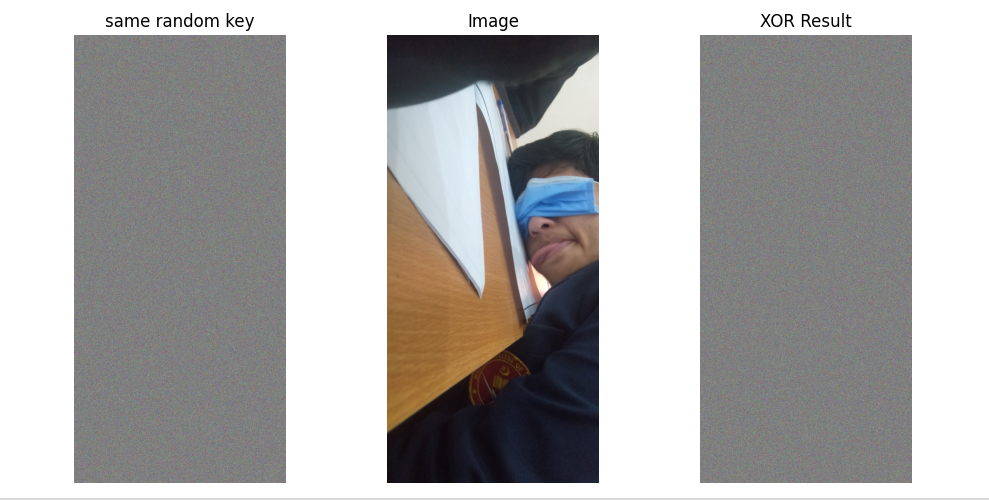
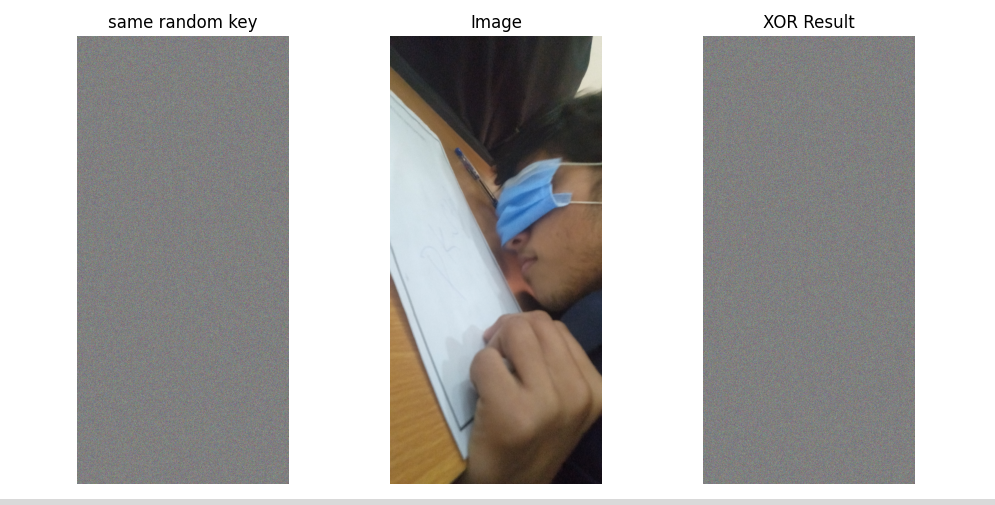
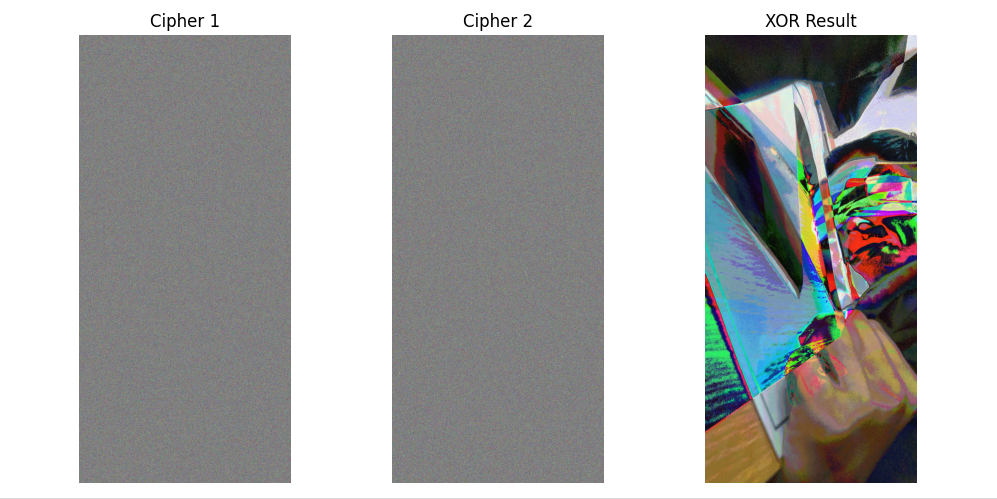
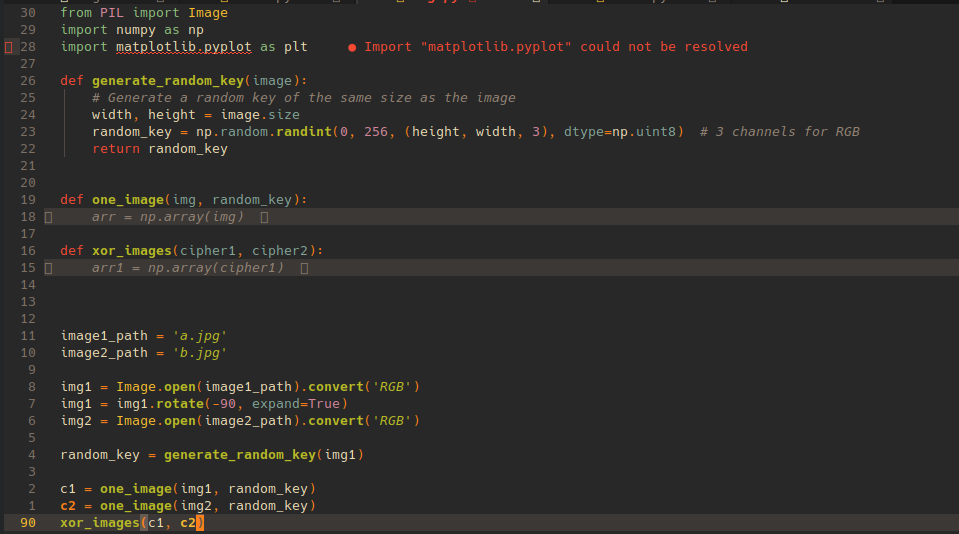
**Question 1:**







**Code:** [full implemetation: https://github.com/muhammadawa242/foc\_asg1/blob/main/q1/img.py]

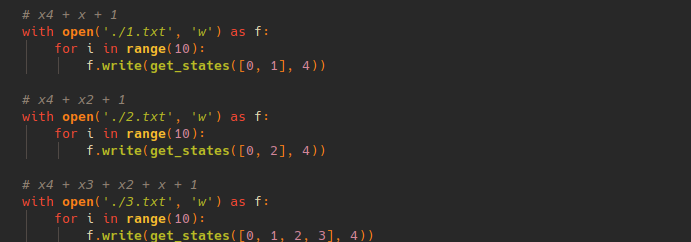
****

**Question 2:**

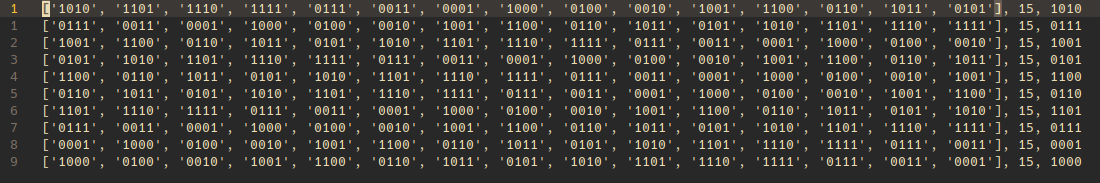
**(2.8)**

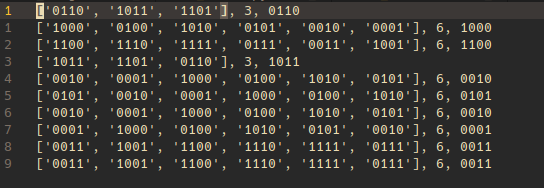
**Code:**

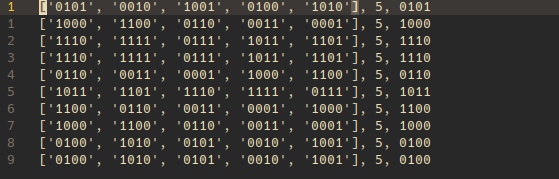




The code when run multiple times generates following pattern of text files:





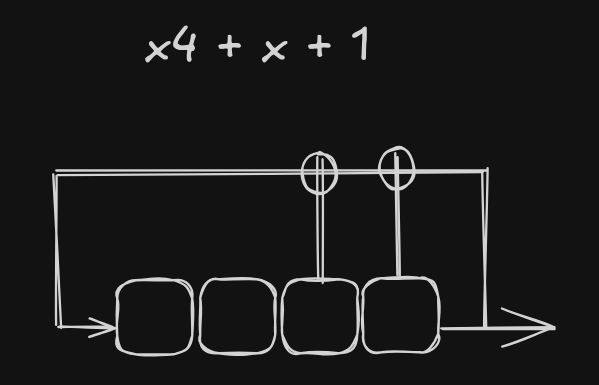


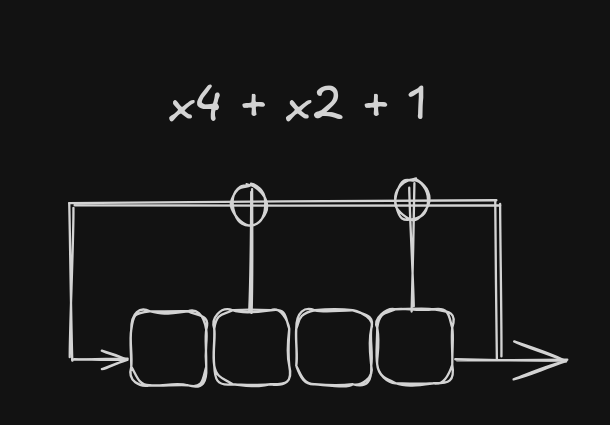
Based on the data, it is safe to assume the following:

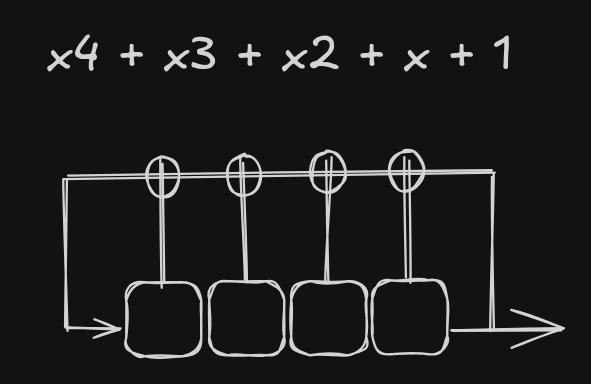
**1.** x4 + x + 1 (Primitive since all sequences are maximum length)

**2.** x4 + x2 + 1 (Reducible since its length changes with initial value)

**3.** x4 + x3 + x2 + x + 1 (Irreducible but not polynomial since length is constant but not maximum)







**(2.9)**

**(a)**

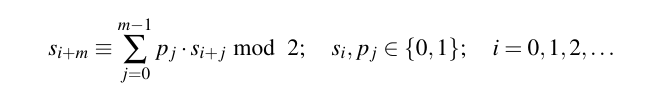
If the LFSR has a degree m, it can generate a maximum of 2m−1 unique states. For a maximum-length sequence generated by the LFSR, all states will be utilized in the keystream. In order to launch a successful attack, we need 2m ciphertext/plaintext bit pairs.

**(b)**

An attack is possible if the attacker knows the plaintext and the corresponding ciphertext. With the 2m pairs of plaintext and ciphertext bits, attacker can reconstruct the first 2m key stream bits:

si = xi (xor) yi (i = 0, 1, . . . , 2m − 1)

The goal is now to find the key which is given by the feedback coefficients pi using:



We get a different equation for every value of i. Moreover, the equations are linearly independent. With this knowledge, attacker can generate 256 equations for the first 256 values of i:

i = 0, s256 ≡ p255s255 + . . . + p1 s1 + p0 s0 mod 2

i = 1, s257 ≡ p255s256 + . . . + p1 s2 + p0 s1 mod 2

.

.

.

i = 255, s511 ≡ p255s510 + . . . + p1 s256 + p0 s255 mod 2

We have now 256 linear equations in 256 unknowns p0 , p1, . . . , p255 . This system can

easily be solved using Gaussian elimination, matrix inversion or any other algorithm for solving systems of linear equations.

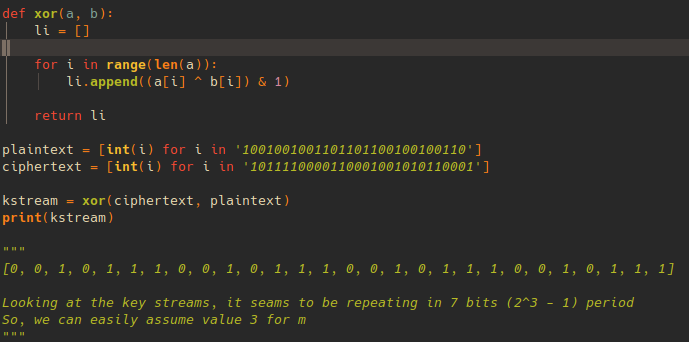
Once computed these feedback coefficients, one can build the LFSR and load it with any 256 consecutive output bits that they already knows and clock the LFSR and produce the entire output sequence.

**(c)**

If we use an LFSR as a stream cipher, the secret key k is the feedback coefficient vector (p255 , . . . , p1 , p0 ). If the initial state is known or can be easily guessed (especially if it is derived from a predictable source), the entire keystream can be reconstructed. This makes the cipher vulnerable to attacks, as an adversary could potentially generate the keystream without needing to know the full key. The same plaintext would always encrypt to the same ciphertext if the initial state is reused. This makes the system susceptible to replay attacks, where an attacker could capture and replay valid ciphertexts, leading to potential decryption of the original plaintext.

**(2.10)**

**(a)**



**(b)**

The initial state can be retrieved by reversing the first 3 bits input as these are the first outputs in the key stream starting from the leftmost bit. Hence the reversal.

Initial vector ≡ 100

**(c)**

i = 0, s3 ≡ p2s2 + p1s1 + p0s0 mod 2

i = 1, s4 ≡ p2s3 + p1s2 + p0s1 mod 2

i = 2, s5 ≡ p2s4 + p1s3 + p0s2 mod 2

Substituting the values for :  
**Equation 1 (for i=0)**

0≡p2⋅1+p1⋅0+p0⋅0mod2

This simplifies to:

0≡p2mod2

Thus, p2≡0.

#### Equation 2 (for i=1)

1≡p2⋅0+p1⋅1+p0⋅0mod2

This simplifies to:

1≡p1mod2

Thus, p1≡1.

#### Equation 3 (for i=2)

1≡p2⋅1+p1⋅0+p0⋅1mod2

This simplifies to:

1≡0⋅1+1⋅0+p0⋅1mod2

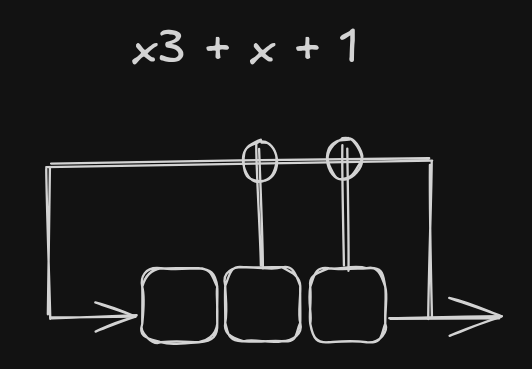
This further simplifies to:

1≡p0mod2

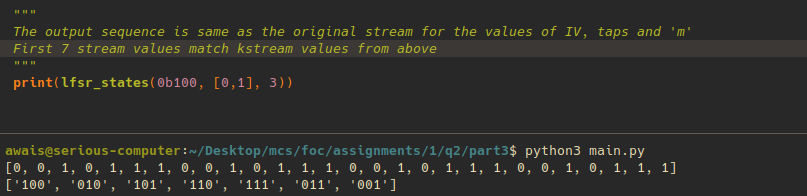
Thus, p0≡1.

=> p0, p1, p2 = (1, 1, 0)

**(d)**

****

**Comparing last bits of states generated from LFSR using the values:**

****

**Question 3:**

**(a)**

**Location:** (<https://github.com/muhammadawa242/foc_asg1/blob/main/q3/triv.txt>)

**Summary**

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

The minimum pass rate for each statistical test with the exception of the

random excursion (variant) test is approximately = 18 for a

sample size = 20 binary sequences.

The minimum pass rate for the random excursion (variant) test

is approximately = 13 for a sample size = 15 binary sequences.

For further guidelines construct a probability table using the MAPLE program

provided in the addendum section of the documentation.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**(b)**

**Location:** (<https://github.com/muhammadawa242/foc_asg1/blob/main/q3/zipfile.txt>)

**Summary**

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

The minimum pass rate for each statistical test with the exception of the

random excursion (variant) test is approximately = 0 for a

sample size = 1 binary sequences.

The minimum pass rate for the random excursion (variant) test is undefined.

For further guidelines construct a probability table using the MAPLE program

provided in the addendum section of the documentation.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

**(c)**

**Location:** (<https://github.com/muhammadawa242/foc_asg1/blob/main/q3/eng.txt>)

**Summary**

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

The minimum pass rate for each statistical test with the exception of the

random excursion (variant) test is approximately = 18 for a

sample size = 20 binary sequences.

The minimum pass rate for the random excursion (variant) test is undefined.

For further guidelines construct a probability table using the MAPLE program

provided in the addendum section of the documentation.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

20 streams of 1 million bits produced through trivim cipher (<https://github.com/cbouilla/trivium>)

does a lot better compared to other 2 since it the output stream from trivium cipher is supposed to be random statistically. When run on zip file, the test still shows randomness. However, the compression process aims to reduce the entropy of the data, meaning that the resulting binary output has a lower level of unpredictability than truly random data. For our particular english text file, in order to adhere to the 20 streams of 1million bits rule, the story was trimmed from the end after converting ASCII to streams of bits. The starting text contained lots of white space and ended up being the least random file on the test. The code for conversion of Ascii text to binary text file with 20 streams of 1000000 bits sits at (<https://github.com/muhammadawa242/foc_asg1/tree/main/q3/inputs/eng_txt>)

**Question 4:**

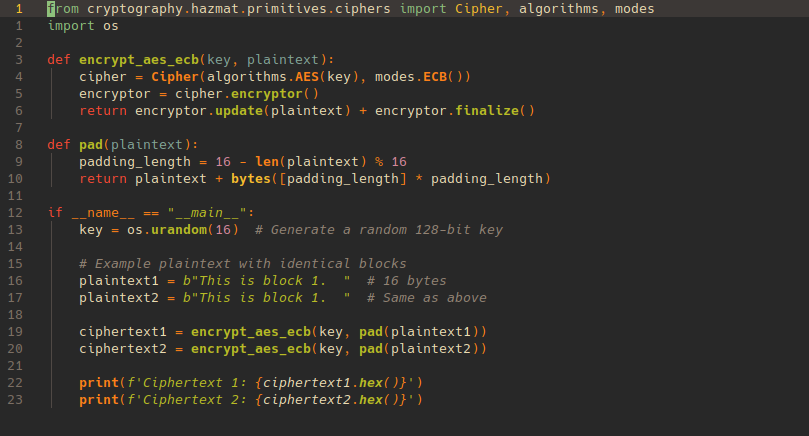
**Question 5:**

AES-128-ECB (Electronic Codebook) mode has notable drawbacks, primarily related to its handling of data patterns.

### Drawbacks of AES-128-ECB

1. **Lack of Diffusion**:
   * ECB encrypts identical plaintext blocks into identical ciphertext blocks. This means that patterns in the plaintext are preserved in the ciphertext, which can reveal information about the data.
2. **Vulnerability to Frequency Analysis**:
   * Since the same plaintext block always encrypts to the same ciphertext block, attackers can analyze the frequency of blocks and infer information about the content.
3. **No IV or Randomization**:
   * ECB does not use an initialization vector (IV), which means every time the same data is encrypted, the output will be identical, making it less secure than other modes that use randomness.

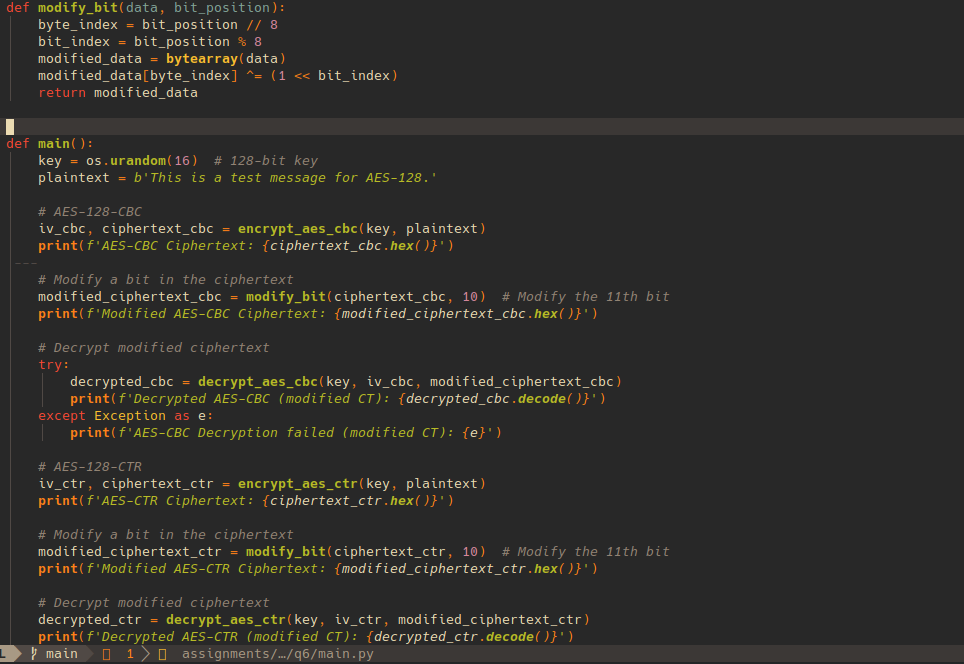
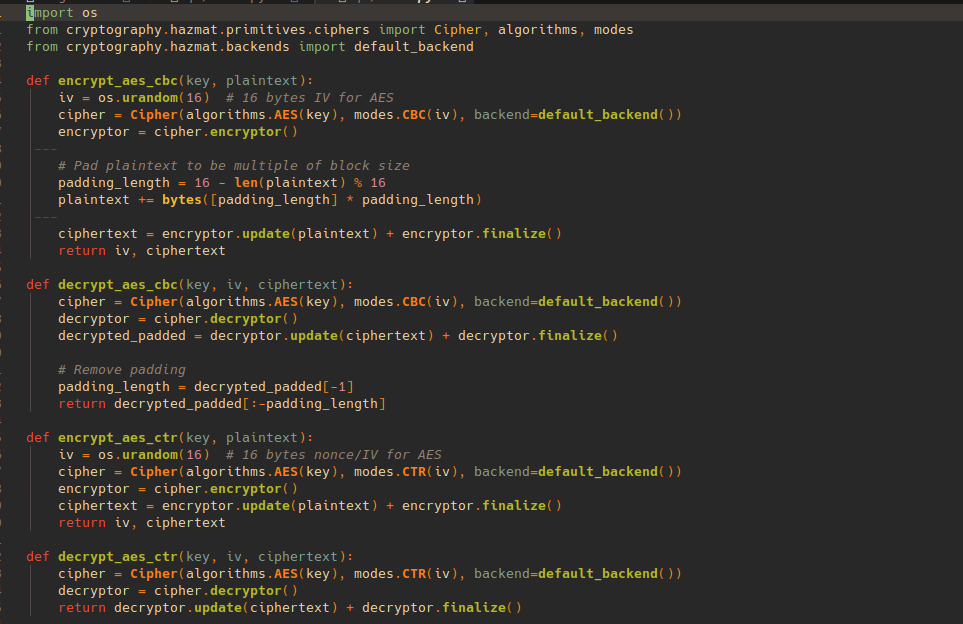
**Code:**



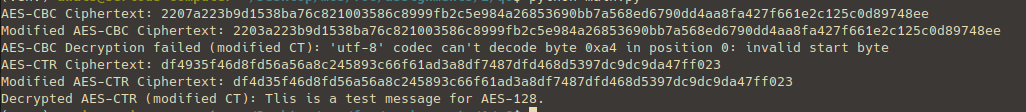
**Output ciphers are same for same plaintext:**



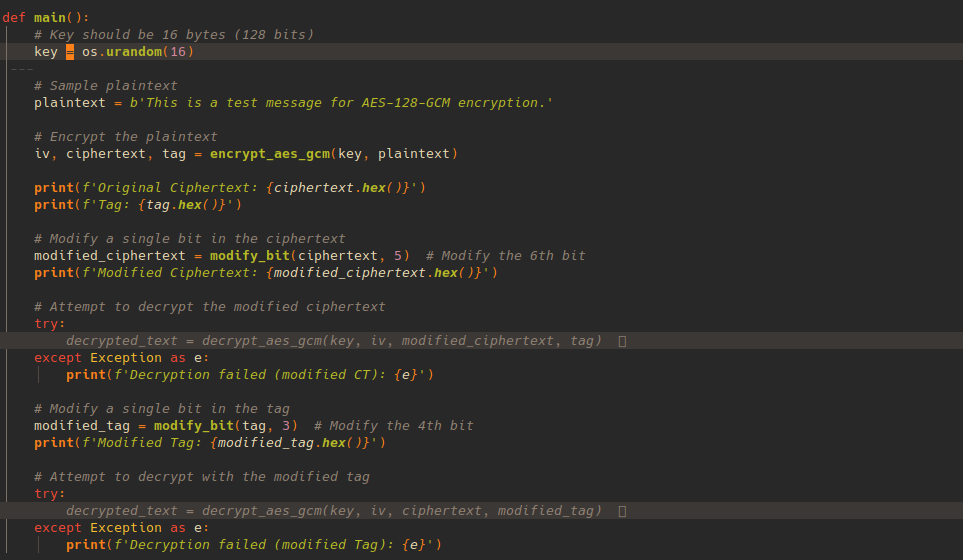
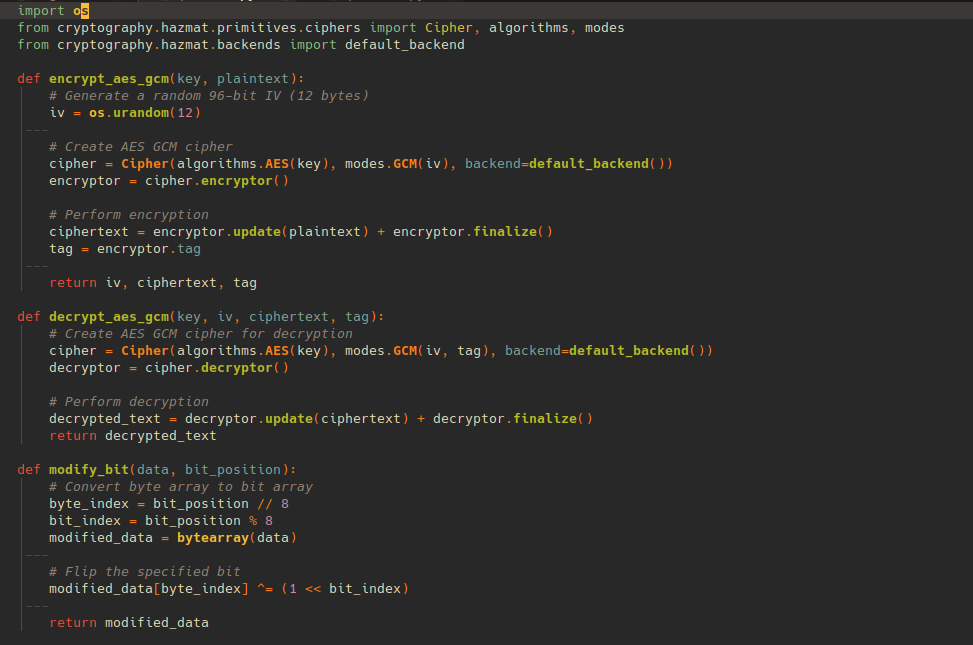
**Question 6:**



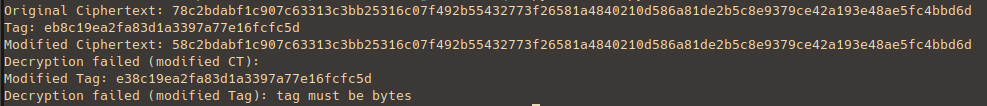
**Output:**



**Question 7:**



**Output:**

****