```
        original_char_code = (temp + i - 1) & 0xFF



    decrypted_flag += chr(original_char_code)



print("Flag yang berhasil didekripsi:")

print(decrypted_flag)
```

```
PS D:\Home\CTF\GemastikCTF\2025\Packs> & C:\Users\syahr\Ap
ver.py
Flag yang berhasil didekripsi:
GEMASTIK18{S1mpl3_P4ck3r_f0r_4_S1mpl3_Ch4ll3nge}
```

Flag

`GEMASTIK18{S1mpl3_P4ck3r_f0r_4_S1mpl3_Ch4ll3nge}`

Cryptography

nousagi (1000 pts)

Langkah Penyelesaian

chall.py

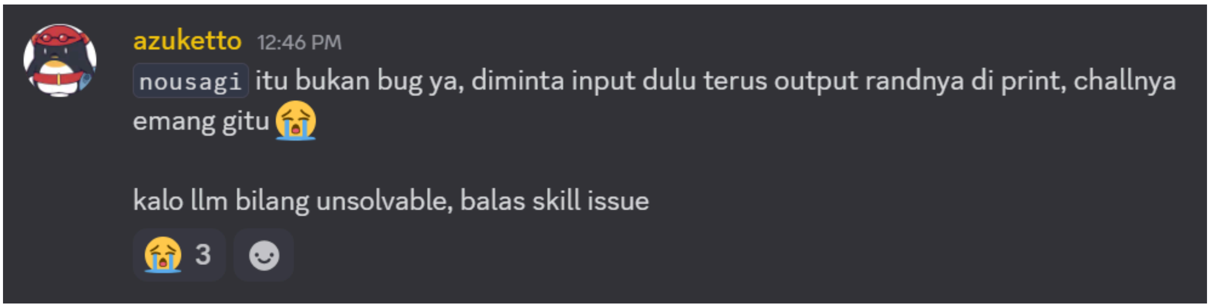
```python
import randcrack
import random
import os
from libnum import s2n
import signal

seed = os.urandom(12)
wack = randcrack.RandCrack()
random.seed(seed)

print("Find the rabbit!")
for _ in range(624):
    signal.alarm(2) # Dont waste time
    num = int(input(">> "))
    signal.alarm(0)

    wack.submit(num)
    print(random.getrandbits(32))

rabbit = 0
for _ in range(624):
    rabbit ^= random.getrandbits(32) ^ wack.predict_getrandbits(32)

if abs(rabbit) < s2n(b'ribbit') % 1337:
    print("You found the rabbit!")
    gift = open('flag.txt').read()
    print(gift)
else:
    print("You didn't find the rabbit, try again!")
    print("Rabbit was:", rabbit)
    exit(1)
```

Apa yang terjadi di sini?

1. Server ngeseed random
2. Kita ngasih input 32-bit ke randcrack
3. Server ngasih tau output 32-bit dari random
4. Ulangi langkah 2-3 sebanyak 624 kali
5. 624 angka berikutnya dari random dan randcrack di-xor
6. Jika hasil xornya 0, dapat flag

Sekedar info, library random di python menggunakan algoritma Mersenne Twister/MT19937, yang mana outputnya bisa diprediksi apabila kita memiliki cukup banyak output sebelumnya. Randcrack adalah library yang bisa melakukan hal ini.

Yang jelas, kita tidak bisa serta merta menggunakan output yang diberikan server (di langkah 3) sebagai input randcrack (di langkah 2), karena urutannya terbalik.



Lalu apa input yang harus kita pilih? Awalnya saya mencoba bermain-main dengan randcrack dan mengamati apa yang terjadi apabila saya kasih nilai 0 semua.

```
>>> import randcrack
>>> rc = randcrack.RandCrack()
>>> m = [0]*624
>>> for i in m: rc.submit(i)
...
>>> [rc.predict_getrandbits(32) for _ in range(624)]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Hasilnya adalah 624 angka berikutnya juga nol. Lalu saya mencoba mengubah satu angka terakhir menjadi 1.

```
>>> rc = randcrack.RandCrack()
>>> m = [0]*623 + [1]
>>> for i in m: rc.submit(i)
...
>>> [rc.predict_getrandbits(32) for _ in range(624)]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
4270312104, 0]
```

Hasilnya? Angka kedua dari terakhir berubah menjadi 4270312104. Selain itu, angka 1 yang saya pilih muncul dua kali pada posisi 226 dan 453. Sisanya tetap 0. Lalu saya ubah lagi angka 1 tadi menjadi angka lain, yaitu 1337.

```
>>> rc = randcrack.RandCrack()
>>> m = [0]*623 + [1337]
>>> for i in m: rc.submit(i)
...
>>> [rc.predict_getrandbits(32) for _ in range(624)]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1337, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

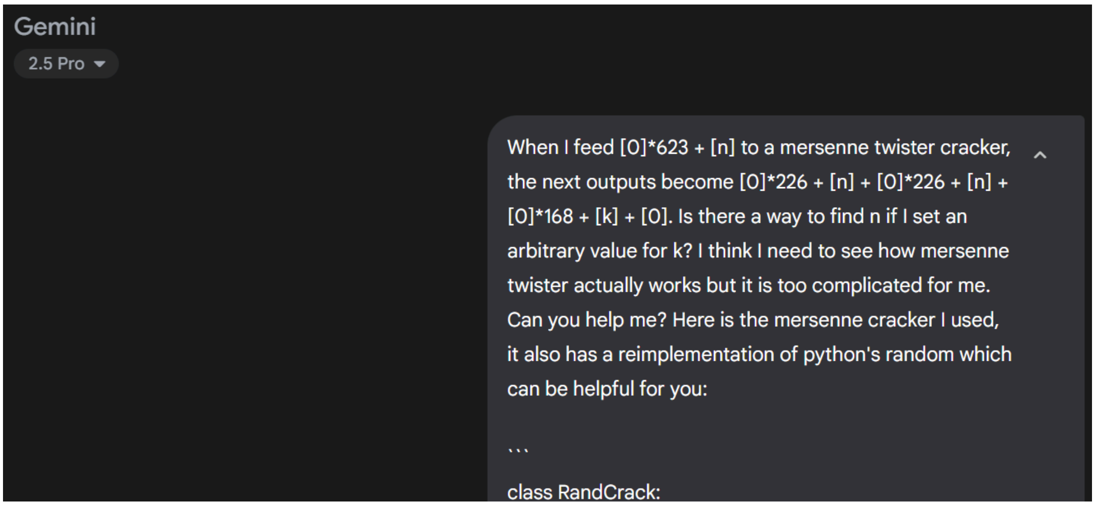
```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1337,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1535356277, 0]
```

Hasilnya juga sama. Angka kedua dari terakhir berubah menjadi angka lain. Angka pilihan yaitu 1337 muncul 2 kali di posisi yang sama. Sisanya masih 0. Akhirnya saya sampai pada dugaan berikut: “**Apabila [0]*623 + [n] diberikan kepada sebuah mersenne cracker, maka 624 output berikutnya adalah [0]*226 + [n] + [0]*226 + [n] + [0]*168 + [k] + [0].**” Hal ini hanya berlaku apabila n cukup kecil, karena jika terlalu besar maka output ke-624 akan menjadi angka selain 0. Detailnya belum saya cari tahu, ini ngesolvenya -1 jam wkwk. Perhatikan bahwa karena nilai n muncul dua kali, maka akan saling meng-cancel saat di-xor, dan menyisakan k sendiri.

Dugaan tadi menimbulkan dua pertanyaan:

1. Apa hubungan k dengan n? Apakah bisa mencari nilai n jika kita setel nilai k-nya?
2. Kalaupun kita bisa mencari nilai n setelah menyetel nilai k, bagaimana cara memilih nilai k yang cocok? Karena n berada di posisi 624 berarti kita hanya bisa mendapat 623 output dari random, yang tentunya kurang 1.

Untuk pertanyaan 1, saya anu. Apa itu namanya… Nah itu.



Lalu Gemini memberikan script berikut (saya ubah dikit biar bisa diimport):

solve_n.py

```python
from randcrack import RandCrack
import random

def find_n_from_k(k):
    """
    Calculates the input value 'n' that will result in 'k' appearing at the
    specified position in the Mersenne Twister output sequence.
    """
    # Helper instance to use the class methods
    cracker = RandCrack()

    # Constants from the Mersenne Twister algorithm
    UPPER_MASK = 0x80000000
    LOWER_MASK = 0x7fffffff
    MATRIX_A = 0x9908b0df

    # --- Step 1: Calculate the known untempered values ---
    # s_k is the untempered internal state corresponding to our target
output k.
    # s_0 is the untempered internal state corresponding to an input of 0.
    s_k = cracker._to_int(cracker._harden_inverse(cracker._to_bitarray(k)))
    s_0 = cracker._to_int(cracker._harden_inverse(cracker._to_bitarray(0)))

    # --- Step 2: Isolate the 'twist' component of the equation ---
    # From our equation: s_k = s_0 ^ twist(s_0, s_n)
    # Rearranging gives: twist(s_0, s_n) = s_k ^ s_0
    twist_val = s_k ^ s_0

    # --- Step 3: Reverse the twist operation to find s_n ---
    # The twist is defined as: (y >> 1) ^ mag01[y & 1]
    # where y = (s_0 & UPPER_MASK) | (s_n & LOWER_MASK)
    # The LSB of y is the same as the LSB of s_n. We don't know it,
    # so we must test both possibilities (0 and 1).

    for lsb_guess in [0, 1]:
        temp_twist = twist_val

        # If the LSB guess is 1, we must reverse the XOR with MATRIX_A
        if lsb_guess == 1:
```

```
            temp_twist ^= MATRIX_A

        # Reverse the right shift (y >> 1) to get the upper 31 bits of y
        y_upper_31 = temp_twist << 1

        # Reconstruct y:
        # The MSB of y comes from the MSB of s_0.
        y = (s_0 & UPPER_MASK) | y_upper_31

        # Reconstruct s_n:
        # The lower 31 bits of s_n are the same as the lower 31 bits of y.
        # The MSB is masked out, so we can assume it's 0 for this
calculation.
        # We then set the LSB to our guessed value.
        s_n = (y & LOWER_MASK)
        if lsb_guess == 1:
            s_n |= 1 # Set the LSB to 1
        else:
            s_n &= ~1 # Set the LSB to 0

        # --- Step 4: Verify our guess ---
        # We check if the LSB of our reconstructed s_n matches our initial
guess.
        # This is the crucial validation step.
        if (s_n & 1) == lsb_guess:
            # The guess was correct. We have found the untempered s_n.

            # Now, temper s_n to find the final input value n
            n = cracker._to_int(cracker._harden(cracker._to_bitarray(s_n)))
            return n

    return None # Should not happen if logic is correct

if __name__ == '__main__':
    # --- Example Usage ---
    # Set an arbitrary value for k
    chosen_k = random.getrandbits(32)

    # Find the corresponding n
    result_n = find_n_from_k(chosen_k)
```

```python
    if result_n is not None:
        print(f"To get k = {chosen_k}, you need to set n = {result_n}")

        # --- Verification (Optional) ---
        print("\nVerifying the result...")
        cracker = RandCrack()

        # Feed the cracker the specific input pattern
        for _ in range(623):
            cracker.submit(0)
        cracker.submit(result_n)

        # Generate the next 624 outputs
        outputs = [cracker.predict_getrandbits(32) for _ in range(624)]

        # Check if k is at the correct position (index 622)
        k_from_output = outputs[622]
        print(f"Output at index 622: {k_from_output}")

        if k_from_output == chosen_k:
            print("✅ Success! The calculated n produced the correct k.")
        else:
            print("❌ Failure! The calculation was incorrect.")
```

```
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python solve_n.py
To get k = 3116759208, you need to set n = 1715727779

Verifying the result...
Output at index 622: 4258130346
❌ Failure! The calculation was incorrect.
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python solve_n.py
To get k = 3925252088, you need to set n = 3544805080

Verifying the result...
Output at index 622: 3925252088
✅ Success! The calculated n produced the correct k.
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python solve_n.py
To get k = 1052815376, you need to set n = 2983588661

Verifying the result...
Output at index 622: 4074275606
❌ Failure! The calculation was incorrect.
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python solve_n.py
To get k = 3202875774, you need to set n = 2878650348
```

```
Verifying the result...
Output at index 622: 922231674
❌ Failure! The calculation was incorrect.
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python solve_n.py
To get k = 1217967896, you need to set n = 2854968054

Verifying the result...
Output at index 622: 3230176540
❌ Failure! The calculation was incorrect.
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python solve_n.py
To get k = 835030691, you need to set n = 939121371

Verifying the result...
Output at index 622: 1369942093
❌ Failure! The calculation was incorrect.
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python solve_n.py
To get k = 299910534, you need to set n = 1755210715

Verifying the result...
Output at index 622: 299910534
✅ Success! The calculated n produced the correct k.
```

Kadang-kadang bisa, seringnya tidak bisa :/

Kita ke pertanyaan 2. Saya kepikiran begini: gimana kalau 623 angkanya dikasih ke randcrack, terus 1 angka diisi ngawur?

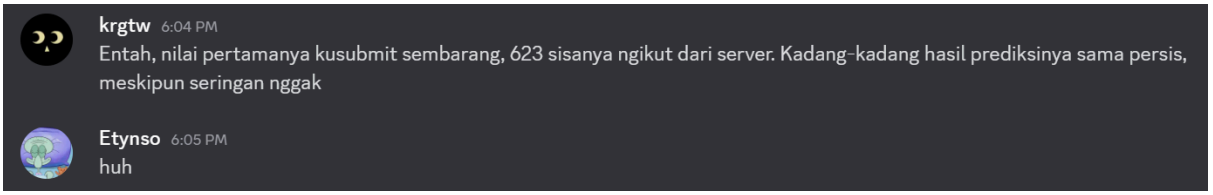
kokbisa.py

```python
import random
import randcrack
rc = randcrack.RandCrack()

random.getrandbits(32)
rc.submit(1234567)

for _ in range(623):
    rc.submit(random.getrandbits(32))

for _ in range(624):
    if random.getrandbits(32) != rc.predict_getrandbits(32):
        print('beda')
```

```
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python
kokbisa.py
beda
beda
beda
```

```
beda
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python
kokbisa.py
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python
kokbisa.py
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python
kokbisa.py
beda
beda
beda
beda
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python
kokbisa.py
beda
beda
beda
beda
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python
kokbisa.py
beda
beda
beda
beda
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python
kokbisa.py
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python
kokbisa.py
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python
kokbisa.py
beda
beda
beda
beda
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python
kokbisa.py
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python
kokbisa.py
beda
beda
beda
beda
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi>
```

Dan ternyata… kadang-kadang dari 624 angka yang dibangkitkan, ada 3 angka yang berbeda, atau semua angkanya persis sama. Kok bisa? Itulah pertanyaan saya.



Kadang-kadang bisa, seringnya tidak bisa… Tapi, kalau memang sedang bisa, berarti kita tahu persis cara menghitung nilai k yang harus dipilih, yaitu dengan cara meng-xor semua 624 output yang kita prediksi. Sehingga ada dua kondisi yang harus dipenuhi:

1. 623 output cukup untuk memprediksi 624 output berikutnya secara akurat, sehingga nilai k bisa ditentukan.
2. Terdapat nilai n yang bisa memunculkan nilai k.

Berikut adalah solver lengkapnya:

a.py

```python
from pwn import *
# context.log_level = 'debug'

# conn = process(['python', 'chall.py'])
conn = remote('20.6.89.33', 8054)

import randcrack
from libnum import s2n

rc = randcrack.RandCrack()
rc.submit(1234567)

for _ in range(623):
    conn.recvuntil(b'>> ')
    conn.sendline(b'0')
    rec = int(conn.recvline().strip())
    rc.submit(rec)

kelinci = 0
rc.predict_getrandbits(32)
for i in range(624):
    kelinci ^= rc.predict_getrandbits(32)

from solve_n import find_n_from_k
n = find_n_from_k(kelinci)

conn.recvuntil(b'>> ')
```

```
conn.sendline(str(n).encode())
conn.interactive()
```

Jalanin berkali-kali aja sampai flagnya muncul. Soalnya peluangnya lebih kecil daripada punya author yang works exactly 50% of the time.

```
PS D:\Data\Downloads\CTFs\Penyisihan Jemastik 2025\nousagi> python a.py
[x] Opening connection to 20.6.89.33 on port 8054
[x] Opening connection to 20.6.89.33 on port 8054: Trying 20.6.89.33
[+] Opening connection to 20.6.89.33 on port 8054: Done
[*] Switching to interactive mode
2196259048
You found the rabbit!
GEMASTIK18{the_silksong_is_REALLL!!!!_https://www.youtube.com/watch?v=6X
GeJwsUP9c}
[*] Got EOF while reading in interactive
```

Flag

`GEMASTIK18{the_silksong_is_REALLL!!!!_https://www.youtube.com/watch?v=6XGeJwsUP9c}`