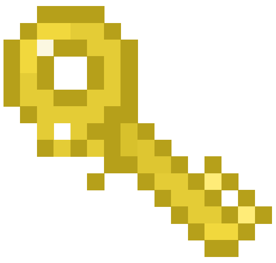
`kh5YhlzvJp`



Type: Cryptography

Difficulty: 🔴 Hard

Flag: `TFCCTF{1n53cur3_rng_15_th3-d0wnf4ll-_0f_m4ny_4pp5.f0ll0w_fsociety}`

Points: 490

Description

VSZjXqmSYy HO9r144FBh 1Gv5vy5U1l uxLV68cO0m vcT2Cdrha7 qliRgqiCZC MZxl0YVJ6u fnSepQQNYh bJjYwWg3PK 4N1PJoBUWQ 8ctNT4y2OV IPDydPtdAt ZGB5uyzMly eSuvvq5h9X Vj9soPan2T chy1HfePMc pjMpOB0s9r NvSyG6iG7O R1zDwOq6El

Solution

The challenge generates a random 4096 bit number four times, and outputs it into a out-file. It then generates a fifth and uses it to encrypt the flag using OTP.

After searching online, you can find that Python uses a Mersenne Twister algorithm for their random() function. Mersenne Twister is not a secure RNG, and can be predicted after 624 numbers. For 4096 bit numbers, though, we only need 5 random numbers to confidently predict the following number. However, we only get 4.

We notice that the flag is being encrypted with just the first 512 digits of the generated number. This is great news!

```
x = xor(flag, str(x)[:512])
```

Even though the Mersenne Twister is only fully predictable for 4096 bit numbers after 5 inputs, we can confidently predict the first few digits of the next number after only 4!

As such, we can easily solve it using a GitHub tool.

https://github.com/kmyk/mersenne-twister-predictor

Solve Script

```
import base64
from pwn import xor
from mt19937predictor import MT19937Predictor

predictor = MT19937Predictor()

with open("c.out", "r") as f:
    for _ in range(4):
        x = f.readline()
        predictor.setrandbits(int(x), 4096)

    y = predictor.getrandbits(4096)
    print(y)
    b64s = f.readline()
    b64s = b64s.split(" ")[1]
    secret = xor(str(y)[:512], base64.b64decode(b64s).decode("utf-8"))
    print(secret)
```