Crypto easy

El cifrao del cuñao

pc1b

1. Descripción

Mi *cuñao* me ha dicho que ha ideado un sistema para guardar sus claves de RSA cifradas con las mismas claves... ya le he dicho que no tiene buena pinta pero, como buen *cuñao*, tiene un amigo que “sabe del tema” y le dice que está perfectamente.

¿Puedes demostrarle que se equivoca?

2. Flag

HackOn{c4m4r3r0_7r4ig4_l4_mUlt4_a7b93daf}

3. Resolución

En primer lugar es necesario obtener el valor de N, ya que es el módulo utilizado en todas las operaciones. Sin embargo, el enunciado solo proporciona el valor de N1.

$$N1 = p \times q \times r$$
$$N = p \times q$$

Como r es un primo de solo 24 bits es fácil de adivinar su valor por fuerza bruta, probando todos los números impares de 24 bits o en páginas como http://factordb.com/ o https://www.dcode.fr/rsa-cipher.

Es posible que, en este punto, algunos participantes crean que han conseguido los factores p y q pero en realidad solo han conseguido r y N.

Una vez se obtenga el valor de N, es necesario darse cuenta del siguiente detalle:

$$c1 = (7 \times p + k_1 \times q)^{e1} (mod\ N) = (7 \times p)^{e1} + (k_1 \times q)^{e1} (mod\ N)$$
$$c2 = (5 \times p + k_2 \times q)^{e2} (mod\ N) = (5 \times p)^{e2} + (k_2 \times q)^{e2} (mod\ N)$$

Siendo K_1 y K_2 dos números aleatorios

Esto es debido a que en la expansión del binomio de Newton, todos los sumandos, excepto el primero y el último, son múltiplos de $p \times q = N$ por lo que, al aplicar el módulo, el resultado de esos sumandos es 0. Con esto en mente, se puede plantear un sistema de ecuaciones de la siguiente forma:

$$c1^{e2} = (7 \times p)^{e1 \times e2} + (k_1 \times q)^{e1 \times e2} (mod\ N)$$
$$c2^{e1} = (5 \times p)^{e2 \times e1} + (k_2 \times q)^{e2 \times e1} (mod\ N)$$

Multiplicando la primera ecuación por $5^{e1 \times e2}$, la segunda por $7^{e1 \times e2}$ y restando ambas ecuaciones, se obtiene:

$$ec = (5 \times k_1 \times q)^{e1 \times e2} (mod\ N) + (7 \times k_2 \times q)^{e2 \times e1} (mod\ N);$$
$$ec = (5 \times k_1 + 7 \times k_2)^{e2 \times e1} \times q^{e2 \times e1} (mod\ N)$$

Es claro que ec es un múltiplo de q por lo que se puede calcular el máximo común divisor entre ec y N obteniendo el valor de q. Finalmente es trivial sacar p como $N \div q$.

Una vez se ha factorizado N, se obtiene el valor de la clave privada d como el inverso multiplicativo de e múdulo $\phi(N) = (p-1) \times (q-1)$, según la implementación de RSA:

$$d = e^{-1}(mod\ \phi(N))$$

Finalmente se obtiene la flag elevando el mensaje cifrado a d módulo N, según la implementación de RSA:

$$flag = flag\_enc^{d}(mod\ N)$$

4. *solver.py*

```
from math import gcd
import rsa
from sympy import mod_inverse
from Crypto.Util.number import long_to_bytes

#Read from file and get params
with open('output.txt', 'r') as archivo:
    lineas = archivo.readlines()
    N1 = int(lineas[0][4:])
    e1 = int(lineas[1][4:])
    e2 = int(lineas[2][4:])
    c1 = int(lineas[3][4:])
    c2 = int(lineas[4][4:])
    flag_enc = int(lineas[5][10:])

#Froce-brute r and get N=p*q
Nsol= 0
for i in range(pow(2, 23) +1, pow(2, 24), 2):
    if (gcd(N1, i) != 1):
        Nsol= N1//i
        break

#pow ecuations their respective oposite exponents
c1mod = pow(c1, e2, Nsol)
c2mod = pow(c2, e1, Nsol)

#Operate between ecuatios to get eq2=pow(k*q, e1*e2, N)
eq2 = (c2mod * pow(7,e1*e2, Nsol) -c1mod* pow(5, e1*e2, Nsol)) % Nsol

#Do gcd between eq2 and N to get q and get p trivially
qsol= gcd(eq2, Nsol)
psol = Nsol // qsol
```

```
print (f'\n\n\np: {psol}\nq: {qsol}\n')

#Get the rsa private key and decrypt the flag

flag_dec = pow(flag_enc, mod_inverse(0x10001, (psol -1)*(qsol-1)), Nsol)
flag_dec = long_to_bytes(flag_dec)
print(flag_dec.decode('utf8'))
```