Challenge: Initialization

Challenge Description :

During a cyber security audit of your government infrastructure, you discover log entries showing traffic directed towards an IP address within the enemy territory of Oumara. This alarming revelation triggers suspicion of a mole within Lusons government. Determined to unveil the truth, you analyze the encryption scheme with the goal of breaking it and decrypting the suspicious communication. Your objective is to extract vital information and gather intelligence, ultimately protecting your nation from potential threats.

Context :

- We are given a questionnaire, based upon a bash_history.txt file and a sshd.log file. You will need to go through both and answer the question based on the log files.

Flag :

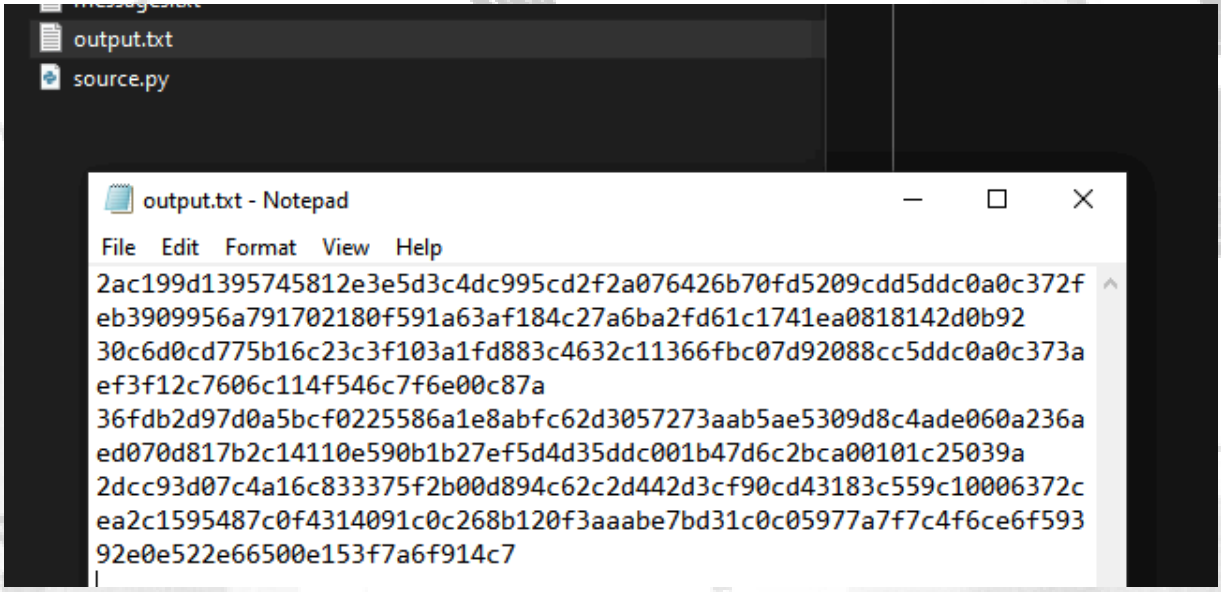
- First, examine the provided Python script used for encrypting the flag. The script reads messages from a file, encrypts them using AES in CTR mode, and writes the ciphertexts to an output file. Here is the source code:

- The script begins by reading messages from messages.txt and storing them in MSG. The main function is then executed.

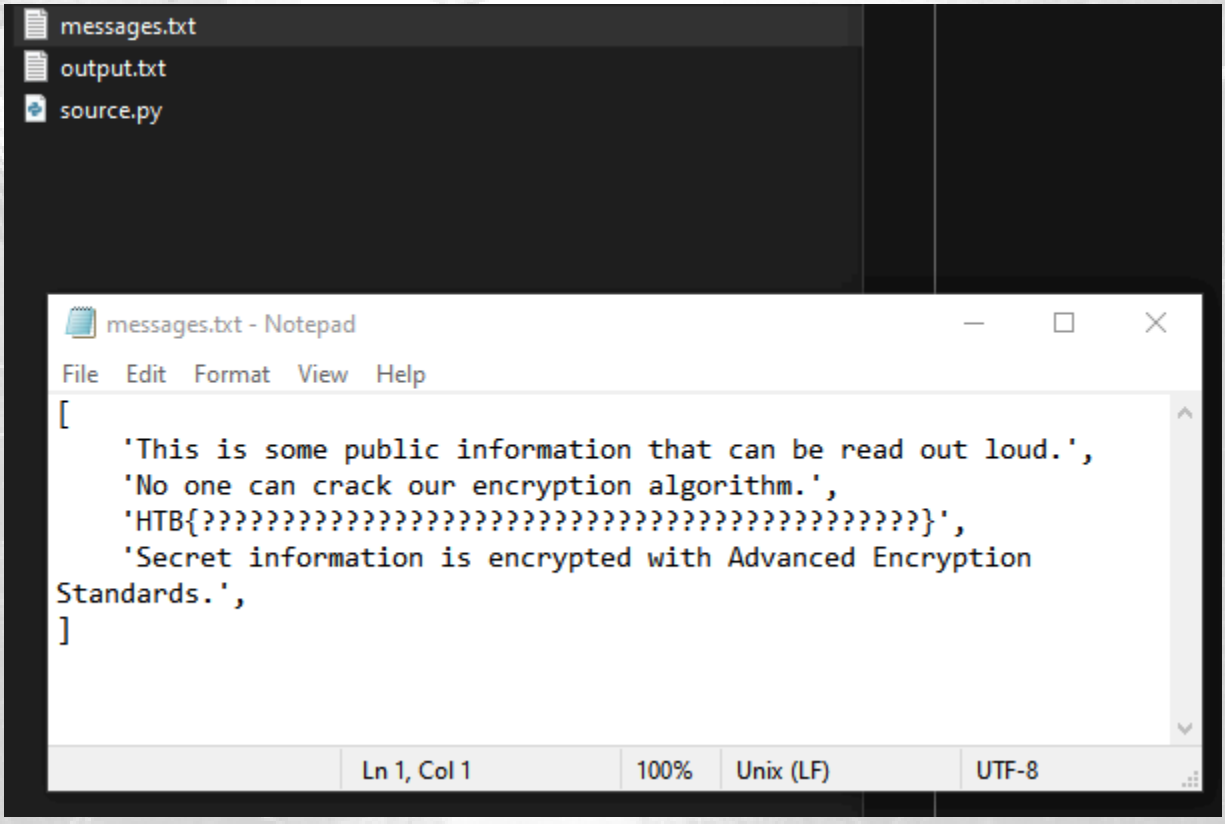
```python
class AdvancedEncryption:
    def __init__(self, block_size):
        self.KEYS = self.generate_encryption_keys()
        self.CTRs = [Counter.new(block_size) for i in range(len(MSG))]

    def generate_encryption_keys(self):
        keys = [[b'\x00']*16] * len(MSG)
        for i in range(len(keys)):
            for j in range(len(keys[i])):
                keys[i][j] = os.urandom(1)
        return keys

    def encrypt(self, i, msg):
        key = b''.join(self.KEYS[i])
        ctr = self.CTRs[i]
        cipher = AES.new(key, AES.MODE_CTR, counter=ctr)
        return cipher.encrypt(pad(msg.encode(), 16))

def main():
    AE = AdvancedEncryption(128)
    with open('output.txt', 'w') as f:
        for i in range(len(MSG)):
            ct = AE.encrypt(i, MSG[i])
            f.write(ct.hex()+'\n')

if __name__ == '__main__':
    with open('messages.txt') as f:
        MSG = eval(f.read())
    main()
```

- The main function initializes an instance of AdvancedEncryption, encrypts each message, and writes the ciphertexts to output.txt

Messages.txt



Output.txt



- The AdvancedEncryption class utilizes AES in CTR mode. This mode is essentially a stream cipher, where plaintext is XORed with a keystream generated from the AES cipher in CTR mode:
- The script intends to use different AES keys for each message, but there is a critical flaw in the key generation.

```
def generate_encryption_keys(self):
    keys = [[b'\x00']*16] * len(MSG)
    for i in range(len(keys)):
        for j in range(len(keys[i])):
            keys[i][j] = os.urandom(1)
    return keys
```

- The line keys = [[b'\x00']*16] * len(MSG) generates multiple references to the same key, resulting in all keys being identical:
- Therefore, all ciphertexts use the same keystream. Knowing one plaintext-ciphertext pair allows us to recover the keystream and decrypt other messages.
- Given known plaintexts and their ciphertexts, we can recover the keystream and decrypt the fl
- We can easily reverse this with the given key and the plaintext in the output.txt and the messages.txt files. We can recover the keysteam and decipher it to get the flag.

```
itialization/crypto_initialization$ python3 Initialization.py
b'HTB{d4mn_th3s3_ins3cur3_bl0ckch41n_p4r4m3t3rs!!!!}\x0e\x0e\x0e
\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\x0e\xa7\x1d\x0ej\xfdK\x
cf\xcfv\xe4b\xf3\xde\x1c\xd9l'
```

The Flag : HTB{d4mn_th3s3_ins3cur3_bl0ckch41n_p4r4m3t3rs!!!!}