3 Challenge 3: Faulty One-Time Pad Distinguishing Attack

3.1 Problem Understanding

This encryption scheme is faulty because the key space is reduced from what it should be to behave as a perfectly secure encryption. We are only using the keys that do not have the 0 bit, thus it is only [0x01, 0xff], and not [0x00, 0xff] which is secure.
Since Bob doesn't use 0x00, the **ciphertext** byte can never be equal to the **plaintext** byte

3.2 Approach

Due to the above mentioned flaw, this scheme is vulnerable to this type of attack:
We can distinguish between encryptions of a known plaintext vs encryption of a random message by checking if any **ciphertext byte equals the corresponding plaintext byte:**

- If yes, $\rightarrow$ **normal OTP is possible**, but Bob's scheme never produces that as the key doesn't contain 0x00.
- So, if we pick our plaintext as **all zero bytes** then:
  ciphertext byte $=$ *0x00* $\oplus$ **keybyte** $\neq$ *0x00*

So, ciphertext bytes produced using this scheme will always be from **0x01** to **0xff** and **never zero**. But if the ciphertext is an encryption of a **random message**, then **cipher bytes** can be anything, including **zero**.

So, our payload should be a large string of 0's and the logic for deciding whether to send c1 or c2 to the server, will be based on choosing the one without 0x00. We are choosing a large string of zero's to ensure that we reduce the possibility of getting cipher texts such that, both c1 and c2 don't have 0x00. (As the cipher text generated from a random message will have lesser probability that it will not contain a 0x00 byte if the payload length is longer)

3.3 Output and Flag

When I have chosen the payload to be: "00"*1024, it only worked till Level 81 in the server, so then I chose it to be 2048 instead of 1024, to make the probability that both c1 and c2 don't have a 0x00 to be even lower.

`Flag: cs409{y0u_h4d_fu11_4dv4nt4g3}`

3.4 Implementation

```
## TODO 1 ##
payload = "00" * 2048  # 2048 zero-bytes

## TODO 2 ##
b1 = bytes.fromhex(c1)
b2 = bytes.fromhex(c2)

has0_1 = (b'\x00' in b1)
has0_2 = (b'\x00' in b2)

# If one contains 0x00 and the other doesn't, choosing the one without
   0x00
if has0_1 and (not has0_2):
    guess = 2
elif has0_2 and (not has0_1):
    guess = 1
else: # Highly unlikely as we are taking a very large string as payload
    guess = 1

sendline(f"c{guess}")
```