

Palestine  
An-Najah National University  
Faculty of Information Technology  
Computerized Information



دولة فلسطين  
جامعة النجاح الوطنية  
كلية تكنولوجيا المعلومات  
قسم نظم المعلومات المحوسبة

## Experiment#1 Secret Key Encryption

<b>Manar Eyad Harb</b>	<b>11924470</b>
<b>Dyaa Al-dein Ashraf Tummazeh</b>	<b>11924899</b>
<b>Yara Mohammad Sholi</b>	<b>11924207</b>

*Supervisor:*

**Dr. Amjad Hawash**

# **Abstract**

In this experiment we practiced on using openssl tool for encrypting and decrypting purposes on several encrypting algorithms and how we can use BLESS tool to modify bits of texts and represent it in many forms.

# **Introduction**

The main goal of this experiment is using openssl tool, openssl is an open-source command line tool that is commonly used to generate private keys, and apply many encrypting algorithms to produce cipher. We were required to use DES, AES, algorithms, ECB, CBC modes of operations, BLESS tool which is used to manipulate files of binary format, and Pseudo Random Number Generation was also required which is an algorithm that uses mathematical formulas to produce sequences of random numbers..

# Procedures

## I. The first is openssl installation:

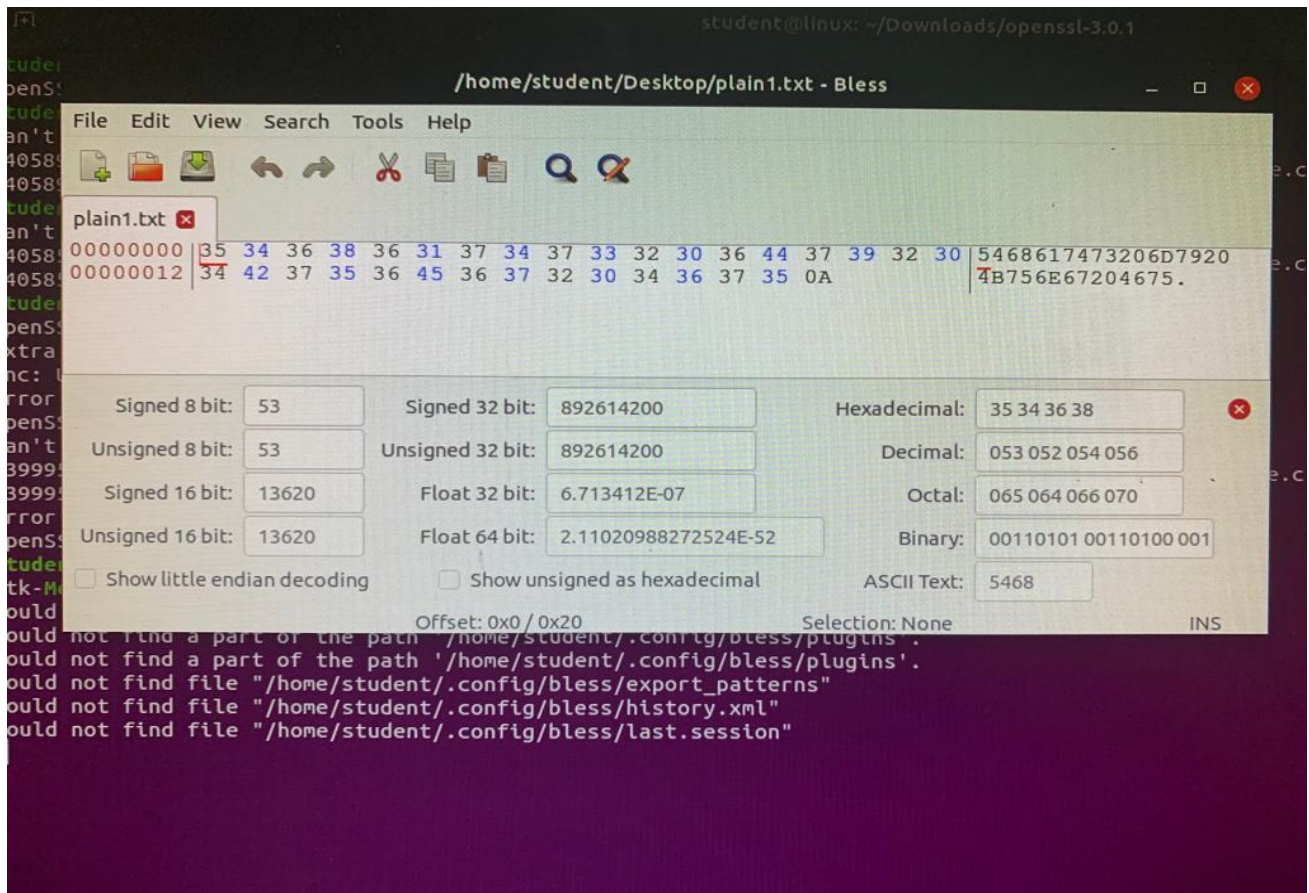
- a. Tar package was downloaded from the official website.
- b. Extraction of the package was done using the following command after directing to the file path.
- c. `sudo tar -xvzf openssl-1.1.1c.tar.gz`
- d. starting the installation process with the commands:  
`sudo ./config --prefix=/usr/local/ssl --openssldir=/usr/local/ssl`  
`shared zlib`  
`sudo make`  
`sudo make test`  
`sudo make install`

- openssl is now installed and ready to use, we made sure that it's installed using the command `openssl version -a`.

## II. Then installing BLESS:

The BLESS is a tool that are used to manipulate files of binary format was needed. There is many of tools do this task such as Frhed, but in this lab we used BLESS tool.

The installation of BLESS tool done by following this commands:  
`sudo apt-get update -y`  
`sudo apt-get install -y bless`



This figure2 shows bless tool running and opening a file with no problem.

### III. Text encryption:

In this section, encryption of a text file using different AES algorithms on OpenSSL was the topic.

- Create a text file called plain1.txt and contain the following text:  
**5468617473206D79204B756E67204675**
- Encrypting the plain1.txt using AES with 128-bit key in ECB mode by using this command:  
**openssl enc -aes-128-ecb -in plain1.txt -out file2.enc**

```
student@linux:~/Desktop$ openssl enc -aes-128-ecb -in plain1.txt -out file2.enc
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
student@linux:~/Desktop$ ls
file2.enc file.enc plain1.txt vmprep.sh
student@linux:~/Desktop$ cat file2.enc
Salted__P+++++31+++++++++b++ vY+R++|+++0+R+cp+ro+student@linux:~/Desktop$
```

Figure3.b shows the result of encrypting the file plain1.txt

**enc:** means that the wanted process is encryption.

**-aes-128-ecb:** the name of the used algorithm

**-in:** the name of the file to encrypt the text inside it

**-out:** the new file to save the cipher text inside it.

- c. After the encrypting, decrypting the cipher from previous step to ensure if the encryption is done successfully or not. Decrypting the file2.enc is done by using this command:

**openssl enc -d -aes-128-ecb -in file2.enc -out filedec.enc**

```
enc
student@linux:~/Desktop$ openssl enc -d -aes-128-ecb -in file2.enc -out
enter aes-128-ecb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
student@linux:~/Desktop$ ls
file2.enc filedec.enc file.enc plain1.txt vmprep.sh
student@linux:~/Desktop$ cat filedec.enc
cat: filedec.enc: No such file or directory
student@linux:~/Desktop$ cat filedec.enc
5468617473206D79204B756E67204675
student@linux:~/Desktop$
```

Figure3.c shows the result of decrypting file2.enc

**-d:** the wanted process is decryption

**-aes-128-ecb:** the name of the used algorithm

**-in:** the name of the file to encrypt the text inside it

**-out:** the new file to save the cipher text inside it.

- d. After of that, encrypt plain1.txt file using AES in CBC, CFB modes with a specified IV equal **0102030405060708**.

The commands that are used here is:



**For CBC:**

```
openssl enc -aes-128-cbc -in plain1.txt -out plain4.enc -iv
0102030405060708
```

### For CFB:

```
openssl enc -aes-128-cfb -in plain1.txt -out plain5.enc -iv
0102030405060708
```

```
*w@dcz0lFstudent@linux:~/Desktop$ openssl enc -aes-128-cbc -in plain1.txt -out plain4.enc -iv 0102030405060708
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
student@linux:~/Desktop$ cat plain4.enc
Salted = `oooookkkkk}oB,00boe`jxxxxU8o4o ooooo<yv-"*student@linux:~/Desktop$
```

Figure3.d1 shows the cipher text when using CBC mode

```
Activities Terminal ▾ Jan 25 16:12
student@linux: ~/Desktop
student@linux:~/Desktop$ openssl enc -aes-128-cfb -in plain1.txt -out plain5.enc -iv 0102030405060708
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
student@linux:~/Desktop$ cat plain5.enc
Salted__5o:0UmF0{v0H00IW80=00e0_0#'7900(student@linux:~/Desktop$
```

Figure3.d2 shows the cipher text when using CBC mode

- e. After the encrypting, decrypting the cipher from previous step to ensure if the encryption is done successfully or not. Decrypting the plain4.enc and plain5.enc are done by using this command:

### For plain4.enc:

```
openssl enc -d -aes-128-cbc -in plain4.txt -out plain40.enc -iv
0102030405060708
```

### For plain5.enc:

```
openssl enc d -aes-128-cfb -in plain5.enc -out plain50.enc -iv
0102030405060708
```

```
student@linux:~/Desktop$ openssl enc -d -aes-128-cbc -in plain4.enc -out plain40.enc -iv 0102030405060708
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
student@linux:~/Desktop$ cat plain40.enc
5468617473206D79204B756E67204675
student@linux:~/Desktop$ openssl enc -d -aes-128-cfb -in plain5.enc -out plain50.enc -iv 0102030405060708
enter aes-128-cfb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
student@linux:~/Desktop$ cat plain50.enc
5468617473206D79204B756E67204675
student@linux:~/Desktop$
```

Figure3.e shows the result of decryption for two files

- f. Now, what if we change the IV value and encrypting plain1.txt in CBC, CFB modes??

If we change the IV value then the cipher text will change refers to avalanche effect.

To encrypting plain1.txt in CBC, and CFB modes using this commands:

**The commands that are used here is:**

**For CBC:**

**openssl enc -aes-128-cbc -in plain1.txt -out plain00.enc -iv 1020304050607080**

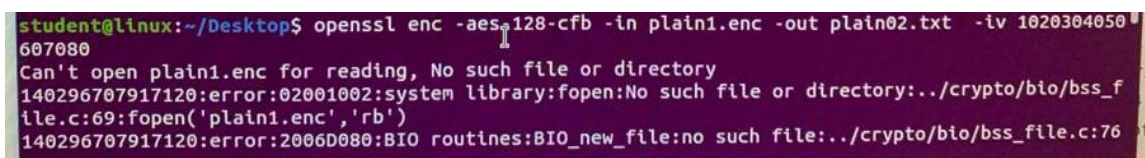
**For CFB:**

**openssl enc -aes-128-cfb -in plain1.txt -out plain02.enc -iv 1020304050607080**



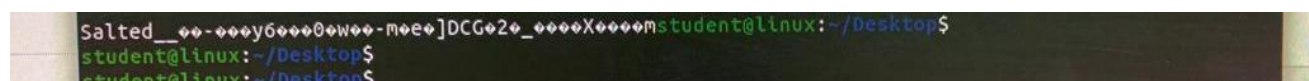
```
student@linux: ~/Desktop
student@linux:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos
student@linux:~$ cd Desktop/
student@linux:~/Desktop$ openssl enc -aes-128-cbc -in plain1.txt -out plain00.enc -iv 1020304050607080
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
student@linux:~/Desktop$ cat plain00.enc
Salted__RWKK<T)7-tX@BVRccs6vv`][00student@linux:~/Desktop$
```

Figure3.f.1 shows the ciphertext when the IV was changed using cbc mode



```
student@linux:~/Desktop$ openssl enc -aes-128-cfb -in plain1.enc -out plain02.txt -iv 1020304050607080
Can't open plain1.enc for reading, No such file or directory
140296707917120:error:02001002:system library:fopen:No such file or directory:../crypto/bio/bss_file.c:69:fopen('plain1.enc','rb')
140296707917120:error:2006D080:BIIO routines:BIIO_new_file:no such file:../crypto/bio/bss_file.c:76:
.
```

Figure3.f.2



```
Salted__yy600w-mee]DCG2_XXmstudent@linux:~/Desktop$
student@linux:~/Desktop$
student@linux:~/Desktop$
```

Figure3.f.3 shows the ciphertext when the IV was changed using cfb mode

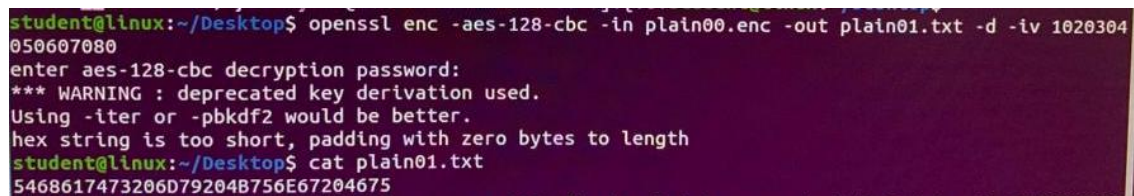
- g. After the encrypting, decrypting the cipher from previous step to ensure if the encryption is done successfully or not. Decrypting the plain4.enc and plain5.enc are done by using this command:

**For plain00.enc:**

**openssl enc -d -aes-128-cbc -in plain00.txt -out plain01.txt -iv  
0102030405060708**

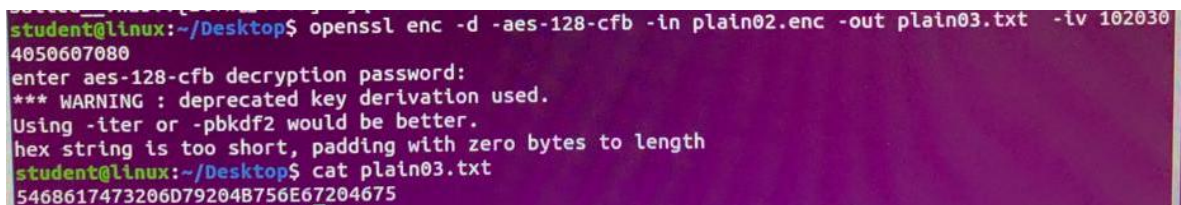
**For plain02.enc:**

**openssl enc d -aes-128-cfb -in plain02.enc -out plain03.txt -iv  
0102030405060708**



```
student@linux:~/Desktop$ openssl enc -aes-128-cbc -in plain00.enc -out plain01.txt -d -iv 1020304050607080
enter aes-128-cbc decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
student@linux:~/Desktop$ cat plain01.txt
5468617473206D79204B756E67204675
```

Figure3.f.4 decrypting plain01.txt



```
student@linux:~/Desktop$ openssl enc -d -aes-128-cfb -in plain02.enc -out plain03.txt -iv 1020304050607080
enter aes-128-cfb decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
student@linux:~/Desktop$ cat plain03.txt
5468617473206D79204B756E67204675
```

Figure3.f.5 decrypting plain03.txt

Finally, as you watch the plain text is the same if the IV was changed or not but the ciphertext was changed in the tow cases.



## IV. Image encryption

a. Download an image from internet with .bmp extension.

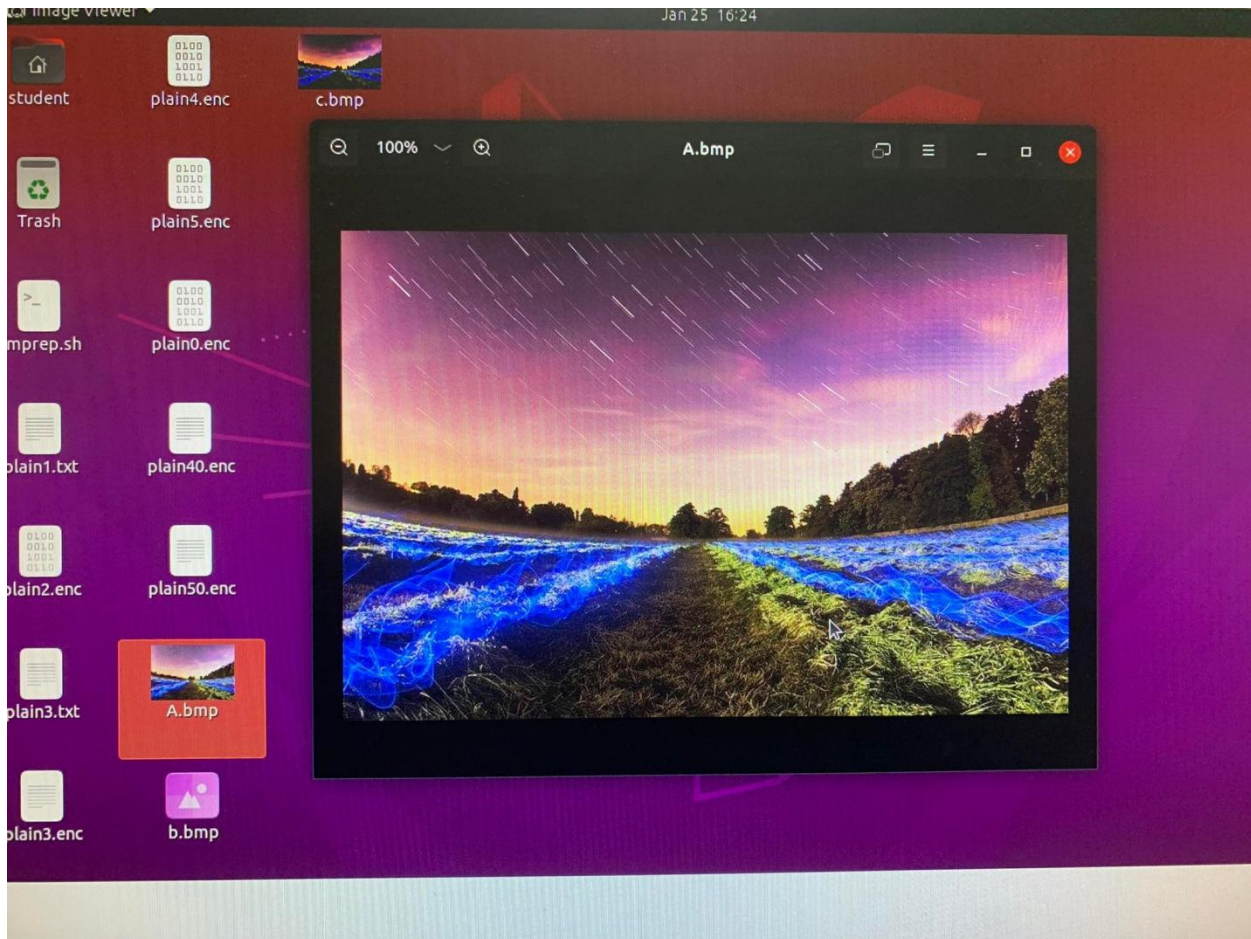


Figure4.a shows the image A.bmp

b. After that, image encryption was done using ECB mode with 3DES and a key value of 0E32932EA6D0D73, this can be done using the following OPENSSL command:

**openssl enc -des-ede3 -in A.bmp -out b.bmp -k 0E32932EA6D0D73**

After executing the previous command, the image could not be viewed.

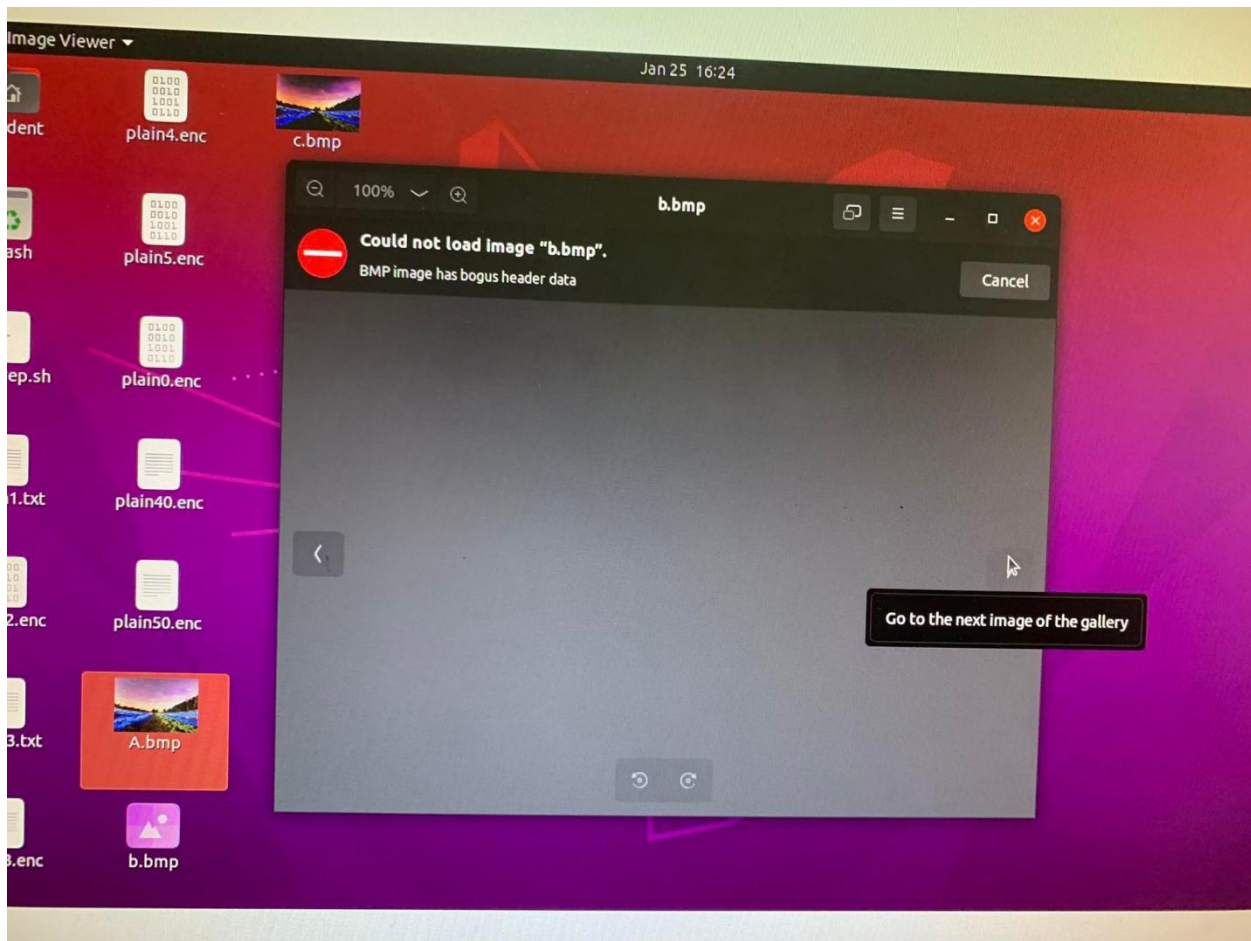


Figure4.b shows the encrypted image b.bmp

As it seems, there is no useful information when opening the encrypted picture, the only useful thing is the image extension.

c. After that, image decryption was done using the following OPENSSEL command :

**openssl enc -d -des-ede3 -in b.bmp -out c.bmp -k 0E32932EA6D0D73**

After executing the previous command, the original image got accessible again and this was the result when opening the image file.

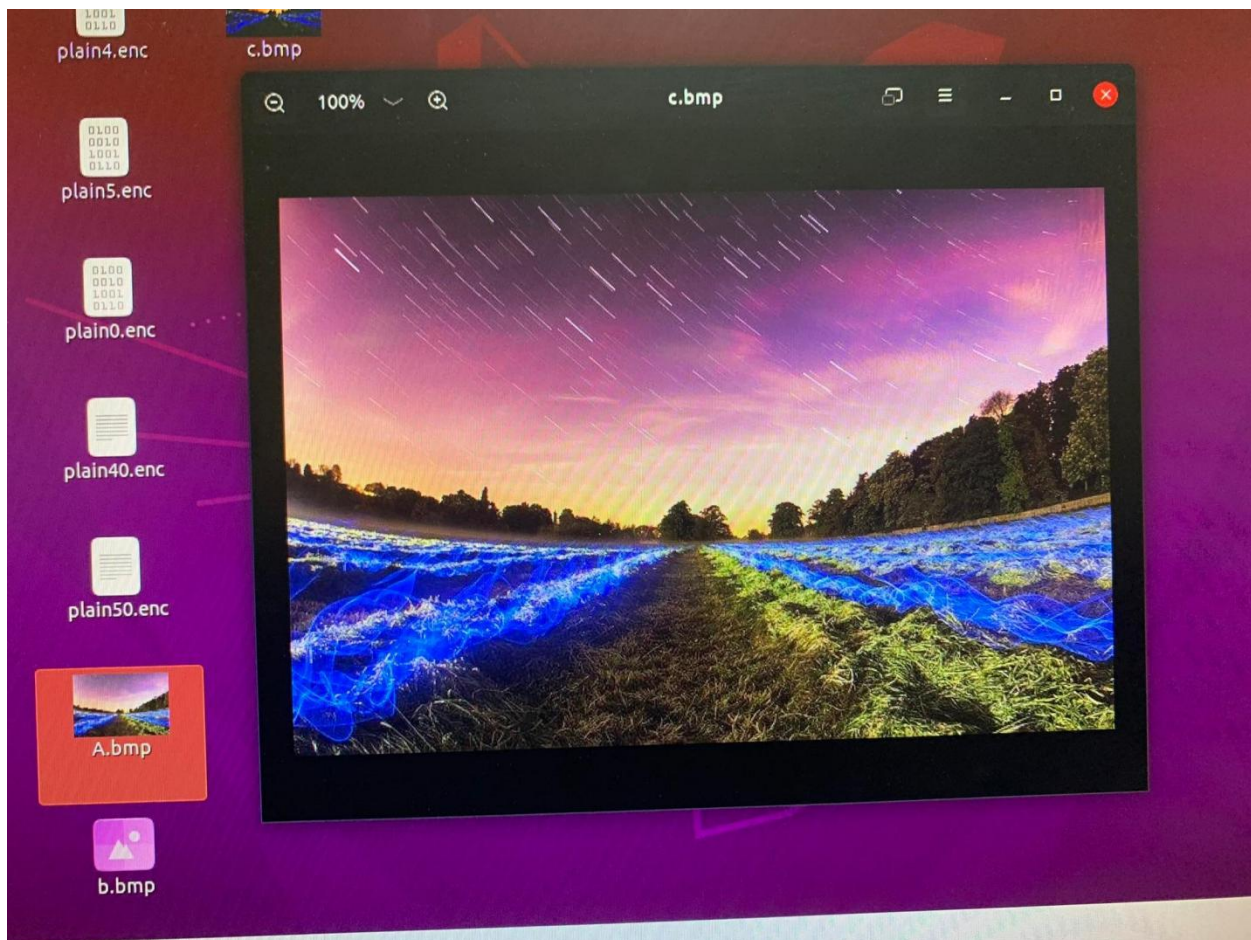


Figure4.c shows the decrypted image b.bmp

As it seems, the image can be viewed without any problem.



## V. Corrupted cipher

In this part of experiment, the 30<sup>th</sup> byte in ciphertext will change to show how much this byte will change the plain text and this effect called avalanche effect.

- a. First, create a text file that has at least 64 bytes length. The plain text in the file is :

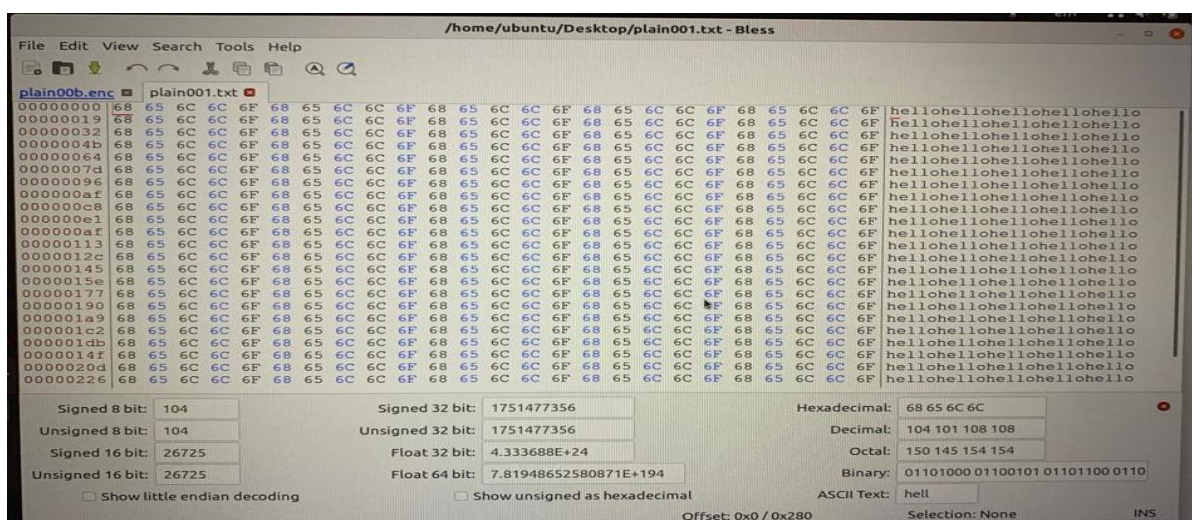


Figure5.a plain text of plain001.txt

- b. Then, Encrypt the file using AES-128 bit cipher with the ECB mode of operation by using this command:
- ```
openssl enc -aes-128-ecb -in plain001.txt -out plain002.enc
```

The result of this command is:

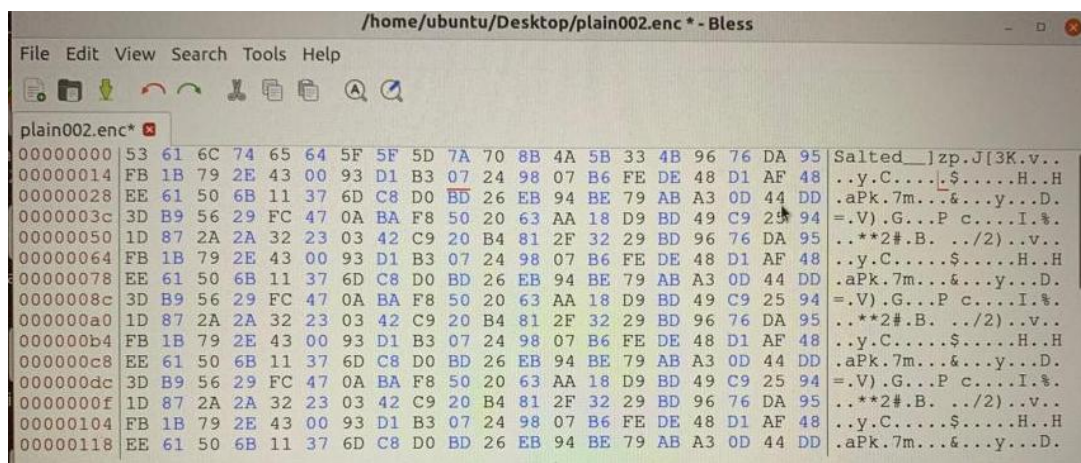


Figure5.b The cipher text of plain001.txt



c. Now, assume that a single bit of the 30<sup>th</sup> byte in the encrypted file got corrupted and change its value manually by using bless hex editor.

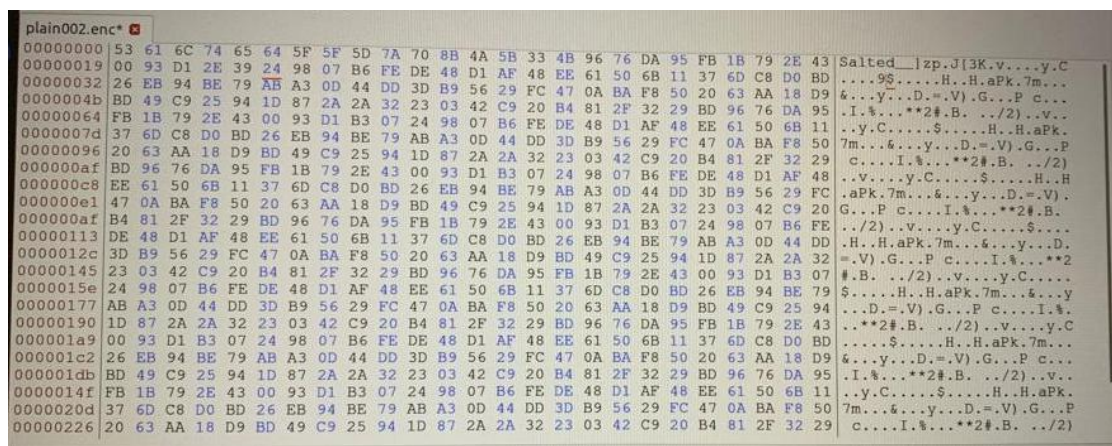


Figure5.c the cipher text after changing 30<sup>th</sup> byte in plain002.enc

d. After that, decrypt the corrupted ciphertext using the key and IV you used in the encryption to show how this byte affected plain text. The decryption is done by using this command:

**openssl enc -d -aes-128-ecb -in plain002.enc -out plain003.txt**



Figure5.d This figure shows the plain text of corrupted file.

e. Repeat the previous steps using CBC, CFB, and OFB modes of operations.

**-Using CBC :**

First, encrypt plain001.txt by using this command:

**openssl enc -aes-128-cbc -in plain001.txt -out plain00c.enc -iv 1020304050607080**

Then, repeat step c on plain00c.enc file.

```
openssl enc -d -aes-128-cbc -in plain00c.enc -out plain01c.enc -iv
1020304050607080
```

Figure 5.e.1 this figure shows how much plain text was changed.

First, encrypt plain001.txt by using this command:

Then, repeat step c on plain00f.enc file.

```
openssl enc -d -aes-128-cfb -in plain00f.enc -out plain01f.enc -iv
1020304050607080
```

Figure5.e.2 this figure shows how much plain text was changed.



### -Using OFB:

First, encrypt plain001.txt by using this command:

```
openssl enc -aes-128-ofb-in plain001.txt -out plain00b.enc -iv
1020304050607080
```

Then, repeat step c on plain00b.enc file.

Finally, decrypt plain00c.enc to show how much plain text was affected by following this command:

```
openssl enc -d -aes-128-cfb -in plain00b.enc -out plain01b.enc -iv
1020304050607080
```

[illegible]

Figure 5.3 this figure shows how much plain text was changed.

As you show in the last two steps, when a 1-byte changed in the ciphertext that is encrypted using ECB, CBC, and CFB modes a lot of bytes will change because these modes are a block cipher mode, and changing one byte means changing one block, but, in OFB, its a block cipher that is convert the block into a synchronous stream cipher and encrypt one byte at a time, so when changing one byte in ciphertext, then only one byte will be changed in plain text.

## VI. Padding analysis

- a. Create a plaintext in a text file that is 20 octets long, 20 octets its equivalent with 20-byte file with text data in octal numbering system.

b. Encrypt the plaintext using AES with the following modes of operations:  
ECB, CBC, CFB, and OFB. Choose the key and IVs that you prefer.

#### **-Using ECB:**

Encrypt the plain text by using this command: (file text called octets-string.txt)



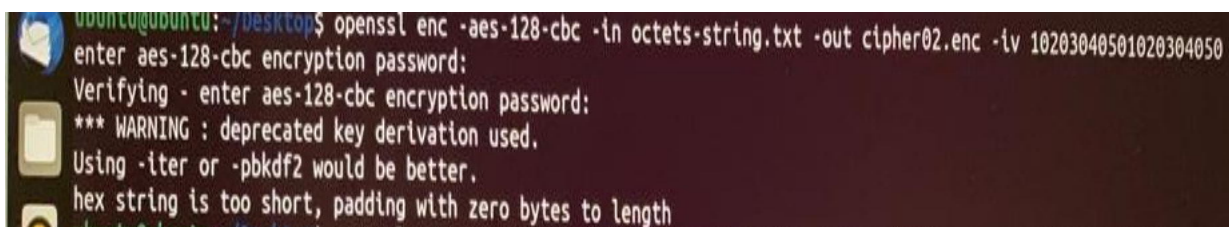
```
ubuntu@ubuntu:~/Desktop$ openssl enc -aes-128-ecb -in octets-string.txt -out cipher01.enc
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

Figure6.b.1 this figure shows the result of encrypting text file using ECB

When using this encryption mode there is no padding happened, because, the length of data that was encrypted was an exact multiple of block length(b).

#### **-Using CBC:**

Encrypt the plain text by using this command:



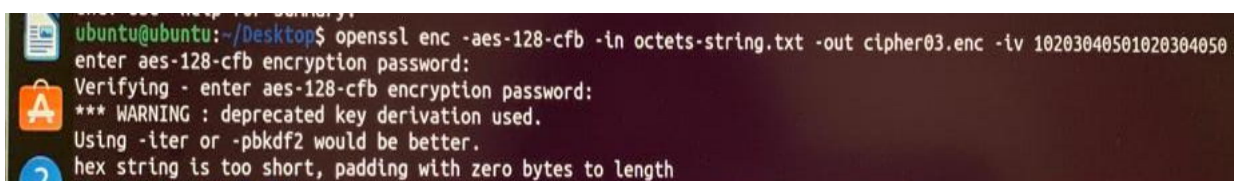
```
ubuntu@ubuntu:~/Desktop$ openssl enc -aes-128-cbc -in octets-string.txt -out cipher02.enc -iv 10203040501020304050
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
```

Figure6.b.2 this figure shows the result of encrypting text file using ECB

But, when using this encryption mode there is padding, because, the hex string is shorter than b ( length of the block).

#### **-Using CFB:**

Encrypt the plain text by using this command:



```
ubuntu@ubuntu:~/Desktop$ openssl enc -aes-128-cfb -in octets-string.txt -out cipher03.enc -iv 10203040501020304050
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
```

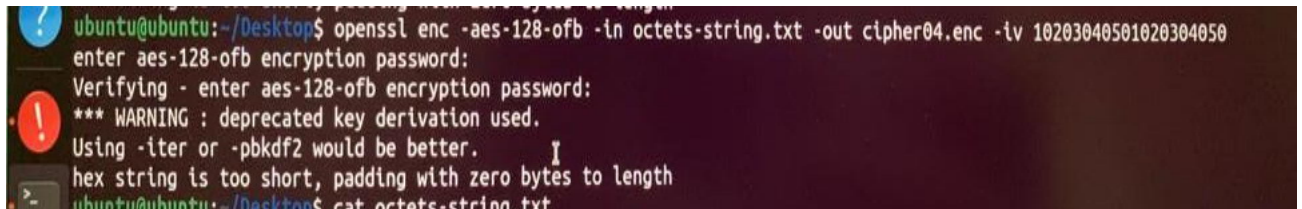
Figure6.b.3 this figure shows the result of encrypting text file using ECB



And, when using this encryption mode there is padding, because, the hex string is shorter than b ( length of the block).

### -Using ECB:

Encrypt the plain text by using this command:



```
ubuntu@ubuntu:~/Desktop$ openssl enc -aes-128-ofb -in octets-string.txt -out cipher04.enc -iv 10203040501020304050
enter aes-128-ofb encryption password:
Verifying - enter aes-128-ofb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
hex string is too short, padding with zero bytes to length
ubuntu@ubuntu:~/Desktop$ cat octets-string.txt
```

Figure6.b.3 this figure shows the result of encrypting text file using ECB

In this encryption there is padding too, because, the hex string is shorter than b ( length of the block).

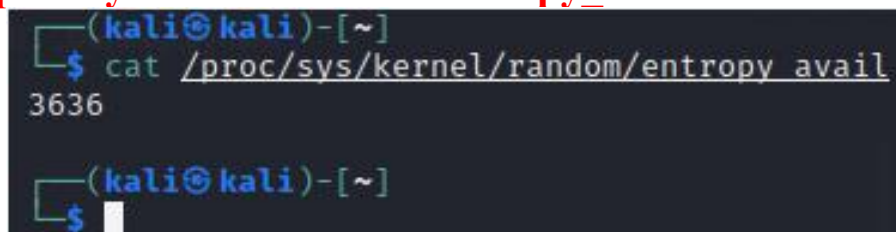
Finally, as you show all of these modes need padding when the data for the final block that will be encrypted was shorter than the block length(b).

## VII. Pseudo random number generation

A pseudo random number generator (PRNG) is an algorithm for generating a sequence of numbers whose properties approximate the properties of sequences of random numbers. In computer security, pseudo randomness is important in encryption algorithms, which create codes that must not be predicted or guessed.

a. To measure the randomness on our machine, the following command was used :

**cat /proc/sys/kernel/random/entropy\_avail**



```
(kali@kali)-[~]
$ cat /proc/sys/kernel/random/entropy_avail
3636
(kali@kali)-[~]
$
```

Figure7.a the output of the previous command

b. After moving the mouse around and typing some text, the re-execution of the above command resulted in 3440 :

```
(kali㉿kali)-[~]  
$ ajscbasc  
ajscbasc: command not found  
  
(kali㉿kali)-[~]  
$ cat /proc/sys/kernel/random/entropy_avail  
3440  
  
(kali㉿kali)-[~]  
$
```

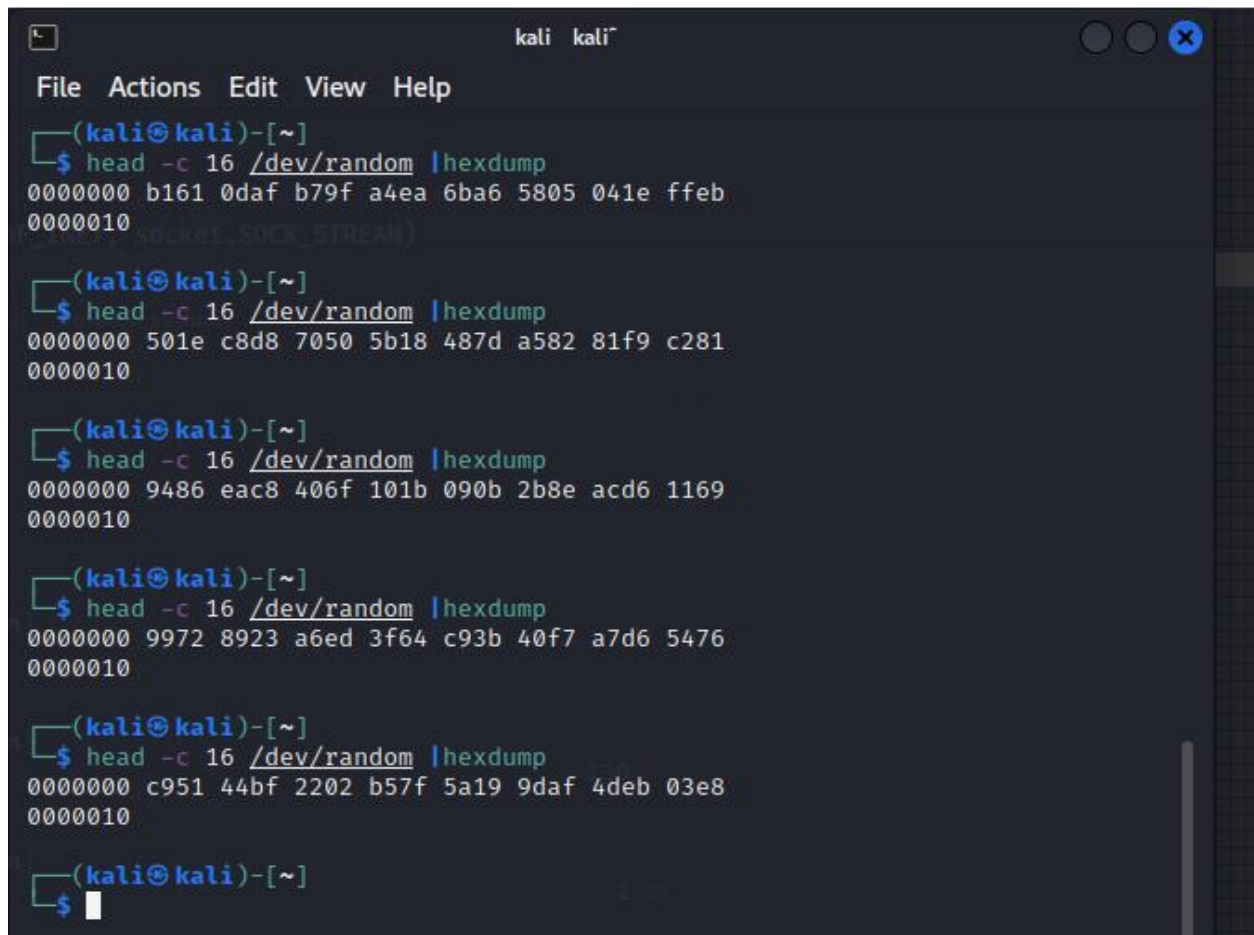
Figure7.b

c. As it seems, the number is changing and it gets effected by whats going on in the machine, like mouse moves and keyboard clicks, etc.

```
(kali㉿kali)-[~]  
$ head -c 16 /dev/random |hexdump  
00000000 9cbd 02a6 f5c3 ceef 72e9 9c94 dc2e a59c  
00000010  
  
(kali㉿kali)-[~]  
$
```

Figure7.c

d. Several tries and excutions of the same above command was done, and there were no any waiting or problems, as it`s shown in the following figure:



```
kali kali~
File Actions Edit View Help

(kali@kali)-[~]
$ head -c 16 /dev/random |hexdump
00000000 b161 0daf b79f a4ea 6ba6 5805 041e ffeb
00000010 socket, SOCK_STREAM)

(kali@kali)-[~]
$ head -c 16 /dev/random |hexdump
00000000 501e c8d8 7050 5b18 487d a582 81f9 c281
00000010

(kali@kali)-[~]
$ head -c 16 /dev/random |hexdump
00000000 9486 eac8 406f 101b 090b 2b8e acd6 1169
00000010

(kali@kali)-[~]
$ head -c 16 /dev/random |hexdump
00000000 9972 8923 a6ed 3f64 c93b 40f7 a7d6 5476
00000010

(kali@kali)-[~]
$ head -c 16 /dev/random |hexdump
00000000 c951 44bf 2202 b57f 5a19 9daf 4deb 03e8
00000010

(kali@kali)-[~]
$
```

Figure7.d

## Conclusion

In the end, using encryption block modes and algorithm to save the data which can be in any type such as image or text file. And the strength of any encryption block modes depends on the key, IV, and the avalanche effect.