Secret Encryption

SEED LABS

Contents

[Environment Setup 2](#_Toc145771218)

[Task 1 3](#_Toc145771219)

[Step 1 3](#_Toc145771220)

[Step 2 3](#_Toc145771221)

[Step 3 4](#_Toc145771222)

[Task 2 8](#_Toc145771223)

[Task 3 9](#_Toc145771224)

[Step 1 9](#_Toc145771225)

[Step 2 10](#_Toc145771226)

[Task 4 12](#_Toc145771227)

[Step 1 12](#_Toc145771228)

[ECB Mode 12](#_Toc145771229)

[CBC Mode 12](#_Toc145771230)

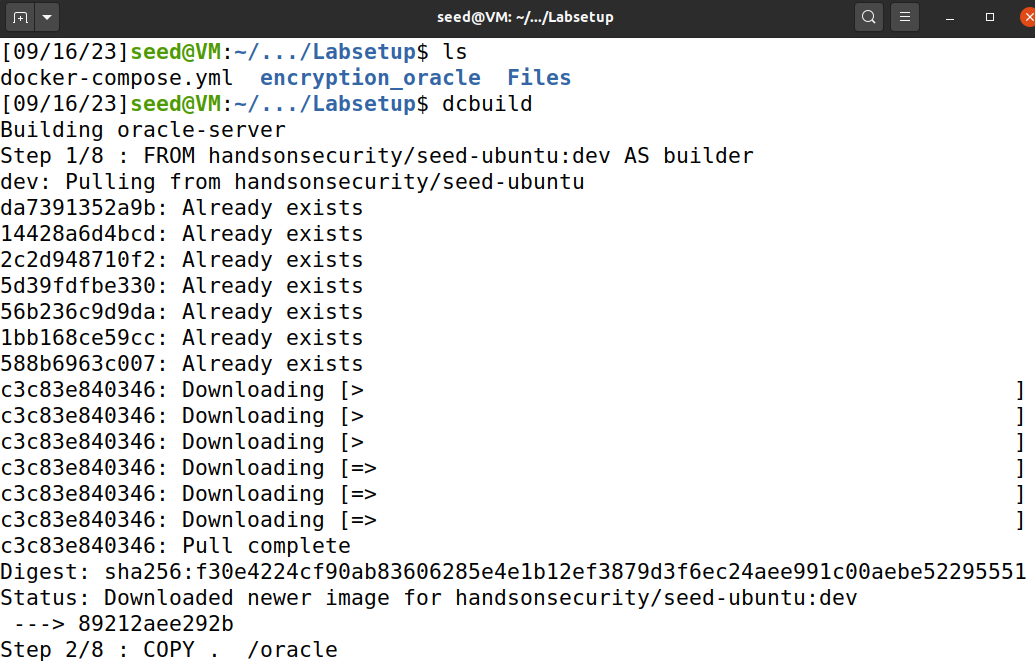
[CFB Mode 12](#_Toc145771231)

[OFB Mode 12](#_Toc145771232)

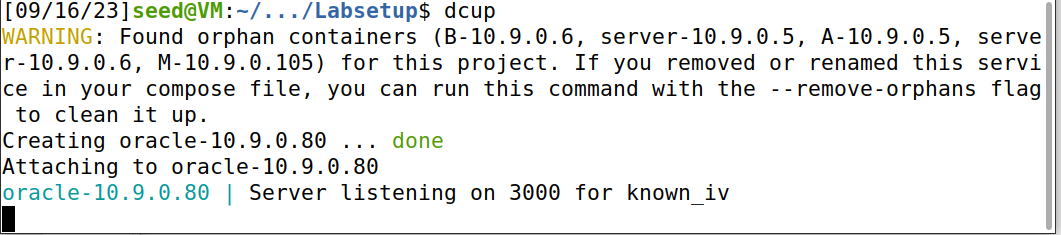
[Step 2 13](#_Toc145771233)

# Environment Setup

Building Container Image.



Initiating Containers.



Connecting to the available Docker in a new terminal.

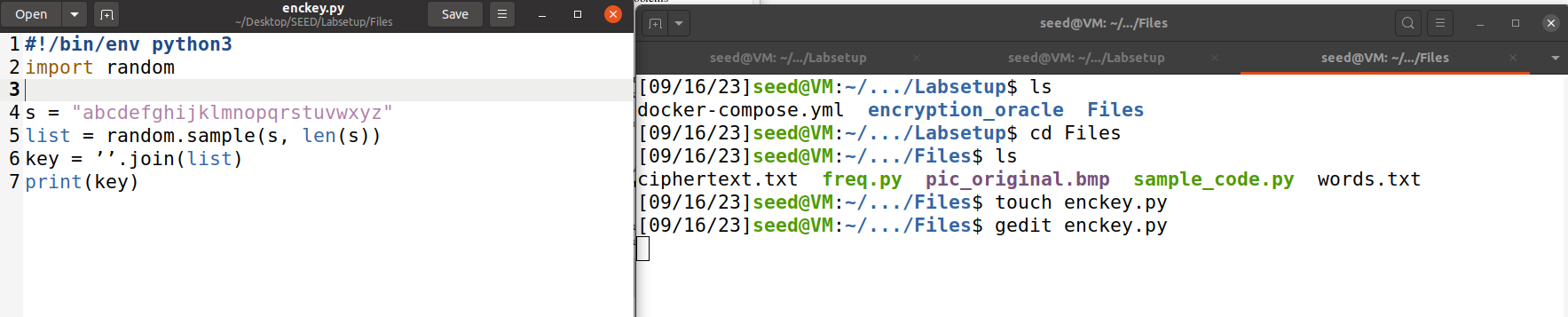
A screenshot of a computer

Description automatically generated

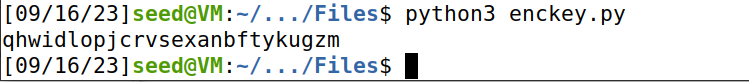
# Task 1

## Step 1

Using the provided code in the manual to generate encryption key.

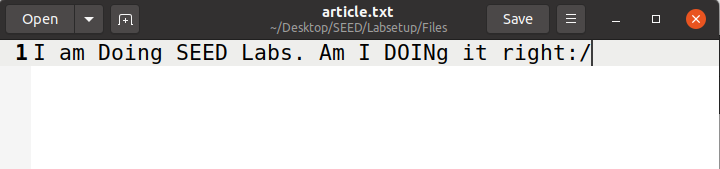


Executing the script to permute the alphabet from a to z using Python and use the permuted alphabet as the key. Each time it is run a randomly generated key is provided.

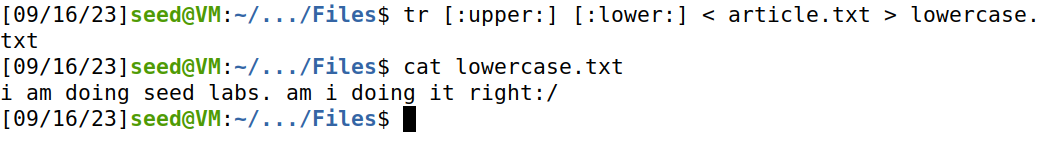


## Step 2

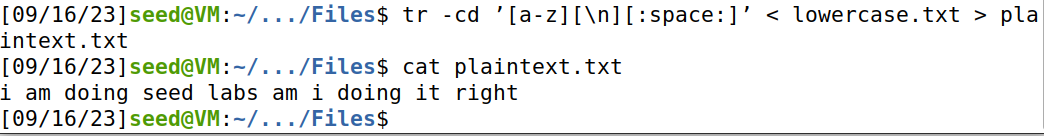
Creating a new text file with some text in it.



As shown in the screenshot the command worked and changed the text to lowercase letters.

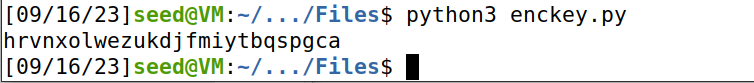


Now with the shown command I have removed special characters and numbers, if any.

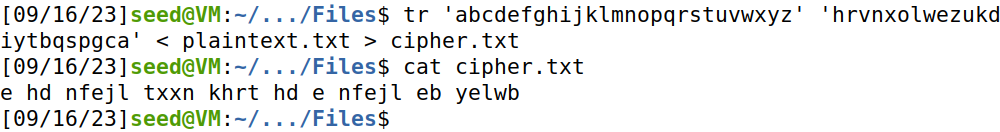


## Step 3

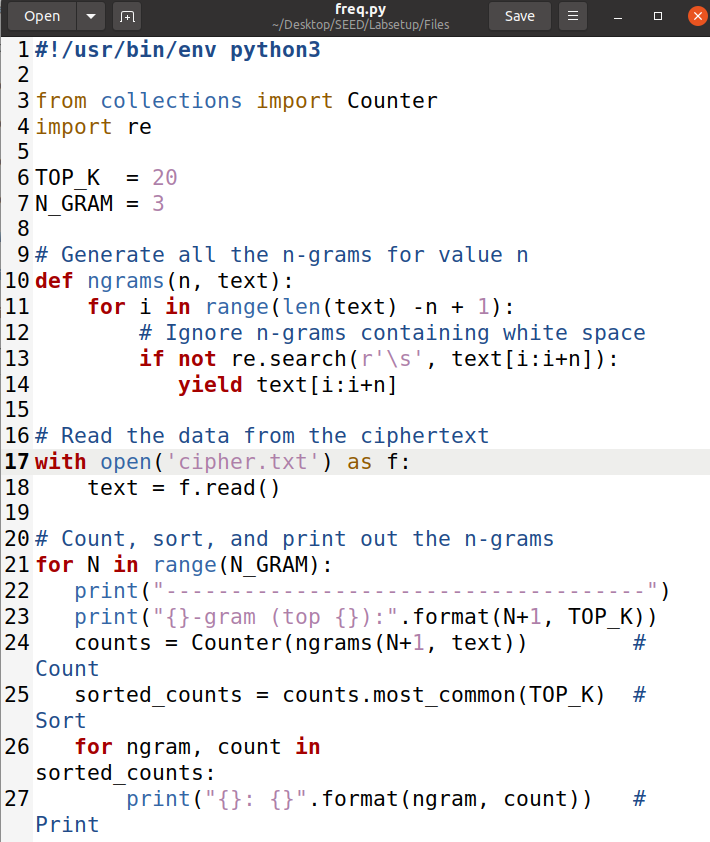
Generating key from the provided script.



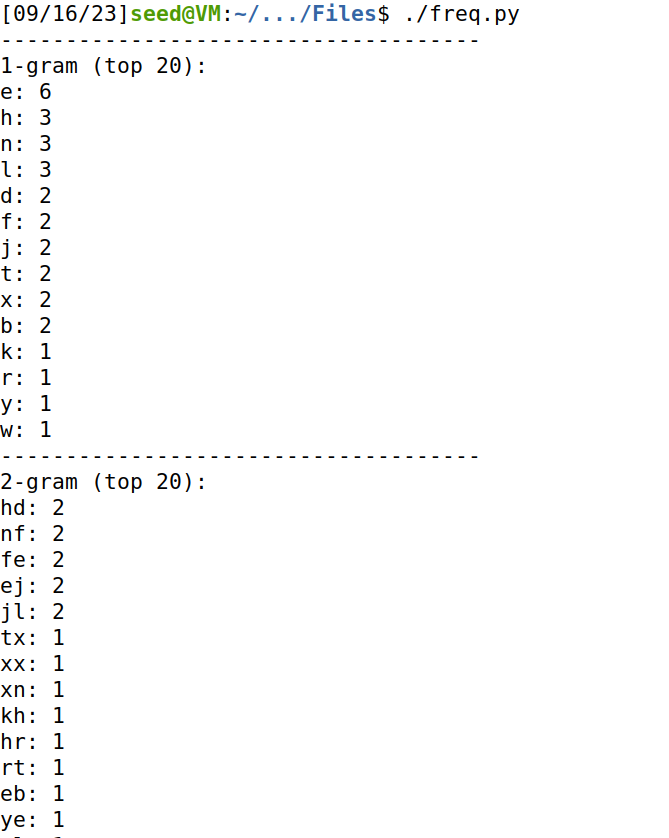
I have encrypted the plaintext and made it ciphertext stored in cipher.txt as there is already provided ciphertext.txt.



Editing the provided frequency analysis code to read from cipher.txt.



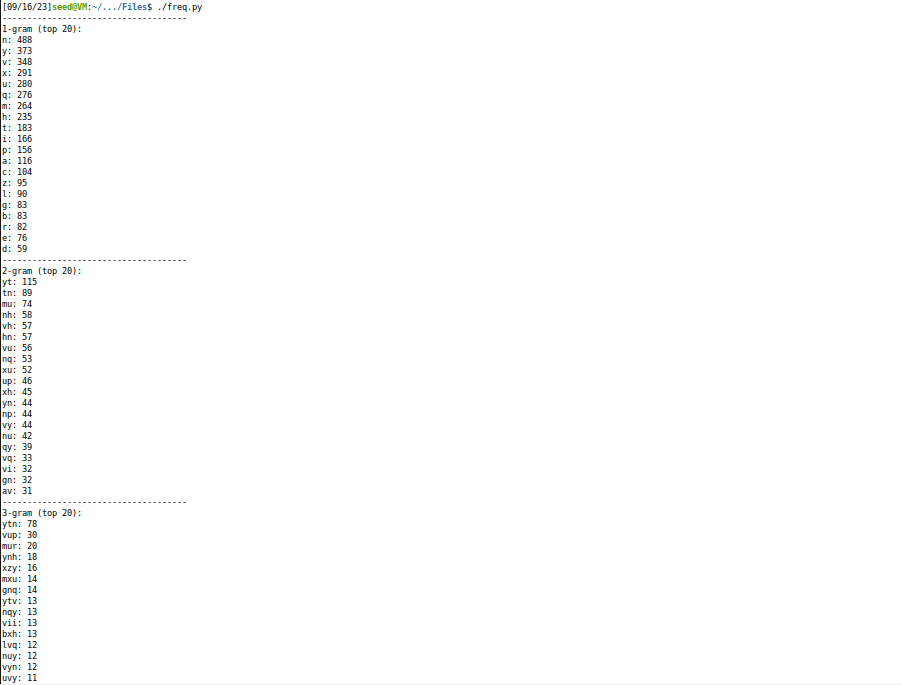
This is the result I got from running Frequency Analysis code.



A screenshot of a computer

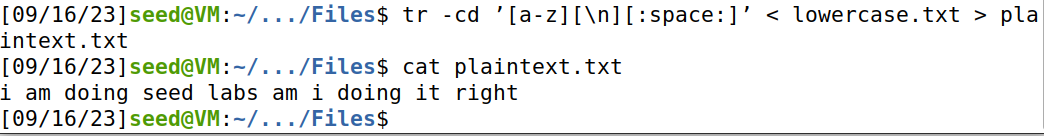
Description automatically generated

Now I changed the Frequency Analysis to read from provided ciphertext.txt and running the script now gave these results.

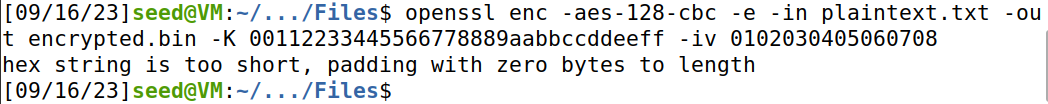


# Task 2

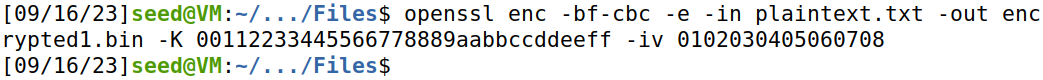
I will be using the plaintext.txt I made in Step 2 of Task 1 and trying encryption using 3 cipher types.



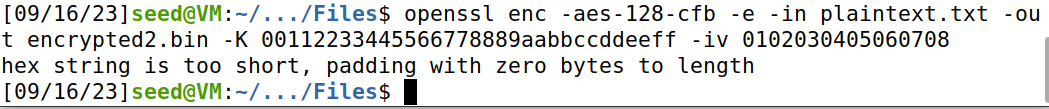
First, I encrypted using aes 128 cbc cipher and stored in encrypted.bin.



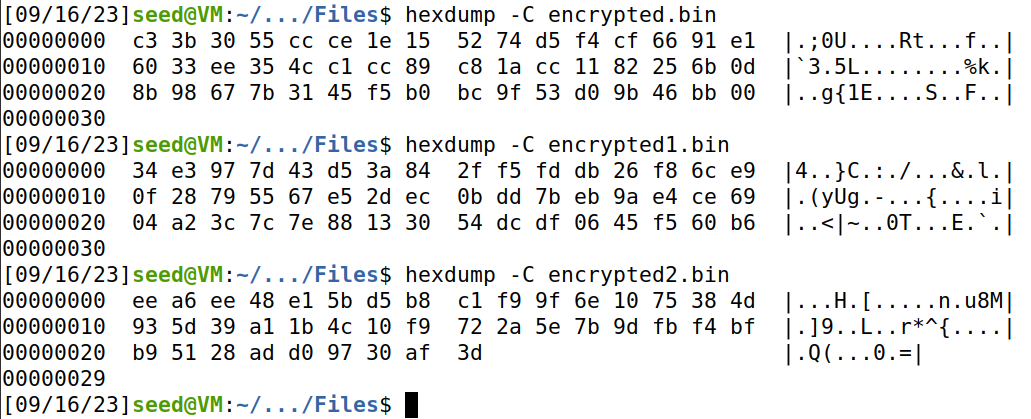
Then I encrypted using aes 128 cbc cipher and stored in encrypted1.bin.



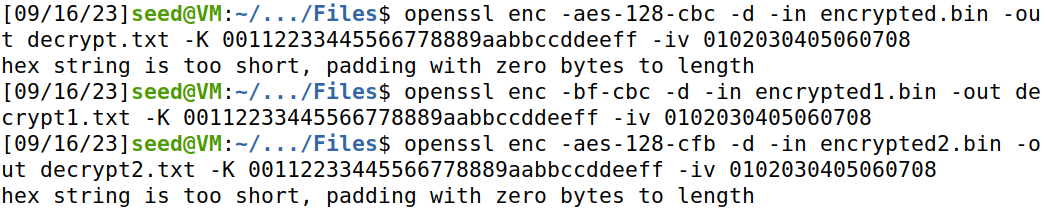
Finally, I encrypted using aes 128 cbc cipher and stored in encrypted2.bin.



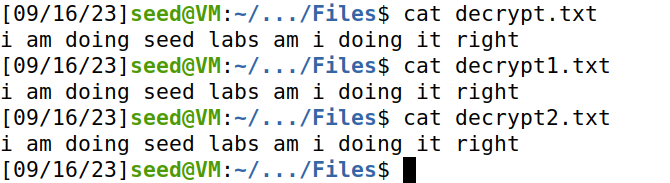
This is what my encrypted files look like using the mentioned ciphers above.



Decrypting the encrypted text for good measure to check if it comes back to the original text which verifies the success of encryption and decryption.



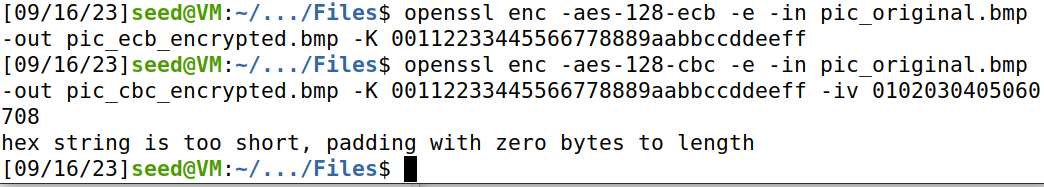
The result for the cipher aes 128 cb, bf cbc and aes 128 cfb are stored in decrypt.txt, decrpyt1.txt and decrypt2.txt, respectively. And the result matches for each cipher encrypted file after decryption with the original plain text.



# Task 3

## Step 1

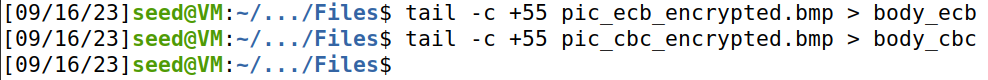
Encrypting original image provided in Labsetup with aes 128 ecb and cbc ciphers and it is worth mentioning aes 128 ecb doesn’t require Initialization Vector known as iv.



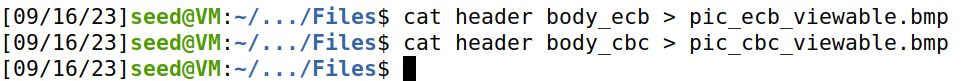
Extracting the Header of the original image file.



Extracting Data from the encrypted images.

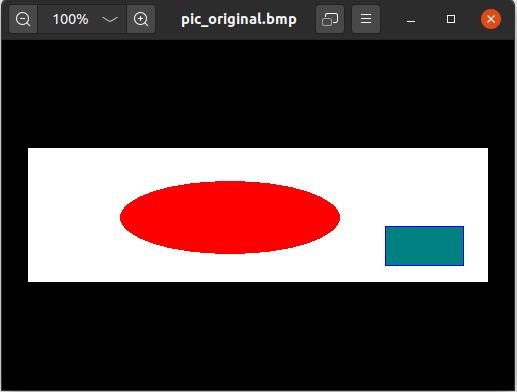


Combining header and body to create a viewable image.



## Step 2

This is the original image.

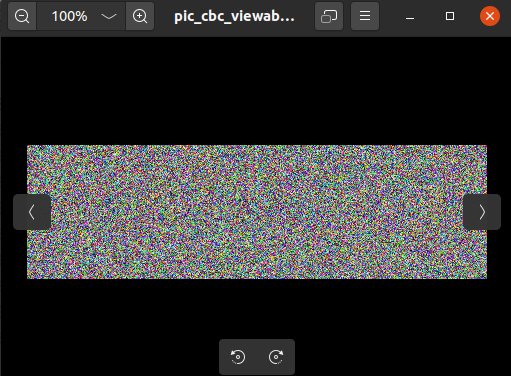


Now the aes 128 ecb image is like this which is blur but reveals the contents of the real image using eob with command *“eog pic\_ecb\_viewable.bmp”.*

A screenshot of a computer

Description automatically generated

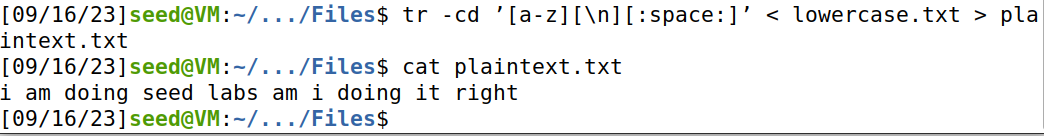
Now the aes 128 cbc image is like this which doesn’t reveal any details regarding the original image as seen by the naked eye by using eob with command *“eog pic\_cbc\_viewable.bmp”.*



# Task 4

## Step 1

I’ll be using again plaintext.txt I created earlier.



Now I will be encrypting in different modes.

### ECB Mode

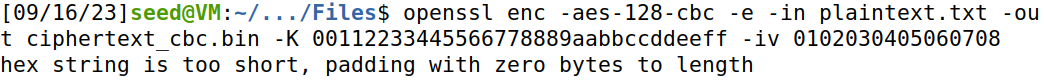
ECB Mode doesn’t require iv.



Padding is not done in this mode as size is a multiple of 16 bytes which is 128 bits in AES. Moreover, this mode uses padding because it uses fixed size.



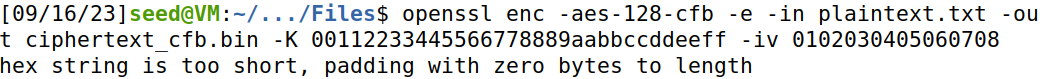
### CBC Mode



Like ECB this mode also requires padding due to requirement of fixed size.



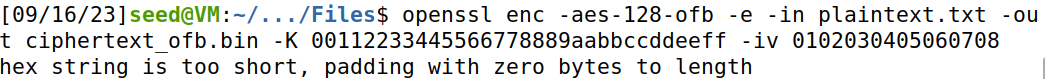
### CFB Mode



Padding isn’t done as this mode turns block cipher into a stream cipher, so it doesn't require padding. It operates on smaller units than the block size and can handle plaintext of any length.



### OFB Mode

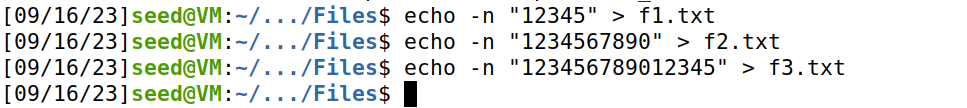


OFB also turns a block cipher into a stream cipher and doesn't require padding. It generates a stream of key bits that XOR with the plaintext, so the length of the plaintext can vary.

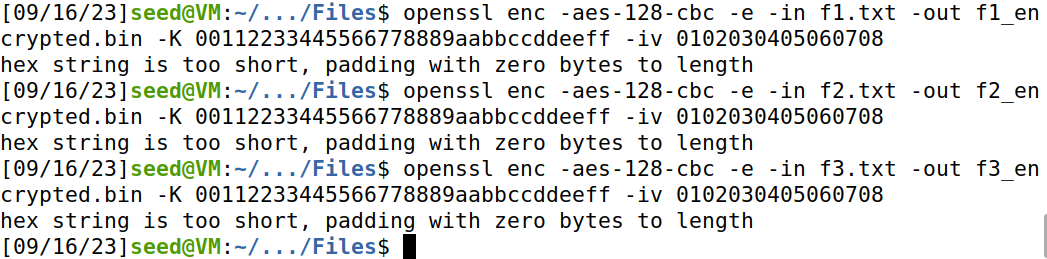


## Step 2

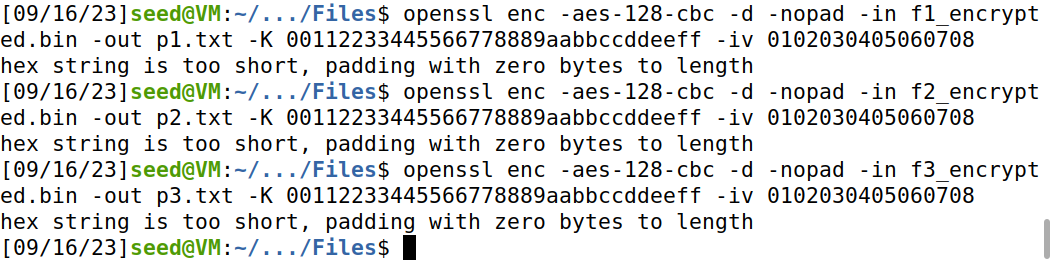
I created three files of size 5, 10 and 15 bytes respectively.



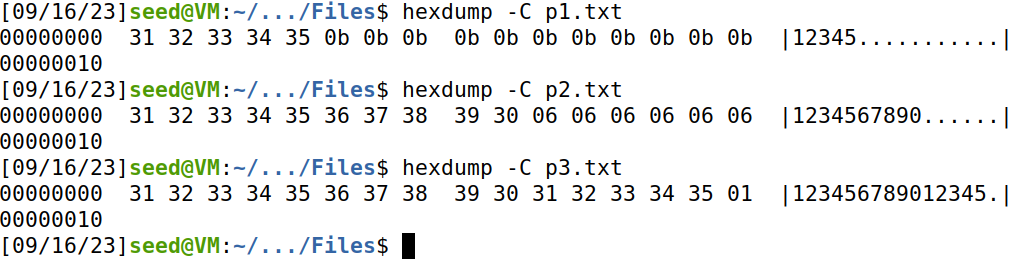
I encrypted all of the created files with AES 128 CBC cipher.



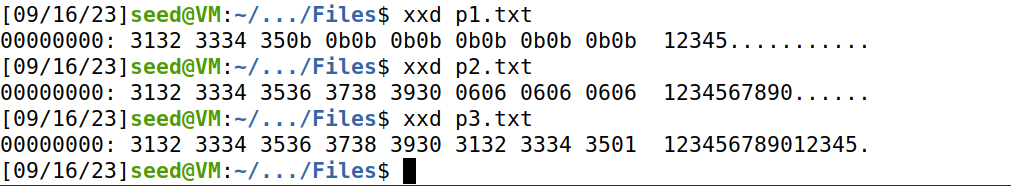
Decrypted the file with -nopad to prevent automatic padding removal during decryption and stored in the files p1.txt, p2.txt and p3.txt, respectively.



Now checking the data in all files using hexdump.



Checking the data in all files using xxd.



**NOTE:** I used hexdump and xxd tools to display the results and padding represented by dots which as whole output is not representable in output files normally as shown in the screenshot below the padding is not displayed without hex tools.

