

## CÁTEDRA DE CIBERSEGURIDAD CIBERUGR, INCIBE-UGR

Nombre	Rivest, Shamir y Adleman
Categoría	CRIPTO
Dificultad	FÁCIL
Puntos	200

### DESCRIPCIÓN DEL RETO

Hemos recibido un correo un tanto extraño de parte del fabricante, la única conclusión que hemos podido sacar es que habla sobre un exponente bajo y nos ha adjuntado un archivo. ¿Podrás averiguar que nos quiere decir el fabricante?

### WRITEUP

1. En este reto nos proporcionan tanto el archivo que cifra la *flag* como la *flag* cifrada.
2. Al visualizar el script que nos proporcionan, observamos como se ha implementado un algoritmo RSA débil, debido a que se utiliza un exponente público bajo.

```
1  from Crypto.Util.number import bytes_to_long, getPrime
2
3
4  # primero se generan p y q para obtener N
5
6  p = getPrime(512)
7  print("\nEl valor de p generado es :", p)
8  q = getPrime(512)
9  print("\nEl valor de q generado es :", q)
10
11 print("\nEl valor de n es :", p*q)
12
13 # ahora pasamos el valor de la flag de bytes a long para poder operar
14
15 flag = b'UGR_ETSIIIT_CTF24{RSA_1s_Fun}'
16 pt = bytes_to_long(flag)
17
18 # realizamos la congruencia entre el valor pt, e con su valor bajo (3,11,...) y N
19
20 e = 3
21
22 print("\nEl valor cifrado es:", pow(pt, e, p*q))
23
```

3. Primero, se generan dos números primos grandes,  $p$  y  $q$ , y se calcula su producto  $N = p \cdot q$ . Luego, se convierte la bandera (mensaje a cifrar) de bytes a un entero largo (pt). El exponente de cifrado ( $e$ ) se establece en 3 en este caso.
4. El mensaje se cifra elevándolo a la potencia del exponente de cifrado ( $e$ ) módulo  $N$ . Esto se hace mediante la función  $\text{pow}(\text{pt}, e, p \cdot q)$ .
5. Para más información, lea <https://irOnstone.gitbook.io/crypto/rsa/public-exponent-attacks/small-e>
6. Sabiendo esto, vamos a crear el script "solver.py" que calcula la raíz cubica entera de "c". Dado que el exponente "e" es 3, esto devuelve la raíz cúbica de "c" como un número entero.

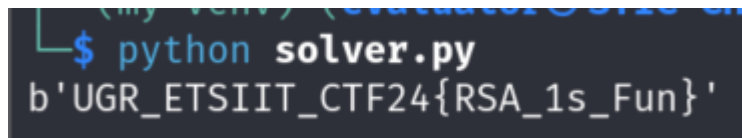
```
1 import gmpy2
2 from Crypto.Util.number import long_to_bytes
3
4 # low public exponent attack
5
6 n = 96288672625869419435629938062971937148872714166828198995826754579597049
7 c = 724364888061568805794282001684615383621935656781107864449032998797098292
8 e = 3
9
10 m = gmpy2.iroot(c, 3)[0]
11 print(long_to_bytes(m))
12
```

7. Ejecutamos el script

---

```
python solver.py
```

---



```
$ python solver.py
b'UGR_ETSIIIT_CTF24{RSA_1s_Fun}'
```