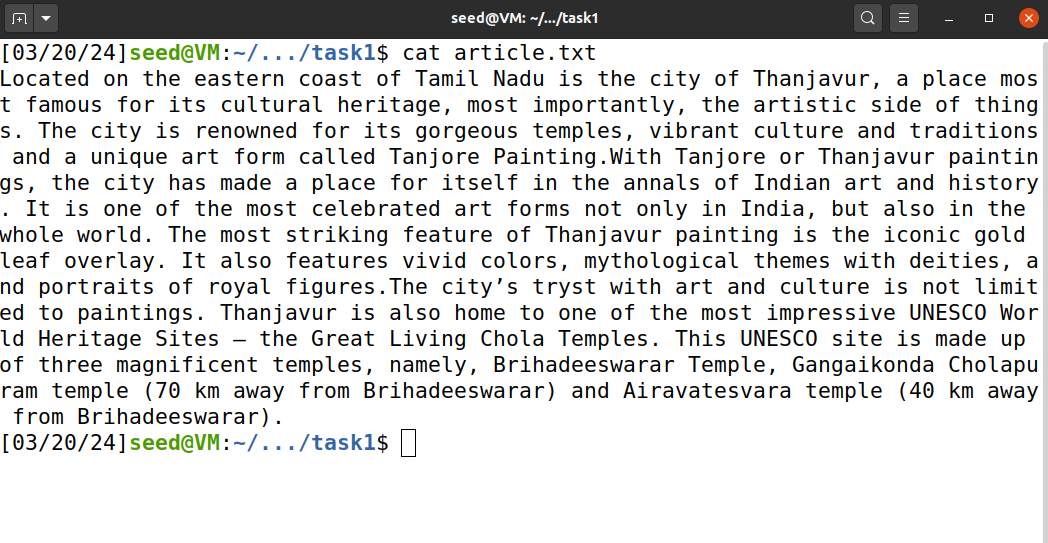
ACS 54500 Cryptography and Network Security

Lab 8: Crypto Lab: Secret-Key Encryption

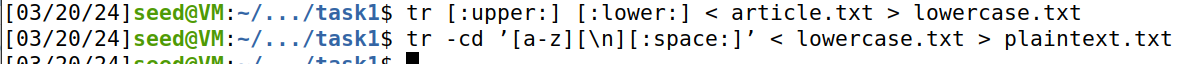
Name: Vijayagiridharan Subramanian

**Task 1: Frequency Analysis**

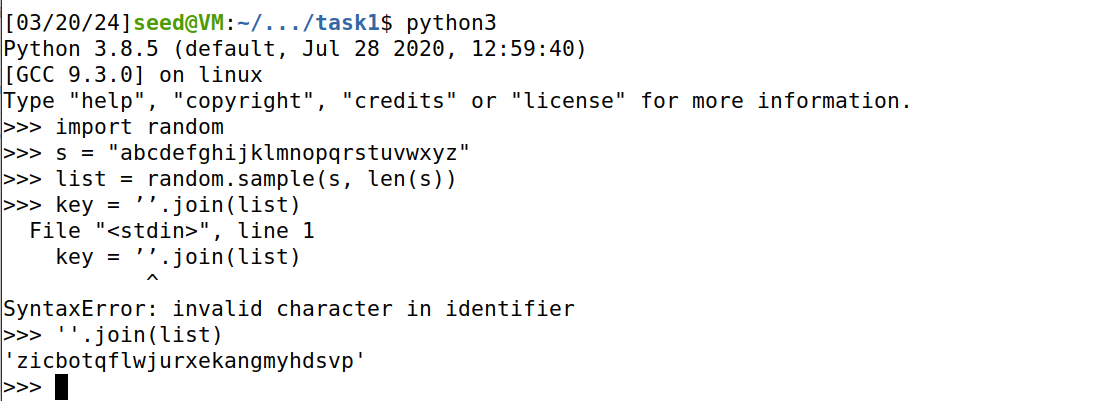
I have chosen a random article for this task and named it article.txt and will convert it into cipher text.



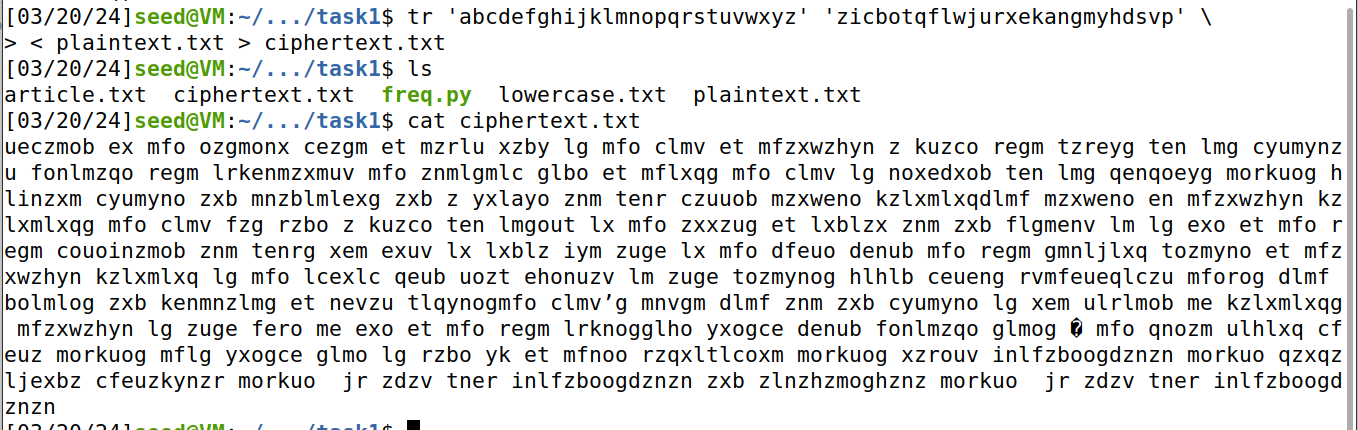
We convert all upper cases to lower cases, and then remove all the punctuations and numbers.



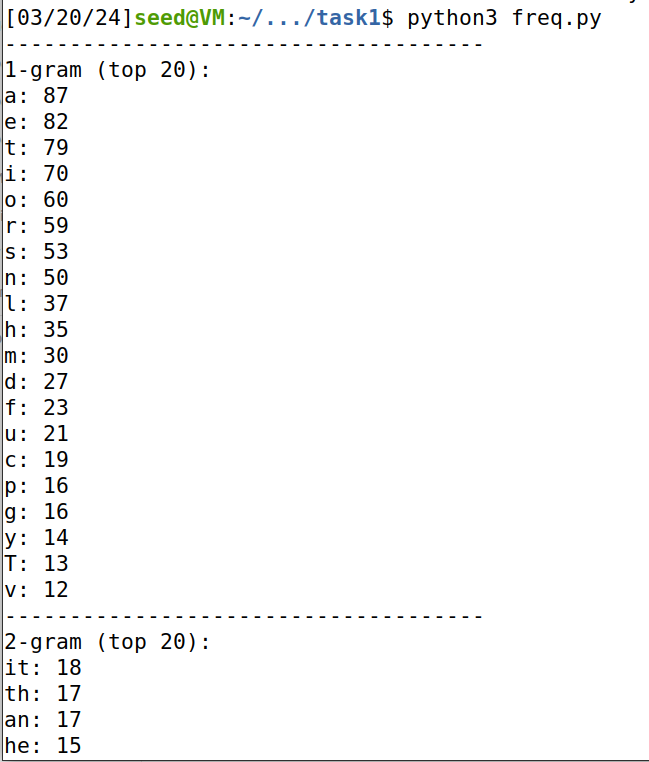
We then generate substitution code using python script and ‘**zicbotqflwjurxekangmyhdsvp**’ is the substitution code.

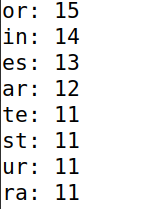


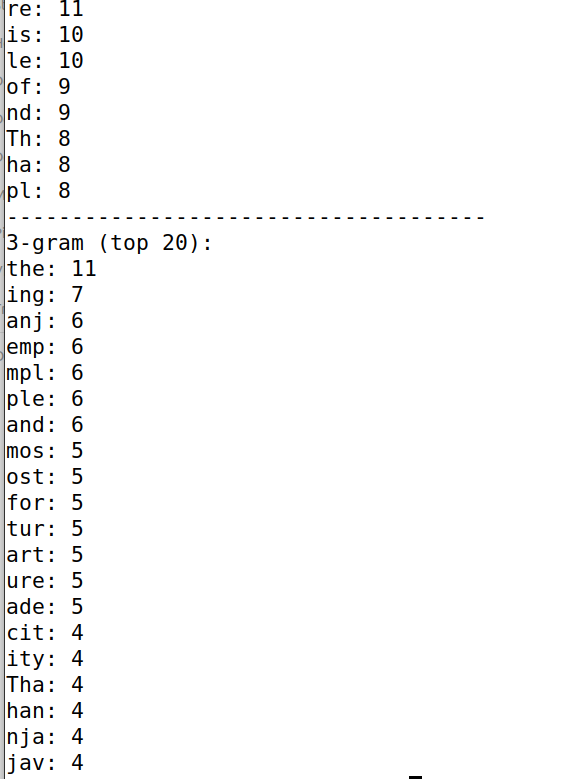
Then we are encrypting the text by the generated substitution code and below that generated cipher text is displayed.



We are running **freq.py** to execute frequency analysis on the cipher text, which produces the statistics for n-grams, including the single-letter frequencies, bigram frequencies (2-letter sequence), and trigram frequencies (3-letter sequence), etc.

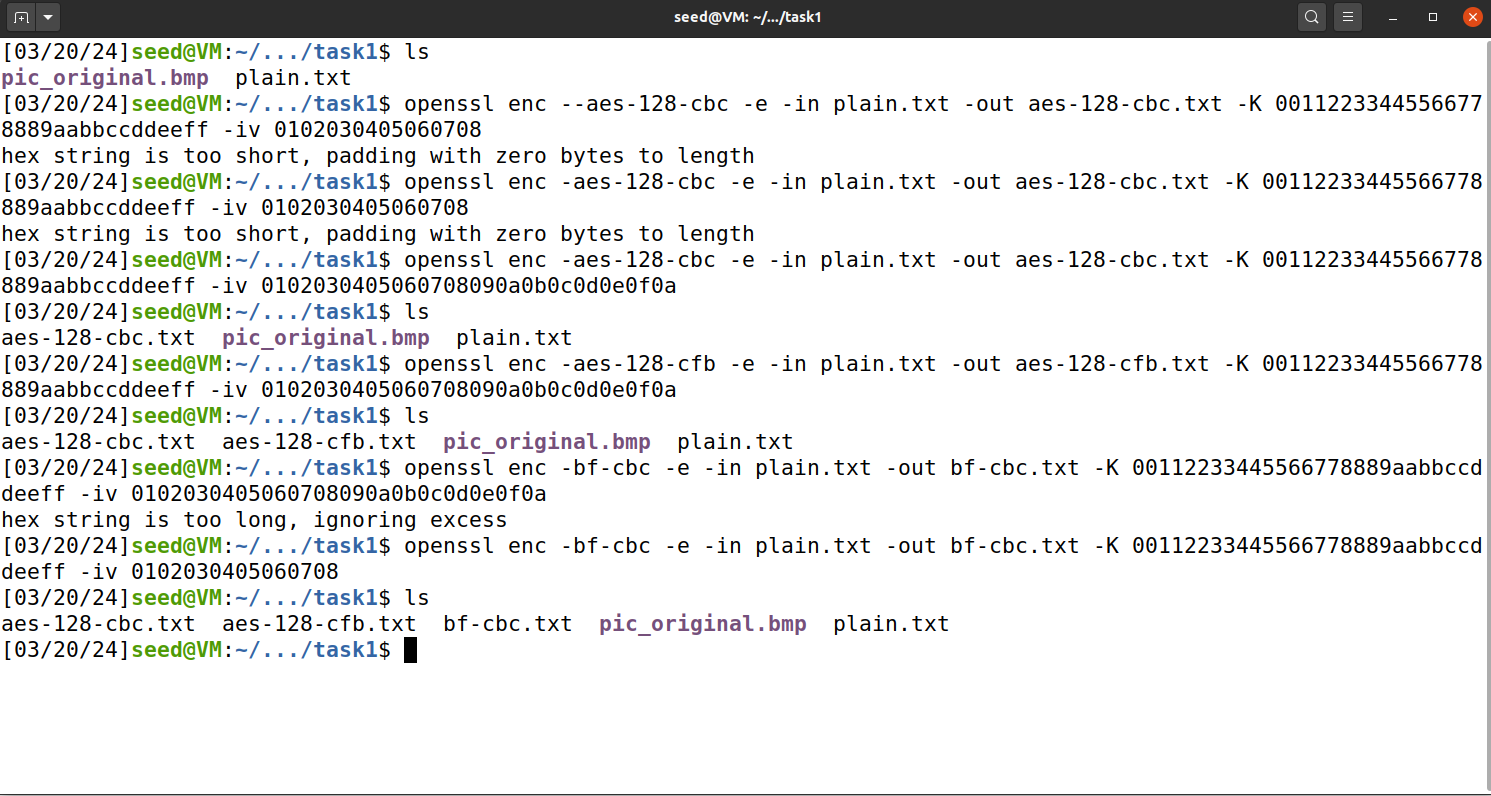
****

****

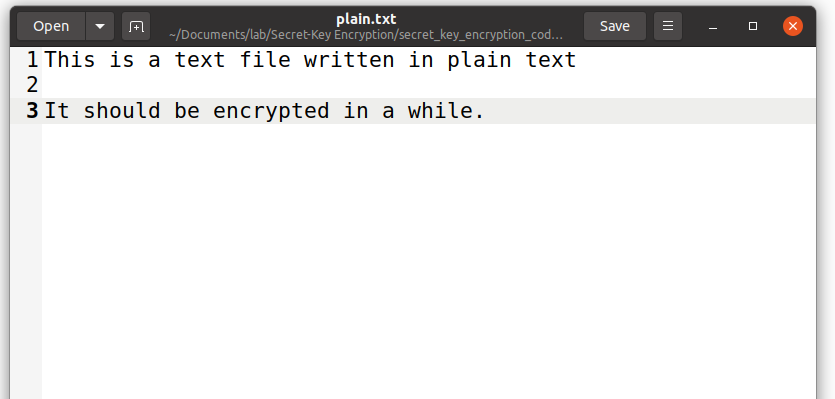
****

**Task 2: Encryption using Different Ciphers and Modes**

Here we are doing encryption with the different ciphers and modes, So first we create plain.txt. We are going to encrypt this with the different modes -aes-128-cbc, -bf-cbc, -aes-128-cfb and aes-128-cbc.txt, bf-cbc.txt, aes-128-cfb.txt are encrypted files of plain.txt in the different modes.



This is initially created plain.txt. We are going to encrypt this.





This aes-128-cbc mode encrypted text of plain.txt:



This aes-128-cfb mode encrypted text of plain.txt:



This bf-cbc mode encrypted text of plain.txt:

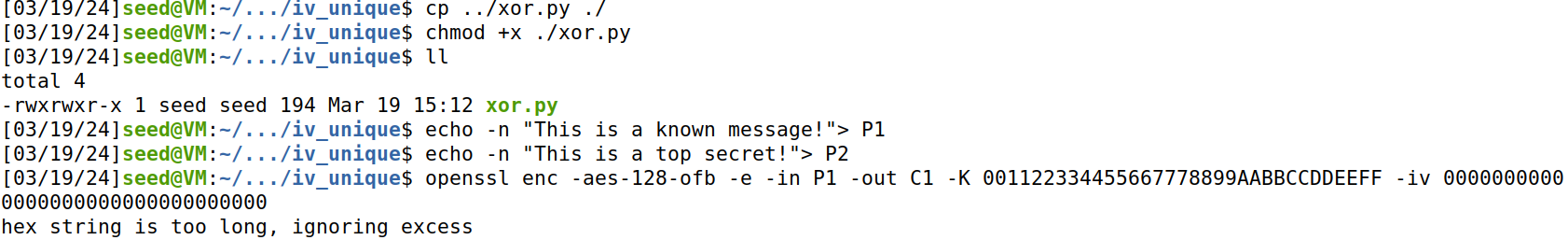


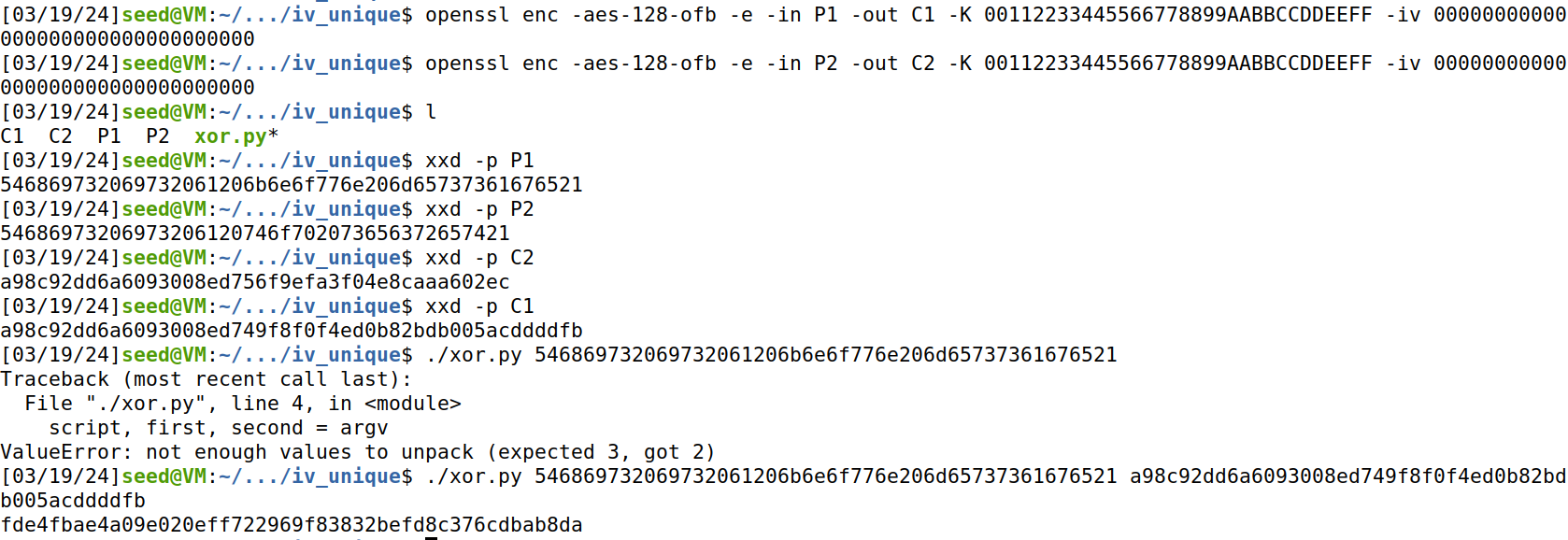
**Task 6: Initial Vector (IV) and Common Mistakes**

**Task 6.1. IV Experiment**

**1.Encrypting the same plaintext using two different IVs.**

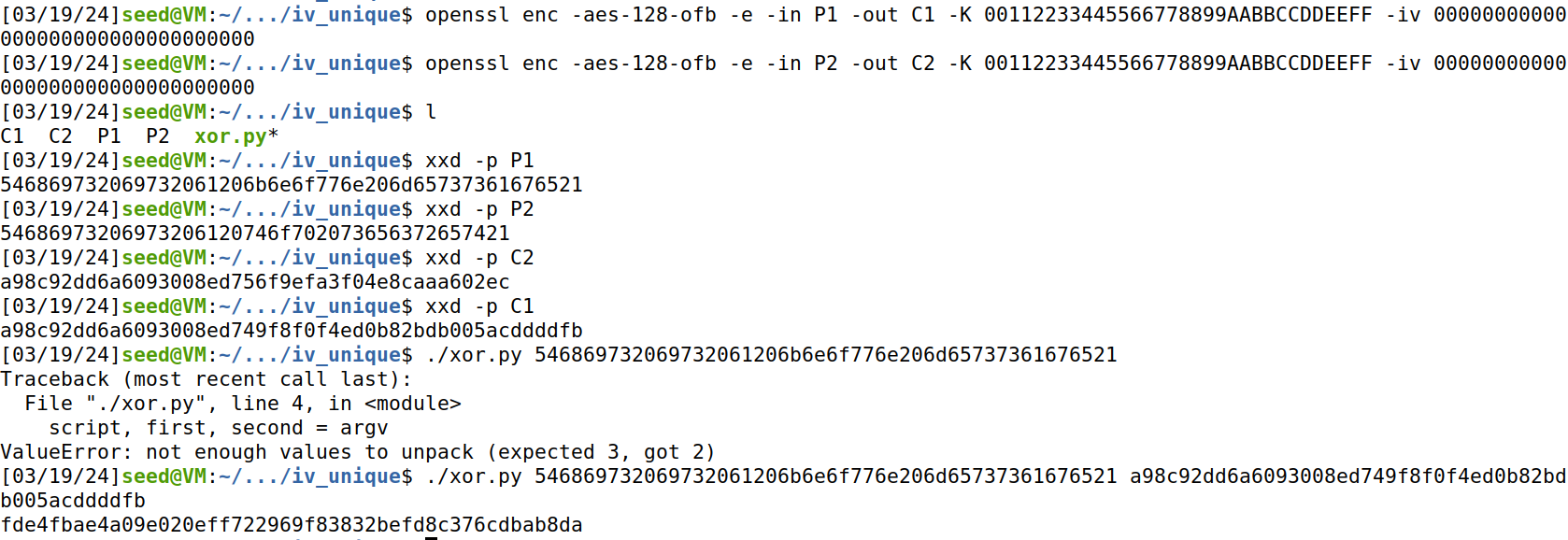
We attempted to encrypt the plaintext file P1 using different IVs (00000000000000000000000000000000 and 00112233445566778899AABBCCDDEEFF) but encountered errors related to IV length initially.After correcting the IV length issue, we successfully encrypted P1 with two different IVs and saved the ciphertext as C1.

****

****

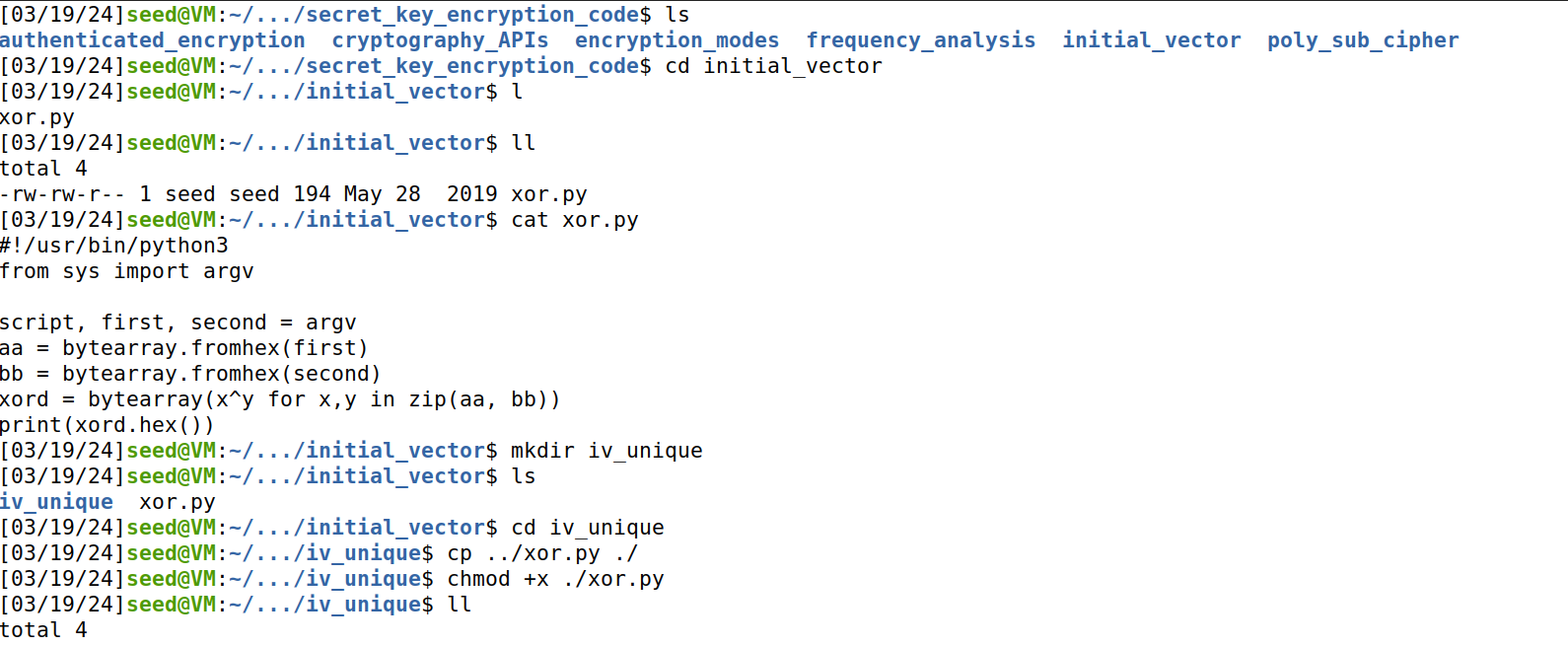
**2.Encrypting the same plaintext using the same IV.**

We encrypted the plaintext files P1 and P2 using the same IV (00000000000000000000000000000000) but with different keys. The resulting ciphertexts were saved as C1 and C2, respectively.

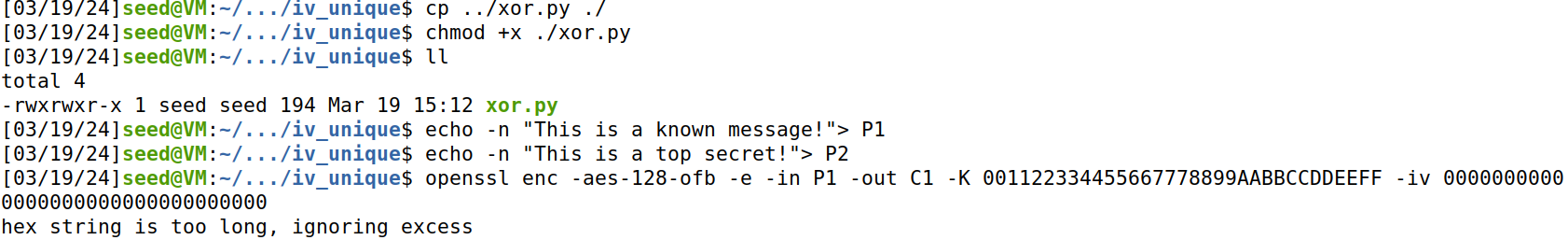
****

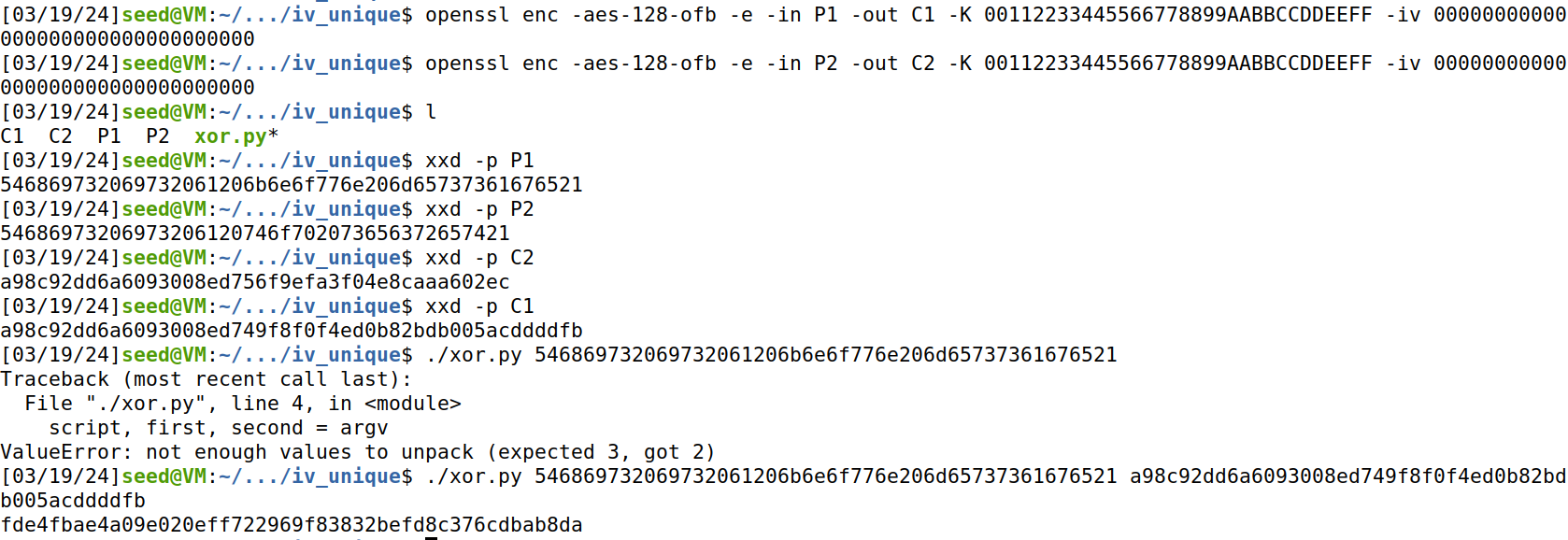
**Task 6.2. Common Mistake: Use the Same IV**

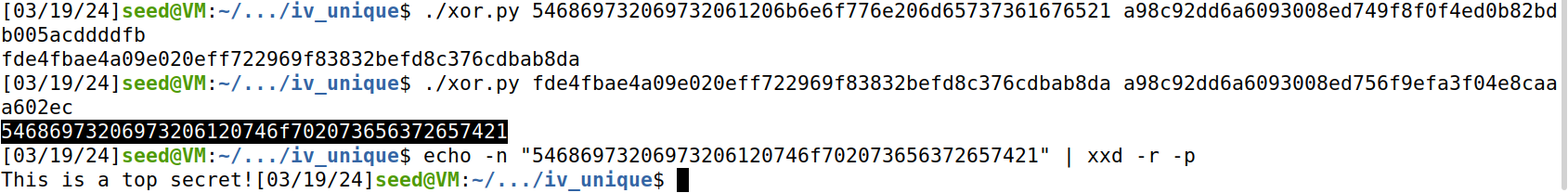
The attack used in this experiment is called the known-plaintext attack, where the attacker has access to both the plaintext and its ciphertext.



Now when we try to create plaintext files with messages ”This is a known message” and “This is a top secret”. Then we tried to encrypt P1 and P2 with the corrected key and IV. Then we converted the contents of P1, P2, C1 and C2 to hexadecimal using xxd. Then we perform XOR operation between P1 and C1; we perform XOR operation between P2 and C2.

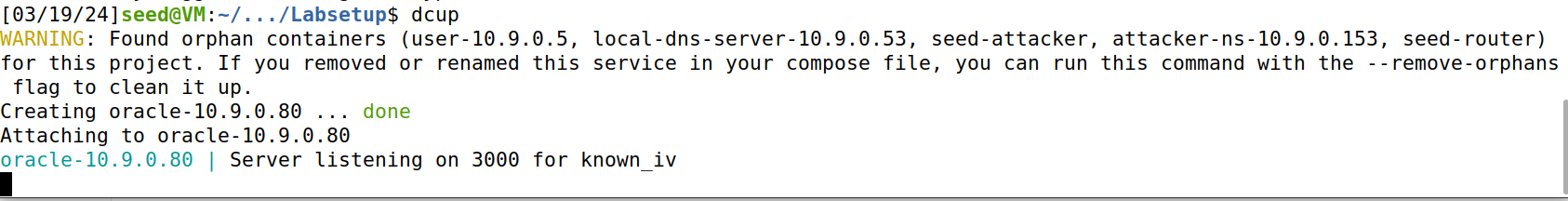
****

****

****

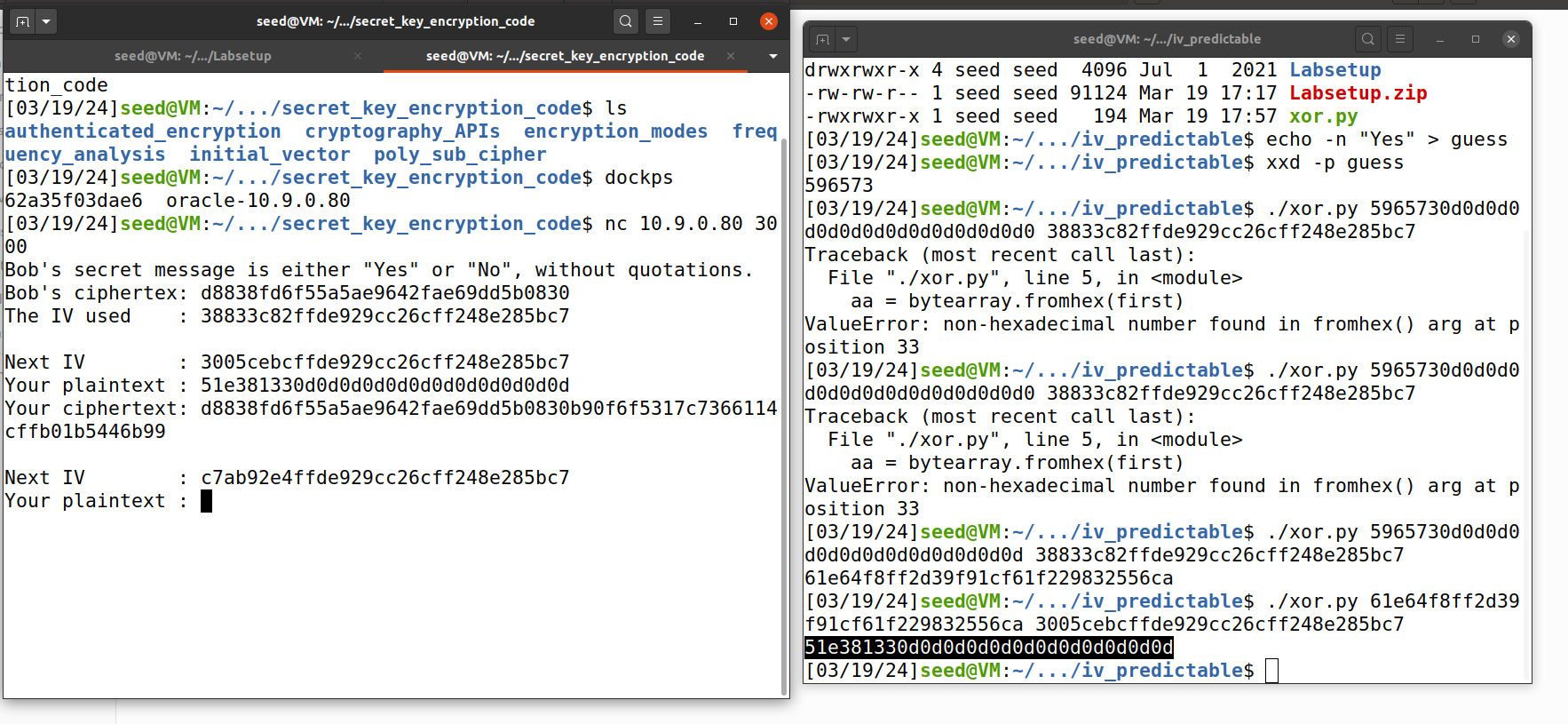
**Task 6.3. Common Mistake: Use a Predictable IV**

We run the docker container and then we are creating 10.9.0.80



In the left side terminal, We are trying to connect through 3000 through netcat 10.9.0.80,we enter the your plaintext like **51e381330d0d0d0d0d0d0d0d0d0d0d0d0d** and it outputs the next ciphertext.

In the right side terminal, we are trying to encrypt message “Yes” using echo -n “YES”, we run the XOR encryption operation with the provided IV, and likely generates the plain text we entered “**51e381330d0d0d0d0d0d0d0d0d0d0d0d0d**”



**Task 7:Programming using the Crypto Library**

I tried 2 methods, you can find that.

**(method 1)**

we change the code according to the question condition of (

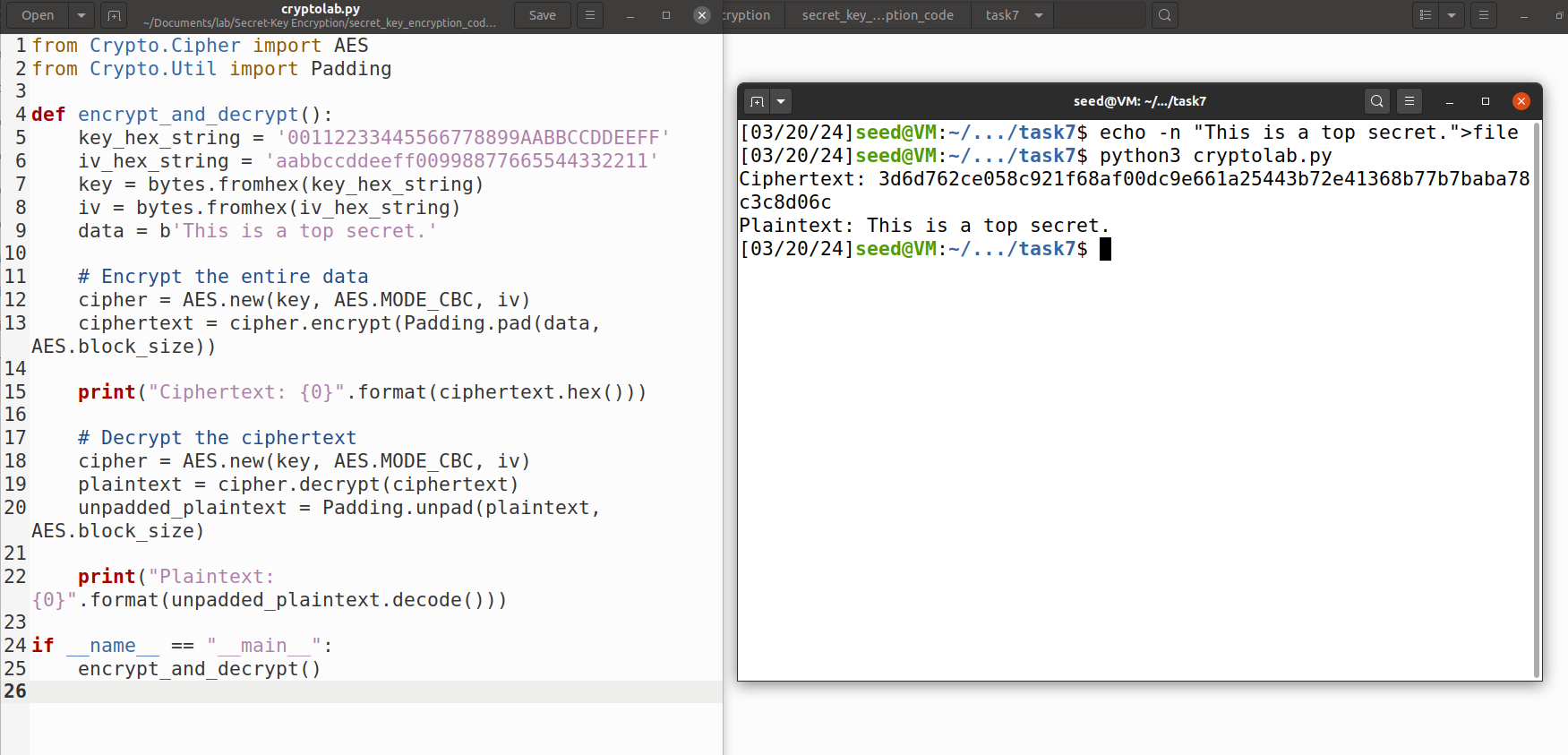
Plaintext (total 21 characters): This is a top secret.

Ciphertext (in hex format):

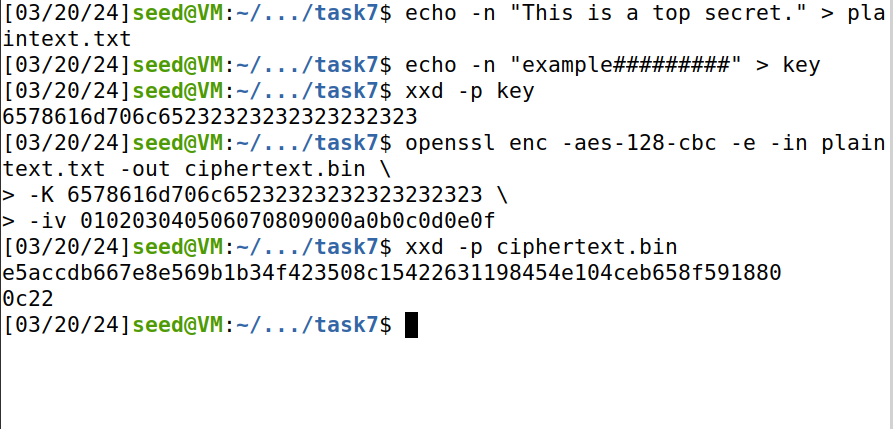
3879c71b232cd0d2fc6f5ffcc1d76f074c0fcbe007d9cc53939fdeebf1d6ffd2

IV (in hex format): aabbccddeeff00998877665544332211

)When we run the code the ciphertext and plaintext is displayed.

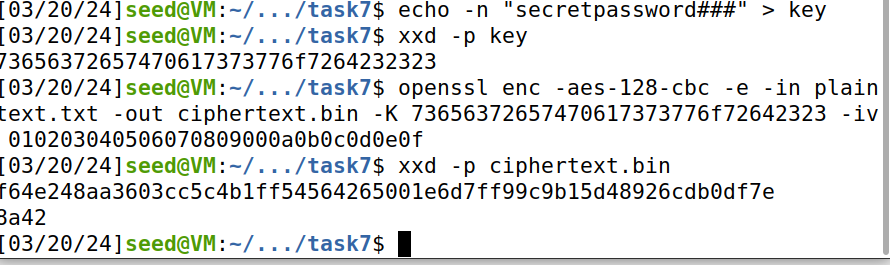
****

A key file is generated using the passphrase "example#########" and converted into hexadecimal format. Next, the plaintext stored in plaintext.txt is encrypted using AES-128-CBC encryption with the derived key and a specified initialization vector. Finally, the resulting ciphertext is displayed in hexadecimal format, showcasing the encryption process.

****

We replace example to secretpassword according to the question.

A key file is generated using the passphrase "secretpassword###" and converted into hexadecimal format. Next, the plaintext stored in plaintext.txt is encrypted using AES-128-CBC encryption with the derived key and a specified initialization vector. Finally, the resulting ciphertext is displayed in hexadecimal format, showcasing the encryption process.

****

**(method 2)**

**Python code:**

**from Crypto.Cipher import AES**

from Crypto.Util.Padding import unpad

import binascii

def decrypt\_aes\_ciphertext(ciphertext\_hex, key, iv):

try:

ciphertext = binascii.unhexlify(ciphertext\_hex)

cipher = AES.new(key, AES.MODE\_CBC, iv)

decrypted = cipher.decrypt(ciphertext)

unpadded\_plaintext = unpad(decrypted, AES.block\_size, style='pkcs7')

return unpadded\_plaintext.decode('utf-8')

except Exception as e:

return None

ciphertext\_hex = "3879c71b232cd0d2fc6f5ffcc1d76f074c0fcbe007d9cc53939fdeebf1d6ffd2"

iv\_hex = "aabbccddeeff00998877665544332211"

with open('words.txt', 'r') as f:

english\_words = f.read().splitlines()

for word in english\_words:

key = (word.encode() + b"#" \* (16 - len(word)))

decrypted\_text = decrypt\_aes\_ciphertext(ciphertext\_hex, key, binascii.unhexlify(iv\_hex))

if decrypted\_text:

print(f"Key found: {word}")

print(f"Decrypted plaintext: {decrypted\_text}")

break

else:

print("Key not found in English word list.")

The code attempts to decrypt a given ciphertext using AES encryption in CBC mode. It iterates through a list of English words, trying each one as a potential decryption key. If decryption succeeds with a key, it prints the key and the decrypted plaintext. Otherwise, it reports that the key wasn't found

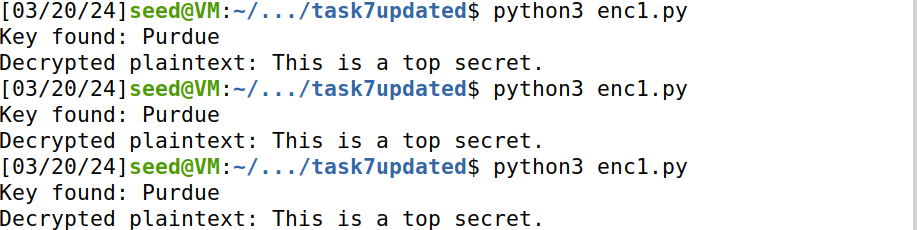
We updated the provided data in the code enc1.py:

Plaintext (total 21 characters): This is a top secret.

Ciphertext (in hex format):

3879c71b232cd0d2fc6f5ffcc1d76f074c0fcbe007d9cc53939fdeebf1d6ffd2

IV (in hex format): aabbccddeeff00998877665544332211

****

When I run **enc1.py,** then **key: purdue** and **decrypted plaintext: This is a top secret.**

Hence we are able to get the encrypted plaintext.