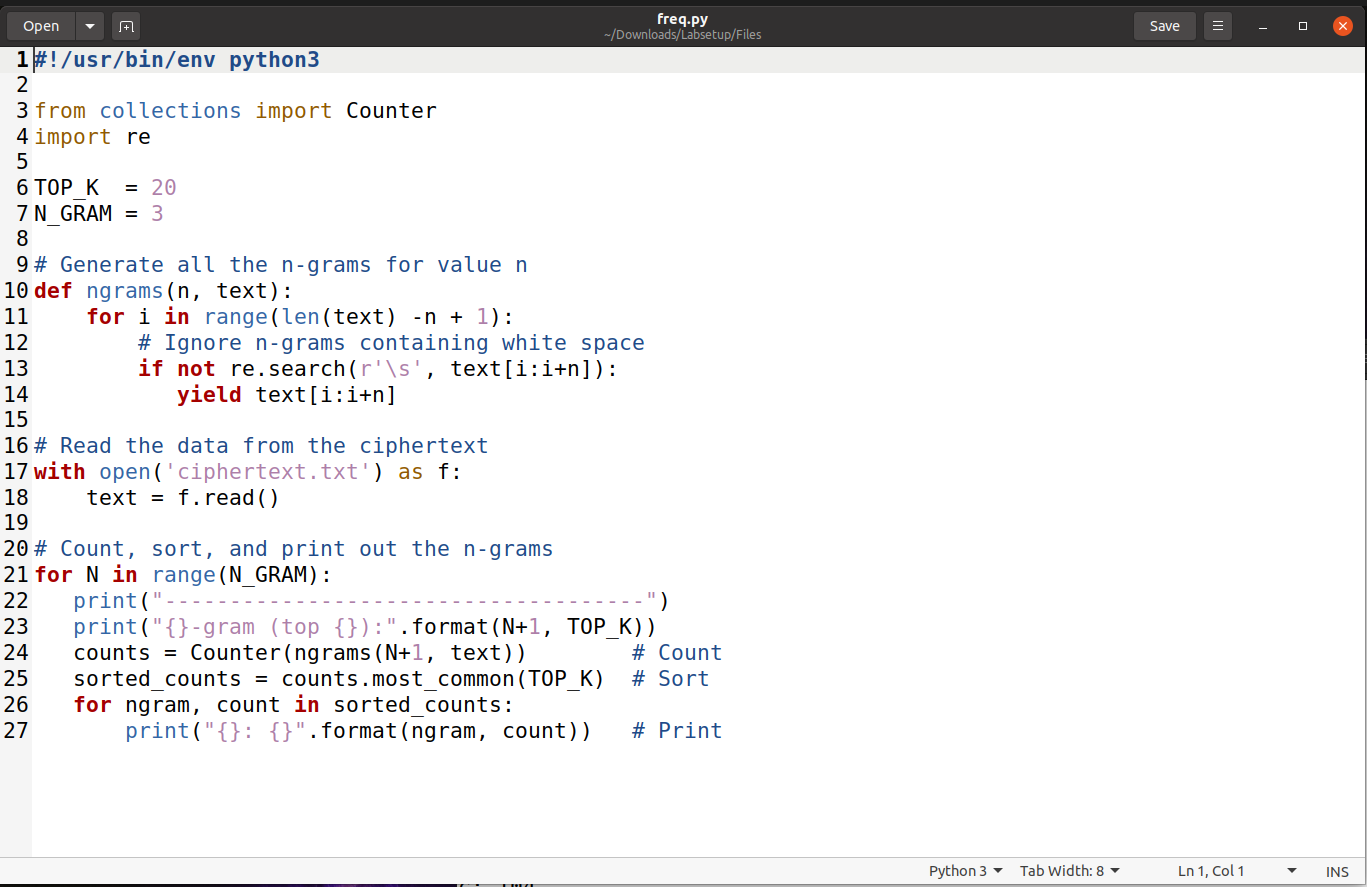
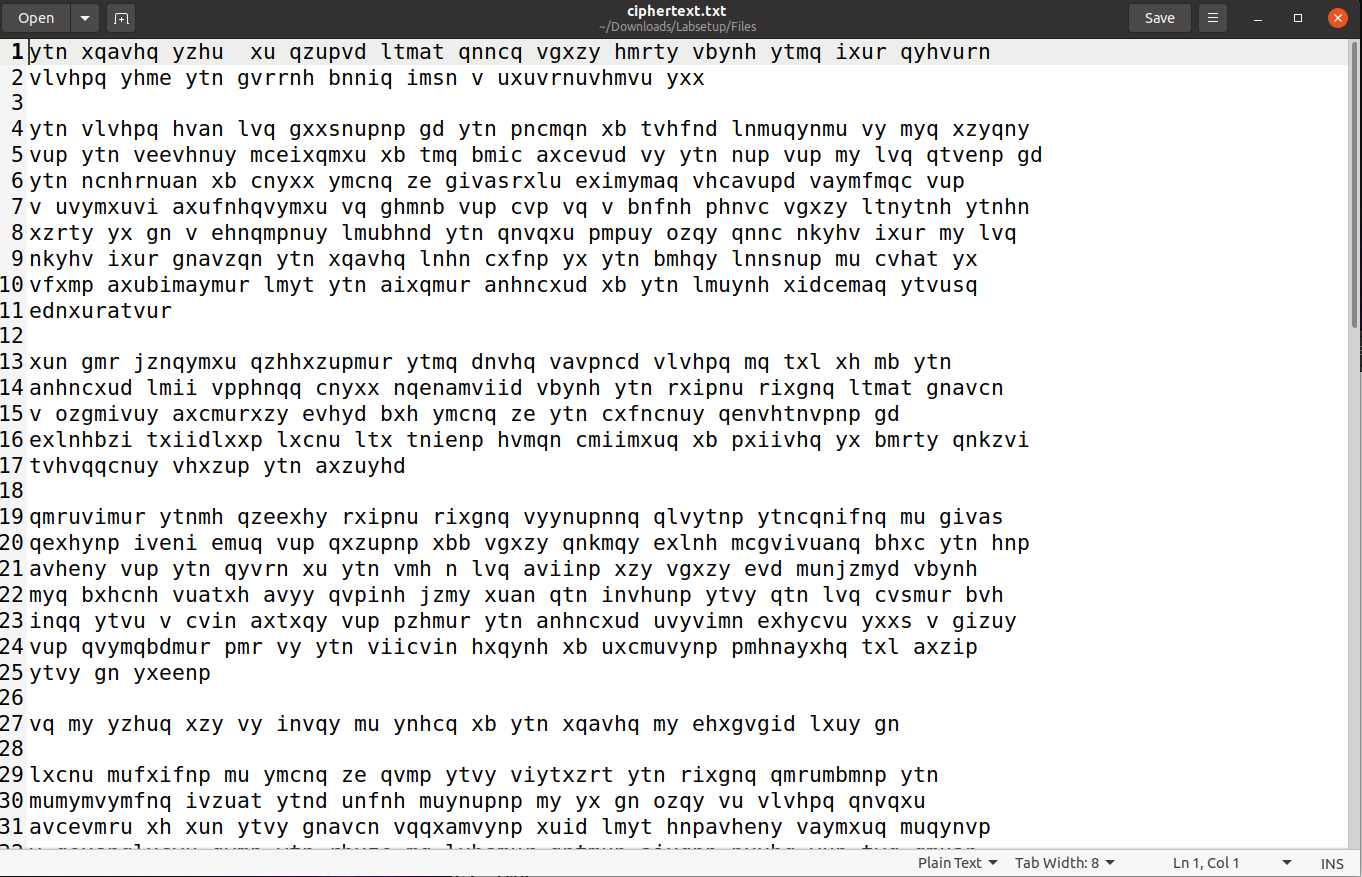
**Task 1:**

Code:





Output:

**A screenshot of a computer

Description automatically generated**

**A white background with black lines

Description automatically generated**

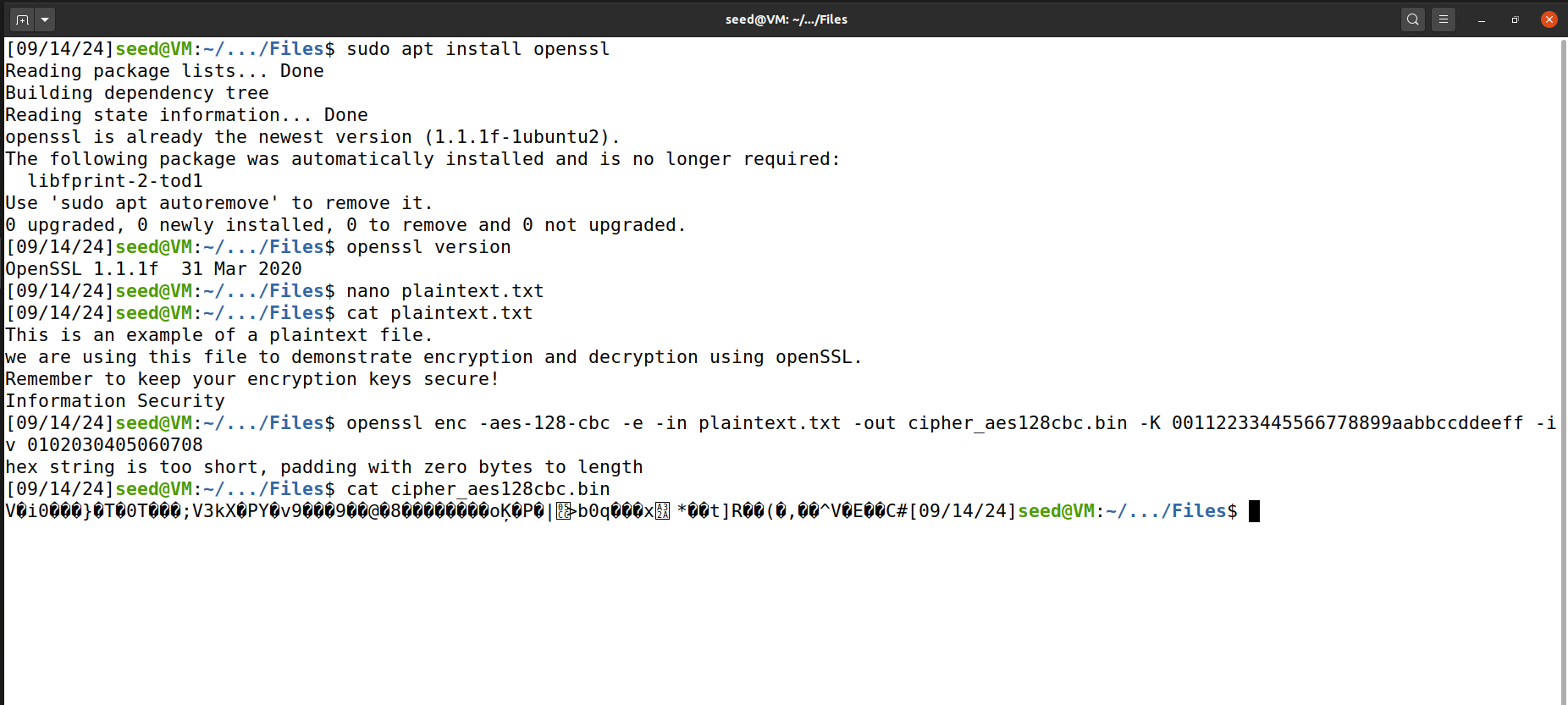
**A white background with black and white clouds

Description automatically generated**

**Task 2:**

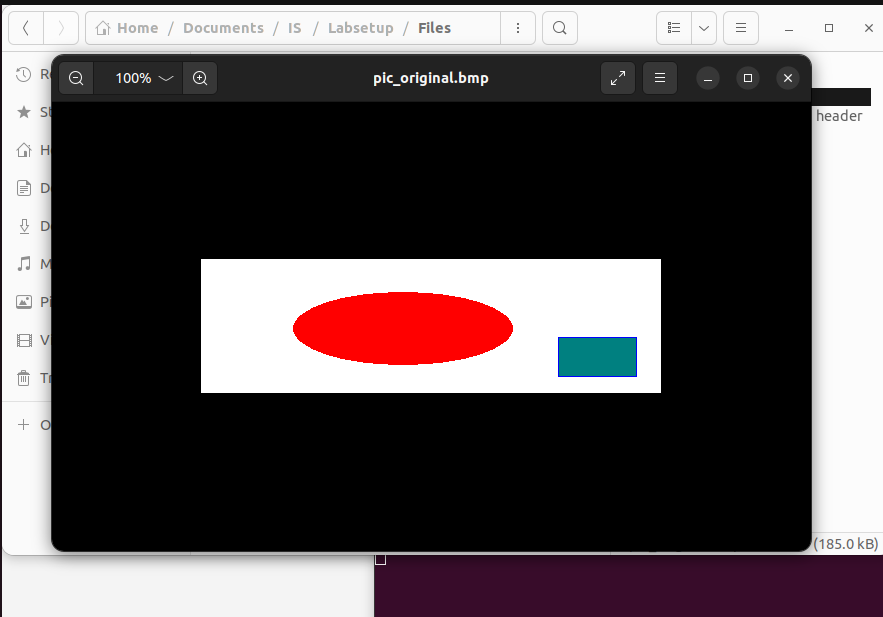
The process of converting plaintext into ciphertext is achieved through the use of the AES-128 encryption algorithm in Cipher Block Chaining (CBC) mode. This algorithm uses a 16-byte encryption key, represented as 00112233445566778889aabbccddeeff, along with an 8-byte initialization vector (IV) 0102030405060708 to ensure the encryption is secure and unique for each session.

Linux Commands:

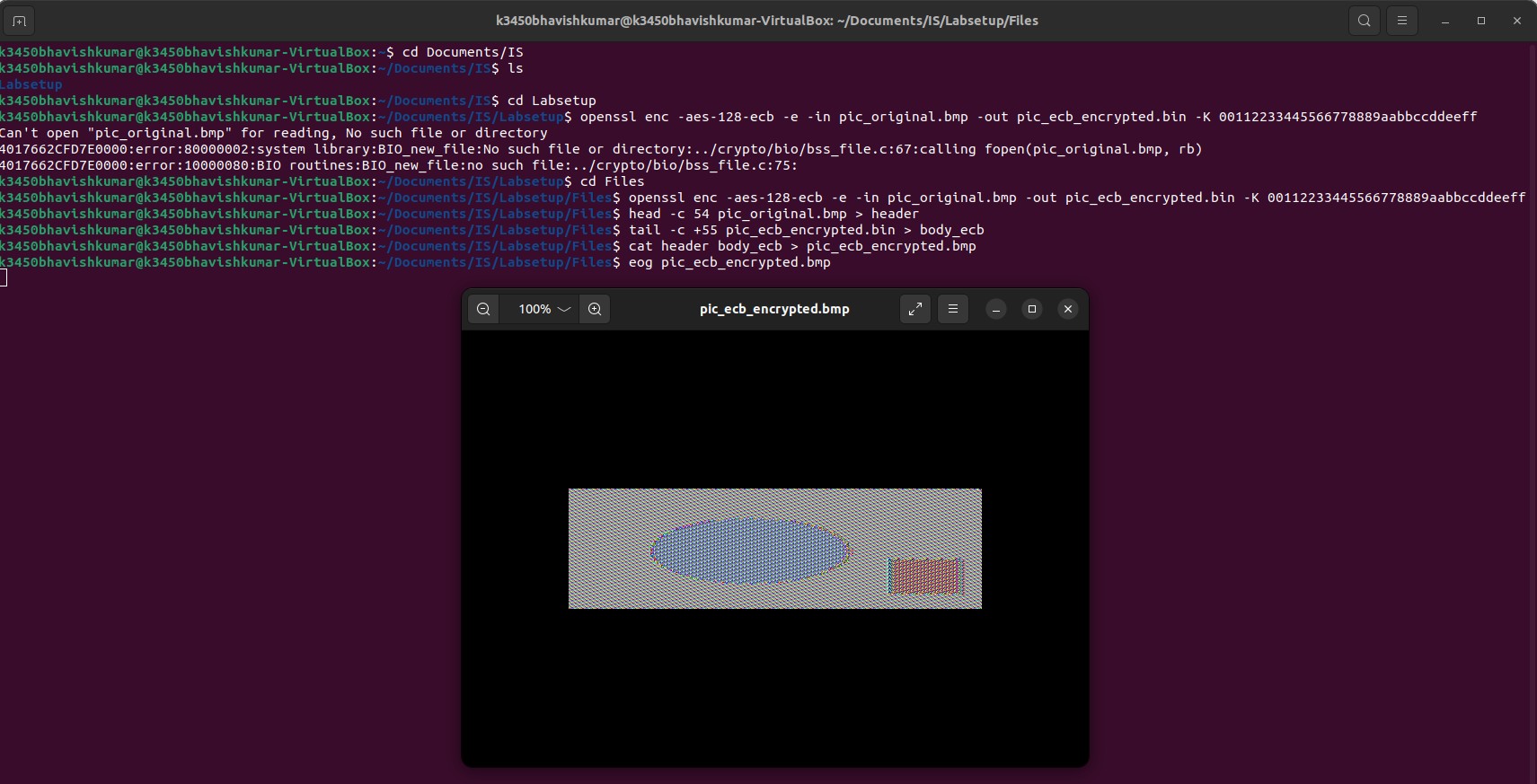


**Task 3:**

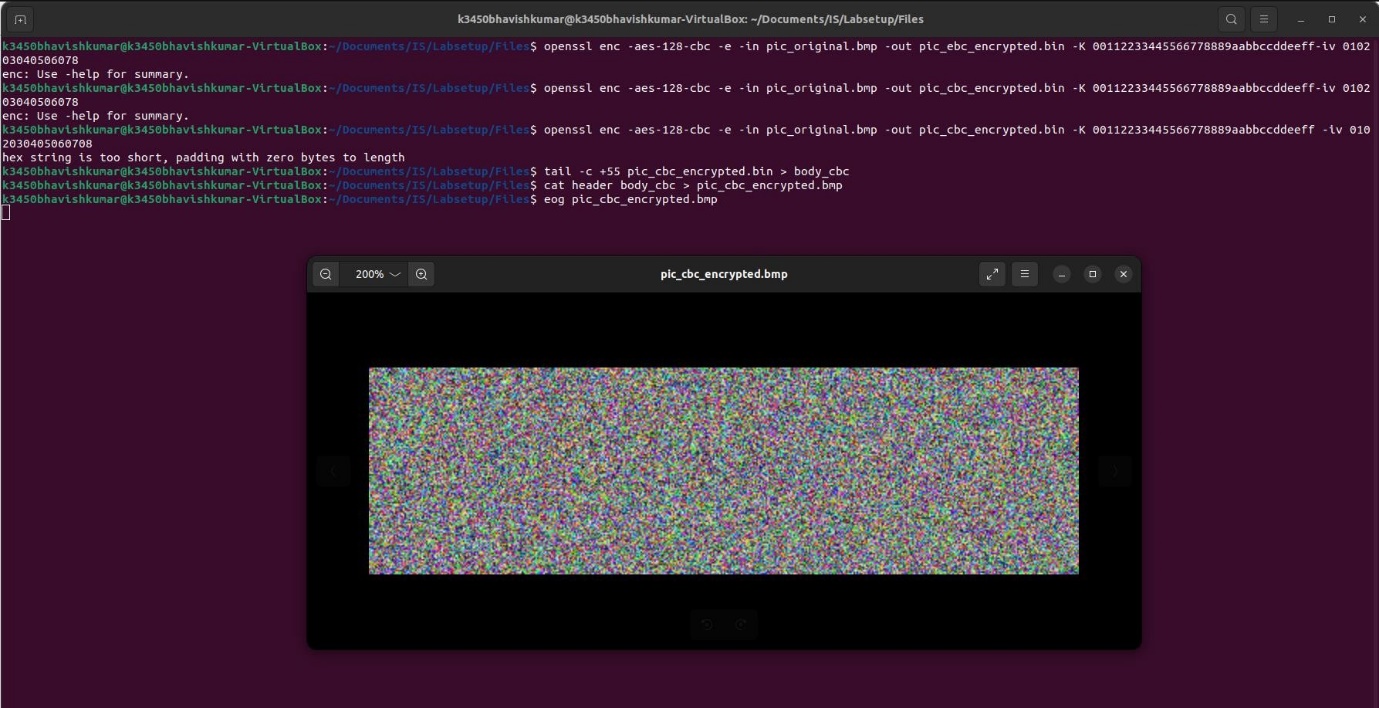
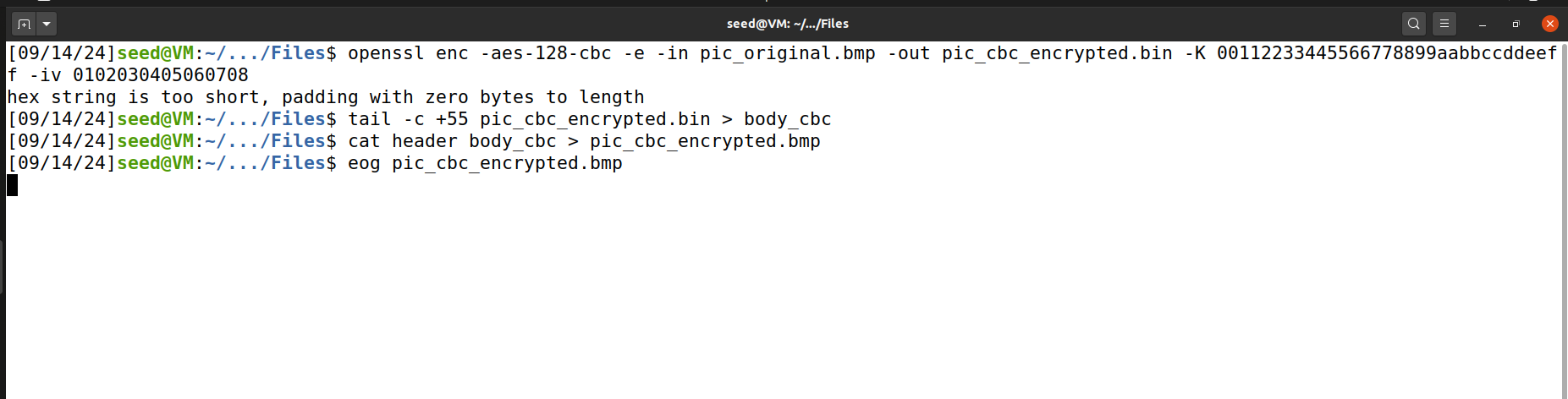
Picture:



Encrypted Image Using ECB:

****

Encrypted Image Using CBC:

****

**Task 4:**

**Encrypted File Size Calculations:**

#### **File 1: 7 Bytes**

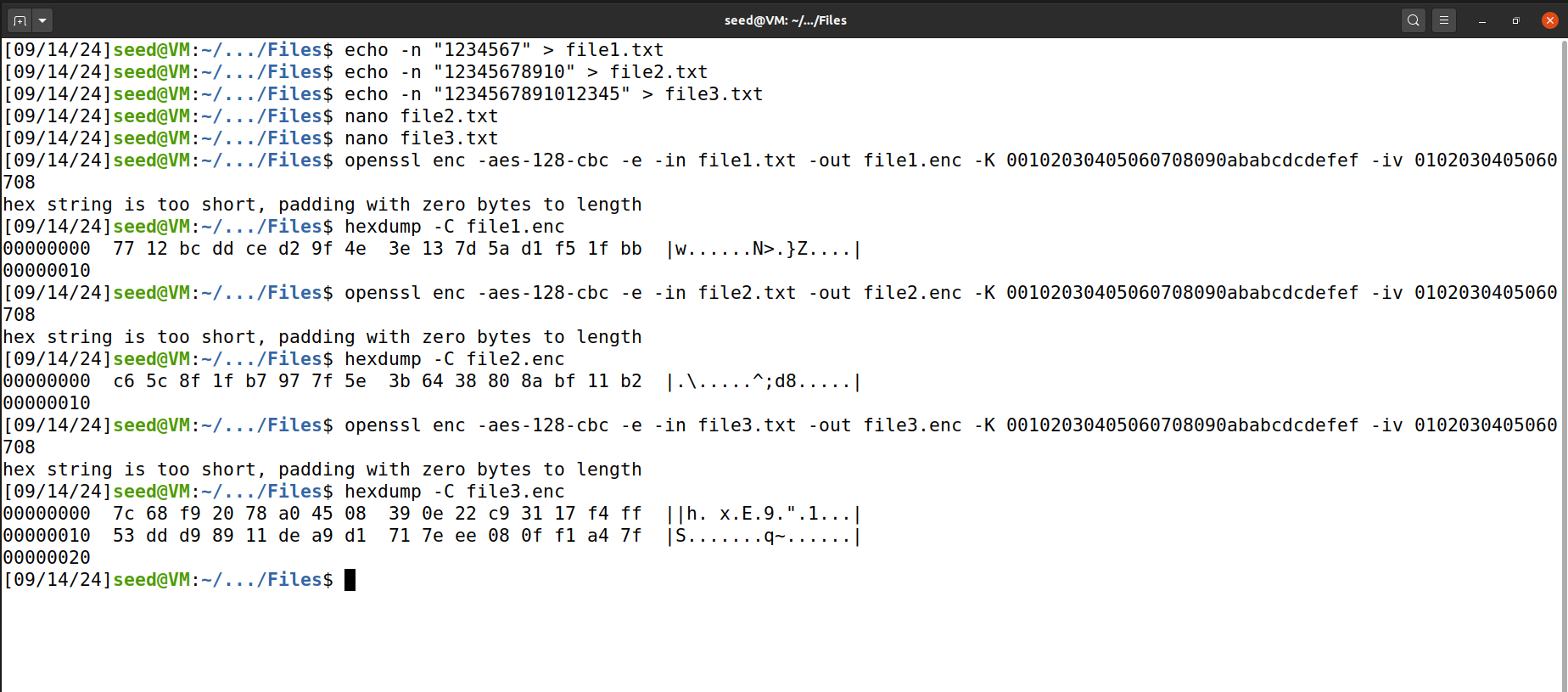
* **Original Data Size**: The original data is 7 bytes in length.
* **Padding Needed**: To reach the required block size of 16 bytes for AES encryption, an additional 9 bytes of padding is applied (since 7 + 9 = 16).
* **Final Size with Padding**: After padding is added, the total size becomes 16 bytes, which completes one full encryption block.
* **Encrypted File Size**: The encrypted file is 16 bytes in size, as AES encrypts in fixed 16-byte blocks.

#### **File 2: 11 Bytes**

* **Original Data Size**: This file contains 11 bytes of unencrypted data.
* **Padding Needed**: Since the AES algorithm operates on 16-byte blocks, 5 bytes of padding are added to make the data fit a complete block (11 + 5 = 16).
* **Final Size with Padding**: Once padding is applied, the total data size becomes 16 bytes, forming one full block.
* **Encrypted File Size**: The encrypted version of the file will be 16 bytes in size, corresponding to the size of one AES block.

#### **File 3: 16 Bytes**

* **Original Data Size**: The original file size is exactly 16 bytes.
* **Padding Needed**: Even though the file already matches the block size, AES-CBC encryption (with PKCS#7 padding) requires an additional full block of 16 bytes of padding.
* **Final Size with Padding**: The total size after padding becomes 32 bytes (16 bytes of data + 16 bytes of padding), which forms two full blocks.
* **Encrypted File Size**: The final encrypted file is 32 bytes in size, as it contains two full blocks of data.



**Task 5:**

### **Encryption Modes and the Impact of Data Corruption:**

#### **Mode 1: Electronic Codebook (ECB)**

* **Effect of Corruption**:
  + In ECB mode, each block of plaintext is encrypted independently of the others. Therefore, if a single bit or block of ciphertext is corrupted, the error only affects that particular block.
  + **Recovery**: You can recover all the other blocks of plaintext except the corrupted one. Since there’s no connection between blocks, errors do not spread, meaning only the damaged block is unreadable or garbled.

#### **Mode 2: Cipher Block Chaining (CBC)**

* **Effect of Corruption**:
  + CBC mode works by XORing each plaintext block with the previous block’s ciphertext before encryption. Due to this chaining, a corruption in one block has a ripple effect, impacting both the corrupted block and the next block during decryption.
  + **Recovery**: While you can recover the plaintext before and after the corrupted block, the corrupted block and the one immediately following it will be affected. The first corrupted block results in incorrect decryption, and the second one becomes garbled due to the disruption in the chaining process.

#### **Mode 3: Cipher Feed back (CFB)**

* **Effect of Corruption**:
  + CFB mode operates as a self-synchronizing stream cipher, meaning the encryption of the current block relies on the previous block’s ciphertext. A corruption in a single bit of the ciphertext results in errors that affect the corresponding bit in the decrypted plaintext and continue to propagate throughout the rest of the decrypted data.
  + **Recovery**: You can recover plaintext up until the corrupted bit. However, from the point of corruption onward, the rest of the decrypted data will be incorrect. The extent of corruption depends on the feedback size, meaning the more data affected by the error, the greater the cascading corruption effect.

#### **Mode 4: Output Feed back (OFB)**

* **Effect of Corruption**:
  + OFB mode converts a block cipher into a synchronous stream cipher. A corruption in the ciphertext only impacts the corresponding bit in the plaintext, and it does not propagate beyond that single bit.
  + **Recovery**: While the bit affected by the corruption will be garbled, all other parts of the plaintext remain intact. OFB mode is unique in that errors are confined to specific bits, and there is no spread or chain reaction due to corruption.

### **Summary of Corruption Impact and Recovery:**

**Electronic Codebook Mode:**

* **Recoverable**: All blocks, except for the corrupted one.
* **Reason**: Each block is encrypted independently, so corruption is isolated to the affected block, with no impact on the rest of the file.

#### **Cipher Block Chaining Mode:**

* **Recoverable**: All plaintext before the corrupted block and the block that follows it.
* **Reason**: Corruption impacts the decrypted output of the corrupted block and spreads to the next block due to the chaining mechanism in CBC.

**Cipher Feed back Mode**:

* **Recoverable**: All plaintext before the corrupted bit; corruption continues to propagate through the rest of the file.
* **Reason**: Corruption in one bit propagates because CFB relies on a feedback loop that affects subsequent decryption.

#### **Output Feed back Mode:**

* **Recoverable**: All plaintext before the corrupted bit; only the specific bit is affected.
* **Reason**: OFB confines corruption to a single bit, as it doesn’t involve feedback chaining, preventing errors from spreading.

### **Justification for Corruption Behavior:**

These behaviors are rooted in the distinct way each encryption mode processes and links blocks of data:

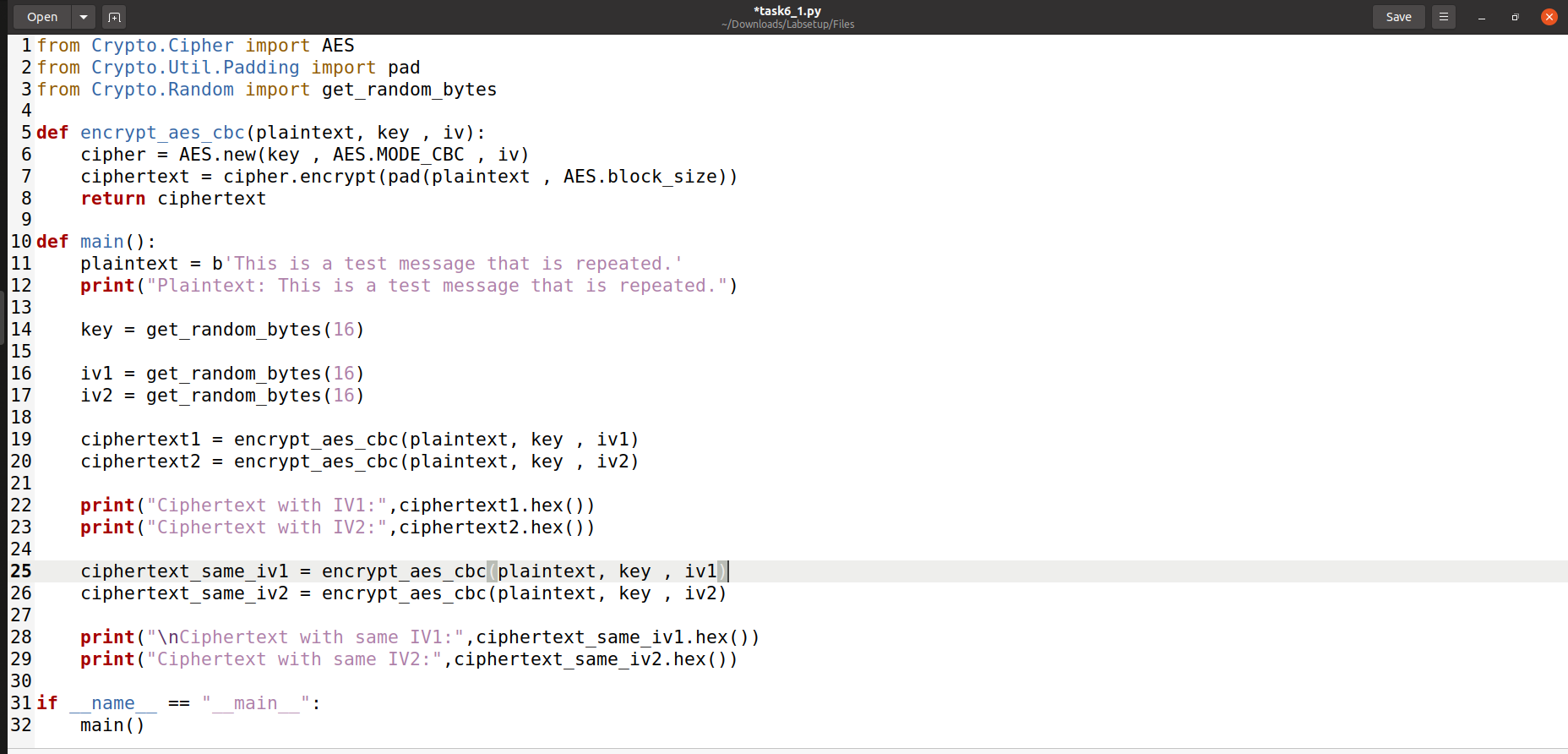
* **ECB Mode**: Encrypts each block independently without referencing other blocks, leading to isolated errors.
* **CBC Mode**: Chains blocks together using XOR operations with the previous block’s ciphertext, causing errors to ripple between adjacent blocks.
* **CFB Mode**: Converts the block cipher into a stream cipher using feedback from previous ciphertext, which makes corruption propagate beyond the original point of error.
* **OFB Mode**: Similar to CFB, but operates as a synchronous stream cipher where corruption does not propagate, limiting the impact to specific bits without affecting other parts of the plaintext.

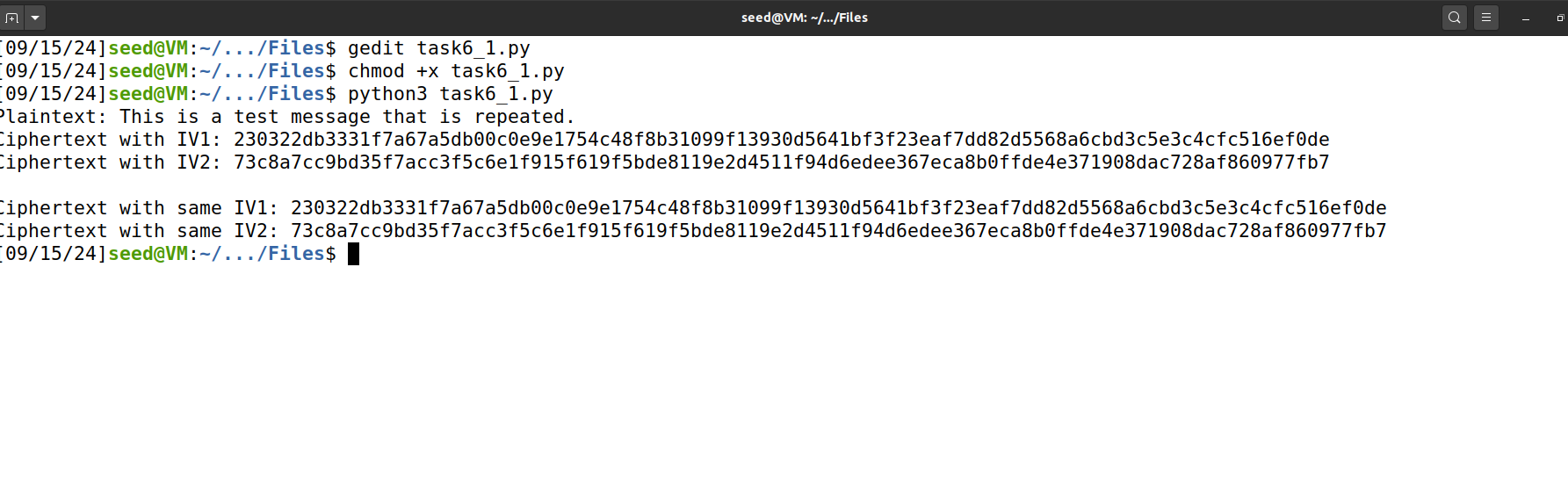
**Task 6:**

**6.1: IV Experiment**

To understand the role of the Initialization Vector (IV) in cryptographic schemes, follow this plan for Task 6.1:

* Encrypt the same plaintext twice: once with different IVs and once with the same IV.
* Observation: Notice how using the same or different IVs affects the cipher text.
* When different IVs are used, the cipher text should vary each time, even for the same plaintext.
* When the same IV is used for the same plaintext, the cipher text will be identical, which is insecure.

****

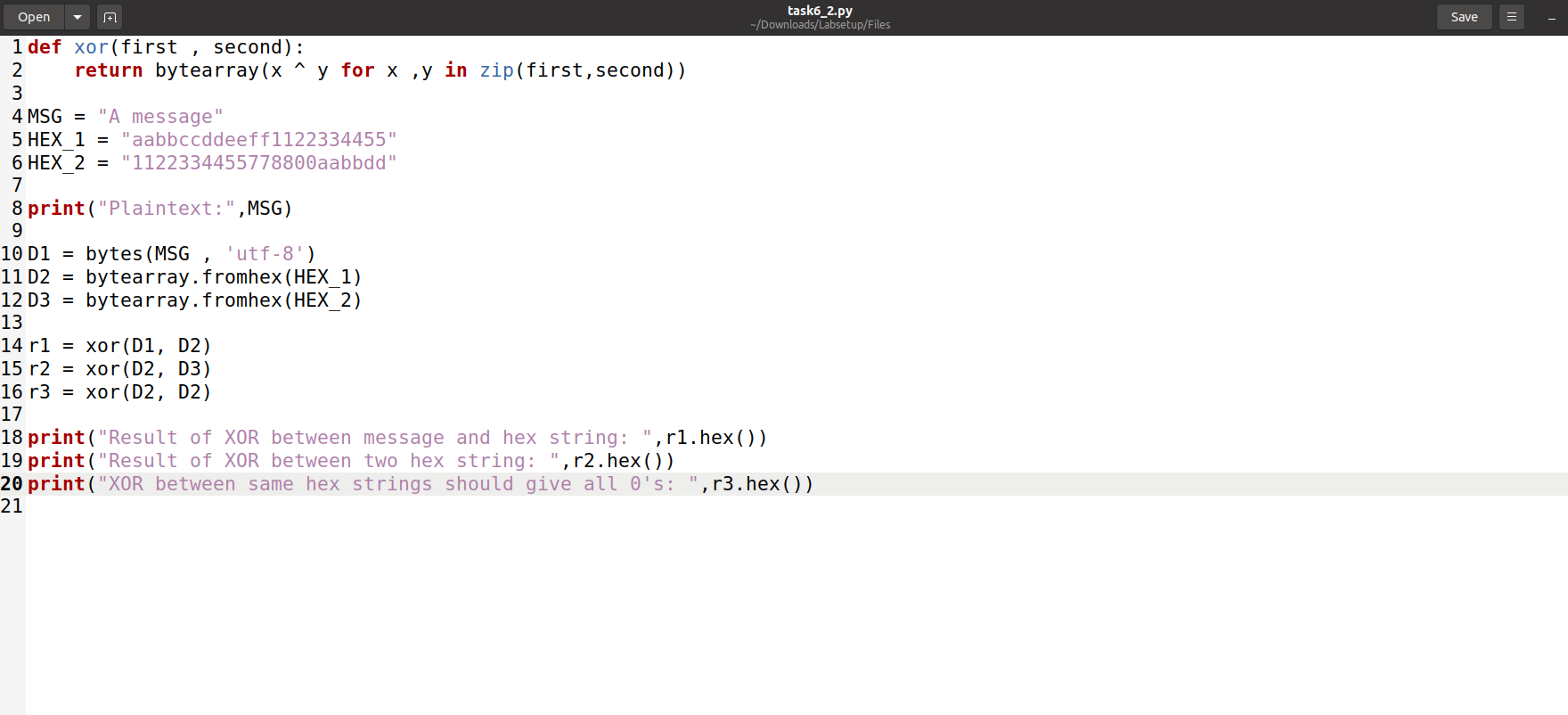
****

**6.2: Common Mistake Using Same IV**

Scenario: If an attacker knows a plaintext (P1) and its ciphertext (C1) using the same IV, they can decrypt another ciphertext (C2) encrypted under the same IV.

In **OFB (Output Feedback) mode**, the encryption of plaintext blocks involves XORing each plaintext block with a key stream value K1K1K1, which is derived from the IV. Specifically, the equations are C1=P1⊕K1C1 = P1 \oplus K1C1=P1⊕K1 and C2=P2⊕K1C2 = P2 \oplus K1C2=P2⊕K1. Given plaintext P1P1P1, ciphertext C1C1C1, and C2C2C2, an attacker can easily compute the key stream K1K1K1 as K1=P1⊕C1K1 = P1 \oplus C1K1=P1⊕C1, and then use it to recover the second plaintext block P2P2P2 with P2=C2⊕K1P2 = C2 \oplus K1P2=C2⊕K1. In contrast, **CFB (Cipher Feedback) mode** introduces additional complexity because each ciphertext block is influenced by the preceding ciphertext block due to its feedback mechanism. While this feedback makes it harder to fully reveal P2P2P2 from a single ciphertext block alone, it can still result in partial leakage of plaintext information if enough data is available.

A computer screen shot of a code

Description automatically generated 

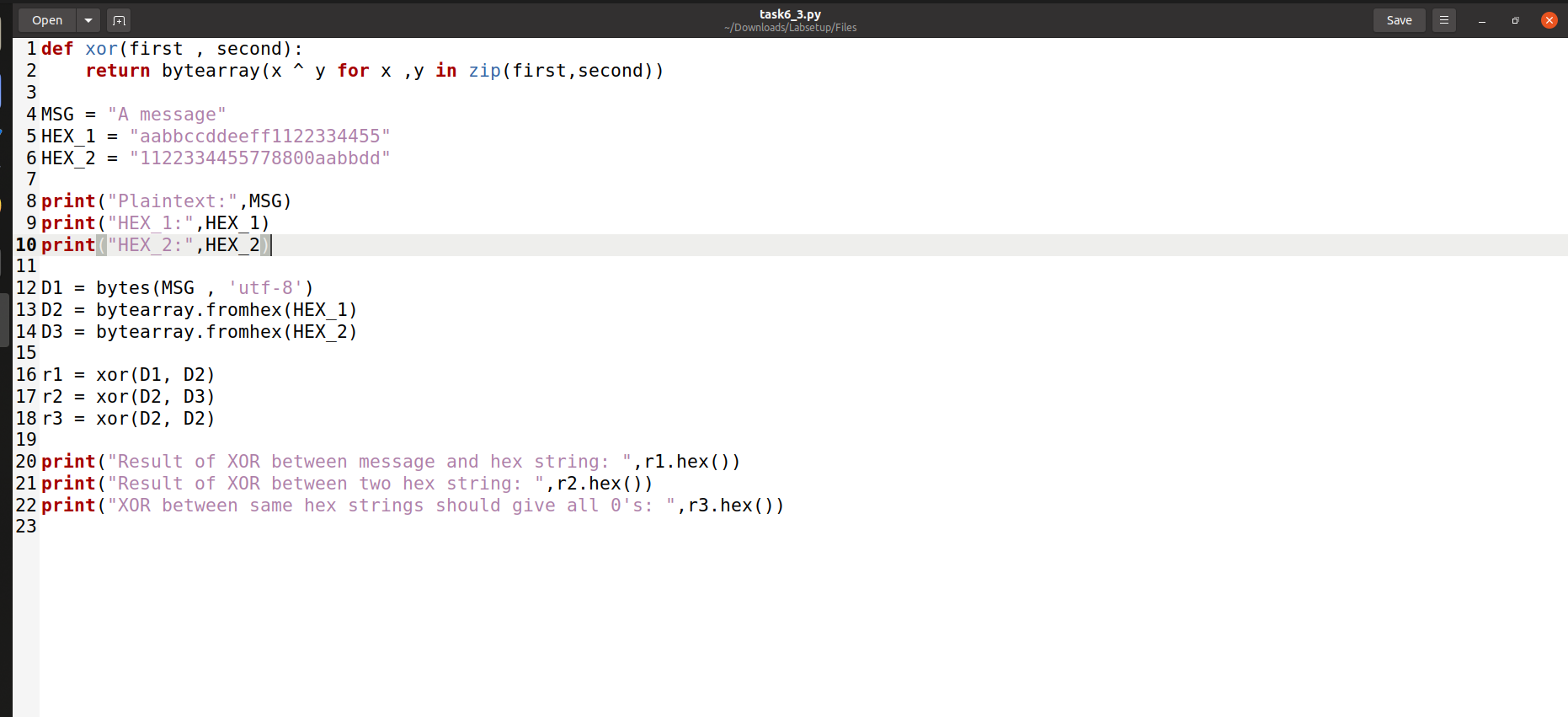
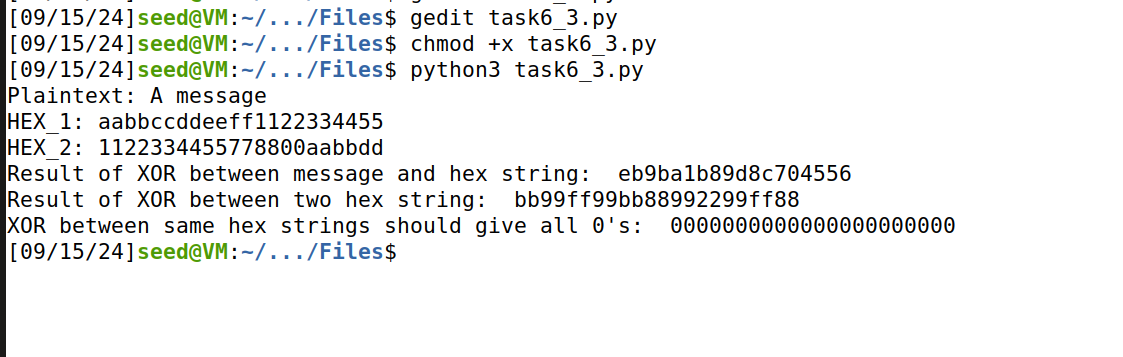
**6.3: Common Mistake Predictable IV**

You are provided with an encryption oracle that simulates Bob’s encryption using AES-128-CBC mode. The objective is to exploit the predictability of Bob’s IV to figure out whether his secret message is "Yes" or "No."

Here’s how you can approach this:

Since Bob uses predictable IVs, you can create a plaintext message and observe the corresponding ciphertext. By analyzing the predictable IV, you can determine the relationship between the plaintext and the ciphertext.

This approach simulates a chosen-plaintext attack, where the attacker leverages knowledge of predictable IVs to infer additional encrypted messages.



**Key Insights:**

* **OFB Mode**: If the Initialization Vector (IV) is reused, the entire key stream can be compromised. This is because the same IV will generate identical key streams for each encryption operation, making it possible to reveal all the encrypted data.
* **CFB Mode**: Reusing the IV in CFB mode may lead to partial exposure of the plaintext. Although this mode does not reveal the entire message as readily as OFB mode, some segments of the plaintext can still be uncovered. The feedback mechanism in CFB mode limits the extent of plaintext that can be decrypted compared to OFB mode.

**Task 7:**

**Steps Involved:**

1. **Generate Key**: Convert each word from the wordlist into a 16-byte key by appending pound signs (#) to pad it as needed.
2. **Decrypt**: Use the generated key and the provided IV to decrypt the ciphertext. Check if the decrypted plaintext matches the known plaintext.
3. **Find the Key**: When a match is found, display the key and the decrypted plaintext.

This task is crucial for understanding how cryptographic keys function, the significance of key security, and how brute-force attacks can exploit weaknesses in key selection.

The main objectives of this task are:

1. **Understand AES-128-CBC Encryption**: AES (Advanced Encryption Standard) is a widely-used symmetric encryption algorithm that operates on 128-bit (16-byte) blocks. CBC (Cipher Block Chaining) mode is one of its most commonly used variants. In CBC mode, each plaintext block is XORed with the previous ciphertext block before encryption, enhancing security over ECB (Electronic Codebook) mode by introducing inter-block dependencies.
2. **Work with Known Plaintext and Ciphertext**: You are given:
   * A plaintext message: "This is a top secret." (21 bytes in total).
   * A ciphertext (in hexadecimal format) representing the encrypted plaintext: 764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2
   * An initialization vector (IV) (in hexadecimal format): aabbccddeeff00998877665544332211
3. **Brute-Force the Encryption Key**: The encryption key is an English word shorter than 16 characters. To reach the required length, append pound signs (#, hex value 0x23) to the word. The task involves brute-forcing this key by trying each word from a wordlist, padding it with pound signs until it is 16 bytes long, and checking if it decrypts the ciphertext to reveal the original plaintext.
4. **Decrypting with a Given IV and Key**: Decrypt the ciphertext using the provided IV and each candidate key from the wordlist. If the decrypted text matches the known plaintext ("This is a top secret."), then the correct key has been identified.

