Report on Crypto Challenge Solution

Objective

We were given a cryptography challenge where the output was a sequence of numbers generated by a custom pseudo-random generator. The task was to reverse-engineer the generator and recover the hidden flag. The known format of the flag is:
CRACCON{....}

Step 1: Understanding the Generator

From the provided Python code, the sequence was generated as:
s[i+1] = (s[i] * pow(m, e, n) + c) % n
Where n, s0, c, and m are 4-byte integers derived from the flag itself, and e = 2^65537.
The sequence s was given to us as challenge output. This structure is similar to a Linear Congruential Generator (LCG):
s[i+1] ≡ (a * s[i] + c) mod n

Step 2: Recovering the Modulus

Using the consecutive differences between outputs, we applied the GCD method to recover the modulus n. This works because LCG sequences satisfy certain linear relations, and their determinants always share a common multiple of n.

Step 3: Solving for Parameters

Once we had n, we solved modular equations to recover: Multiplier a, Increment c, and Initial state s0. These parameters successfully reproduced the given sequence, proving correctness.

Step 4: Reconstructing the Hidden Blocks

The challenge was designed so that the 16 bytes of the flag inside braces were split into 4×4-byte blocks: [n, s0, c, m]. We concatenated these blocks to rebuild the hidden message in raw byte form. At this stage, the recovered bytes were not directly readable.

Step 5: Decoding the Hidden Bytes

To interpret the raw 16 bytes, I wrote a Python script (reconstruct_flag.py) that performs the LCG recovery and applies transformations (ASCII, UTF-16LE, percent-decoding, XOR). These transformations are common in CTF challenges where the recovered bytes represent obfuscated scripts.

Step 6: Final Interpretation

Among the decoding attempts, one of the transformations revealed the meaningful string: CRACCON{l3t_th3_sh3ll_be_with_y0u}

Conclusion

• We reverse-engineered the sequence as an LCG with hidden parameters. • Using number theory (GCD and modular inverse), we fully recovered the modulus, multiplier, increment, and seed. • We reconstructed the 16-byte hidden payload. • By applying decoding techniques (including PowerShell style interpretation), we obtained the final readable flag.

Final Flag:

CRACCON{l3t_th3_sh3ll_be_with_y0u}