R09921A10 鄭翔予
CTF Account: stanleymusic

1. nLFSR
   - Description
     - Get the RNG algorithm, try to guess the right output multiple times
   - Solution
     - The step() and random() function can be combined and rewrote as multiplication in GF(2^64) where reducing polynomial = 0x1fd07d87ee65cb055
       - 0x1fd07d87ee65cb055 = 2^{65}+ (little endian of 0xaa0d3a677e1be0bf in bits), '+' denotes concatenation
       - random() function can be represented as $\text{state} * \text{x}^{43}$
     - The new RNG algorithm
       - init_state = init_state * (x^43)
       - ret = init_state.integer_representation()
       - return (ret >> 63) & 1
     - The state is 64 bit, which means there are 64 variables. We need to construct 64 equations to solve the variable
     - Get consecutive 64 outputs from RNG. These outputs are generated by the same algorithm we rewrote, therefore we can solve the state by applying matrix multiplication on 64 equations
     - After running 64 steps from the init_state, the RNG we wrote locally has the same state as the RNG in the remote
       - Generate the next 300 output, and use it as the following input to make money > 2.4
   - Flag
     - FLAG{2iroO742LwA2ES1Cwewx}
   - Reference
     - https://hackmd.io/@cXpZn6ltSku4Vwx_OL0bqA/SyyxioFgI#winner-winner-chicken-dinner
2. Single
   - Description
     - Given three points A, B on ECC, generator G, and prime number p and ciphertext, try to get the plaintext
   - Solution
     - Solve the curve parameter a, b of ECC by given points A and B (two unknown parameters and two equations)
       - $\mathrm{A.y}^2 \equiv A.x^3 + a * A.x + b \pmod p$

- $B.y^2 \equiv B.x^3 + a * B.x + b \ (mod\ p)$
- Find out that $4 * a^3 + 27 * b^2 \equiv 0 \ (mod\ p)$, so it's a singular curve
- Determine whether it's node or cusp
  - It's node since the curve can be denoted as $(x - \alpha)^2 * (x - \beta)$
- Follow the formula taught, and reduce it to DLP on ($\mathbb{F}$p, x)
  - The DLP problem can be solved using the built-in library of sagemath
- Finally, we get the private key dA and the key of sha512, the rest is simple, just XOR each byte of ciphertext and the hash value generated by sha512, then we can get the original flag

- Flag
  - FLAG{adbffefdb46a99fad0042dd3c10fdc414fadd25c}
- Reference
  - https://ctftime.org/writeup/12563

3. HNP-revenge

- Description
  - Given the ECDSA algorithm, try to get the right signature of "Kuruwa"
- Solution
  - The highest 128 bits of the ephemeral key k are fixed, known value
    - int(md5(b'secret').hexdigest()
  - Construct lattice basis and use LLL to solve the two ephemeral keys
    - Since the ephemeral key k1 and k2 are now denoted by (fixed_prefix + k1') and (fixed_prefix+k2'), the value of u is different from the lecture
    - $a = \text{fixed\_prefix} \ll 128$
    - $u = s_1^{-1} * r_1 * h_2 * r_2^{-1} - s_1^{-1} h_1 + a - s_1^{-1} * s_2 * r_1 * r_2^{-1} * a$
    - k1 and k2 can be found where one of LLL's output row = (-k1, k2, K), K = upper bound = 2^128
  - And then the private key d can be retrieved
    - $d = (s_1 * k_1 - h_1) * \ r_1^{-1} \ mod\ n = \ (s_2 * k_2 - h_2) * \ r_2^{-1} \ mod\ n$
  - The solution of LLL is not always determined but still has a high probability to be correct. If the calculated point P by d*G is not equivalent to the value received from the server, reconnect and try it again. Try a few more times to get the right d
  - Sign the input "Kuruwa" locally with the correct private key to get the corresponding r and s
- Flag
  - FLAG{adfc9b68bd6ec6dbf6b3c9ddd46aafaea06a97ee}