向量IV来加密）

python中的pycryptodome库可以实现AES-CBC的加密和解密：

```
key =
iv =
plaintext =
#加密
cipher = AES.new(key, AES.MODE_CBC, iv)
ciphertext = cipher.encrypt(pad(plaintext, AES.block_size))
#解密
decipher = AES.new(key, AES.MODE_CBC, iv)
decrypted_data = unpad(decipher.decrypt(ciphertext), AES.bloc
```

exp：

```
from Crypto.Cipher import AES

key = b'\xb8r' * 16
iv = b"\t\x17\x87\xeb\xe0'P#2\xa9\x83\xdf\xad\x04\xa1\xa9"
enc_flag = b"\xc4']\xab!\x02/\x1d\x04\xef,\xf4^mb\x0f\x9b?\xd

cipher = AES.new(key, AES.MODE_CBC, iv)
decrypted_flag = cipher.decrypt(enc_flag).rstrip(b'0')#移除填充
print("flag = ", decrypted_flag)

#flag = b'BasicallyNotationOfAES'
```

ECC

题目：

```
from fastecdsa.curve import P256 as Curve
from fastecdsa.point import Point
from Crypto.Util.number import bytes_to_long, isPrime
from os import urandom
from random import getrandbits
from secret import flag

file_out = open("ecc-rsa.txt", "w")

def gen_rsa_primes(G):
    urand = bytes_to_long(urandom(256//8))
    while True:
        s = getrandbits(521) ^ urand
        print("load...")

        Q = s*G
        if isPrime(Q.x) and isPrime(Q.y):
            print("ECC Private key:", hex(s))
            print("RSA primes:", hex(Q.x), hex(Q.y))
            print("Modulo:", hex(Q.x * Q.y))
            return (Q.x, Q.y)

ecc_p = Curve.p
a = Curve.a
b = Curve.b

Gx = Curve.gx
Gy = Curve.gy
G = Point(Gx, Gy, curve=Curve)

e = 0x10001
p, q = gen_rsa_primes(G)
n = p*q
```

```
file_out.write("ECC Curve Prime: " + hex(ecc_p) + "\n")
file_out.write("Curve a: " + hex(a) + "\n")
file_out.write("Curve b: " + hex(b) + "\n")
file_out.write("Gx: " + hex(Gx) + "\n")
file_out.write("Gy: " + hex(Gy) + "\n")
file_out.write("e: " + hex(e) + "\n")
file_out.write("p * q: " + hex(n) + "\n")

c = pow(flag, e, n)
file_out.write("ciphertext: " + hex(c) + "\n")
file_out.close()

ECC Curve Prime: 0xffffffff00000001000000000000000000000000ff
Curve a: -0x3
Curve b: 0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63b
Gx: 0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a1394
Gy: 0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406
e: 0x10001
p * q: 0x2c57c7758740e0699e9374ad86648bba759f803ab97b0a467cb9
ciphertext: 0x111b723c688279c2d5c9bc70018609b49ff9f836f7e7193
```

一点点简单的学习笔记：

- ECC椭圆曲线加密，属于非对称加密，可以用较短的密钥就可以达到使用了较长密钥的RSA加密的安全效果。
- 该加密方式是基于形如y^2 = x^3 + ax + b的椭圆曲线方程进行操作（a,b要满足4*a^3 +27b^2 != 0的要求，这样是确保不会出现奇点，也就是不可导的点）
- 加密及解密过程：

1.选择一条椭圆曲线Ep(a,b)，并选取在曲线的一点作为基点P

2.选择一个大数k作为私钥，并计算公钥Q = kP

3.加密：选择一个随机数r，将明文M生成加密后的点对C(rP,M+rQ)

4.解密：使用密文点对：y-kx = M M + rQ - k(rP) == M +rkP - krp = M

题目分析：

读题得到RSA中的p，q 就是Q = kP的点对，已知了a，b，p那这个椭圆曲线就是已知的

又因为p，q是满足上述的式子，代入进去就是

$$q^2 = p^3 + ap + b$$

又给出了p*q = n的值

$$q = \sqrt{p^3 + ap + b}$$

$$\sqrt{p^3 + ap + b} * p = n$$

$$p^5 + ap^3 + bp^2 = n^2$$

化成了一个解一元五次方程（在模P下成立），所以放在sage上解一下

```
In [4]: a = -0x3
        b = 0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b
        n = 0x2c57c7758740e0699e9374ad86648bba759f803ab97b0a467cb993888e964021cee19606b2762ec3a513c8a80aeaf9b056019d6334f5ba32ba2c48de0854a6d9
        p = 0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff
        PR.<x> = PolynomialRing(Zmod(p))
        f=x^5+a*x^3 + b*x^2 - n^2
        roots=f.roots()
        print(roots)

[(60949102002512525849174187379731487698048308191913961394910054055845337804149, 1)]
```

得到了p之后，后面就是普通rsa问题了

exp：

```
a = -0x3
b = 0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3
n = 0x2c57c7758740e0699e9374ad86648bba759f803ab97b0a467cb9938
p = 0xffffffff00000001000000000000000000000000ffffffffffff
PR.<x> = PolynomialRing(Zmod(p))
f=x^5+a*x^3 + b*x^2 - n^2
roots=f.roots()
print(roots)

#6094910200251252584917418737973148769804830819191396139491000

import gmpy2
from Crypto.Util.number import long_to_bytes

n = 0x2c57c7758740e0699e9374ad86648bba759f803ab97b0a467cb9938
p =6094910200251252584917418737973148769804830819191396139491
```

```
q = n//p
e = 65537
c = 0x111b723c688279c2d5c9bc70018609b49ff9f836f7e71930be90318
d = gmpy2.invert(e,(p-1)*(q-1))
print(long_to_bytes(pow(c,d,n)))

#b'Aha!YouAmostCheatedByMe'
```

DSA

题目：

```
from secret import r, t,flag
from Crypto.Util.number import *

flag = bytes_to_long(flag.encode())
e = 65537

def gen_keys():
    p = getPrime(1024)
    q = getPrime(1024)
    phi = (p-1)*(q-1)
    d = inverse(e,phi)
    n = p*q
    print(f'n = {n}')
    Gensin_imapct = (d ** 6 + 7) % phi
    print(f'Gensin_imapct= {Gensin_imapct}')
    return d, n, Gensin_imapct

def sign_in(n,d):
    m = flag * pow(r,e**3+d**4,n) % n
    s = pow(m,d**2,n)
    return s

def clue():
    assert t > 0,r > 1
    clue = pow(r,t)+1
    #print(t)
```

```
    print(isPrime(clue))

d,n,Gensin_imapct = gen_keys()
clue()
sign = sign_in(n,d)
print(f'sign = {sign}')


'''
n = 196396003282238446717044891235469885019032914733498723610
Gensin_imapct= 186155554283607372038584831197613798031066441
1
sign = 182612882045389811815720304827597984265847901365421186
'''
```

分析：

题目已知

$$
\begin{gathered}
\text{Gensin imapct} = (d^6 + 7) \mod \phi \\
m = \text{flag} \times r^{e^3+d^4} \mod n \\
s = m^{d^2} = \left(\text{flag} \times r^{e^3+d^4}\right)^{d^2} \mod n \\
s = \text{flag}^{d^2} \times r^{(e^3+d^4)d^2} \mod n
\end{gathered}
$$

由

$$
\begin{gathered}
d \cdot e \equiv 1 \mod \phi(n) \\
m^{ed} = m \mod n
\end{gathered}
$$

可以推出指数上ed = 1。构造ff，整体带入G换出来flag

$$
\begin{gathered}
\text{令} ff = s \times (2^{e+G-7})^{-1} \mod n \\
ff = s \times (2^{e+G-7})^{-1} \mod n \\
ff = \text{flag}^{d^2} \mod n \\
\text{flag} = ff^{e^2} \mod n
\end{gathered}
$$

exp：

```
from Crypto.Util.number import *
n = 1963960032822384467170448912354698850190329147334987236101
G= 1861555542836073720385848311976137980310664412340875167993
e = 65537
sign = 1826128820453898118157203048275979842658479013654211862
ff = sign * pow(pow(2,e+G -7,n),-1,n)
flag = pow(ff,e**2,n)
print(long_to_bytes(int(flag)))

#b'SYC{G0od_Math_h3lps_S1gnature_in_RSA}'
```