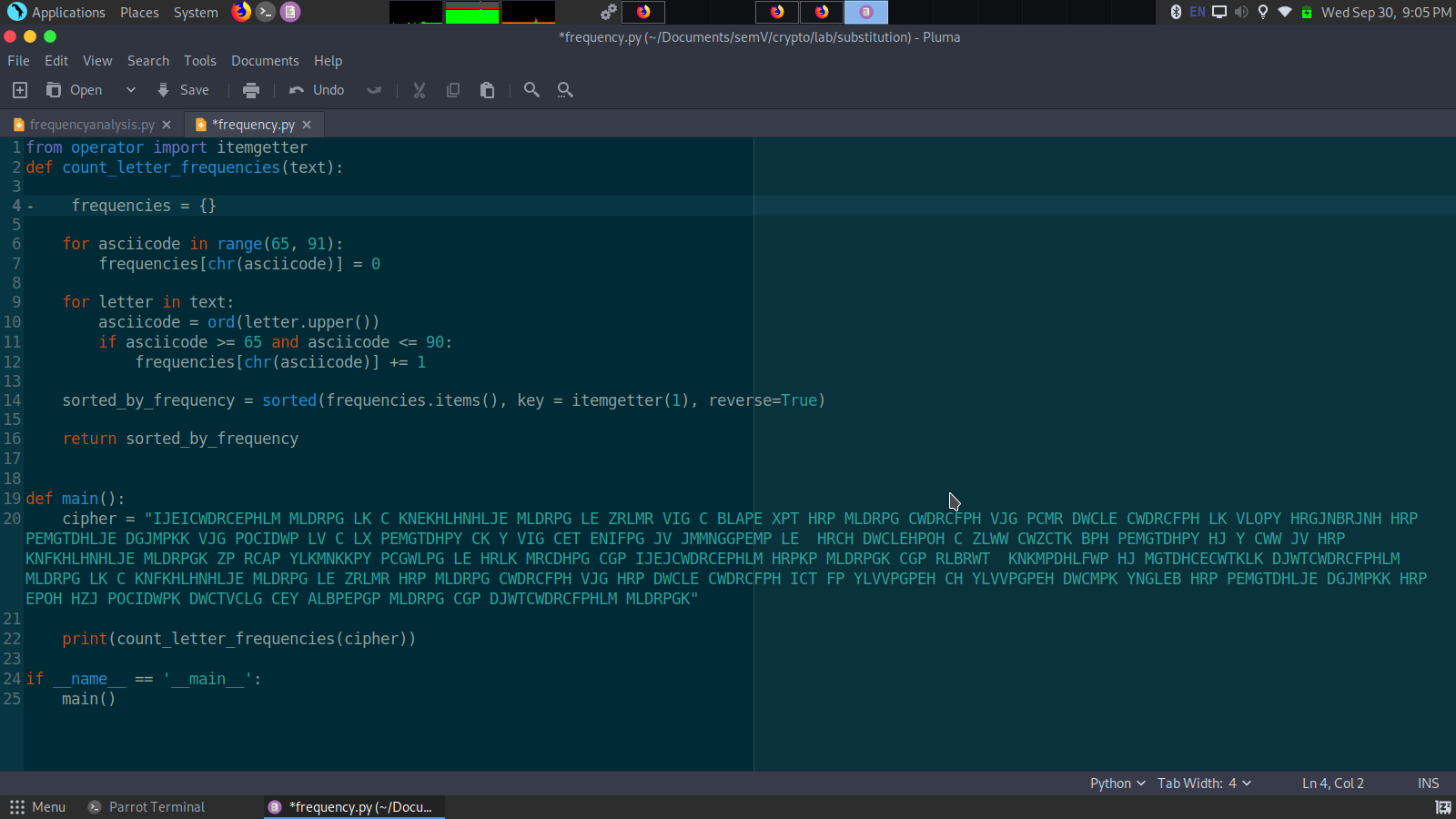
Task1 . Frequency Analysis Against Mono-alphabetic Substitution Cipher

Wrote my own python code to find frequency 

Cipher text:

IJEJCWDRCFPHLM MLDRPG LK C KNFKHLHNHLJE MLDRPG LE ZRLMR VJG C BLAPE XPT HRP MLDRPG

CWDRCFPH VJG PCMR DWCLE CWDRCFPH LK VLOPY HRGJNBRJNH HRP PEMGTDHLJE DGJMPKK VJG

POCIDWP LV C LK PEMGTDHPY CK Y VJG CET ENIFPG JV JMMNGGPEMP LE HRCH DWCLEHPOH C

ZLWW CWZCTK BPH PEMGTDHPY HJ Y

CWW JV HRP KNFKHLHNHLJE MLDRPGK ZP RCAP YLKMNKKPY PCGWLPG LE HRLK MRCDHPG CGP

IJEJCWDRCFPHLM HRPKP MLDRPGK CGP RLBRWT KNKMPDHLFWP HJ MGTDHCECWTKLK

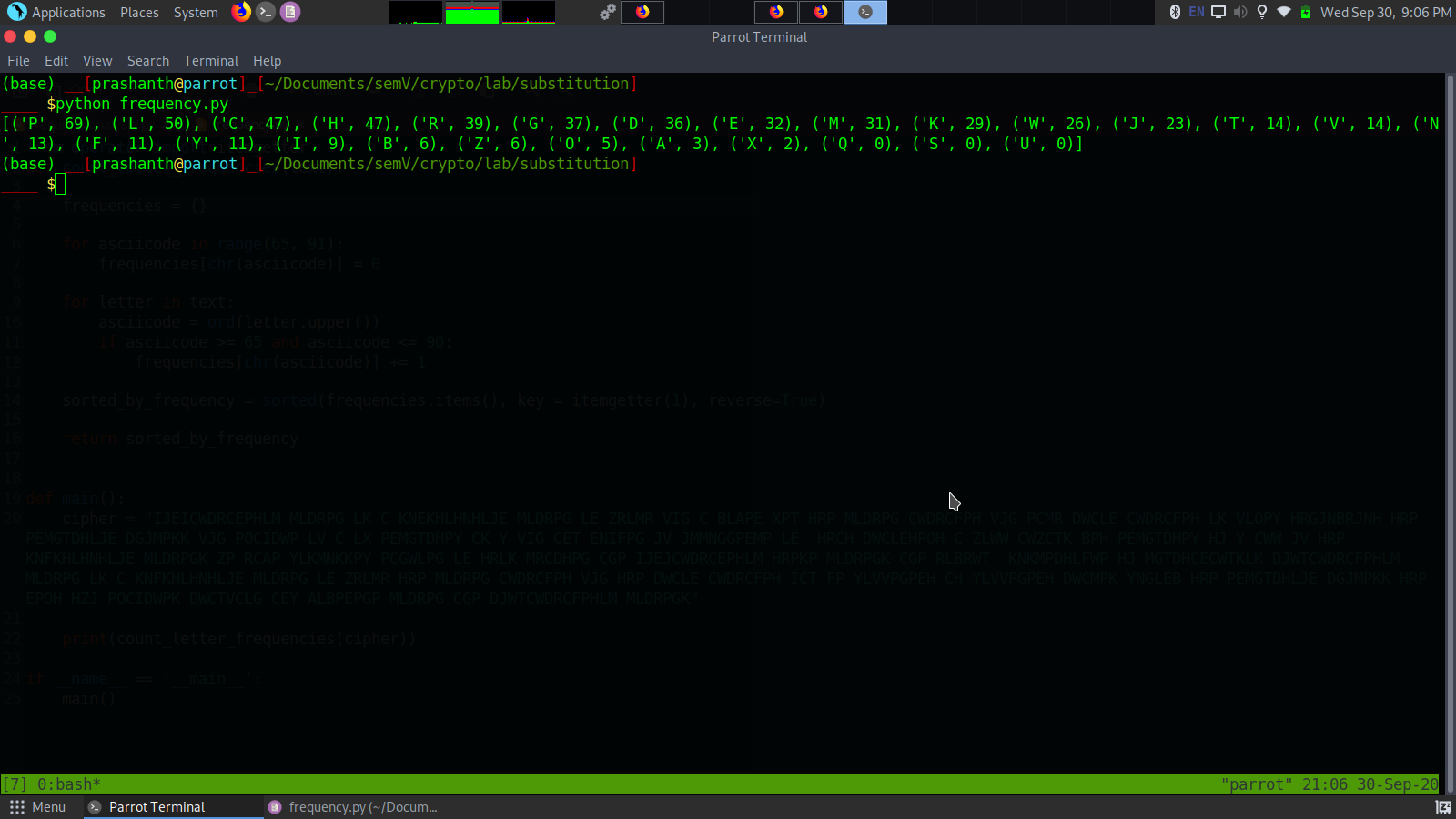
DJWTCWDRCFPHLM MLDRPG LK C KNFKHLHNHLJE MLDRPG LE ZRLMR HRP MLDRPG CWDRCFPH VJG

HRP DWCLE CWDRCFPH ICT FP YLVVPGPEH CH YLVVPGPEH DWCMPK YNGLEB HRP PEMGTDHLJE

DGJMPKK HRP EPOH HZJ POCIDWPK DWCTVCLG CEY ALBPEPGP MLDRPG CGP DJWTCWDRCFPHLM

MLDRPGK

Frequency analysis output



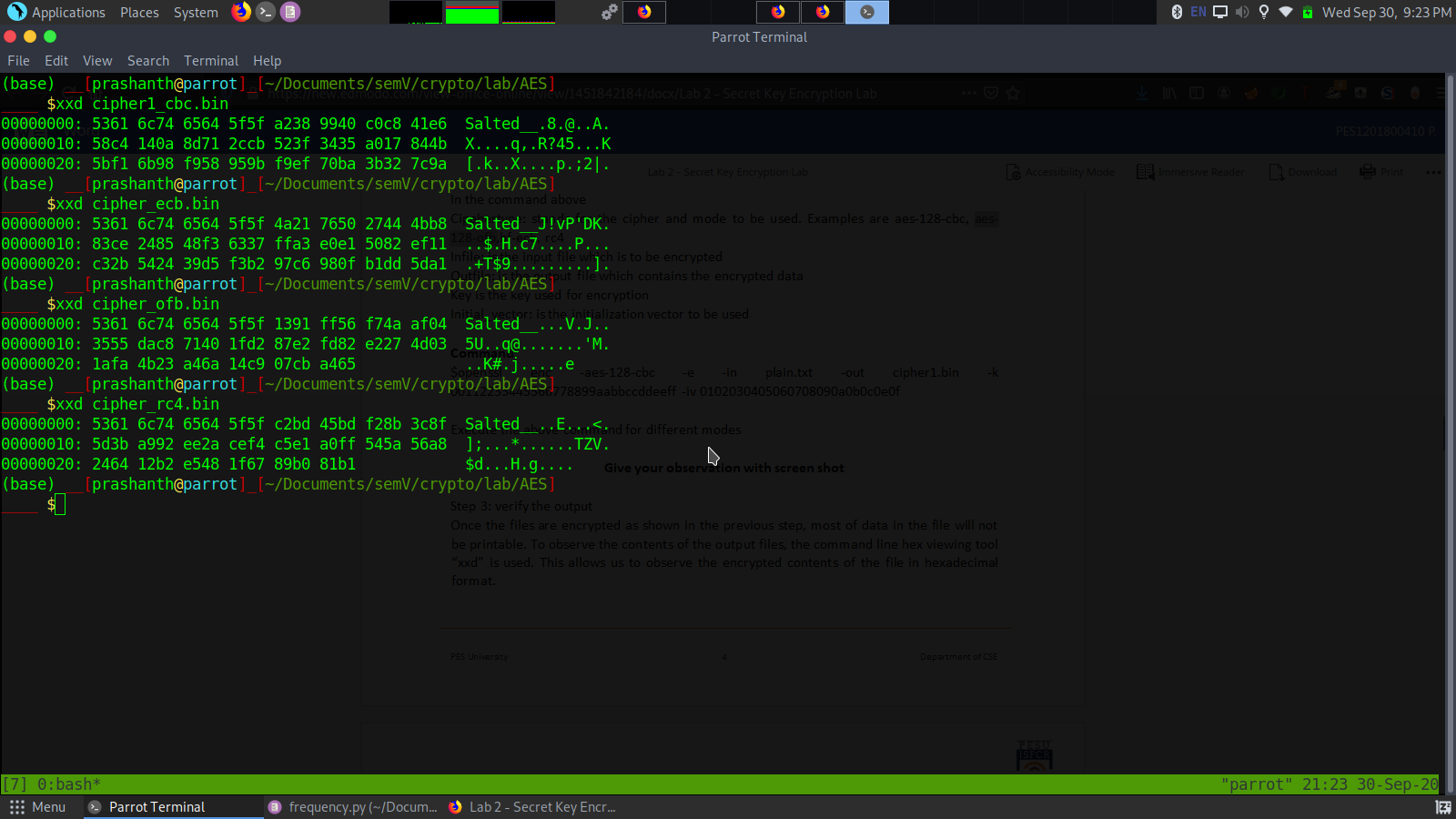
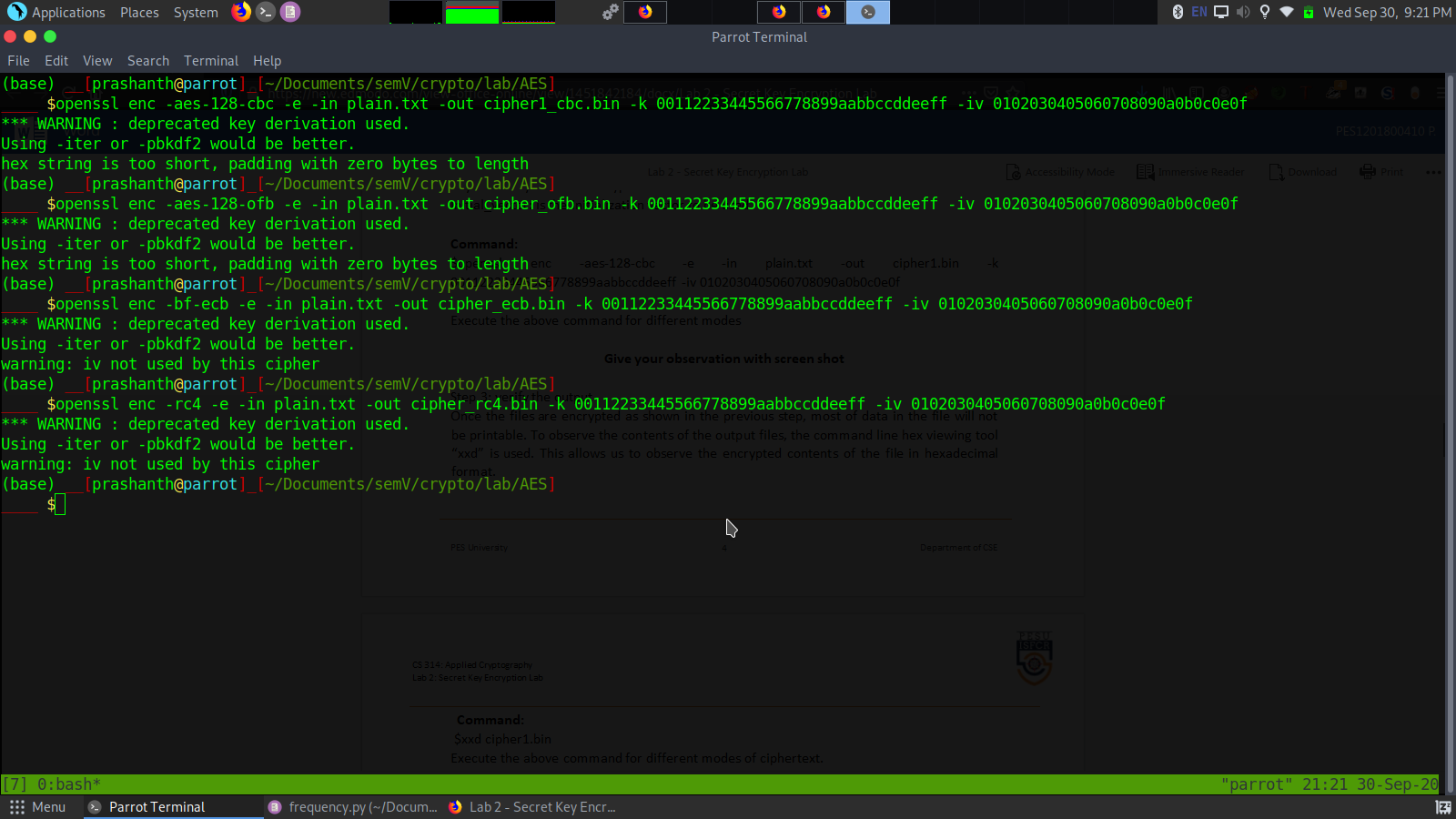
Plaintext:

monoalphabetic cipher is a substitution cipher in which for a given key the cipher alphabet for each plain alphabet is fixed throughout the encryption process for example if a is encrypted as d for any number of occurrence in that plaintext a will always get encrypted to d

all of the substitution ciphers we have discussed earlier in this chapter are monoalphabetic these ciphers are highly susceptible to cryptanalysis

polyalphabetic cipher is a substitution cipher in which the cipher alphabet for the plain alphabet may be different at different places during the encryption process the next two examples playfair and vigenere cipher are polyalphabetic ciphers

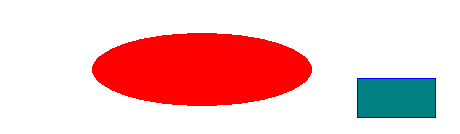
Task2 Encryption using different ciphers and modes



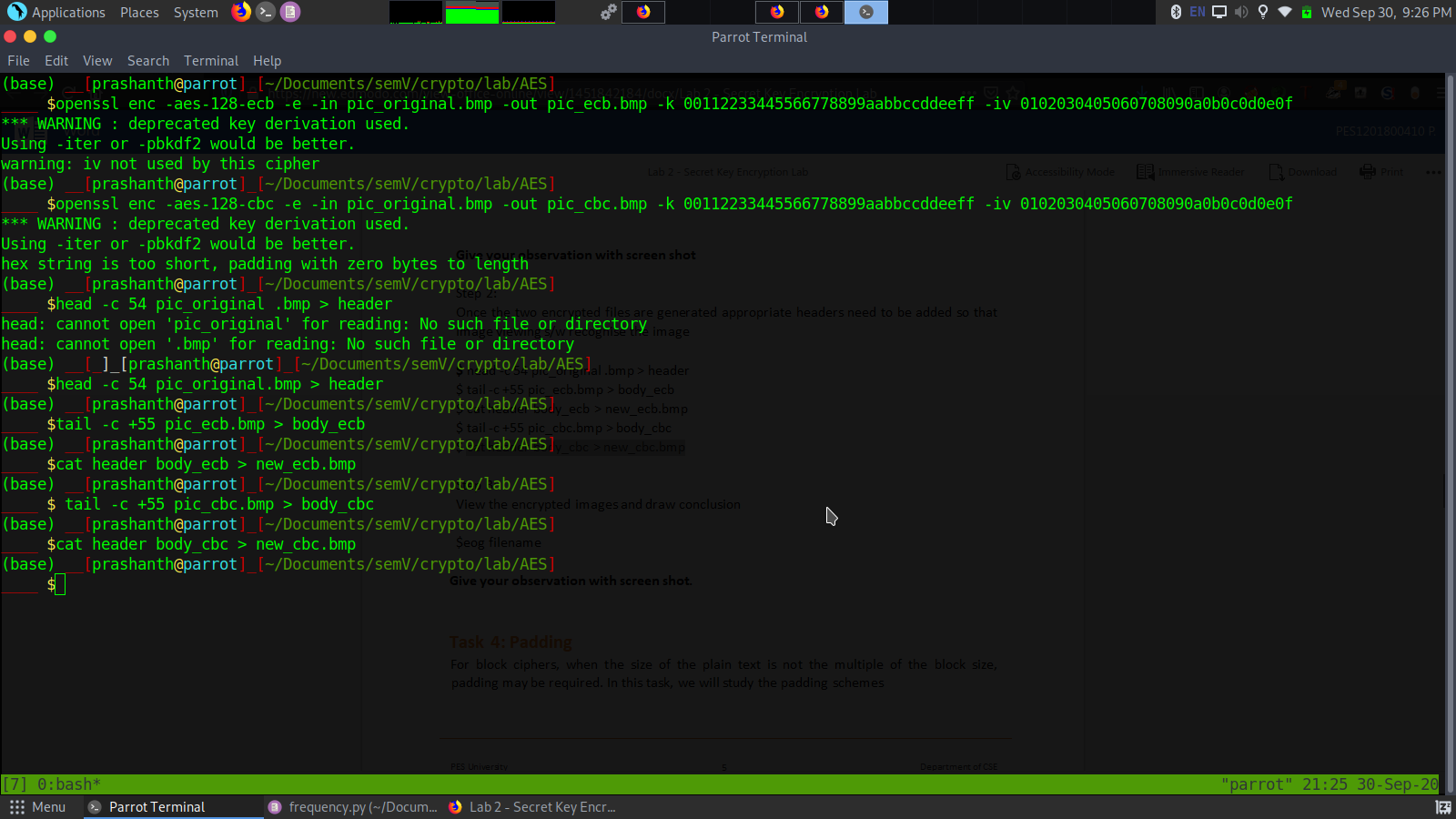
Observation: There is no similarities from plaintext and ciphertext.

Task3: Encryption Mode ECB vs CBC

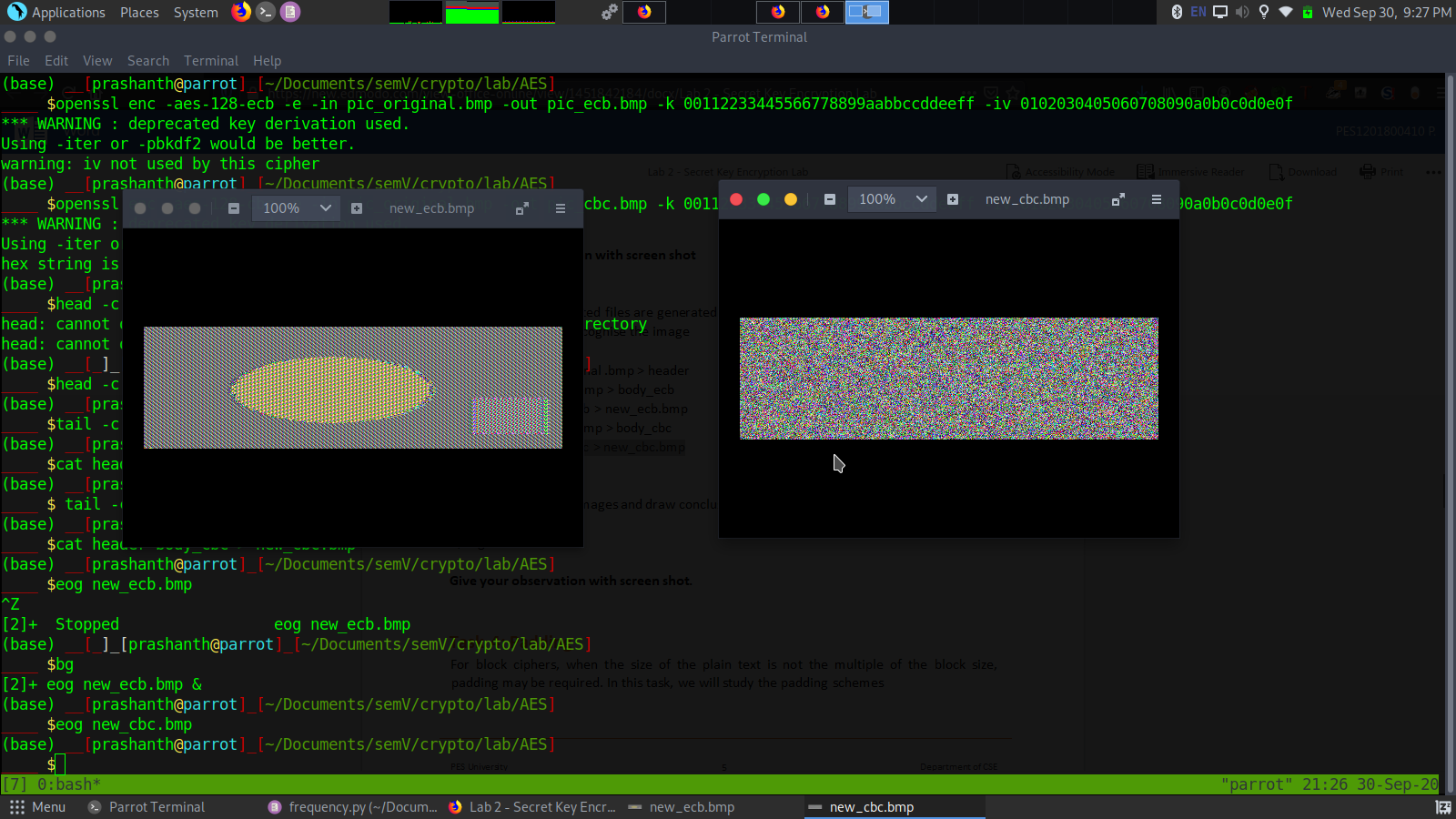
Original image



Procedure



Ecb vs cbc



Observation

In ecb mode we can make out something about the plaintext. But in CBC we can’t make out anything from the cipher text .

Task4: Padding

I wrote a shell script to do my job simple

Task4.sh

echo -n "12345">f1.txt

echo -n " 1234567890">f2.txt

echo -n "1234567890abcde">f3.txt

ls -l f\*.txt

openssl enc -aes-128-cbc -e -in f1.txt -out f1.bin -k 001122334455667788899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f

openssl enc -aes-128-cbc -e -in f2.txt -out f2.bin -k 001122334455667788899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f

openssl enc -aes-128-cbc -e -in f3.txt -out f3.bin -k 001122334455667788899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f

openssl enc -aes-128-cbc -d -in f1.bin -out p1.txt -nopad -k 001122334455667788899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f

openssl enc -aes-128-cbc -d -in f2.bin -out p2.txt -nopad -k 001122334455667788899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f

openssl enc -aes-128-cbc -d -in f3.bin -out p3.txt -nopad -k 001122334455667788899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f

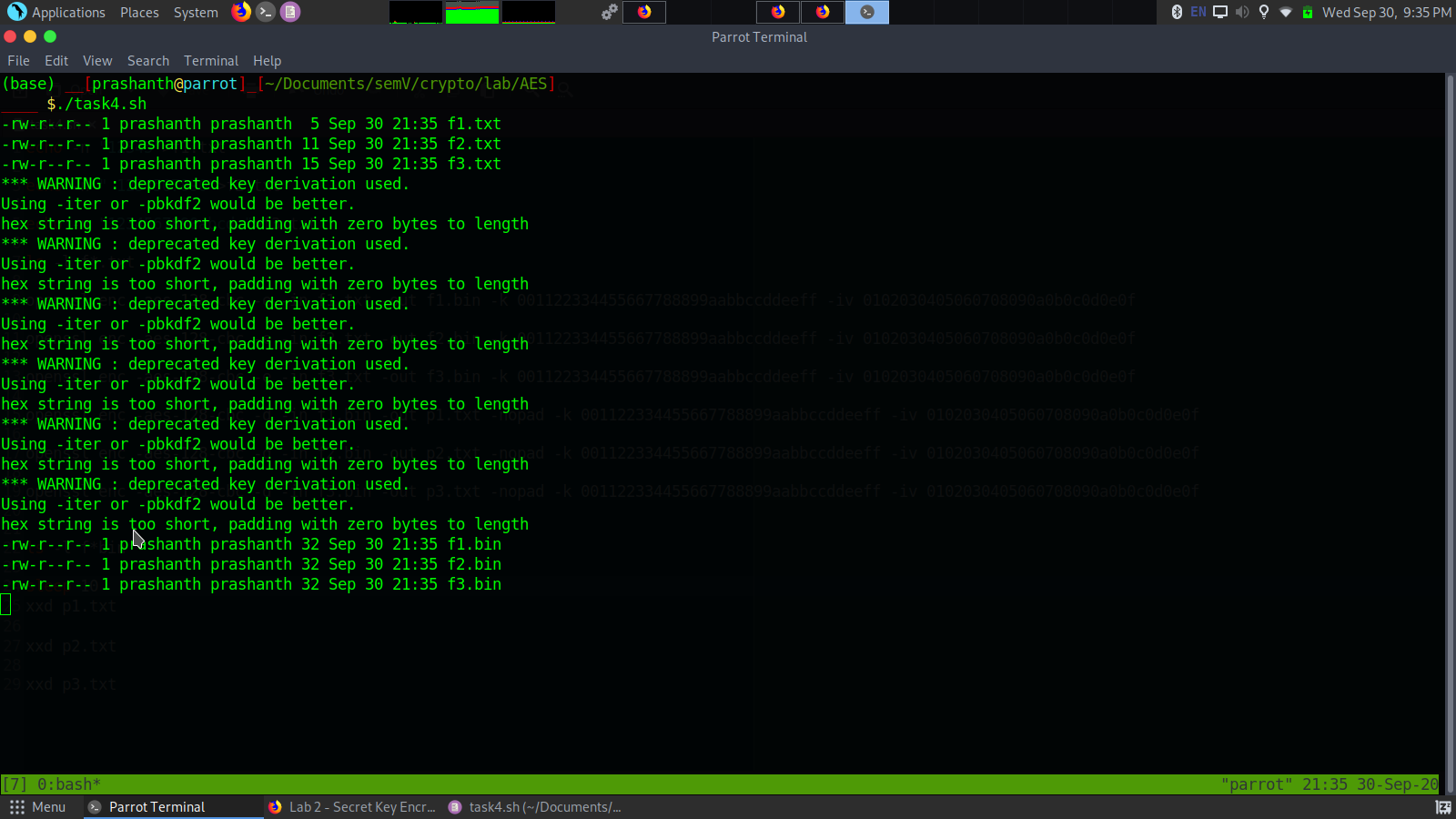
ls -l f\*bin

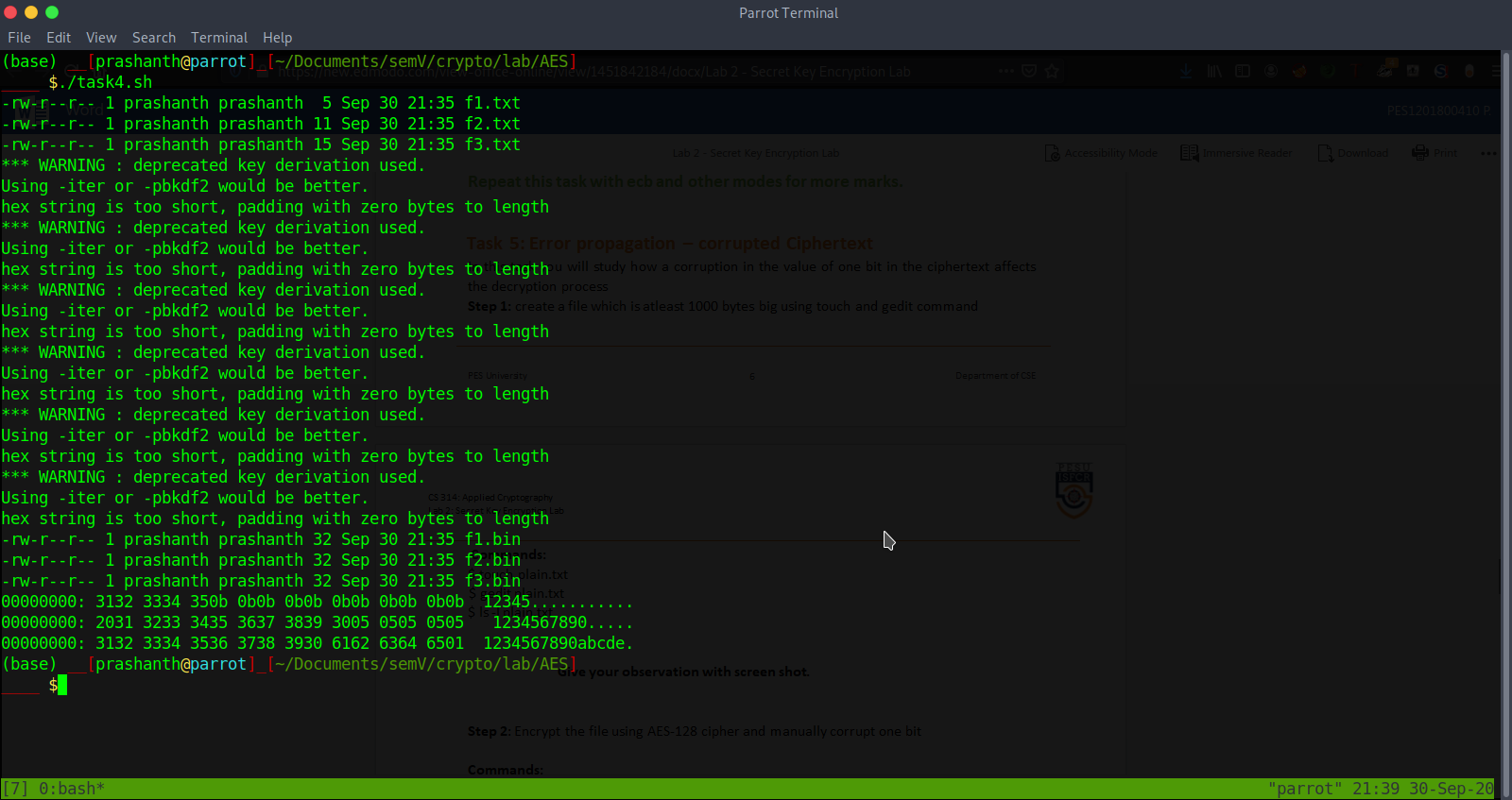
sleep 10

xxd p1.txt

xxd p2.txt

xxd p3.txt

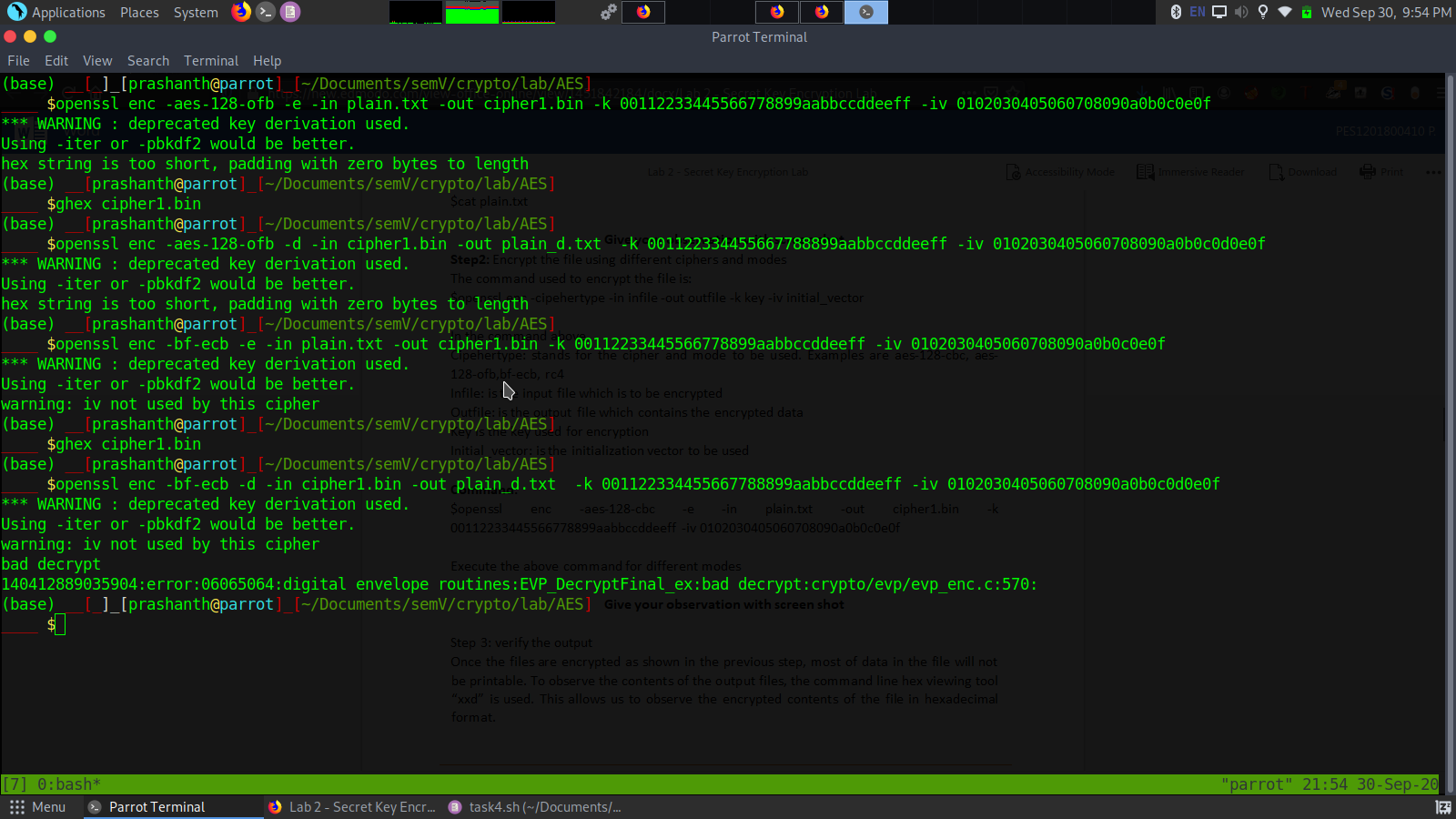


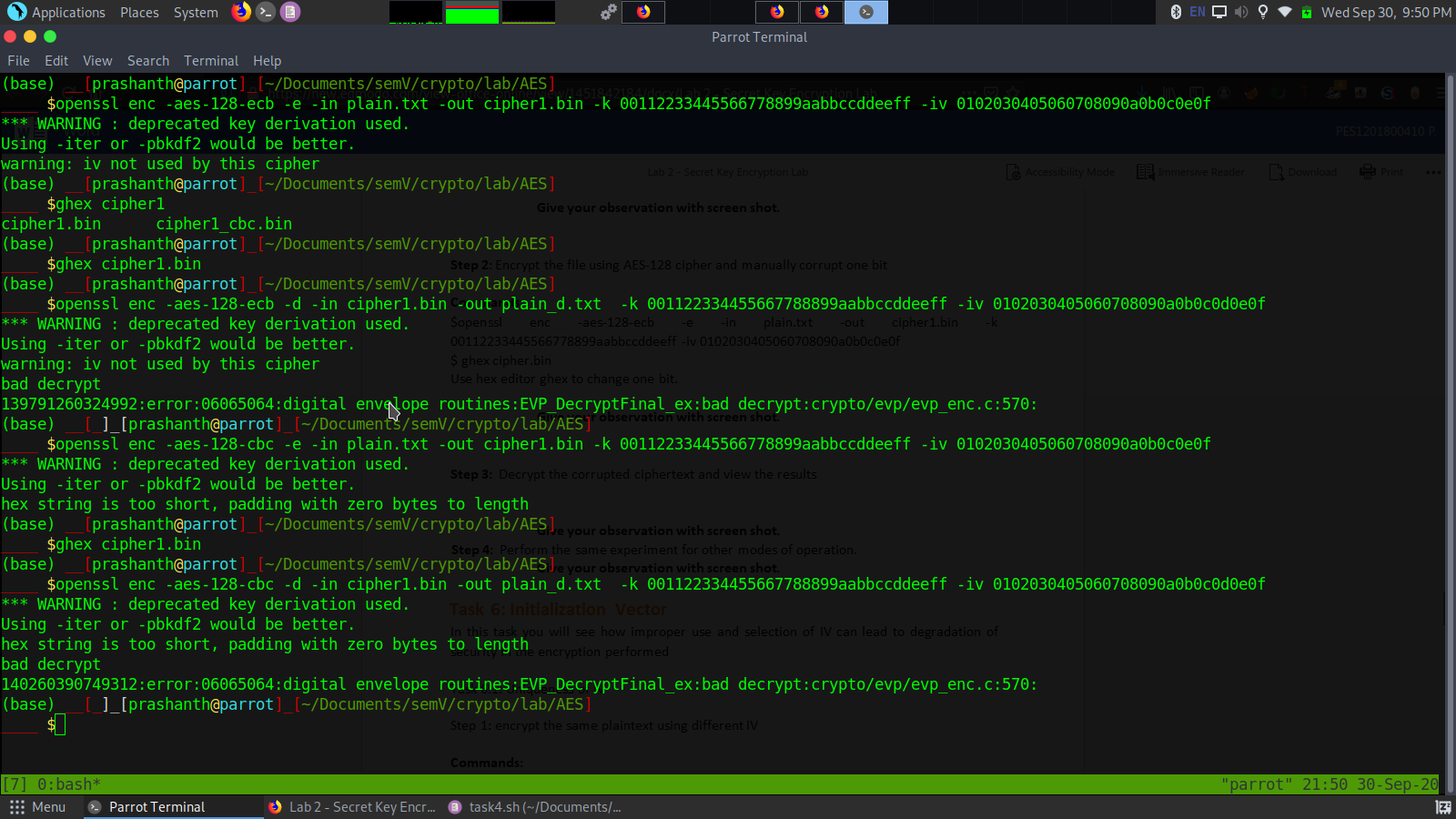


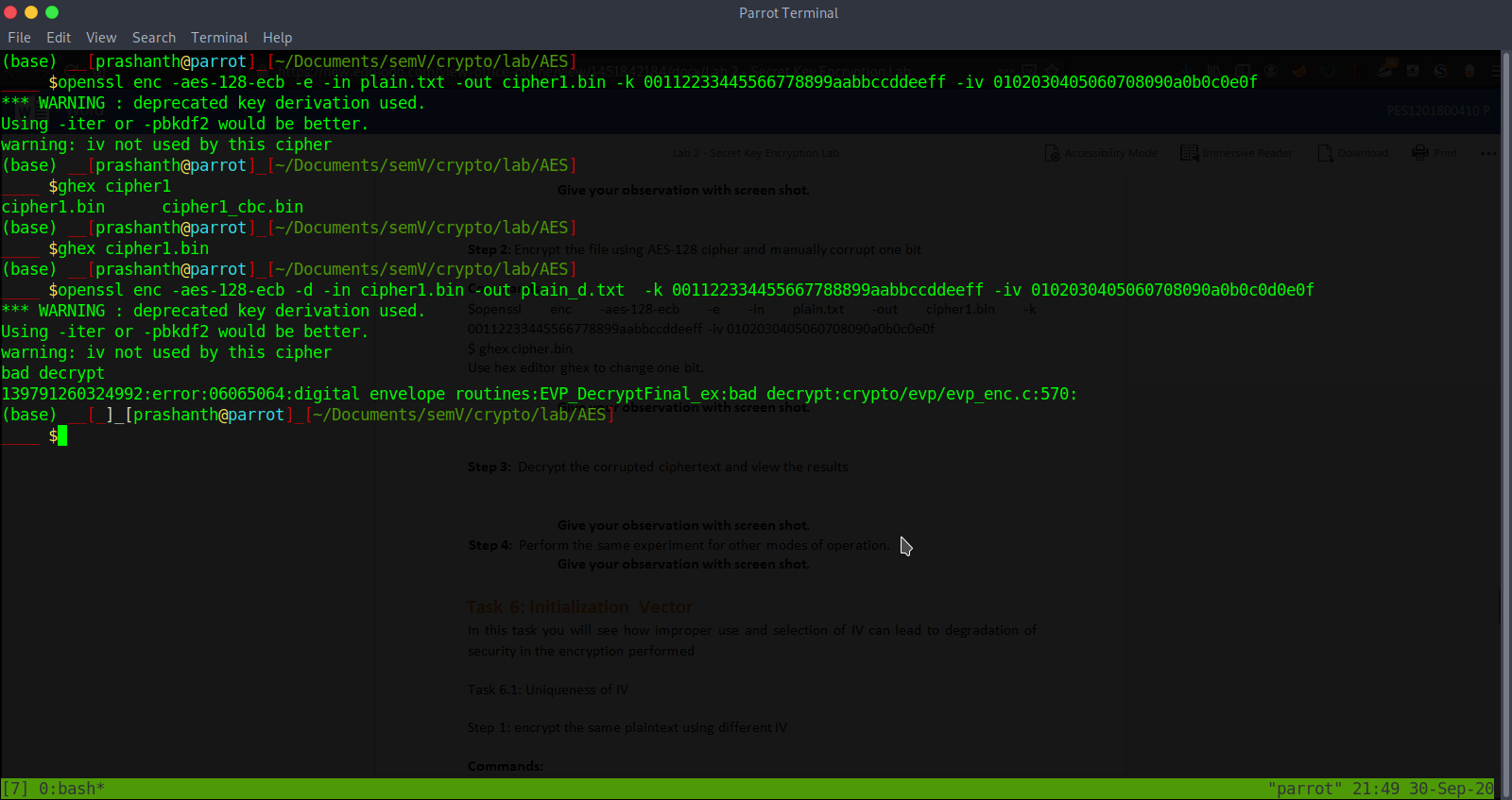
Observation

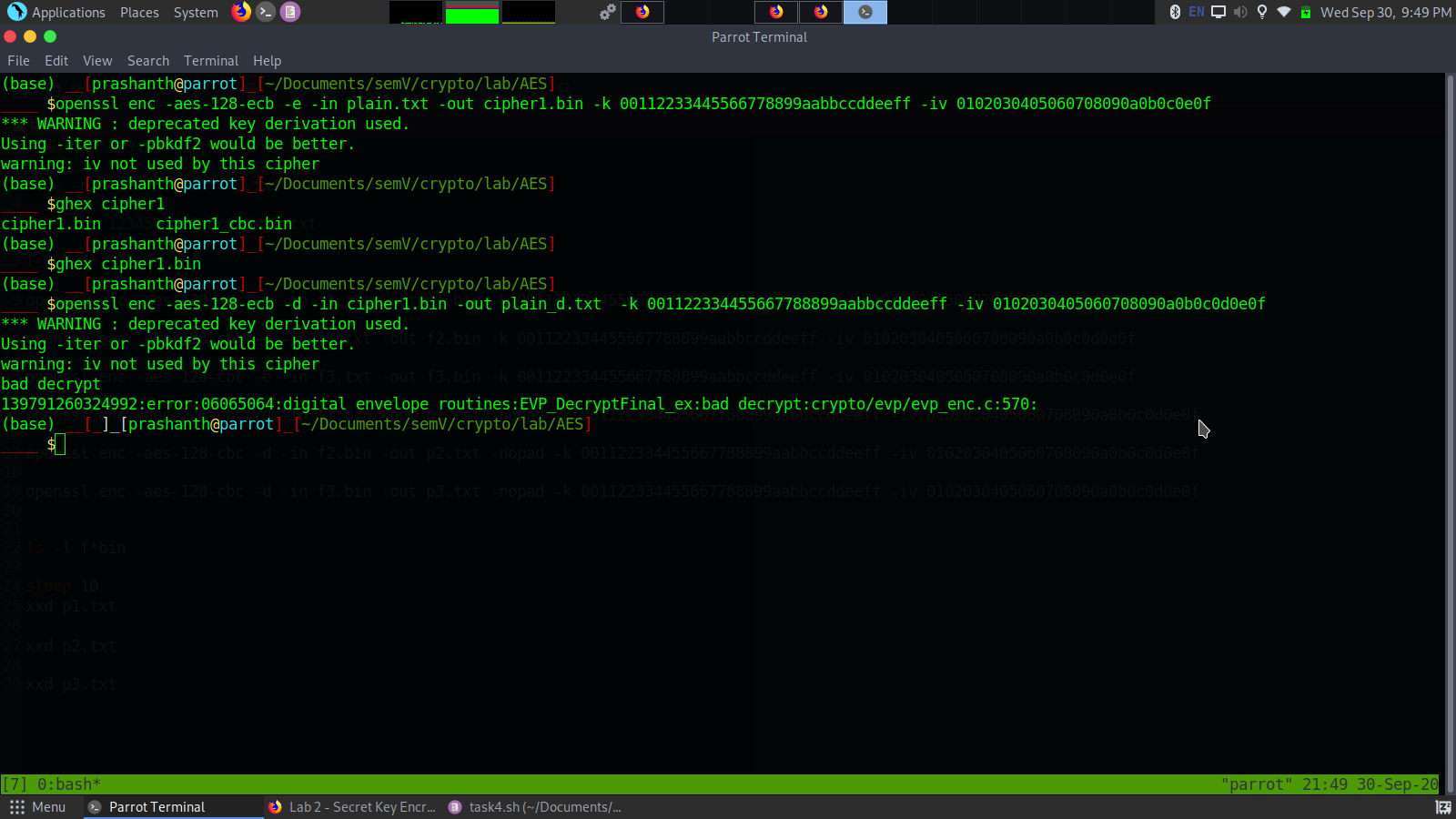
We can see some dummy padding on every plain text after decryption because of –nopad option mentioned in while decrypting

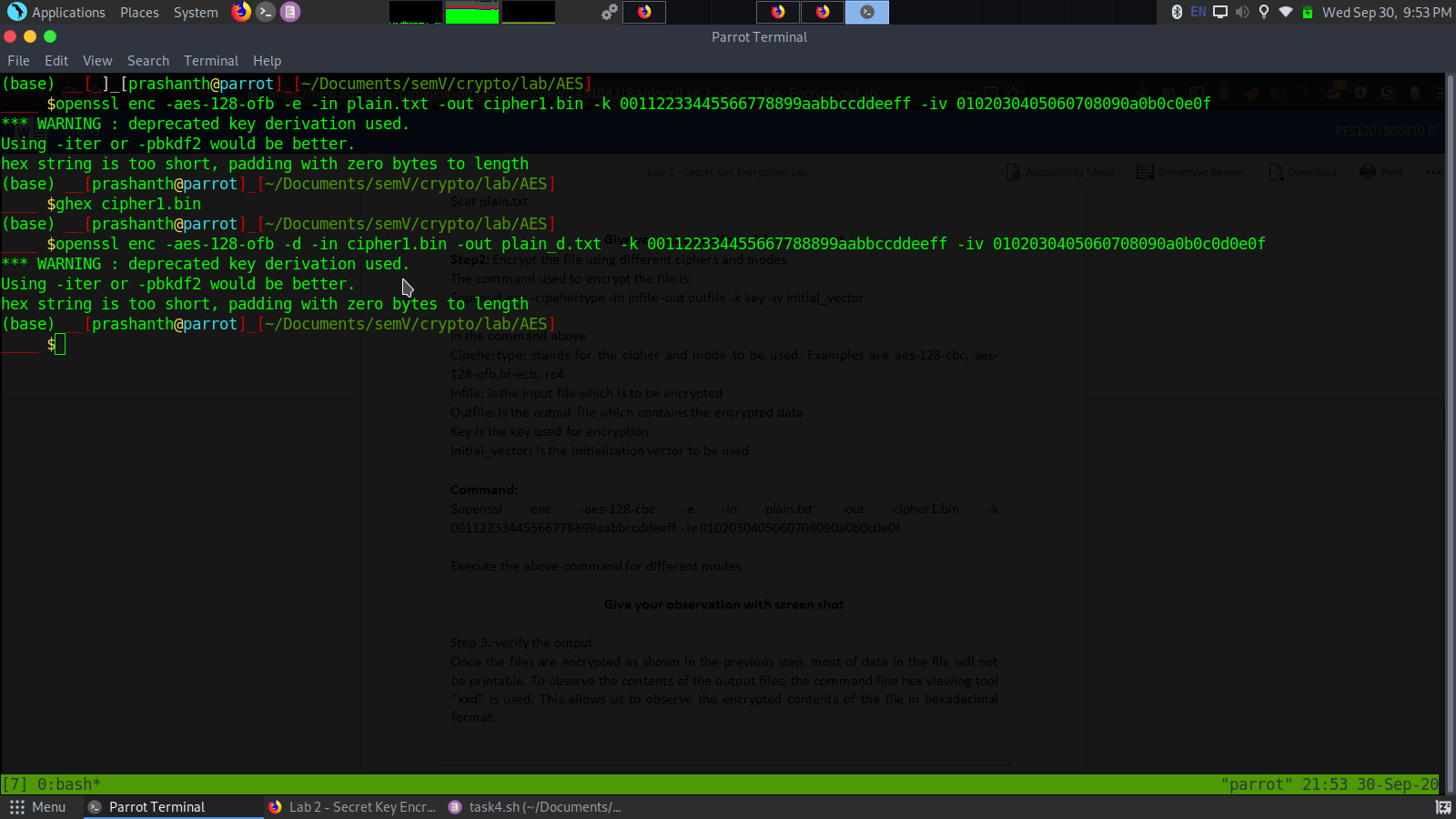
Task 5: Error propagation – corrupted Ciphertext

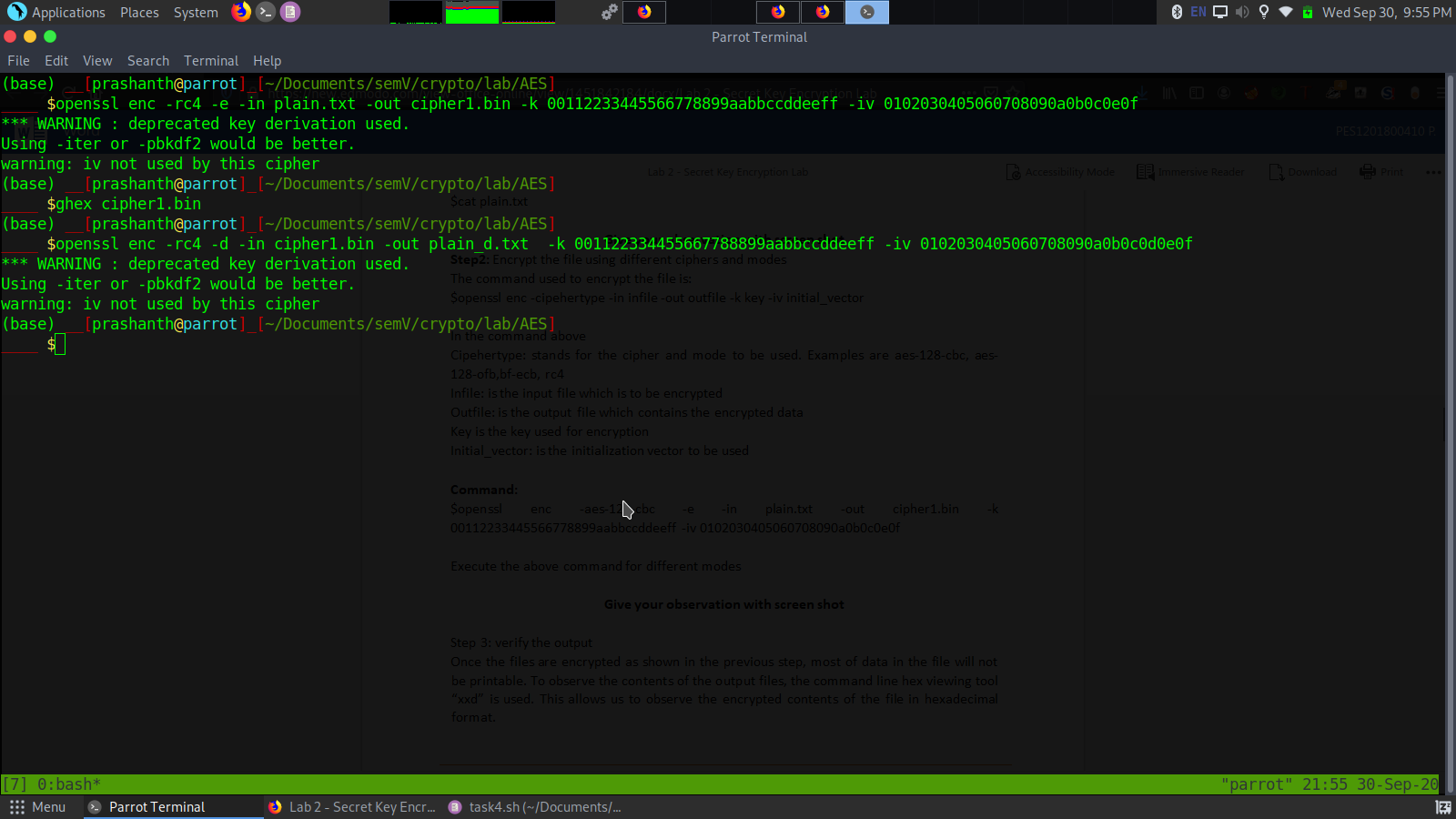










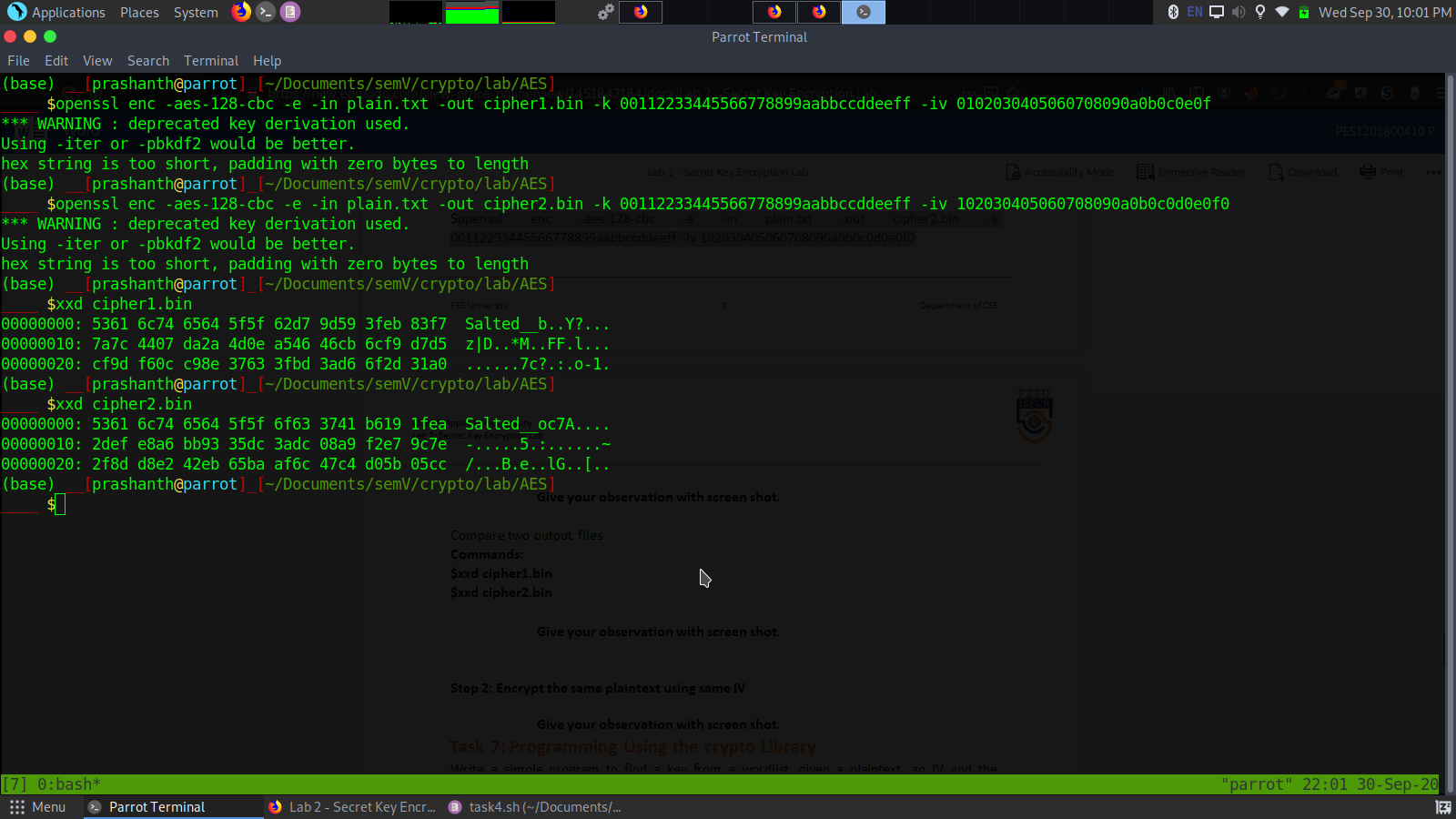


Observation

We can see rc4 and aes-ofb do not go through the error detection . Other 3 can make out or detect changes in the cipher text.

Task 6: Initialization Vector

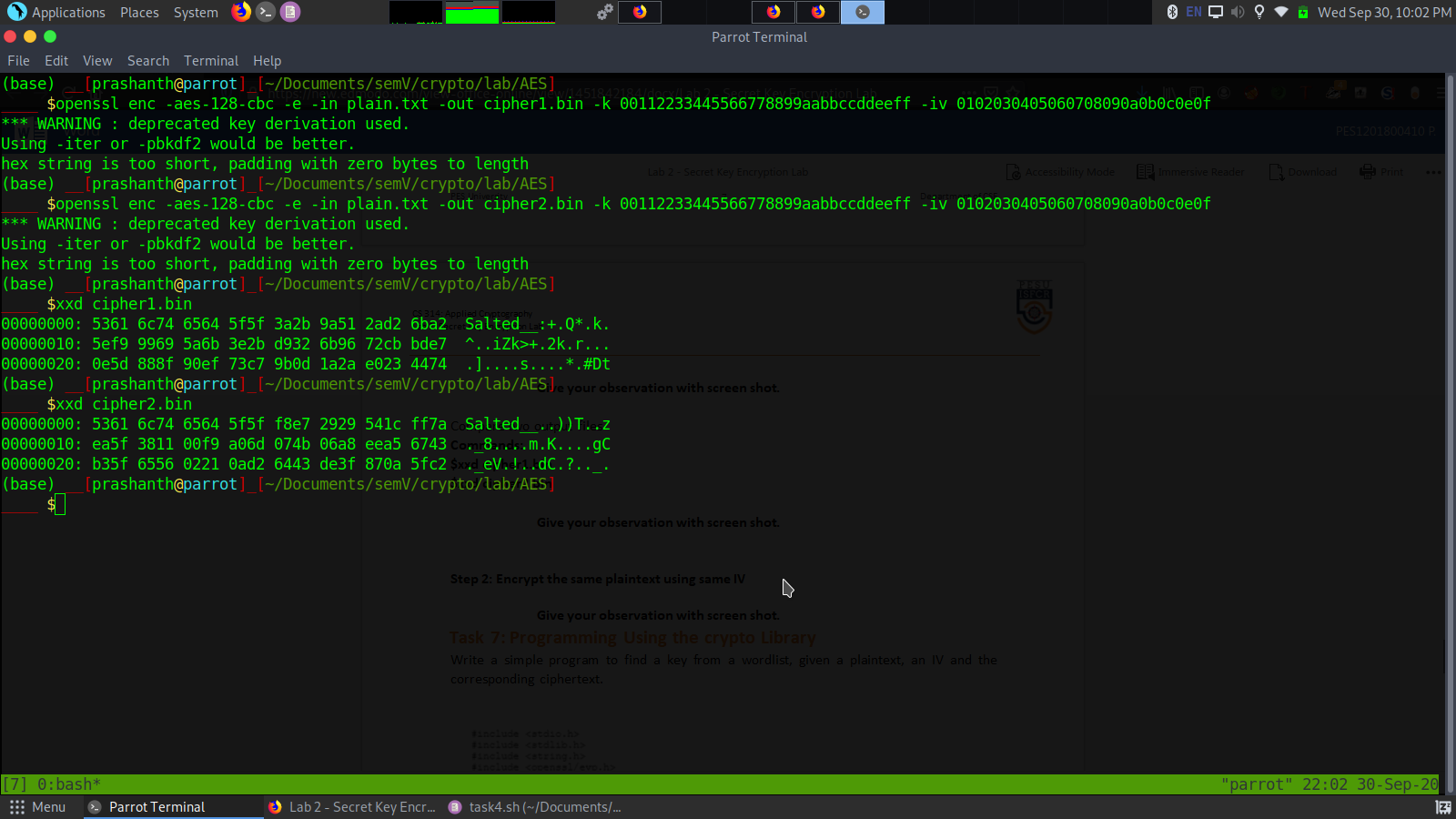
Different IV



Observation

Different IV gives different cipher text

Same IV



With same iV still we are getting different cipher text which is very interesting.

This happens because it uses salt. Which makes hackers difficult to crack.

Task 7: Programming Using the crypto Library

