LABORATORY CEL62: Cryptography and System Security Winter 2021

Experiment 3:	**Crypto Encryption**

Name: Divya Shah
Roll No: 2019130057
TE Comps

Note: Students are advised to read through this lab sheet before doing experiment. On-the-spot evaluation may be carried out during or at the end of the experiment. Your performance, teamwork effort, and learning attitude will count towards the marks.

Experiment 4: Crypto Lab – Secret-Key Encryption

1 OBJECTIVE

The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms, encryption modes, paddings, and initial vector (IV). Moreover, students will be able to use tools and write programs to encrypt/decrypt messages.

2 LAB ENVIRONMENT

Installing OpenSSL. In this lab, we will use openssl commands and libraries. We have already in-stalled openssl binaries in our VM. It should be noted that if you want to use openssl libraries in your programs, you need to install several other things for the programming environment, including the header files, libraries, manuals, etc. We have already downloaded the necessary files under the directory openssl-1.0.1. To configure and install openssl libraries, run the following commands.

```
You should read the INSTALL file first:

% ./config
% make
% make test
% sudo make install
```

Installing GHex. In this lab, we need to be able to view and modify files of binary format. We have installed in our VM GHex a hex editor for GNOME. It allows the user to load data from any file, view and edit it in either hex or ascii.

3 LAB TASKS

3.1 Task 1: Encryption using different ciphers and modes

In this task, we will play with various encryption algorithms and modes. You can use the following openssl enc command to encrypt/decrypt a file. To see the manuals, you can type man openssl and man

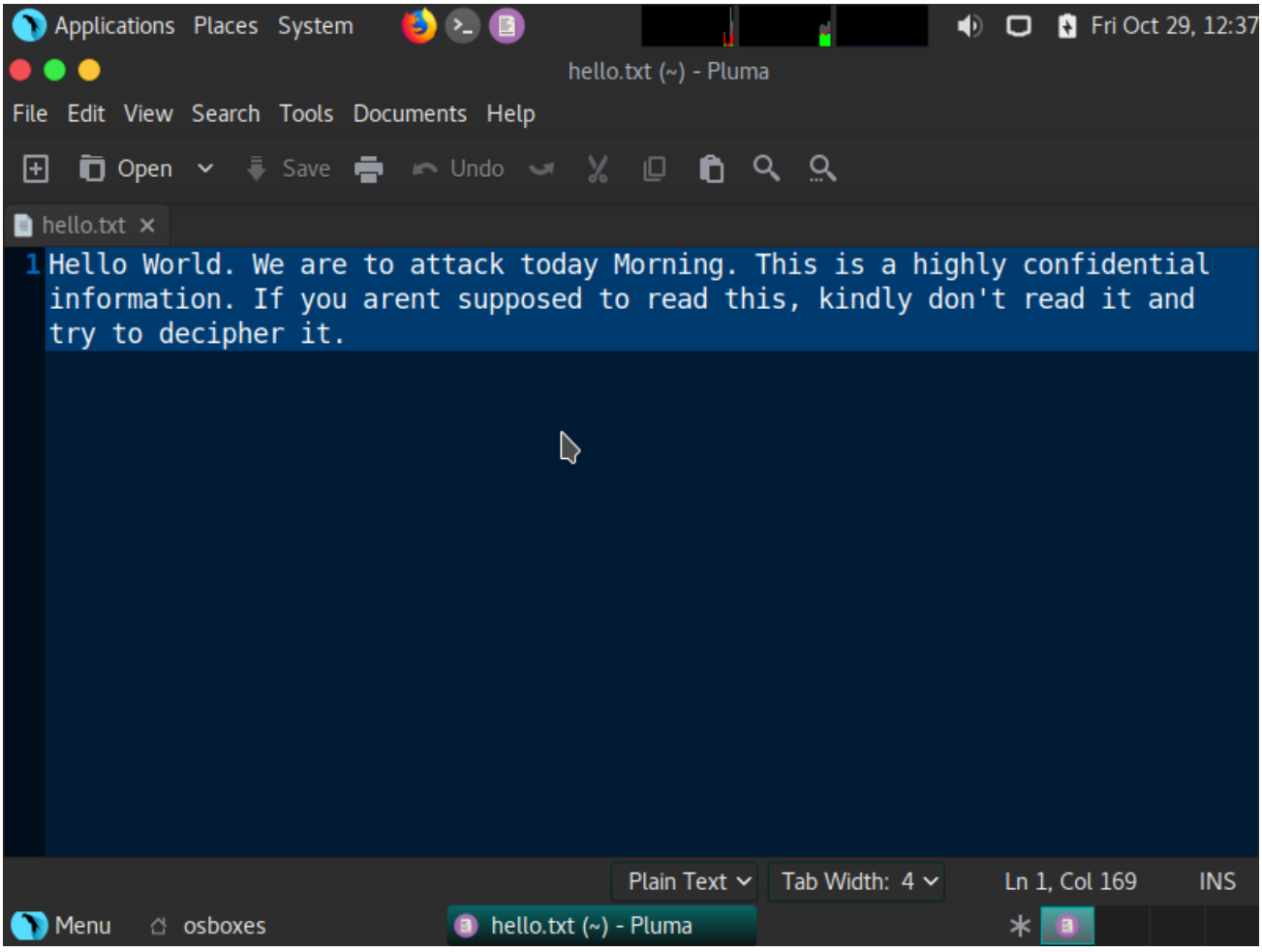
```
% openssl enc ciphertype -e -in plain.txt -out
                                cipher.bin \ -K
        00112233445566778889aabbccddeeff \
-iv 0102030405060708
```

enc.

Please replace the ciphertype with a specific cipher type, such as -aes-128-cbc, -aes-128-cfb, -bf-cbc, etc. In this task, you should try at least 3 different ciphers and three different modes. You can find the meaning of the command-line options and all the supported cipher types by typing "man enc". We include some common options for the openssl enc command in the following:

-in <file>	input file
-out <file>	output file
-e	encrypt
-d	decrypt
-K/-iv	key/iv in hex is the next argument
-[pP]	print the iv/key (then exit if -P)

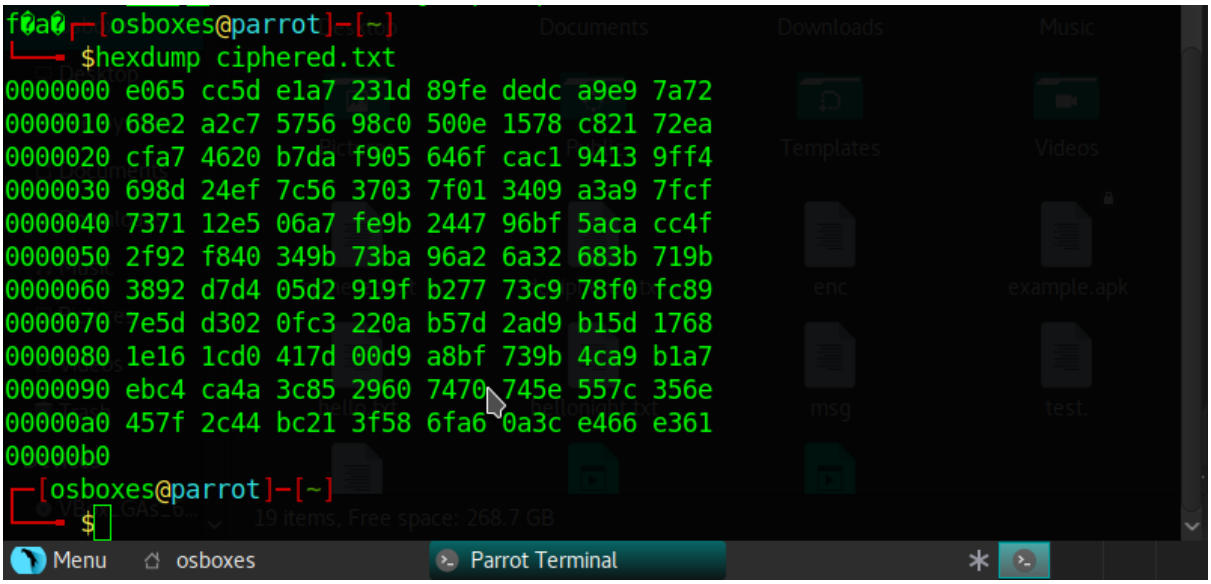
Encryption of plain text using different Ciphers.
The plain text is as follows:



1.Encryption using aes-128-ecb

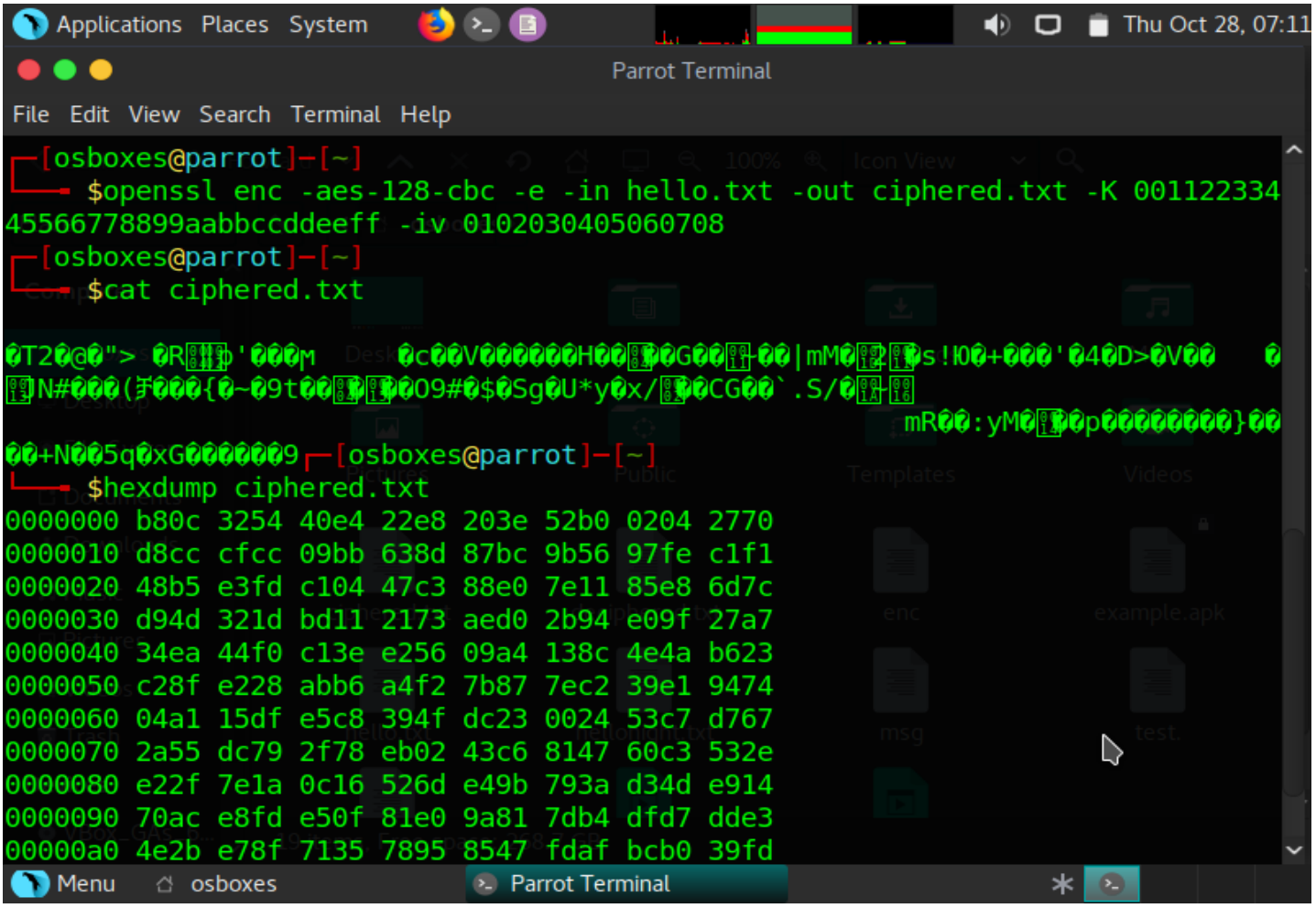
The Encrypted text is :

The hexdump for the above cipher text is:



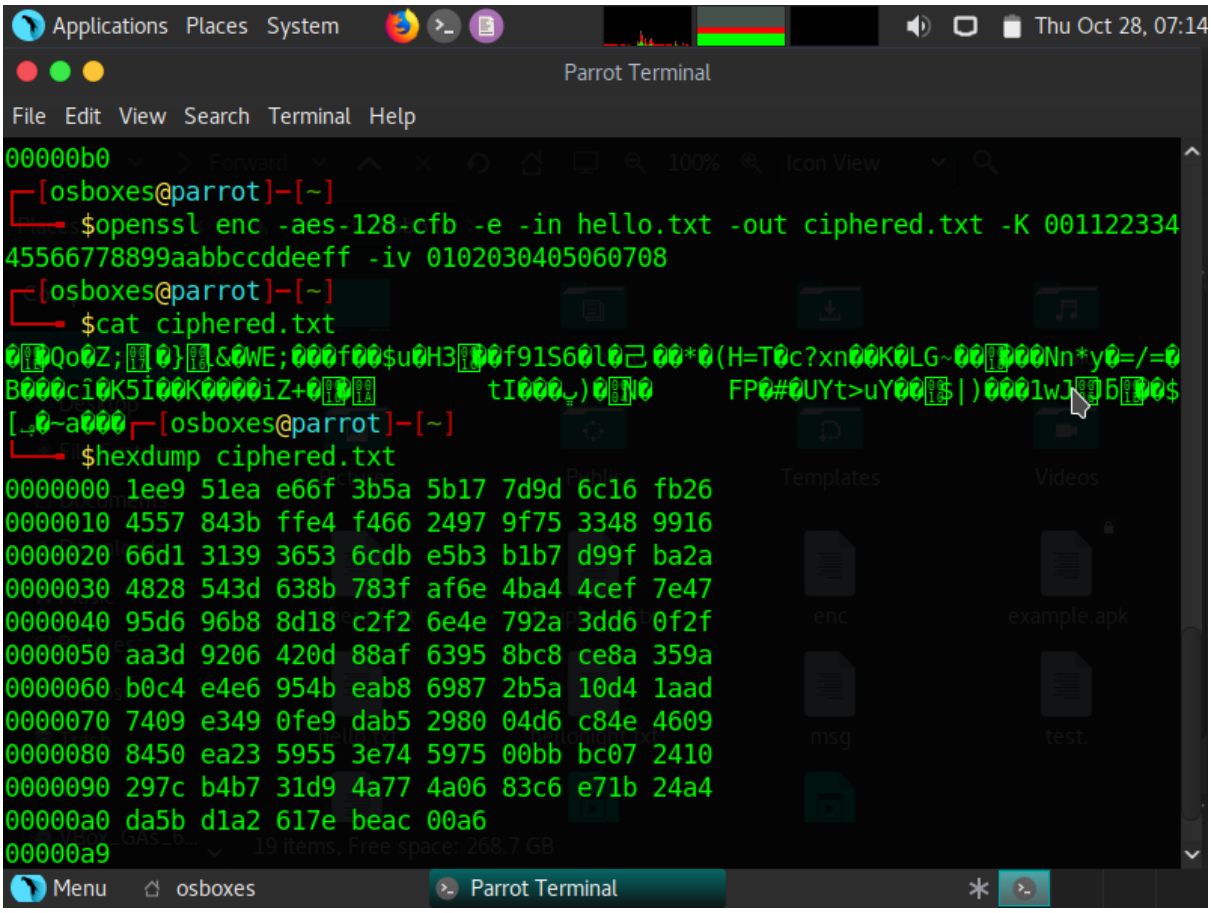
2. Encryption using aes-128-cbc

The cipher text and the hexdump of the cipher text is as follows:



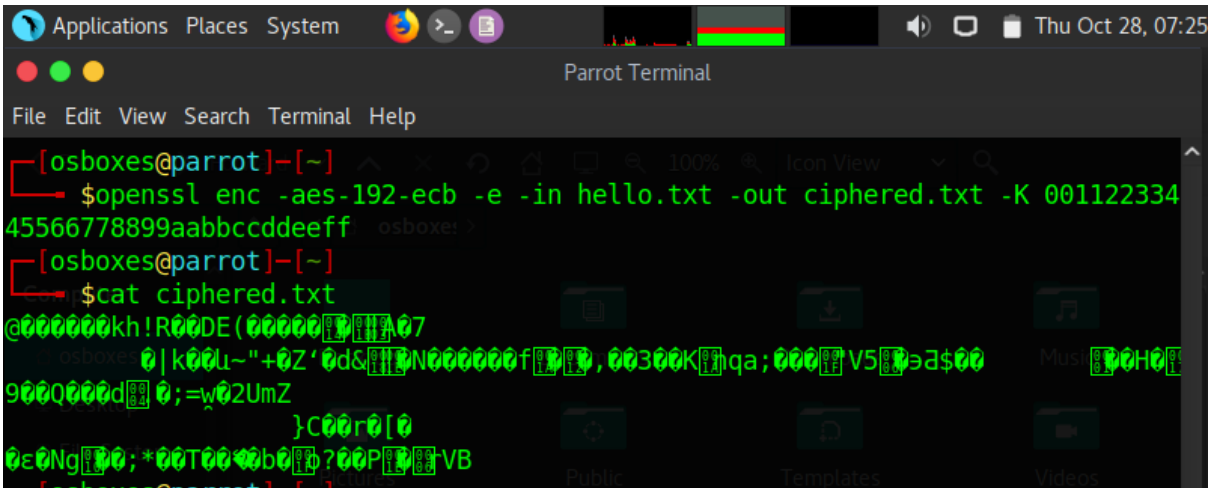
3. Encryption using aes-128-cfb

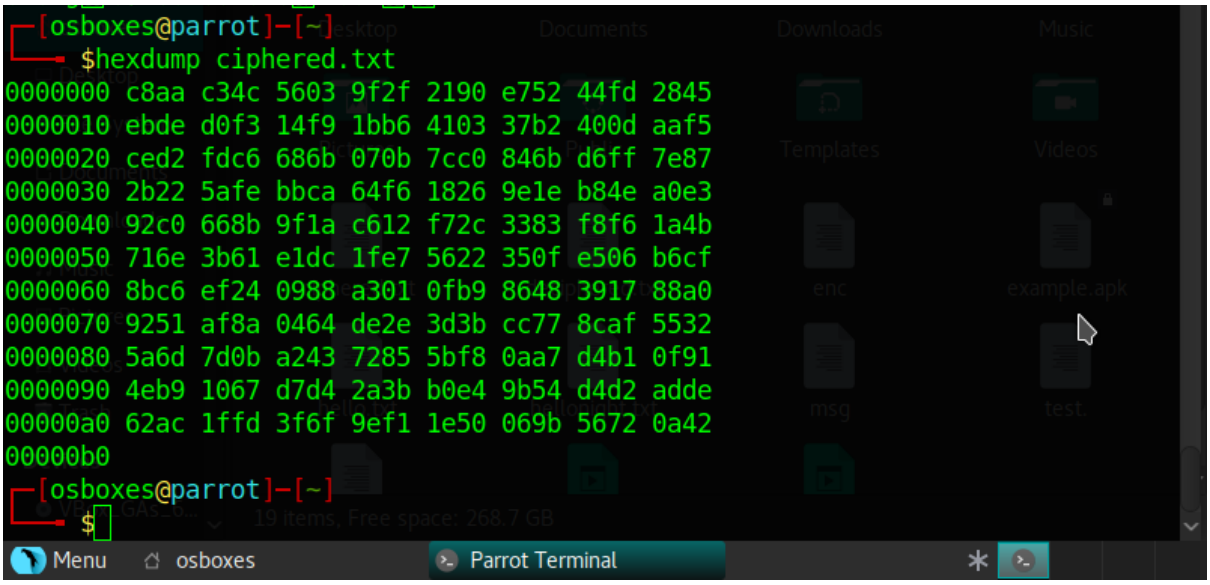
The cipher text and the hexdump for the same is as follows:



4. Encryption using aes-192-ecb

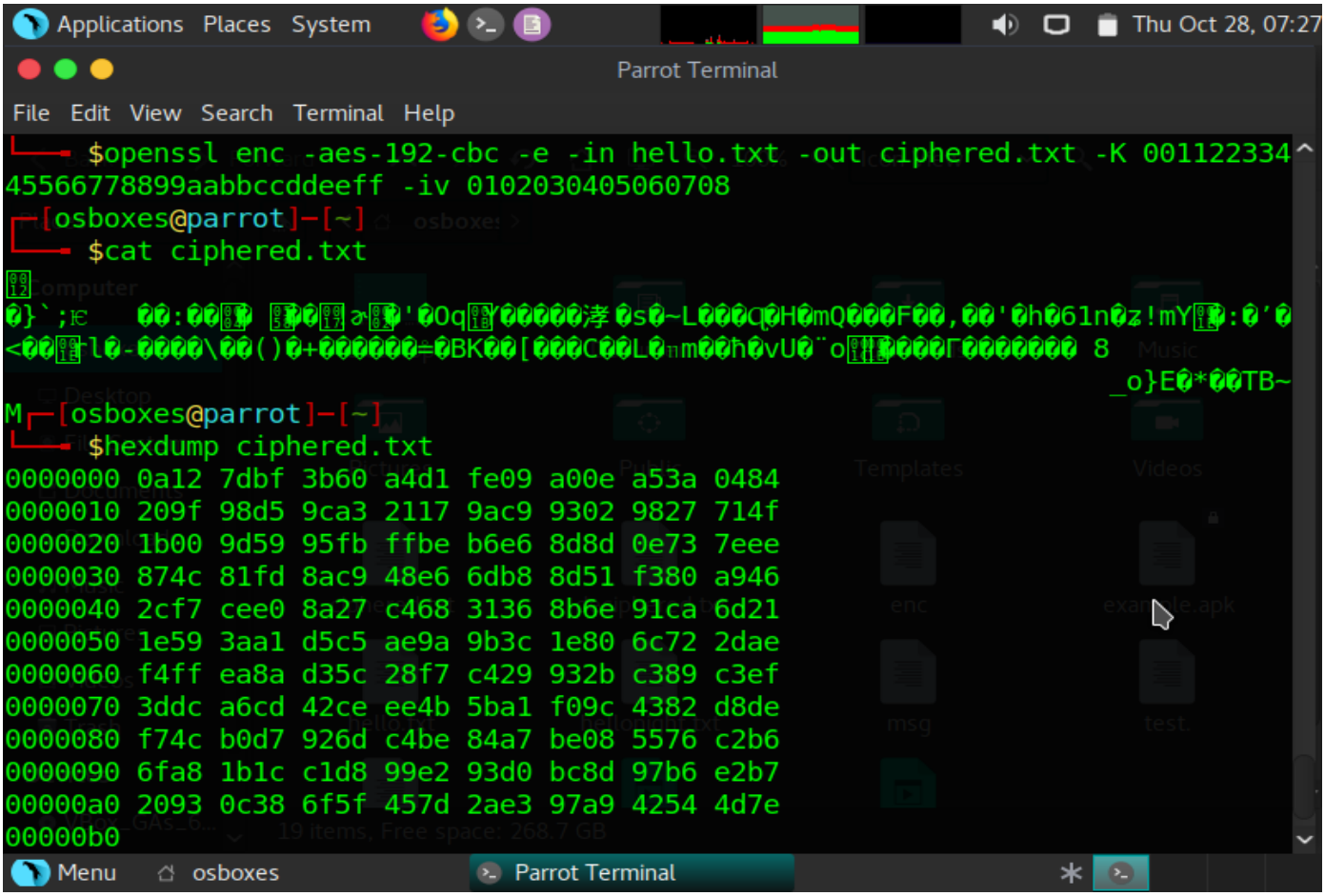
The cipher text and hexdump of the same is as follows:



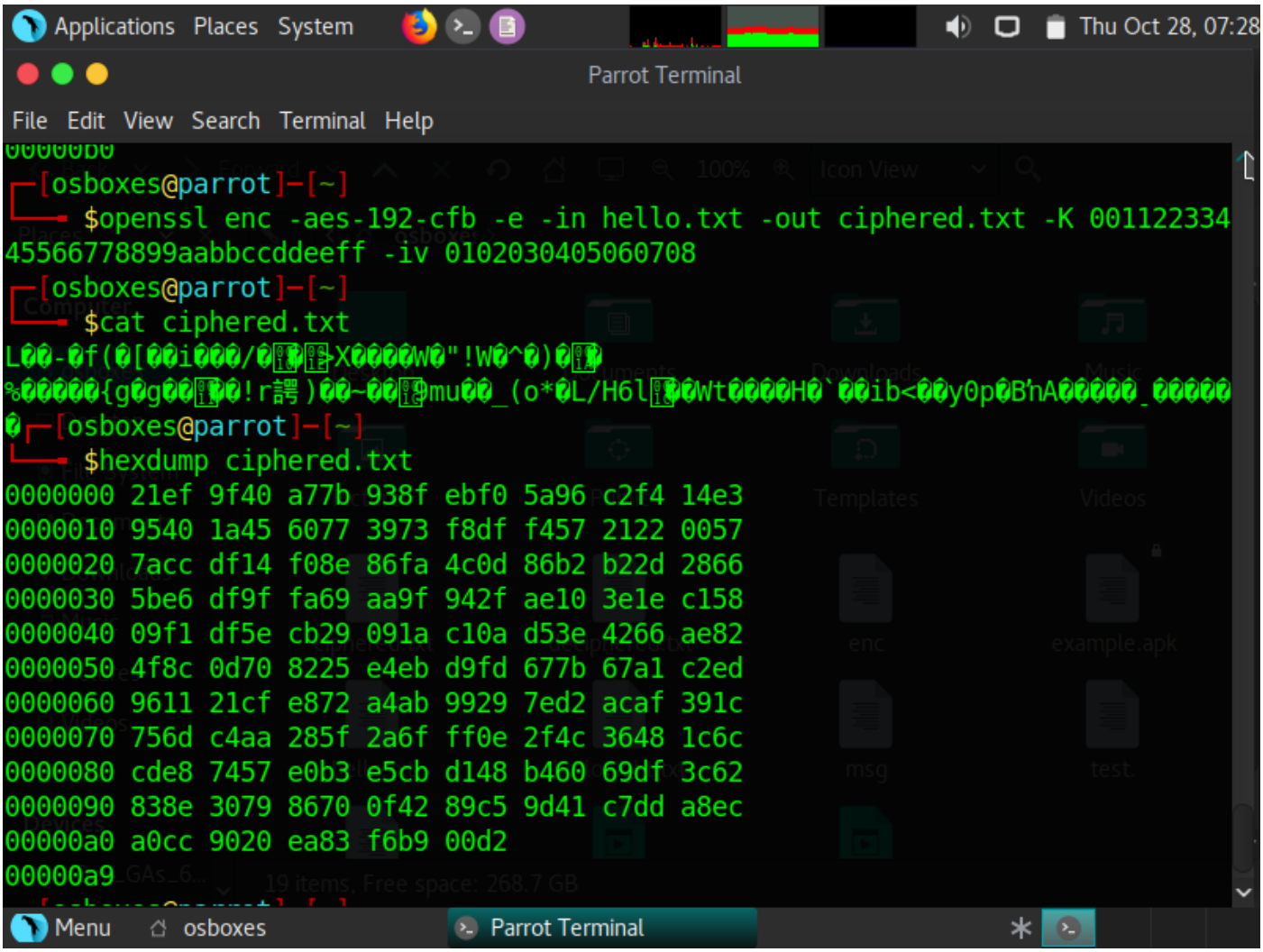


5. Encryption using aes-192-cbc

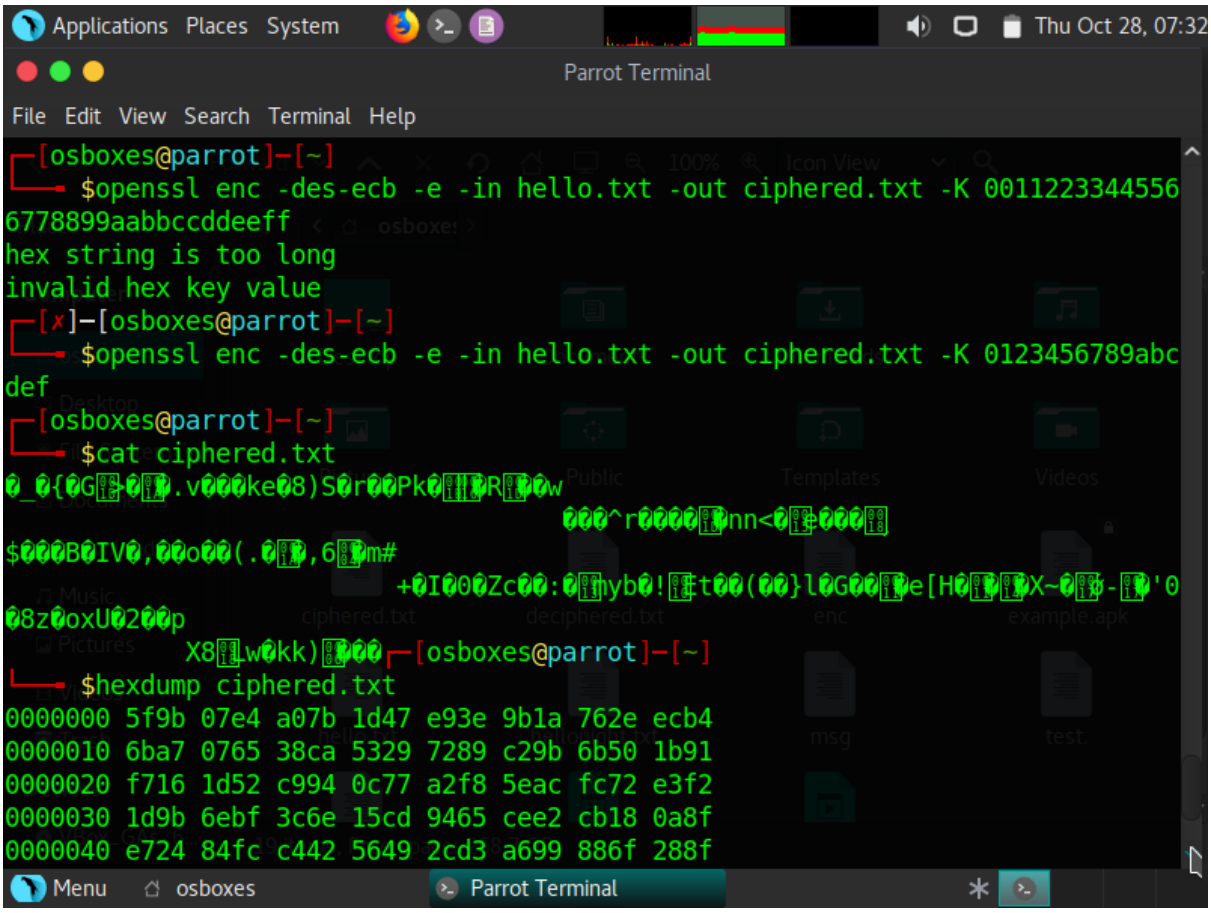
The cipher text and hexdump of the same is as follows:



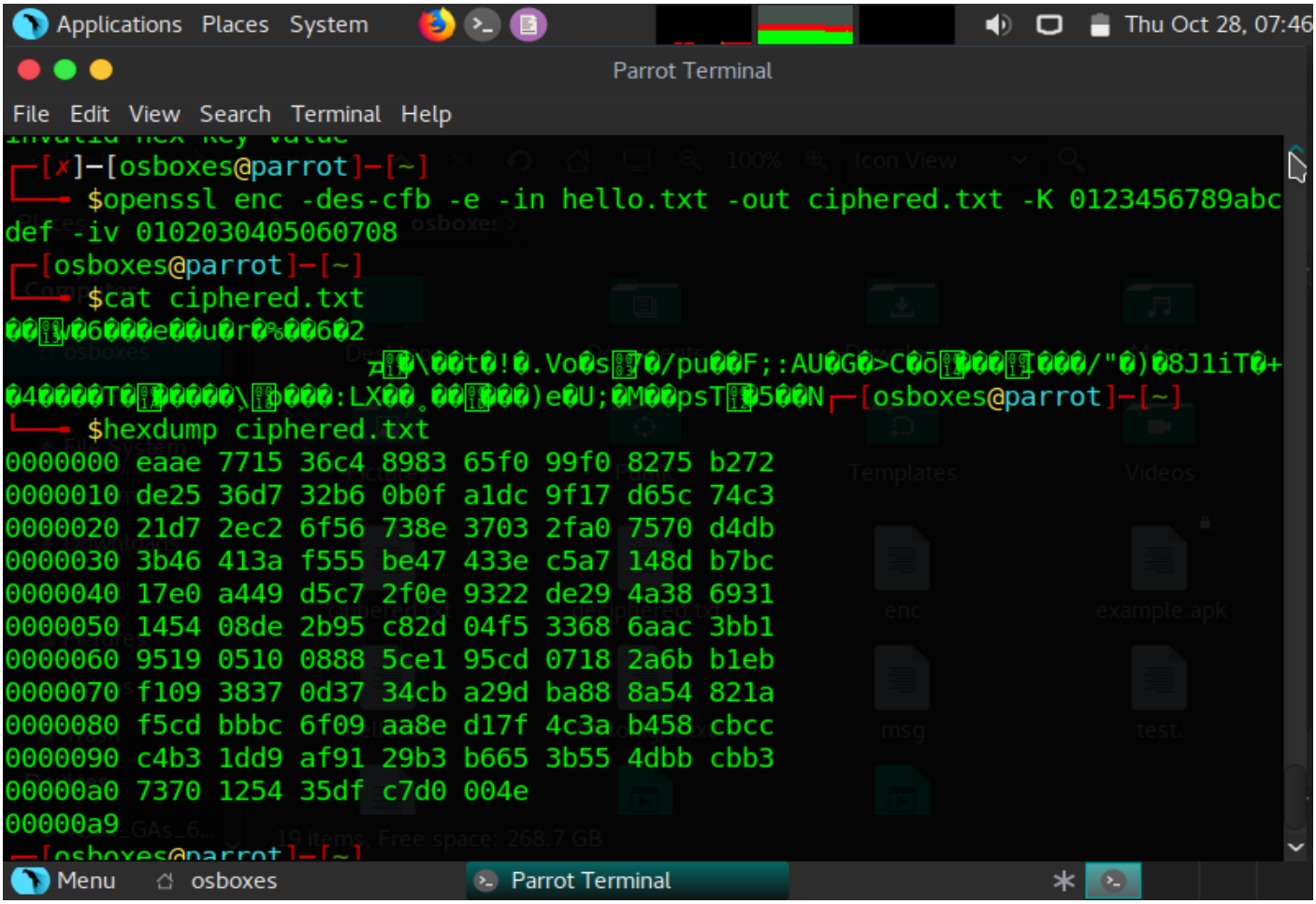
6. Encryption using aes-192-cfb
The cipher text and hexdump of the same is as follows:



7. Encryption using des-ecb
The cipher text and hexdump of the same is as follows:

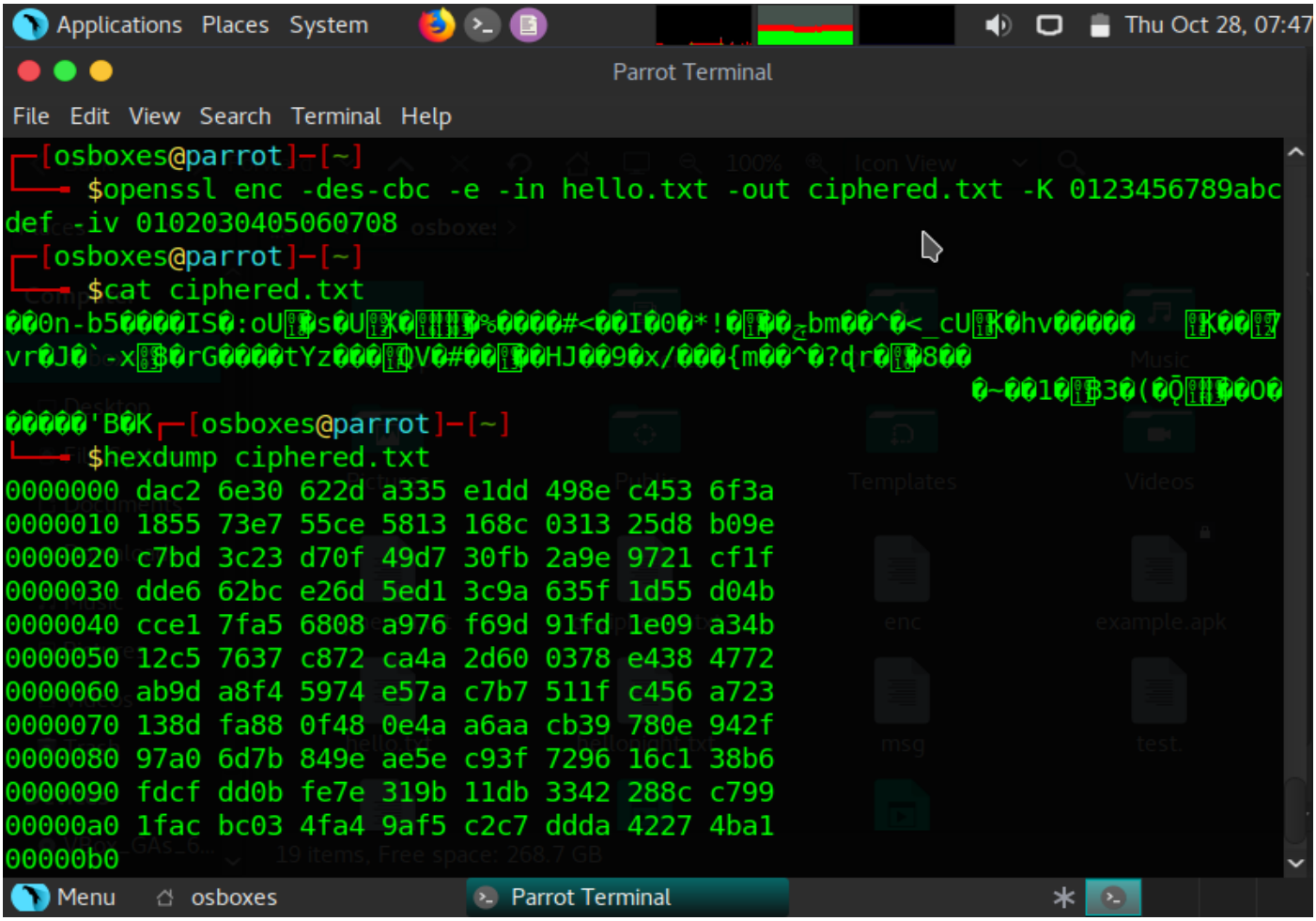


8. Encryption using des-cfb
The cipher text and hexdump of the same is as follows:



9. Encryption using des-cbc

The cipher text and hexdump of the same is as follows:



3.2 Task 2: Encryption Mode – ECB vs. CBC

The file pic original.bmp contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. How- ever, For the .bmp file, the first 54 bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate .bmp file. We will replace the header of the encrypted picture with that of the original picture. You can use the ghex tool to directly modify binary files.

2. Display the encrypted picture using any picture viewing software. Can you derive any useful infor-mation about the original picture from the encrypted picture? Please explain your observations.

Original Picture

Image Encrypted using ecb

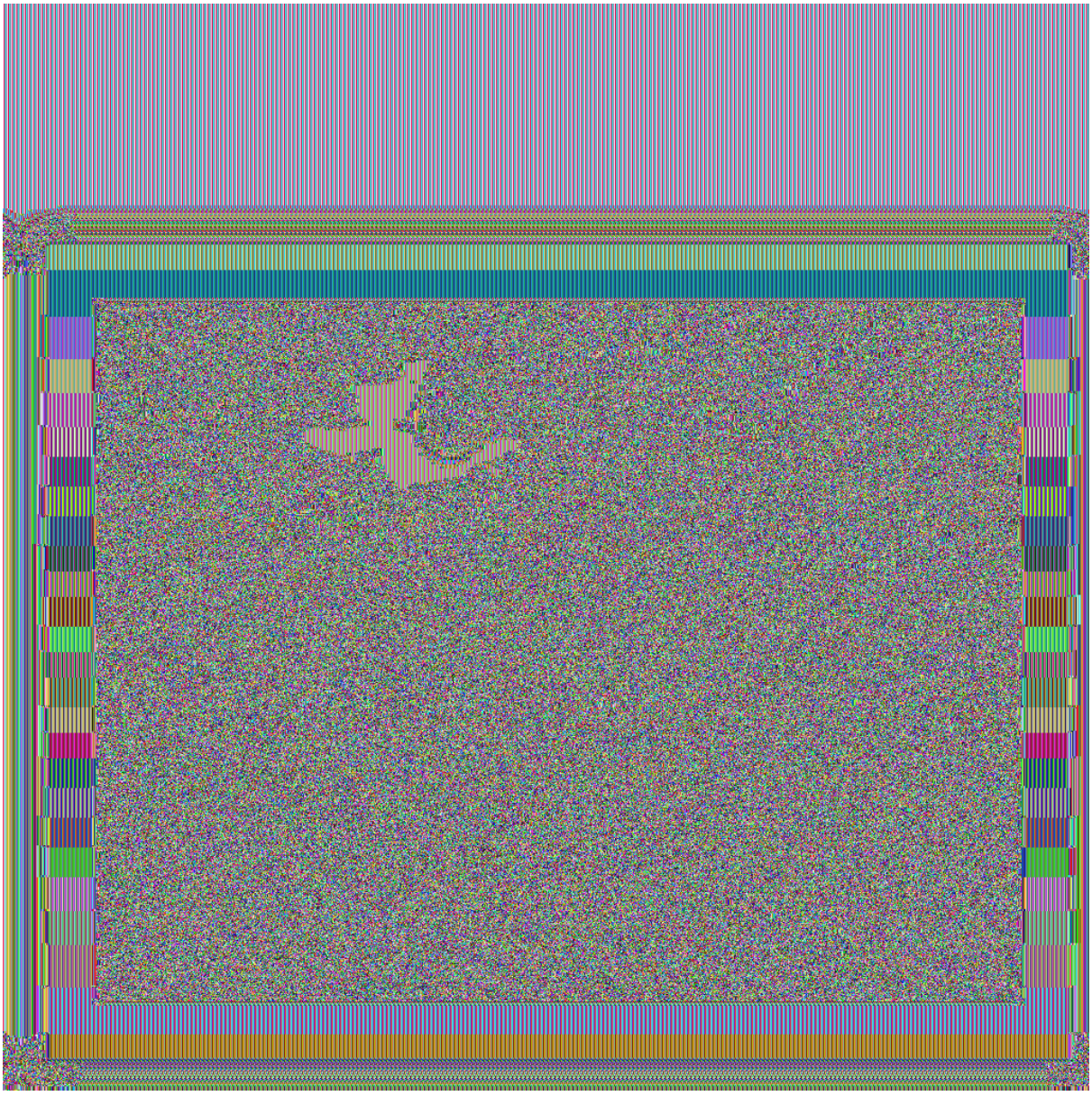
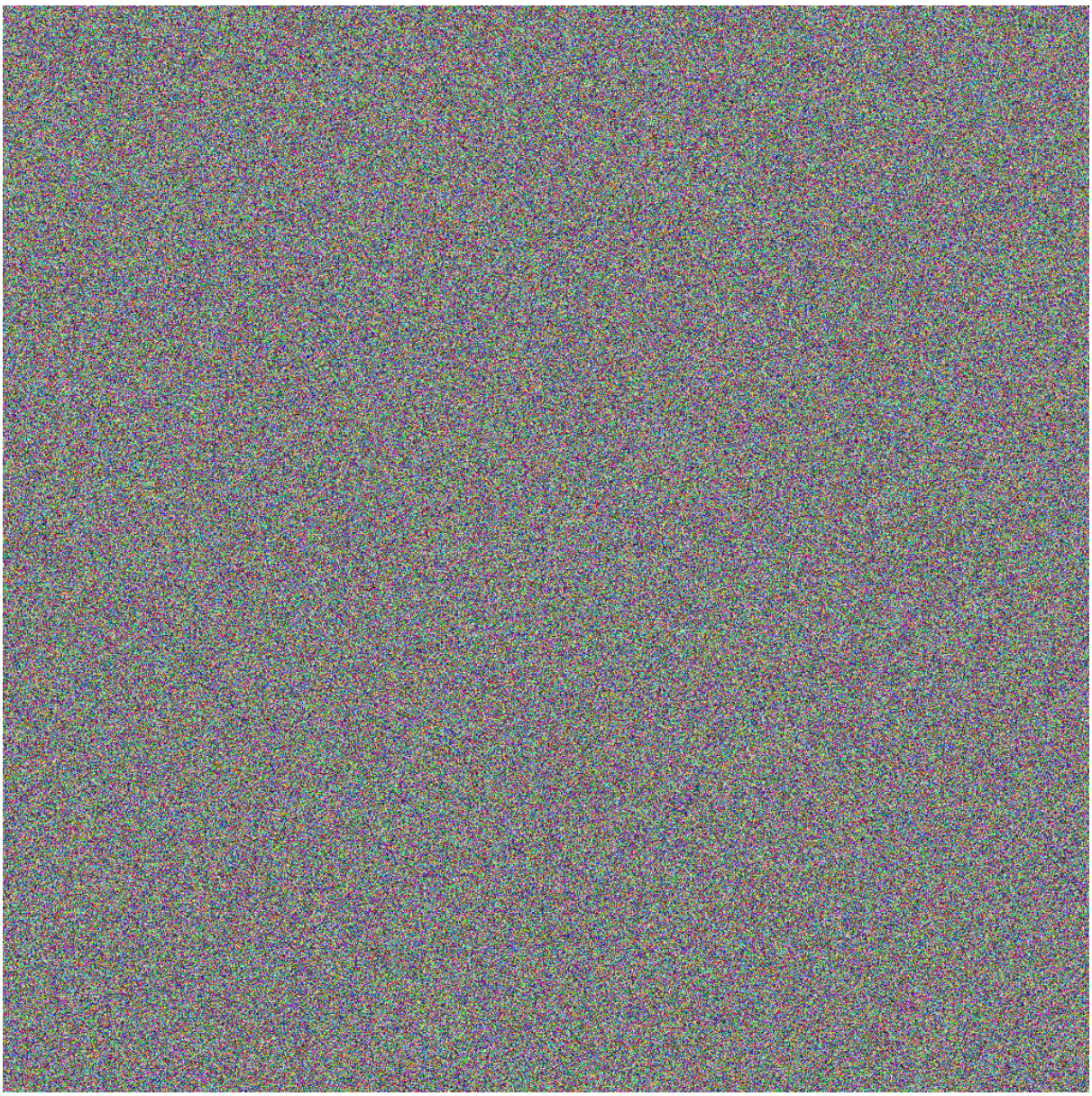


Image encrypted using cbc



Observations:

1) In the image encrypted using ECB, we can still see a set of certain colors and shapes from which we can try to figure out what the original picture could be.

2) In the image encrypted using CBC the entire images is distorted and there is no way by which we can try to figure out the original picture.

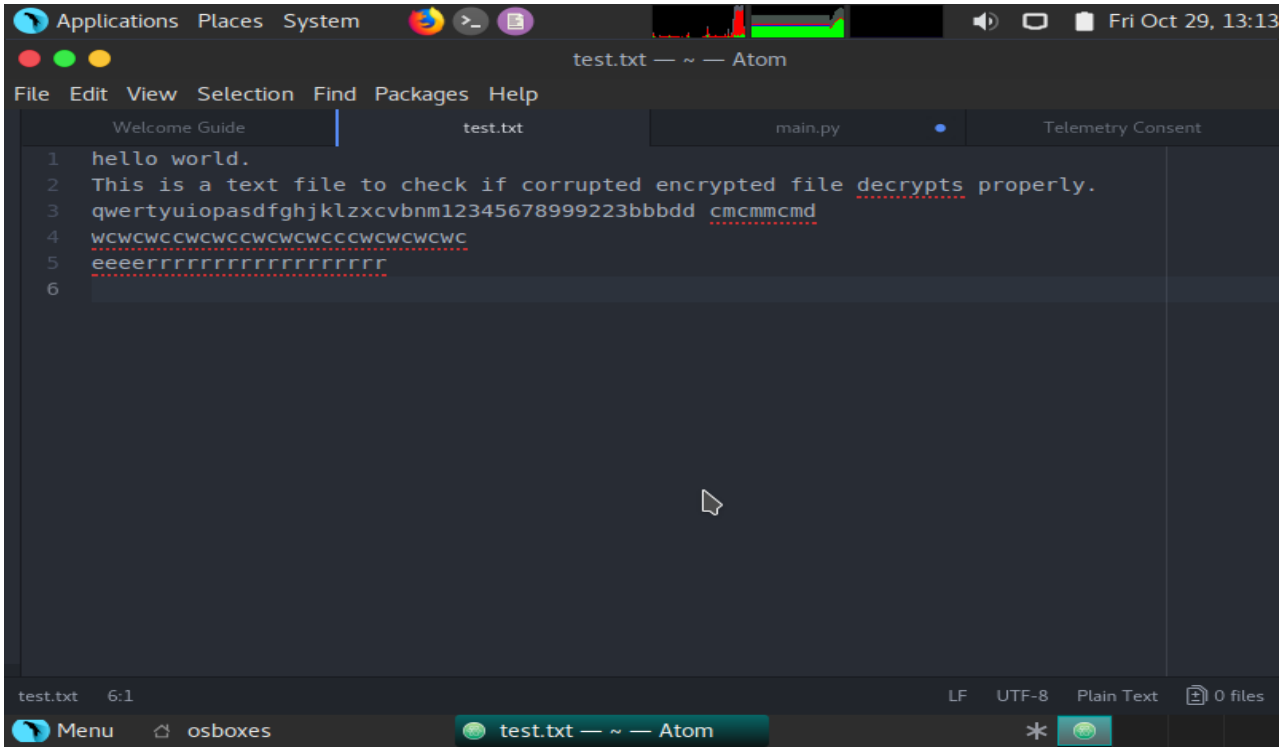
3.3 Task 3: Encryption Mode – Corrupted Cipher Text

To understand the properties of various encryption modes, we would like to do the following exercise:

1. Create a text file that is at least 64 bytes long.
2. Encrypt the file using the AES-128 cipher.
3. Unfortunately, a single bit of the 30th byte in the encrypted file got corrupted. You can achieve this corruption using ghex.
4. Decrypt the corrupted file (encrypted) using the correct key and IV.

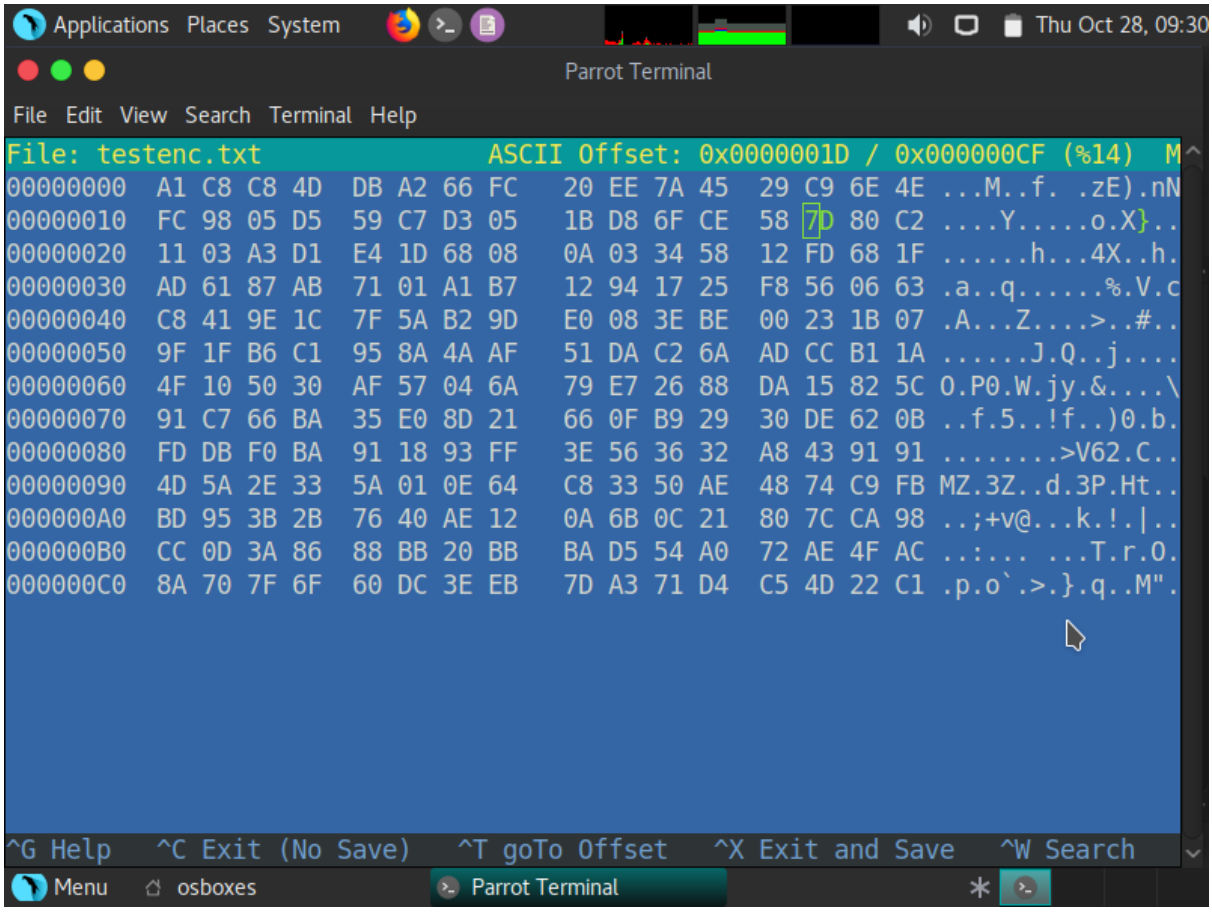
Please answer the following questions: (1) How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task.
(2) Please explain why. (3) What are the implication of these differences?

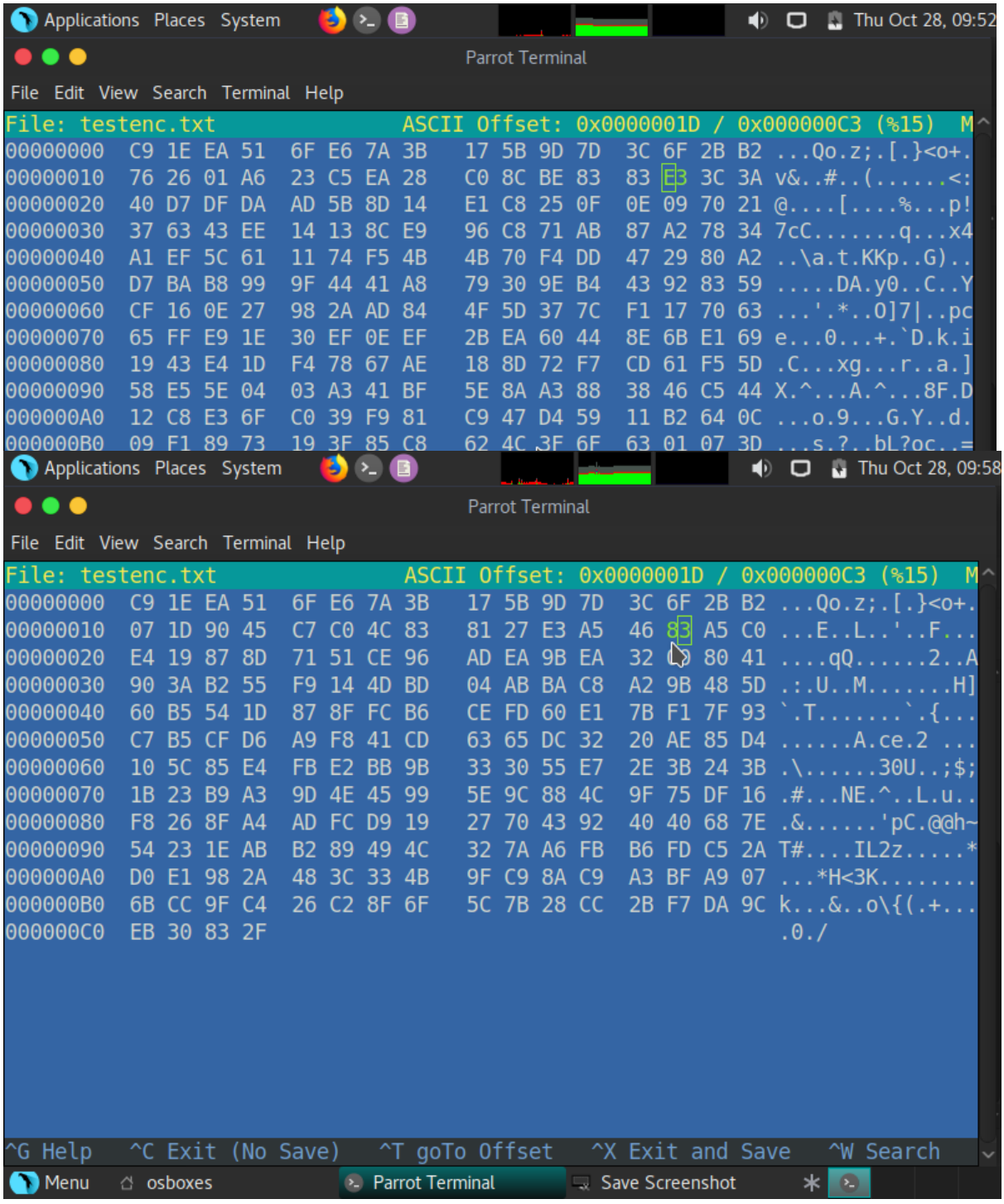
The original plain text is as follows:



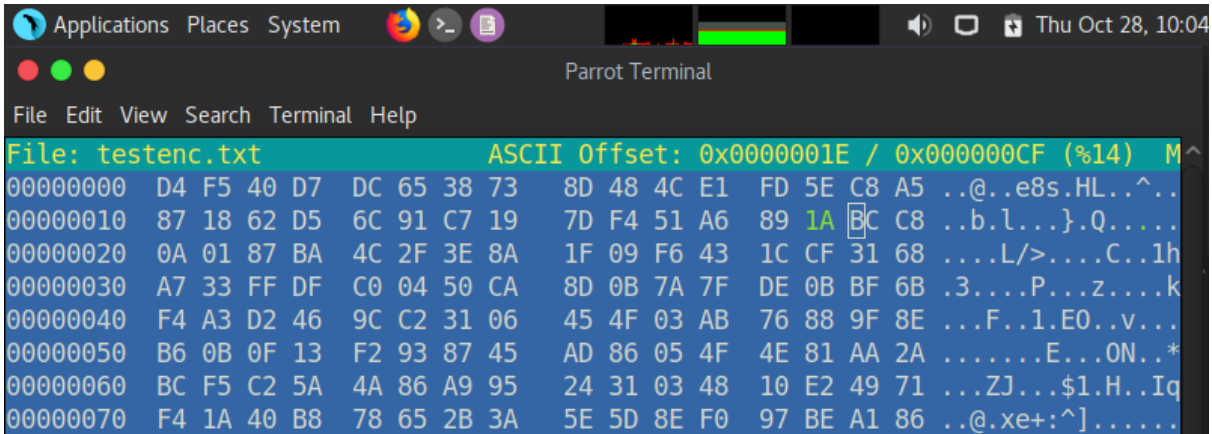
The above text is encrypted using aes-128 in 4 different modes: cbc,cfb,ofb and ecb.

After corrupting 30th byte of each cbc, cfb, ofb and ecb :



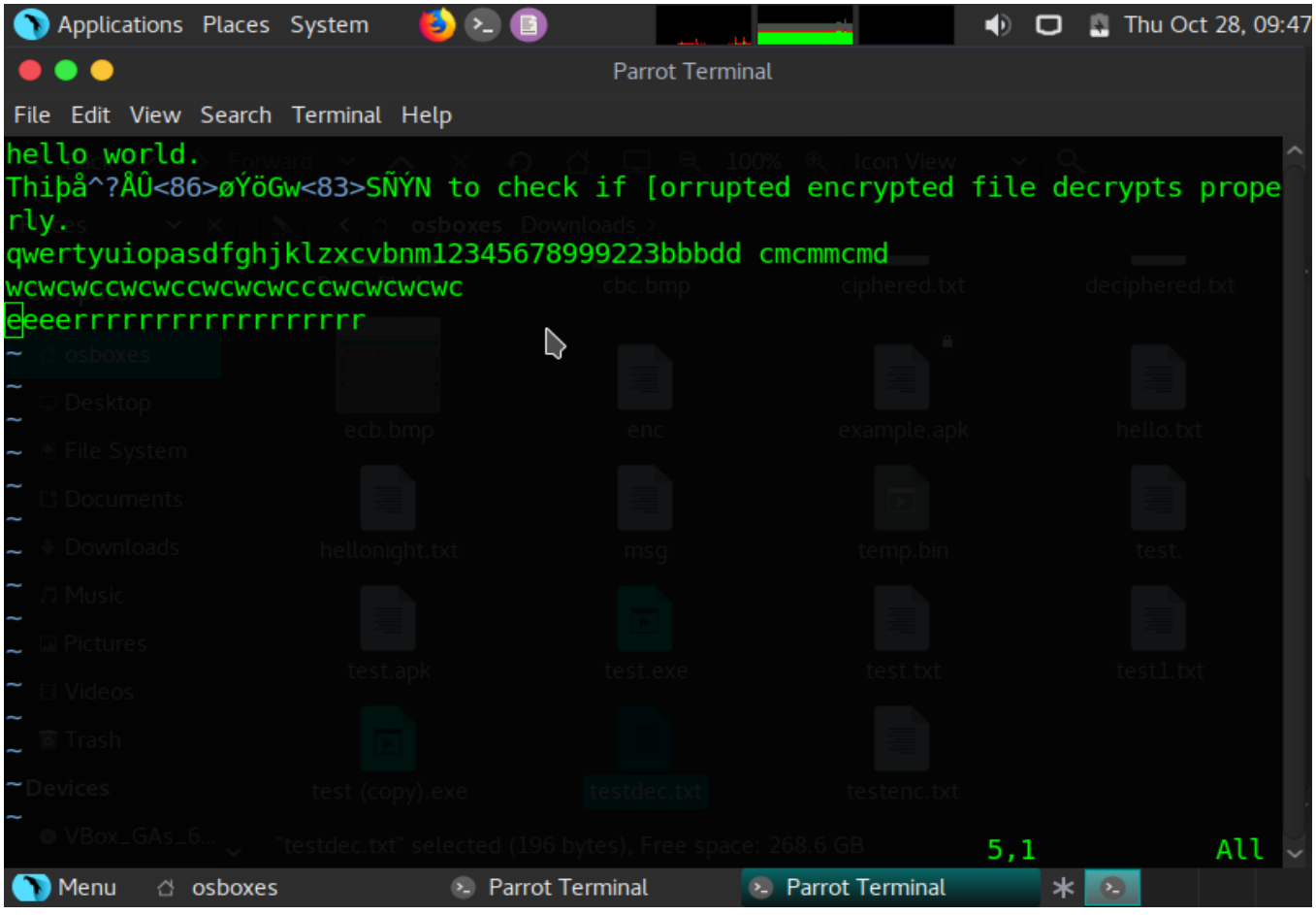

Applications Places System
Thu Oct 28, 09:52
Parrot Terminal
File Edit View Search Terminal Help
File: testenc.txt ASCII Offset: 0x0000001D / 0x000000C3 (%15) M
00000000 C9 1E EA 51 6F E6 7A 3B 17 5B 9D 7D 3C 6F 2B B2 ...Qo.z;.[.}<o+.
00000010 76 26 01 A6 23 C5 EA 28 C0 8C BE 83 83 E3 3C 3A v&..#..(.....<:
00000020 40 D7 DF DA AD 5B 8D 14 E1 C8 25 0F 0E 09 70 21 @....[....%...p!
00000030 37 63 43 EE 14 13 8C E9 96 C8 71 AB 87 A2 78 34 7cC.....q...x4
00000040 A1 EF 5C 61 11 74 F5 4B 4B 70 F4 DD 47 29 80 A2 ..\a.t.KKp..G)..
00000050 D7 BA B8 99 9F 44 41 A8 79 30 9E B4 43 92 83 59 .....DA.y0..C..Y
00000060 CF 16 0E 27 98 2A AD 84 4F 5D 37 7C F1 17 70 63 ...'.*..O]7|..pc
00000070 65 FF E9 1E 30 EF 0E EF 2B EA 60 44 8E 6B E1 69 e...0...+.`D.k.i
00000080 19 43 E4 1D F4 78 67 AE 18 8D 72 F7 CD 61 F5 5D .C...xg...r..a.]
00000090 58 E5 5E 04 03 A3 41 BF 5E 8A A3 88 38 46 C5 44 X.^...A.^...8F.D
000000A0 12 C8 E3 6F C0 39 F9 81 C9 47 D4 59 11 B2 64 0C ...o.9...G.Y..d.
000000B0 09 F1 89 73 19 3F 85 C8 62 4C 3F 6F 63 01 07 3D ...s.?..bL?oc..=

Applications Places System
Thu Oct 28, 09:58
Parrot Terminal
File Edit View Search Terminal Help
File: testenc.txt ASCII Offset: 0x0000001D / 0x000000C3 (%15) M
00000000 C9 1E EA 51 6F E6 7A 3B 17 5B 9D 7D 3C 6F 2B B2 ...Qo.z;.[.}<o+.
00000010 07 1D 90 45 C7 C0 4C 83 81 27 E3 A5 46 83 A5 C0 ...E..L..'..F...
00000020 E4 19 87 8D 71 51 CE 96 AD EA 9B EA 32 [illegible] 80 41 ....qQ......2..A
00000030 90 3A B2 55 F9 14 4D BD 04 AB BA C8 A2 9B 48 5D .:.U..M.......H]
00000040 60 B5 54 1D 87 8F FC B6 CE FD 60 E1 7B F1 7F 93 `.T.......`.{...
00000050 C7 B5 CF D6 A9 F8 41 CD 63 65 DC 32 20 AE 85 D4 ......A.ce.2 ...
00000060 10 5C 85 E4 FB E2 BB 9B 33 30 55 E7 2E 3B 24 3B .\.......30U..;$;
00000070 1B 23 B9 A3 9D 4E 45 99 5E 9C 88 4C 9F 75 DF 16 .#...NE.^..L.u..
00000080 F8 26 8F A4 AD FC D9 19 27 70 43 92 40 40 68 7E .&......'pC.@@h~
00000090 54 23 1E AB B2 89 49 4C 32 7A A6 FB B6 FD C5 2A T#....IL2z.....*
000000A0 D0 E1 98 2A 48 3C 33 4B 9F C9 8A C9 A3 BF A9 07 ...*H<3K........
000000B0 6B CC 9F C4 26 C2 8F 6F 5C 7B 28 CC 2B F7 DA 9C k...&..o\{(.+...
000000C0 EB 30 83 2F .0./
^G Help ^C Exit (No Save) ^T goTo Offset ^X Exit and Save ^W Search
Menu osboxes Parrot Terminal Save Screenshot

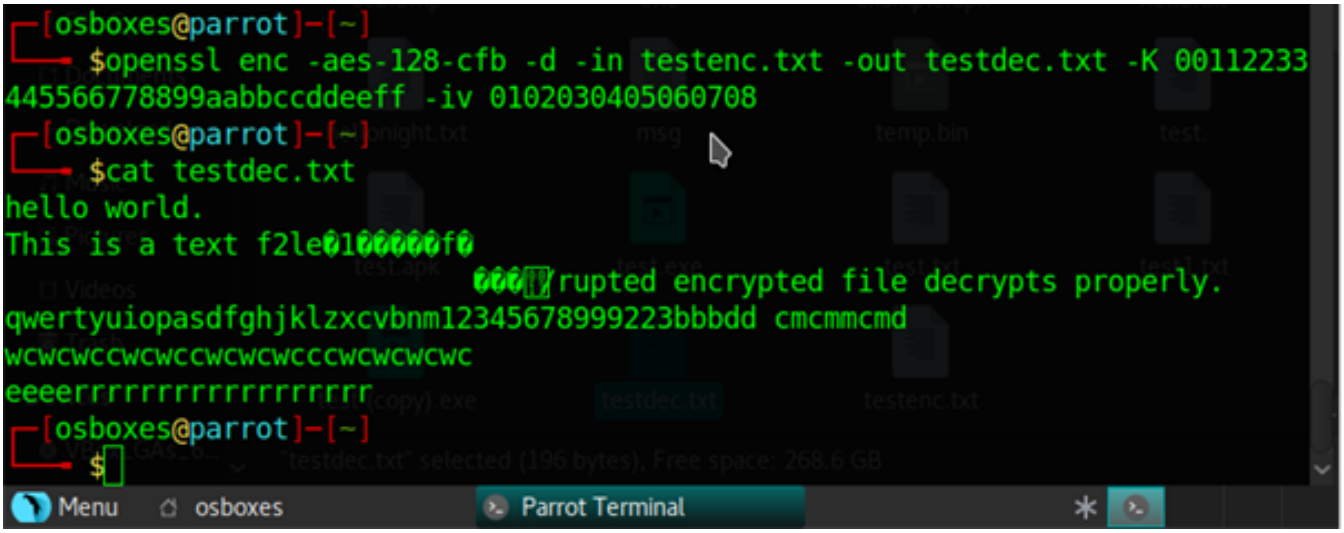

Applications Places System
Thu Oct 28, 10:04
Parrot Terminal
File Edit View Search Terminal Help
File: testenc.txt ASCII Offset: 0x0000001E / 0x000000CF (%14) M
00000000 D4 F5 40 D7 DC 65 38 73 8D 48 4C E1 FD 5E C8 A5 ..@..e8s.HL..^..
00000010 87 18 62 D5 6C 91 C7 19 7D F4 51 A6 89 1A BC C8 ..b.l...}.Q.....
00000020 0A 01 87 BA 4C 2F 3E 8A 1F 09 F6 43 1C CF 31 68 ....L/>....C..1h
00000030 A7 33 FF DF C0 04 50 CA 8D 0B 7A 7F DE 0B BF 6B .3....P...z....k
00000040 F4 A3 D2 46 9C C2 31 06 45 4F 03 AB 76 88 9F 8E ...F..1.EO..v...
00000050 B6 0B 0F 13 F2 93 87 45 AD 86 05 4F 4E 81 AA 2A .......E...ON..*
00000060 BC F5 C2 5A 4A 86 A9 95 24 31 03 48 10 E2 49 71 ...ZJ...$1.H..Iq
00000070 F4 1A 40 B8 78 65 2B 3A 5E 5D 8E F0 97 BE A1 86 ..@.xe+:^]......

The decrypted text for the same is as follows:

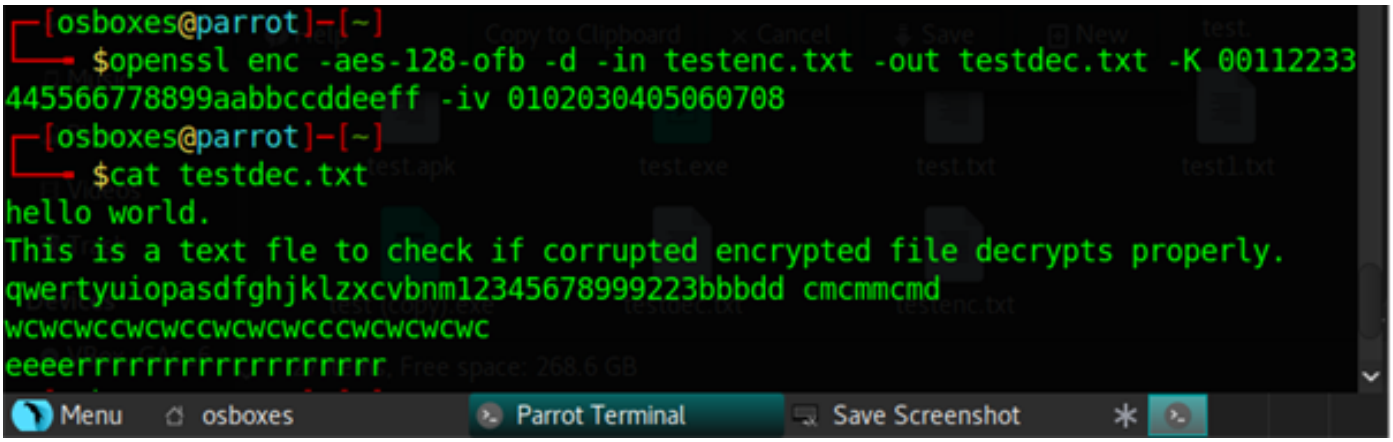
CBC



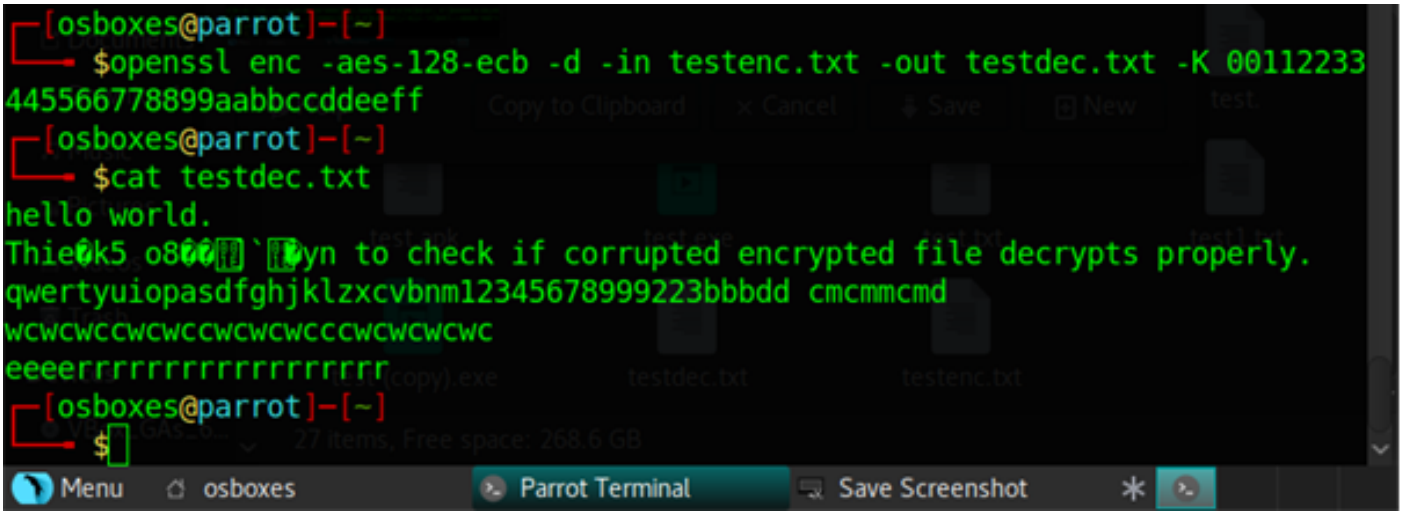
CFB


[osboxes@parrot]-[~]
$openssl enc -aes-128-cfb -d -in testenc.txt -out testdec.txt -K 00112233
445566778899aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~]
$cat testdec.txt
hello world.
This is a text f2le
rupted encrypted file decrypts properly.
qwertyuiopasdfghjklzxcvbnm12345678999223bbbdd cmcmmcmd
wcwcwccwcwccwcwcwcccwcwcwcwc
eeeerrrrrrrrrrrrrrrrr
[osboxes@parrot]-[~]
$
Menu
osboxes
Parrot Terminal

OFB


[osboxes@parrot]-[~]
$openssl enc -aes-128-ofb -d -in testenc.txt -out testdec.txt -K 00112233
445566778899aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~]
$cat testdec.txt
hello world.
This is a text fle to check if corrupted encrypted file decrypts properly.
qwertyuiopasdfghjklzxcvbnm12345678999223bbbdd cmcmmcmd
wcwcwccwcwccwcwcwcccwcwcwcwc
eeeerrrrrrrrrrrrrrrrr
Menu
osboxes
Parrot Terminal
Save Screenshot

ECB


[osboxes@parrot]-[~]
$openssl enc -aes-128-ecb -d -in testenc.txt -out testdec.txt -K 00112233
445566778899aabbccddeeff
[osboxes@parrot]-[~]
$cat testdec.txt
hello world.
Thie k5 o8 `yn to check if corrupted encrypted file decrypts properly.
qwertyuiopasdfghjklzxcvbnm12345678999223bbbdd cmcmmcmd
wcwcwccwcwccwcwcwcccwcwcwcwc
eeeerrrrrrrrrrrrrrrrr
[osboxes@parrot]-[~]
$
Menu
osboxes
Parrot Terminal
Save Screenshot

From the results above I can conclude that:

In ECB mode, only one block is affected when any problem in a ciphertext happens.Each block is decrypted independently. However, the corrupted bit of the 30th byte in ciphertext block 8 bytes might spread to all n bits in plaintext block 8 bytes since we do the decryption one block at a time.

In CBC mode, there was an effect in two blocks.

In CFB mode, there is a problem in n / r number of blocks.

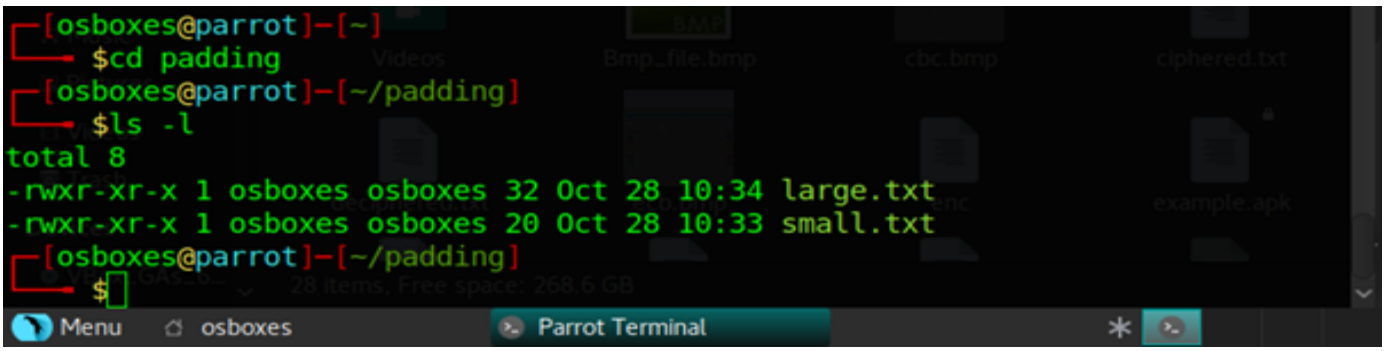
r = number of bits taken at a time to XOR.

In OFB mode, the single digit of the 30th byte is corrupted, then in plain text only that byte or character is corrupted. Thus, only OFB mode shows the most promising result and almost all the texts are recovered.

3.4 Task4 : Padding

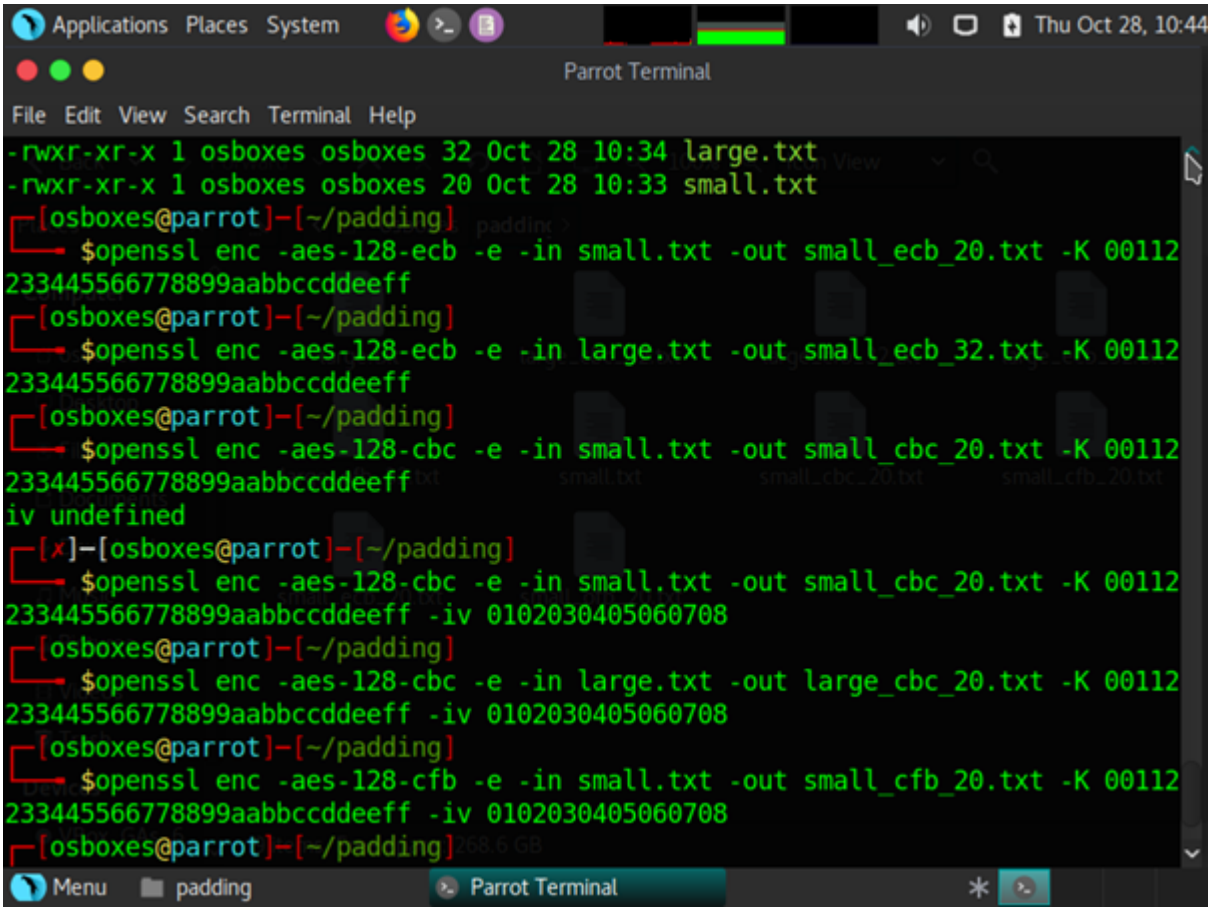
For block ciphers, when the size of the plaintex is not the multiple of the block size, padding may be required. In this task, we will study the padding schemes. Please do the following exercises:

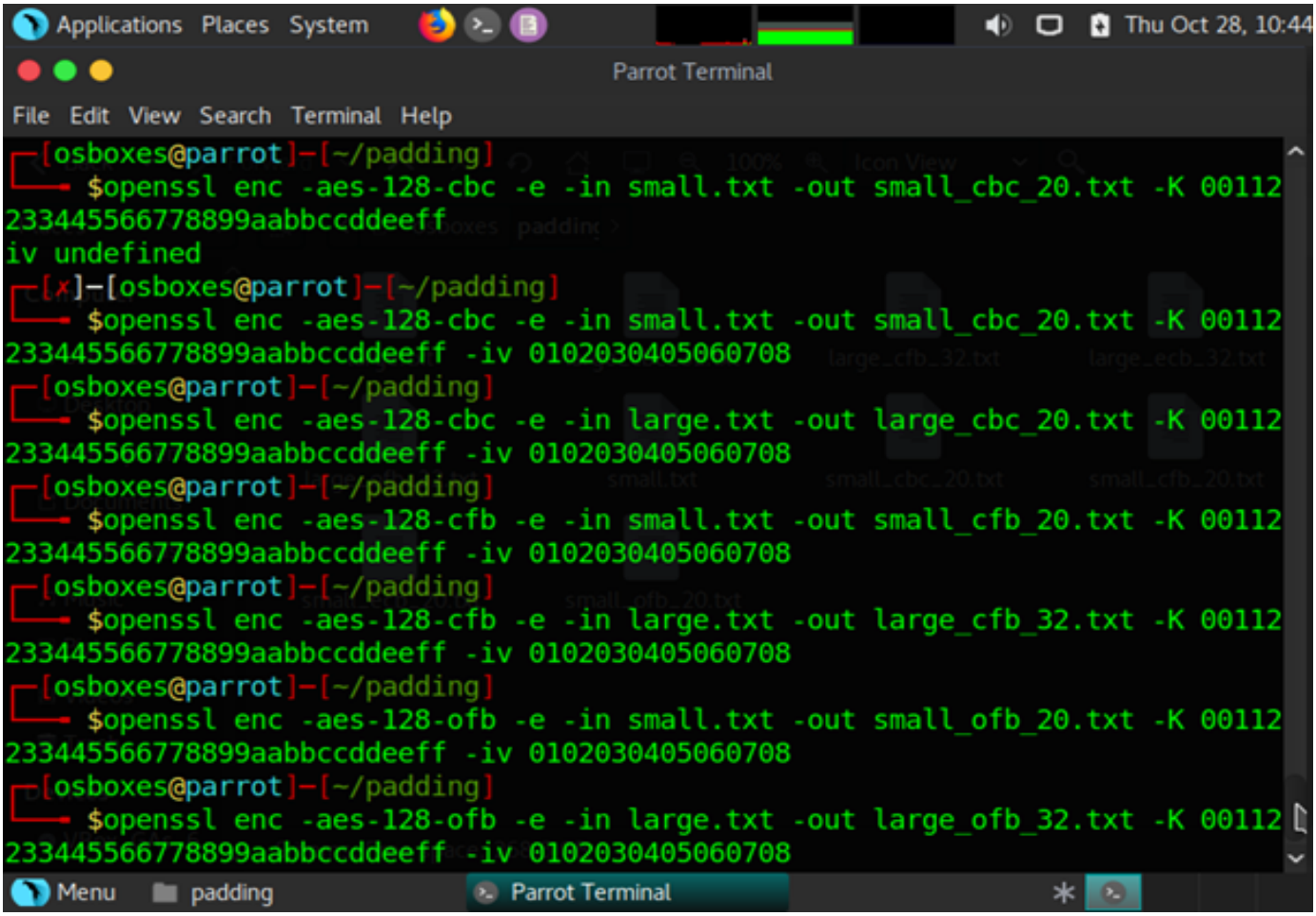
1. The openssl manual says that openssl uses PKCS5 standard for its padding. Please design an experiment to verify this. In particular, use your experiment to figure out the paddings in the AES encryption when the length of the plaintext is 20 octets and 32 octets.
2. Please use ECB, CBC, CFB, and OFB modes to encrypt a file (you can pick any cipher). Please report which modes have paddings and which ones do not. For those that do not need paddings, please explain why.

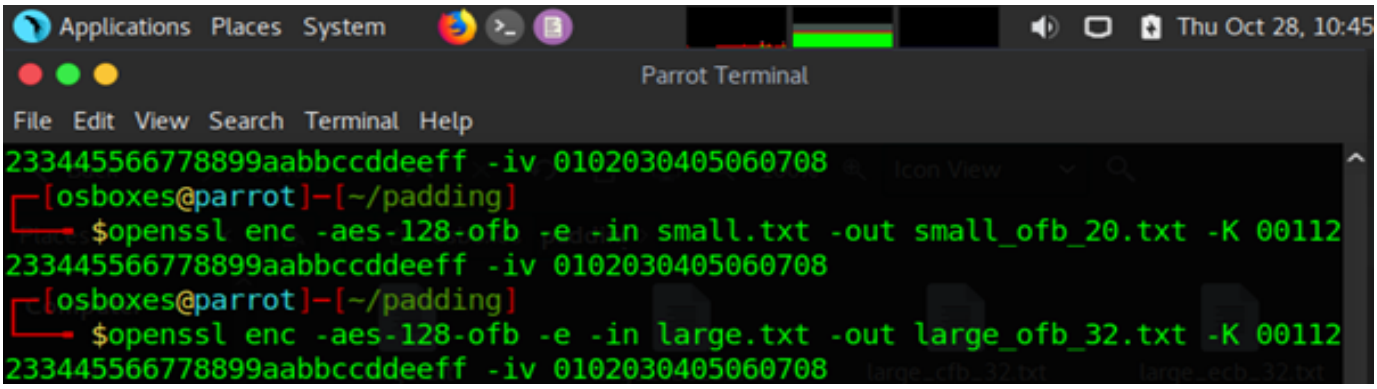


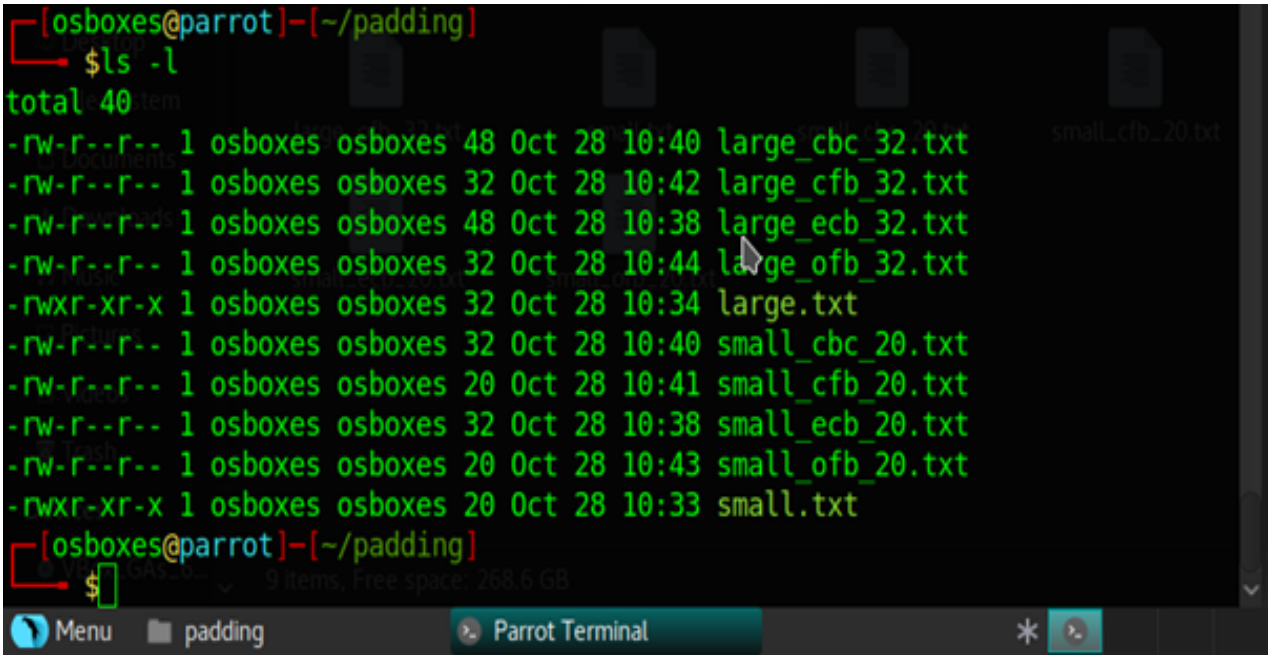
Two files namely large.txt and small.txt are made with each having 32 and 20 bytes respectively.

The files are encrypted using aes-128 in 4 modes namely: ecb, cbc, cfb, ofb

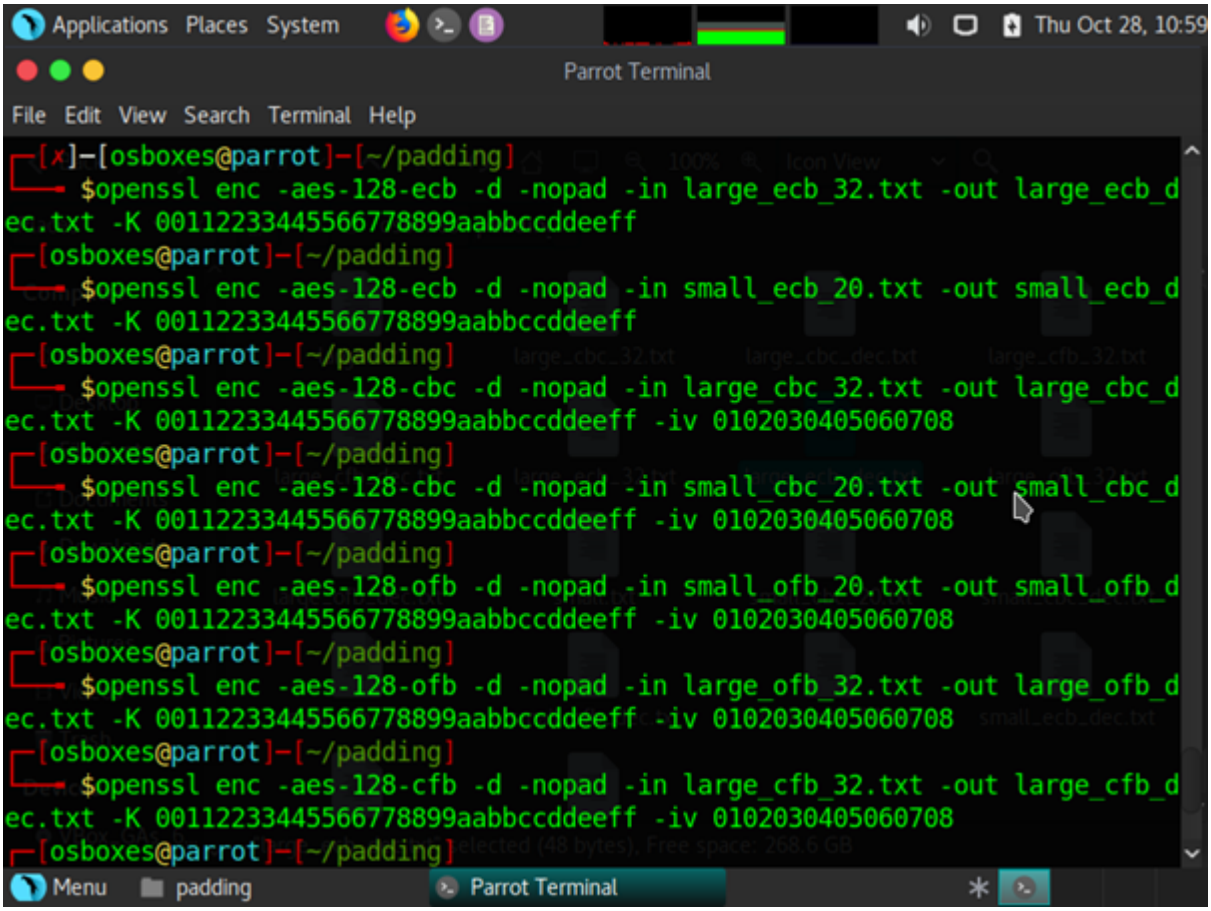

Applications Places System
Thu Oct 28, 10:44
Parrot Terminal
File Edit View Search Terminal Help
-rwxr-xr-x 1 osboxes osboxes 32 Oct 28 10:34 large.txt
-rwxr-xr-x 1 osboxes osboxes 20 Oct 28 10:33 small.txt
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-ecb -e -in small.txt -out small_ecb_20.txt -K 00112
233445566778899aabbccddeeff
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-ecb -e -in large.txt -out small_ecb_32.txt -K 00112
233445566778899aabbccddeeff
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-cbc -e -in small.txt -out small_cbc_20.txt -K 00112
233445566778899aabbccddeeff
iv undefined
[x]-[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-cbc -e -in small.txt -out small_cbc_20.txt -K 00112
233445566778899aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-cbc -e -in large.txt -out large_cbc_20.txt -K 00112
233445566778899aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-cfb -e -in small.txt -out small_cfb_20.txt -K 00112
233445566778899aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
Menu
padding
Parrot Terminal

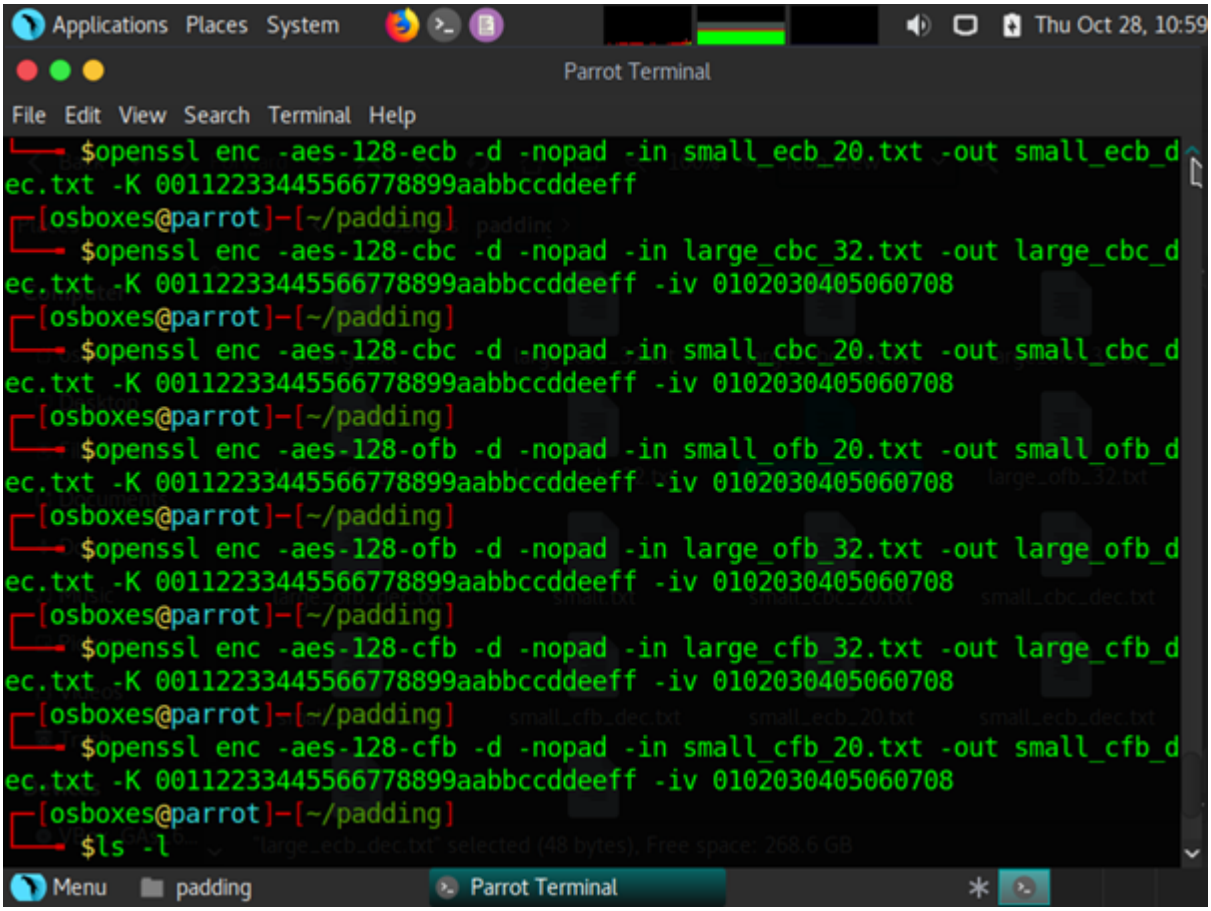

Applications Places System
Thu Oct 28, 10:44
Parrot Terminal
File Edit View Search Terminal Help
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-cbc -e -in small.txt -out small_cbc_20.txt -K 00112
2334455667788 99aabbccddeeff
iv undefined
[x]-[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-cbc -e -in small.txt -out small_cbc_20.txt -K 00112
2334455667788 99aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-cbc -e -in large.txt -out large_cbc_20.txt -K 00112
2334455667788 99aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-cfb -e -in small.txt -out small_cfb_20.txt -K 00112
2334455667788 99aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-cfb -e -in large.txt -out large_cfb_32.txt -K 00112
2334455667788 99aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-ofb -e -in small.txt -out small_ofb_20.txt -K 00112
2334455667788 99aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-ofb -e -in large.txt -out large_ofb_32.txt -K 00112
2334455667788 99aabbccddeeff -iv 0102030405060708
Menu
padding
Parrot Terminal


Applications Places System
Thu Oct 28, 10:45
Parrot Terminal
File Edit View Search Terminal Help
2334455667788 99aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-ofb -e -in small.txt -out small_ofb_20.txt -K 00112
2334455667788 99aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-ofb -e -in large.txt -out large_ofb_32.txt -K 00112
2334455667788 99aabbccddeeff -iv 0102030405060708

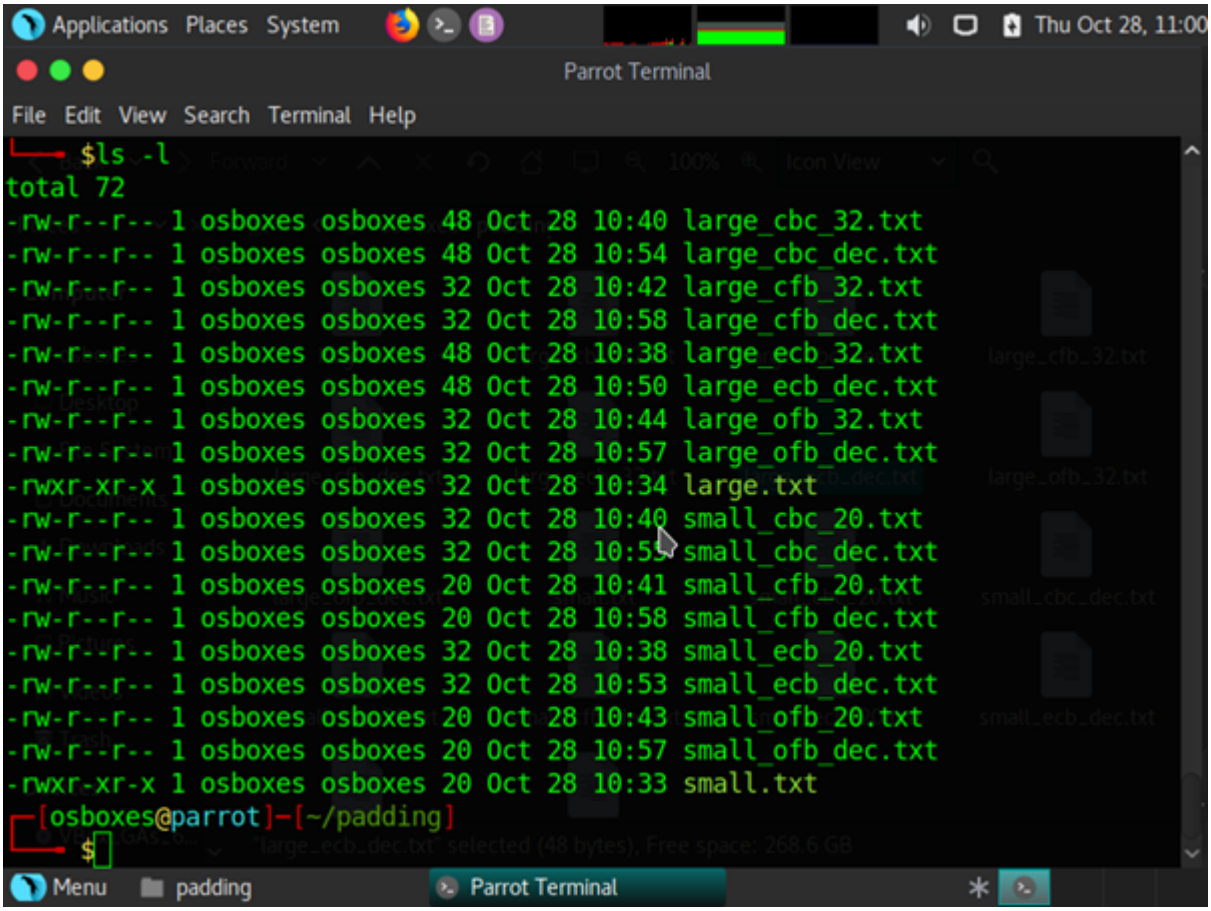


To confirm that openssl uses PKCS5 padding, decrypt the encrypted file with option–nopad. This option turns off the standard block padding. Normally, the padding is included by default during encryption, so if I use the nopad option, I can see the padding in the decrypted file.


Applications Places System
Thu Oct 28, 10:59
Parrot Terminal
File Edit View Search Terminal Help
[x]-[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-ecb -d -nopad -in large_ecb_32.txt -out large_ecb_dec.txt -K 00112233445566778899aabbccddeeff
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-ecb -d -nopad -in small_ecb_20.txt -out small_ecb_dec.txt -K 00112233445566778899aabbccddeeff
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-cbc -d -nopad -in large_cbc_32.txt -out large_cbc_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-cbc -d -nopad -in small_cbc_20.txt -out small_cbc_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-ofb -d -nopad -in small_ofb_20.txt -out small_ofb_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-ofb -d -nopad -in large_ofb_32.txt -out large_ofb_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
$openssl enc -aes-128-cfb -d -nopad -in large_cfb_32.txt -out large_cfb_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[osboxes@parrot]-[~/padding]
Menu
padding
Parrot Terminal



Finally we can see the decrypted and encrypted files as follows:



The screenshot above shows that the size of CBC and ECB encrypted files with the nopad option is 12 bytes more for the 20 bytes file and 16 bytes larger for the 32 bytes file, however the size of OFB and CFB decrypted files is the same.

Result:

1. The experiment shows that padding is needed for ECB and CBC encryption modes. This can be because ECB and CBC are block ciphers and for a block cipher length of input must be an exact multiple of block length. If this is not the case then padding must be added to make it so. This padding is removed after decrypting.
2. In OFB and CFB, the padding is not required because they are stream ciphers and the ciphertext is always the same length as plain text.

3.5 Task 5: Programming using the Crypto Library

So far, we have learned how to use the tools provided by openssl to encrypt and decrypt messages. In this task, we will learn how to use openssl's crypto library to encrypt/descrypt messages in programs.

OpenSSL provides an API called EVP, which is a high-level interface to cryptographic functions. Although OpenSSL also has direct interfaces for each individual encryption algorithm, the EVP library provides a common interface for various encryption algorithms. To ask EVP to use a specific algorithm, we simply need to pass our choice to the EVP interface. A sample code is given in http://www.openssl.org/docs/crypto/EVP_EncryptInit.html. Please get yourself familiar with this program, and then do the

following exercise.

You are given a plaintext and a ciphertext, and you know that aes-128-cbc is used to generate the ciphertext from the plaintext, and you also know that the numbers in the IV are all zeros (not the ASCII character '0'). Another clue that you have learned is that the key used to encrypt this plaintext is an English word shorter than 16 characters; the word that can be found from a typical English dictionary. Since the word has less than 16 characters (i.e. 128 bits), space characters (hexadecimal value 0x20) are appended to the end of the word to form a key of 128 bits. Your goal is to write a program to find out this key. You can download a English word list from the Internet. We have also linked one on the web page of this lab. The plaintext and ciphertext is in the following:

```
Plaintext (total 21 characters): This is a top secret. Ciphertext
(in hex format): 8d20e5056a8d24d0462ce74e4904c1b5
13e10d1df4a2ef2ad4540fae1ca0aaf9
```

Note 1: If you choose to store the plaintex message in a file, and feed the file to your program, you need to check whether the file length is 21. Some editors may add a special character to the end of the file. If that happens, you can use the ghex tool to remove the special character.

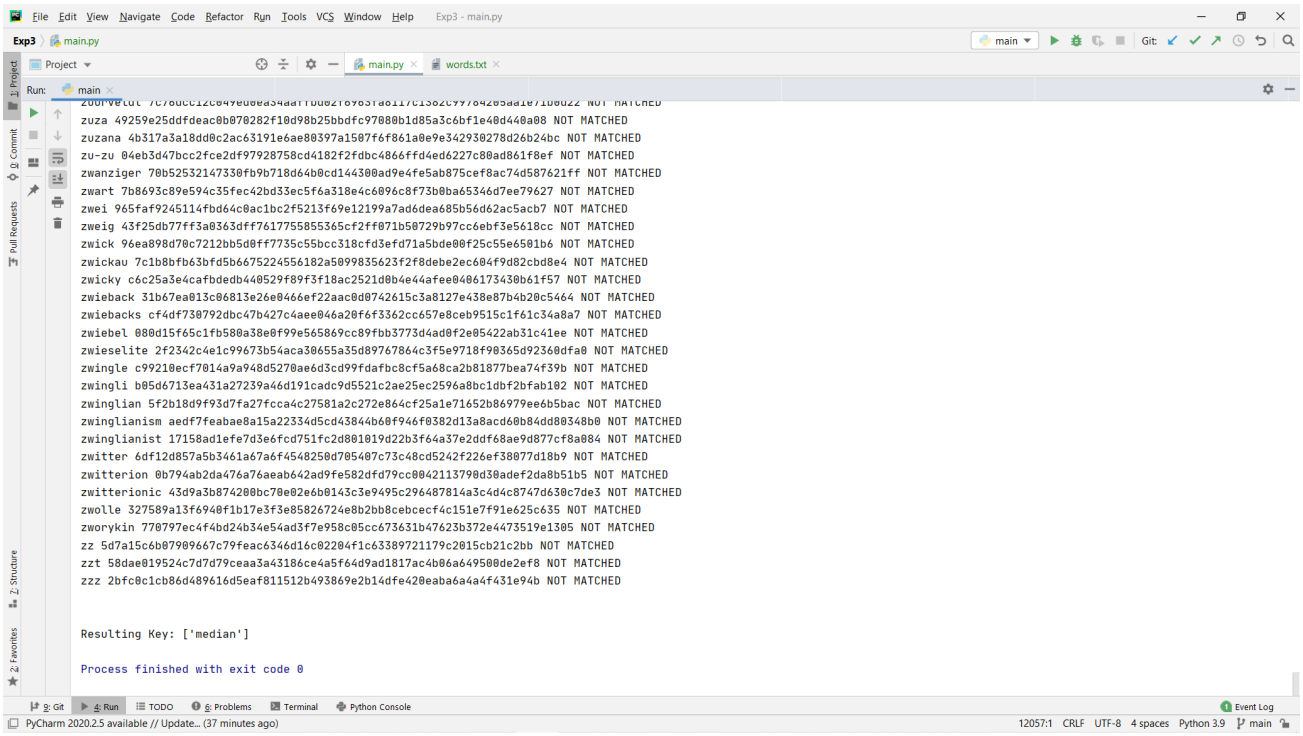
Note 2: In this task, you are supposed to write your own program to invoke the crypto library. No credit will be given if you simply use the openssl commands to do this task.

Note 3: To compile your code, you may need to include the header files in openssl, and link to openssl libraries. To do that, you need to tell your compiler where those files are. In your Makefile, you may want to specify the following:

```
INC=/usr/local/ssl/
 include/
LIB=/usr/local/ssl/
 lib/

all:
gcc -I$(INC) -L$(LIB) -o enc yourcode.c -lcrypto -ldl
```

Code:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad

plain_text = b"This is a top secret."
cipher_hex = "8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9"

result_k = []
file = open('words.txt', 'r')
lines = file.readlines()
words = [str.strip(line) for line in lines]
for word in words:
   if len(word) >= 16:
       continue
   word = word.lower()
   key = word.encode() + b' '*(16-len(word))
   cipher = AES.new(key, AES.MODE_CBC, iv=bytes.fromhex('0'*32))
   ciphertext = cipher.encrypt(pad(plain_text, AES.block_size))
   is_matched = "NOT MATCHED"
   if bytes.hex(ciphertext) == cipher_hex:
       is_matched = "MATCHED"
       result_k.append(word)
   print(word, bytes.hex(ciphertext), is_matched)
print("\n\nResulting Key:",result_k)
```



The key used to encrypt is median.

Conclusion:

1) AES, DES are symmetric key algorithms using the same keys to encrypt and decrypt the data.
2) ECB mode of encryption is the weakest form of encryption in comparison to CBC, CFB and OFB.
3) I could conclude from the experiment that ECB and CBC use padding while encryption while the other two don't. This proves that ECB and CBC are block ciphers while CFB and OFB are stream ciphers.
4) I learned how different modes react to a corrupted bit of a cipher text. The best decryption in such a case is provided by OFB where only the corrupted bit of cipher text is affected while encrypting.
5) I could conclude from this experiment that , If I have the plaintext, ciphertext and iv know , I can easily find the key using brute force method.

Github Link: https://github.com/Divya-127/CSS/