2 Challenge 2: One-Time Pad over Groups

2.1 Problem Understanding

In this question, the OTP scheme is implemented using modulo 128 operation instead of XOR. The encryption is performed by adding each plaintext byte to the corresponding key byte modulo 128. Its flaw is in the fact that, instead of using a truly random key, it starts from a random bit and then deterministically expands it using SHA-26, ensuring key length matches plaintext length.

2.2 Given Data

- `ciphertext.enc`
- `encrypt.py`

2.3 Approach

Now since we know that the encryption algorithm deterministically generates the entire key just from the starting byte, which is generated by random. We can just brute force our way as there are only 128 possible first bits. For each case we can generate a key and then try to decrypt the message by reversing the modulo 128 operation (subtraction instead of addition). Then we can find the plausible plaintext using the fact that the flag contains a { and } in it.

2.4 Implementation

```python
import hashlib

def group_sub(m1: bytes, m2: bytes) -> bytes:
    assert len(m1) == len(m2)
    res = b""
    for i in range(len(m1)):
        res += chr((m2[i] - m1[i]) % 128).encode()
    return res

def generate_key(start_byte: int, length: int) -> bytes:
    key = chr(start_byte).encode()
    for _ in range(1, length):
        key += chr(hashlib.sha256(key).digest()[0] % 128).encode()
    return key

with open("ciphertext.enc", "rb") as f:
    ciphertext = f.read()

# A brute force which checks all possible 128 combinations of the key (
    because it can be generated using the first byte)
# Then comparing it with the flag format, which must contain { and }
for start_byte in range(128):
    key = generate_key(start_byte, len(ciphertext))
    plaintext = group_sub(key, ciphertext)
    if b"{" in plaintext and b"}" in plaintext:
        print(f"[+] Found plausible flag with start byte {start_byte}:
            {plaintext}")

# Hence the flag is: cs409{algebra_enters_the_picture!}
```

2.5 Output and Flag

```
[+] Found plausible flag with start byte 5: b'cs409{algebra_enters_the_picture!}'
[+] Found plausible flag with start byte 48: b'8{A[pQ\x18\x0f\x00fUk,{==\\54sxm(9\x07}$l\x1
[+] Found plausible flag with start byte 59: b'-\x19\x1ad\x16{dOoiN\nPwJ\x04\x14\x08/S}2G\x
[+] Found plausible flag with start byte 69: b"#1PaiG{W2OAU:\x05hBQ_c\x0c'(\x19y\x12\x18}tF
[+] Found plausible flag with start byte 78: b'\x1aN\x03s}04]5,CMk\x1bd:{zQfU\x08-\x1d\x14q
```

```
Flag: cs409{algebra_enters_the_picture!}
```