INCOGNITO CTF Crypto Challenge Write-up

문제 개요: XOR와 ASE의 Encryption 및 Decryption 알고리즘 이해를 바탕으로 Decrypt code를 완성하여 random key 값을 찾아내고 최종적으로 flag 값을 완성한다.

문제 설명:

주어진 flag.bin 파일을 열어서 Encrypt flag를 출력하면 다음과 같은 값이 나온다.

```
b'₩x88₩xae₩x97B₩x94₩xcf₩x95₩xe8₩x98₩x8bS₩xf8₩xf1₩xb3₩xcajl₩xb9₩xe7₩xba₩xda₩xc57
AP₩x18z₩xf5E₩xda1'
```

main 함수를 보면 암호화된 flag를 AES Decryption을 한 후, XOR Decryption을 하여 최종적인 flag 값을 얻을 수 있다.

먼저 XOR와 AES256 class의 Decrypt 함수를 작성해야 한다.

XOR의 경우, 암호화 복호화 과정이 같다. 그렇기 때문에 다음과 같이 함수를 완성 수 있다.

```
def decrypt(self, data: bytes) -> bytes:
        return self.encrypt(data)
```

AES256은 코드를 보면 pycrypto module을 사용하여 암호화를 진행하고 있기 때문에 Encrypt method와 같은 방식으로 코드를 작성하면 된다. 이때 주의할 점은 flag 값 32byte에서 flag 값 길이를 뺀 만큼 padding 값을 붙였으므로 복호화한 값에서 padding을 제거하여 반환해야 한다. 그리고 flag의 길이는 주석에 나와있는 힌트대로 23byte이다. 최종적으로 다음과 같이 함수를 완성할 수 있다.

```
def decrypt(self, enc):
        msg = self.crypto.decrypt(enc)
        return msg[:23]
```

함수를 완성하고 코드를 실행하여 AES decrypt flag를 출력하면 다음과 같이 나온다

```
b'5a65748a83771854b781635f68b68d704852f08b320a4a'
```

현재 XOR class의 key 값은 5byte 길이의 랜덤 바이트 값이다. 즉, 우리는 이 5byte 랜덤 키 값을

구해서 XOR 복호화를 진행해야 한다. 이때 flag의 시작값이 'INCO{'란 것을 알고 XOR의 특성을 이용하면 아래와 같이 key값을 구할 수 있다.

INCO{ (text) → 494e434f7b (hex)

49 XOR 5a → 13

4e XOR 65 → 2b

43 XOR 74 → 37

4f XOR 8a → c5

7b XOR 83 → f8

최종적인 key의 hex 값은 '132b37c5f8'이다. key 값이 현재 byte 형태로 저장되기 때문에 XOR class의 init method에서 key는 다음과 같이 설정한다.

```
self.key = binascii.unhexlify("132b37c5f8")
```

이 코드를 실행하게 되면 decrypt flag를 얻을 수 있다.

```
decrypt flag: b'INCO{d3crypt_succe5s!!}'
```

flag = INCO{d3crypt_succe5s!!}

```
import
os
          import binascii
          from Crypto.Cipher import AES

          class XOR:
              def __init__(self):
                  self.key = os.urandom(5)
                  #print("key = ", binascii.b2a_hex(self.key))
                  self.key = binascii.unhexlify("132b37c5f8")
                  #print("length of key = ", len(self.key))
              def encrypt(self, data: bytes) -> bytes:
                  enc = b''
                  for i in range(len(data)):
                      enc += bytes([data[i] ^ self.key[i % 5]])
                  return enc
              def decrypt(self, data: bytes) -> bytes:
                  #complete this method
```

```
        return self.encrypt(data)

class AES256:
    def __init__(self):
        iv = b'\x00'*16
        self.key = b'12345678901234567890123456789012'
        self.crypto = AES.new(self.key, AES.MODE_CBC, iv)
    def encrypt(self, msg):
        msg = msg + b'\x00'*(32-len(msg))
        cipher = self.crypto.encrypt(msg)
        return cipher
    def decrypt(self, enc):
        #complete this method
        #flag length is 23-byte
        msg = self.crypto.decrypt(enc)
        return msg[:23]

def main():
    pwd = os.getcwd()
    flag = open(pwd+'/desktop/hgy/INCOGNITO/ctf/flag.bin', 'rb').read().strip()
    #print(flag)
    #flag =
binascii.unhexlify("494e434f7b6433637279707 45f737563636535732121 7d")
    #print(len(flag)) #23-byte
    #xore = XOR()
    #xore_flag = xore.encrypt(flag)
    #aese = AES256()
    #encrypt_flag = aese.encrypt(xore_flag)

    #fw = open('output.txt', 'w')
    #fw.write("encrypt flag: ", encrypt_flag)
    #fw.close()
    print(">>>>>encrypt flag: ", flag)

    xord = XOR()
    aesd = AES256()
    decrypt_flag = aesd.decrypt(flag)
    #print("only aes decrypt: ", decrypt_flag)
    print("only aes decrypt: ", binascii.b2a_hex(decrypt_flag))
    xord_flag = xord.decrypt(decrypt_flag)
    print ('>>>>>decrypt flag:', xord_flag)

if __name__ == '__main__':
    main()
```