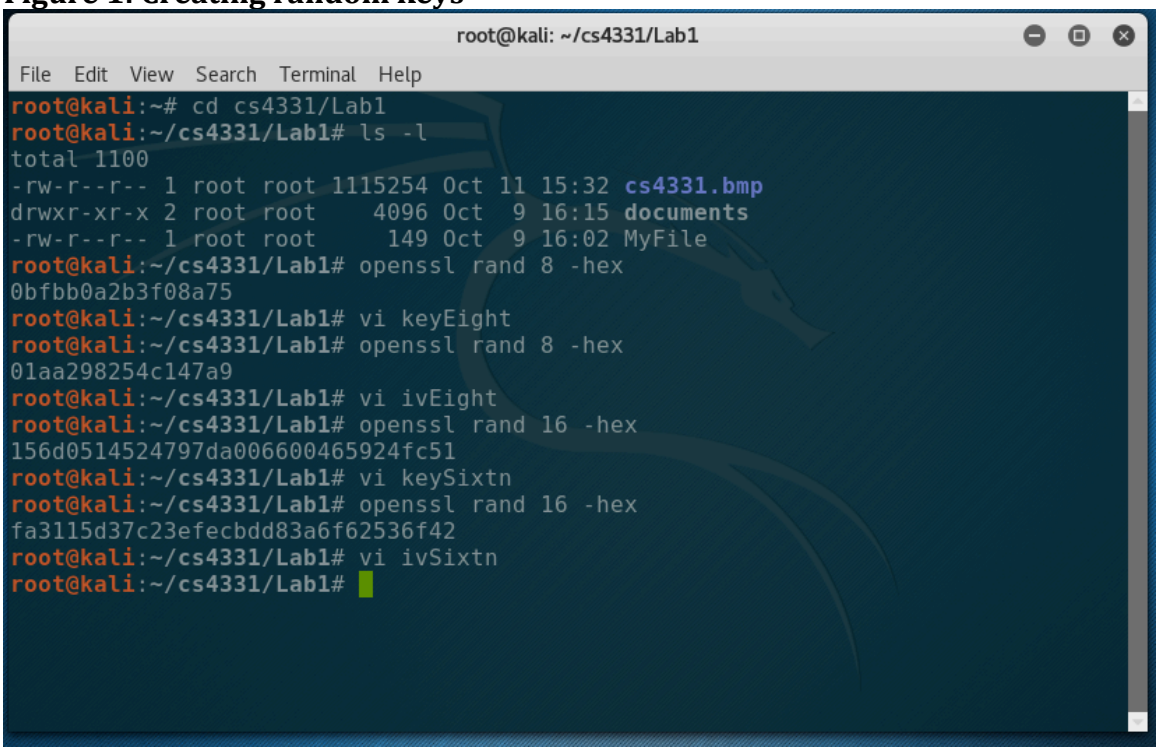
LAB 1: Modes of Operation

1. Creating random keys:
   a) First I created my random keys and the initialization vectors. See **Figure 1** below.
   b) I noticed after creating the random keys that when I want to use them to encrypt in their respective modes. You have to directly type in the key versus typing in the file name where you saved the key.

Figure 1: Creating random keys



2.1 Encrypting the image file
   a) First I viewed the unencrypted image file cs4331.bmp. See **Figure 2.1a** below
   b) I then encrypted the image file using (a) –des-ecb, (b) –aes-128-ecb, (c) –aes-128-cbc, (d) –des-cbc. See **Figure 2.1b** below.

Figure 2.1a: Viewing unencrypted cs4331.bmp

Figure 2.1b: Encrypting cs4331.bmp

```
root@kali:~/cs4331/Lab1# openssl enc -des-ecb -e -in cs4331.bmp -out cs4331desecb.bmp -K 0bfbb0a2b3f08a75
root@kali:~/cs4331/Lab1# openssl enc -aes-128-ecb -e -in cs4331.bmp -out cs4331aesecb.bmp -K 156d0514524797da006600465924fc51
root@kali:~/cs4331/Lab1# ls -l
total 3300
-rw-r--r-- 1 root root 1115264 Oct 11 15:41 cs4331aesecb.bmp
-rw-r--r-- 1 root root 1115254 Oct 11 15:32 cs4331.bmp
-rw-r--r-- 1 root root 1115256 Oct 11 15:40 cs4331desecb.bmp
drwxr-xr-x 2 root root    4096 Oct  9 16:15 documents
-rw-r--r-- 1 root root      15 Oct 11 15:33 ivEight
-rw-r--r-- 1 root root      34 Oct 11 15:34 ivSixtn
-rw-r--r-- 1 root root      18 Oct 11 15:33 keyEight
-rw-r--r-- 1 root root      33 Oct 11 15:34 keySixtn
-rw-r--r-- 1 root root     149 Oct  9 16:02 MyFile
root@kali:~/cs4331/Lab1# openssl enc -aes-128-cbc -e -in cs4331.bmp -out cs4331aescbc.bmp -K 156d0514524797da006600465924fc51 -iv fa31
15d37c23efecbdd83a6f62536f42
root@kali:~/cs4331/Lab1# openssl enc -des-cbc -e -in cs4331.bmp -out cs4331descbc.bmp -K 0bfbb0a2b3f08a75 -iv 01aa298254c147a9
root@kali:~/cs4331/Lab1# ls -l
total 5484
-rw-r--r-- 1 root root 1115264 Oct 11 15:41 cs4331aescbc.bmp
-rw-r--r-- 1 root root 1115264 Oct 11 15:41 cs4331aesecb.bmp
-rw-r--r-- 1 root root 1115254 Oct 11 15:32 cs4331.bmp
-rw-r--r-- 1 root root 1115256 Oct 11 15:42 cs4331descbc.bmp
-rw-r--r-- 1 root root 1115256 Oct 11 15:40 cs4331desecb.bmp
drwxr-xr-x 2 root root    4096 Oct  9 16:15 documents
```

2.2 Viewing the encrypted cs4331.bmp

a) First I attempted to view the encrypted files using an image view of my choice. The image viewer was not able to load the encrypted files because the headers to the files were unrecognizable.

b) Next installed bless hex editor in order to view the encrypted files. The files open in bless and the binary code that makes up the images is represented by hexadecimal.

c) I then copied the first 54 bytes of the unencrypted cs4331.bmp and used them to replace the first 54 bytes of by encrypted cs4331.bmp files. I was then able to view the images in a image view of my choice. See **Figure 2.2a-2.2d** below.

Figure2.2a Viewing encrypted cs.4331 file (Image Viewer: -des-ebc)

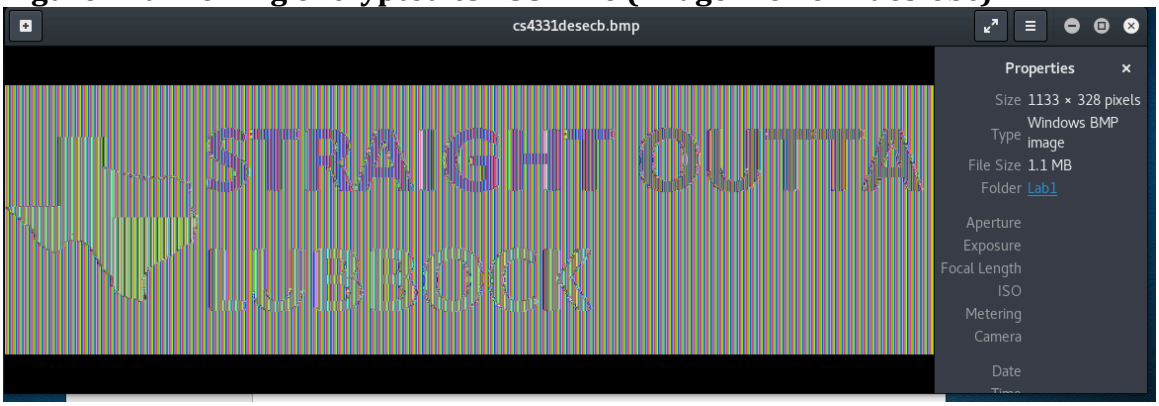


Figure2.2b: Viewing encrypted cs.4331 file (Image Viewer: -aes-128-ebc)

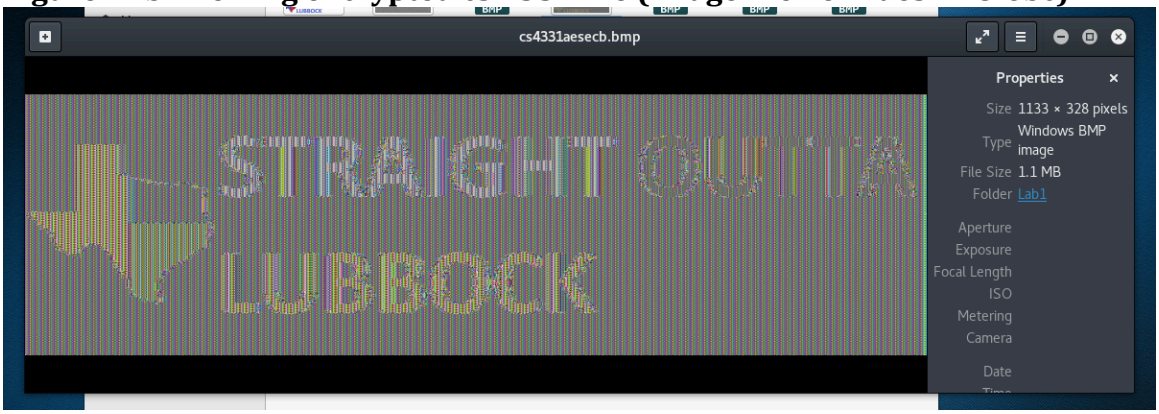
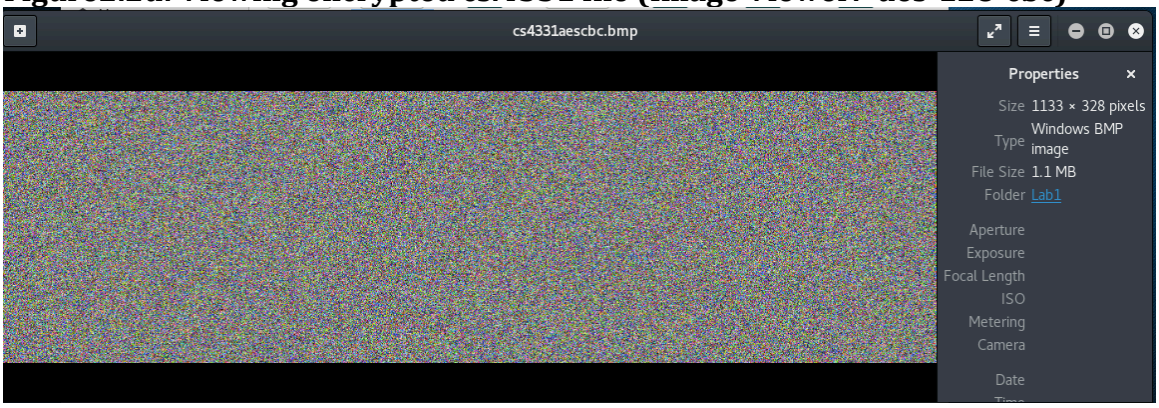


Figure2.2c: Viewing encrypted cs.4331 file (Image Viewer: -des-cbc)

Figure2.2d: Viewing encrypted cs.4331 file (Image Viewer: -aes-128-cbc)



d) I am not impressed with the –ebc mode. The images did not encrypt well in this mode. As you can tell, the image is still viewable. The only difference is the color scheme and the arrangement of the pixels. The structure of the pixels in the original image is still visible. In other words the structure of the binary form the original image is still represented after encrypting with –des-ebc, and –aes-128-ecb.

e) I am very impressed with the –cbc mode. The images were encrypted very successfully in this mode. When viewing these images the pixels are completely scrambled and the image is not recognizable. I would use this mode of operation either in –aes-128-cbc or –des-128-cbc. The images original binary structure is not represented in the encrypted image.

f) The differences in my observations for –ecb and –cbc are as follows. The –ebc mode only randomized the colors scheme of the pixels. The random placement of the pixels still held a structure as well. The overall structure of the original binary is also still represented. The –cbc mode randomized everything and did not hold a structure after doing so. The original images binary structure is no longer recognizable and the encrypted image is completely rearranged. No color recognizable color scheme or structure of the image. You can see these differences in the hexadecimal representation as well. See **Figure 2.3a-2.3d** below.

Figure2.3a: Viewing encrypted cs.4331 file (Bless: -des-ecb)

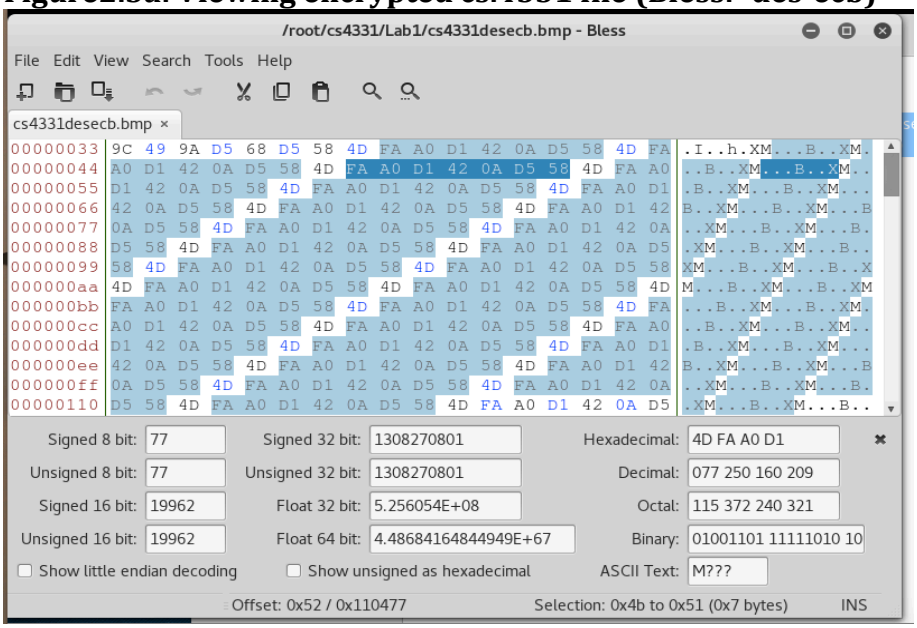


Figure2.3b: Viewing encrypted cs.4331 file (Bless: -aes-128-ecb)

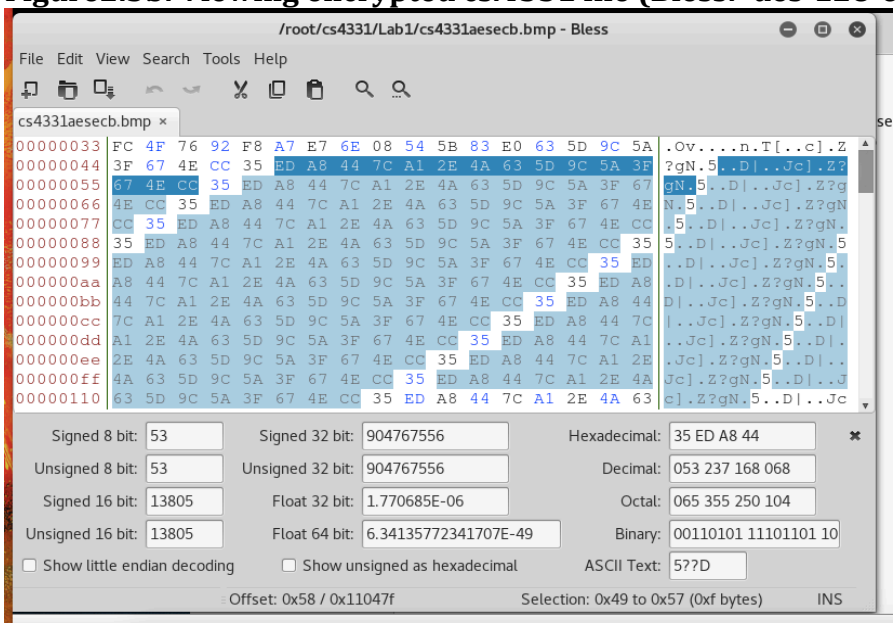


Figure2.3c: Viewing encrypted cs.4331 file (Bless: -des-cbc)

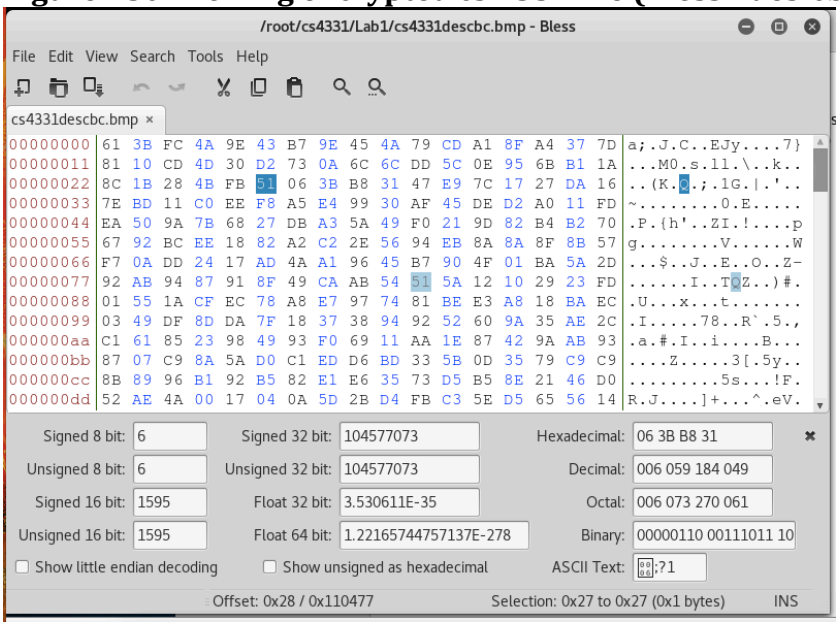
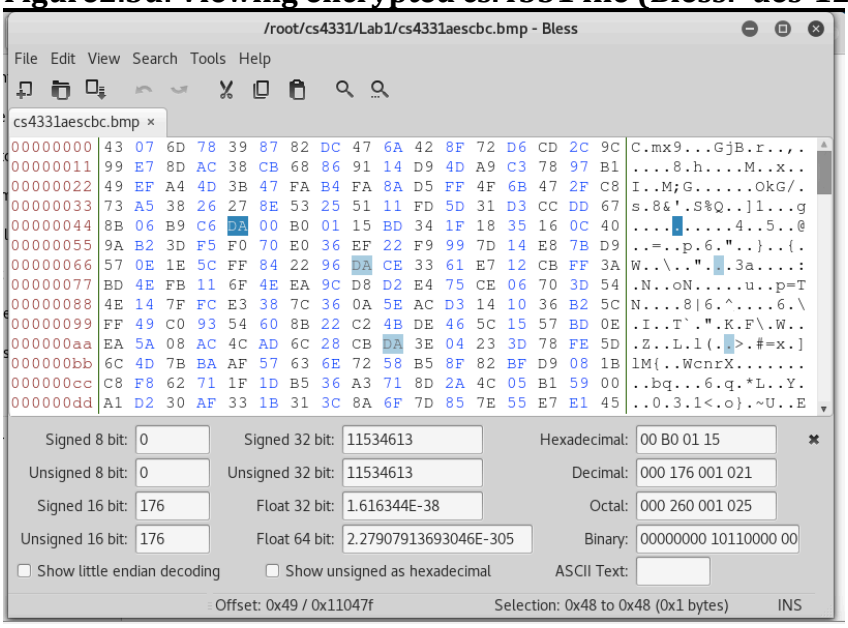


Figure2.3d: Viewing encrypted cs.4331 file (Bless: -aes-128-cbc)



3. Comparing –ecb, -cbc, -ofb, -cfb
   a) First I created a small text file named MyFile.bin. See **Figure 3.1** below.
   b) Then I encrypted the MyFile using –aes in the following modes, -ecb, -cbc, -ofb, and –cfb. See **Figure 3.2** below.

Figure 3.1: MyFile.bin

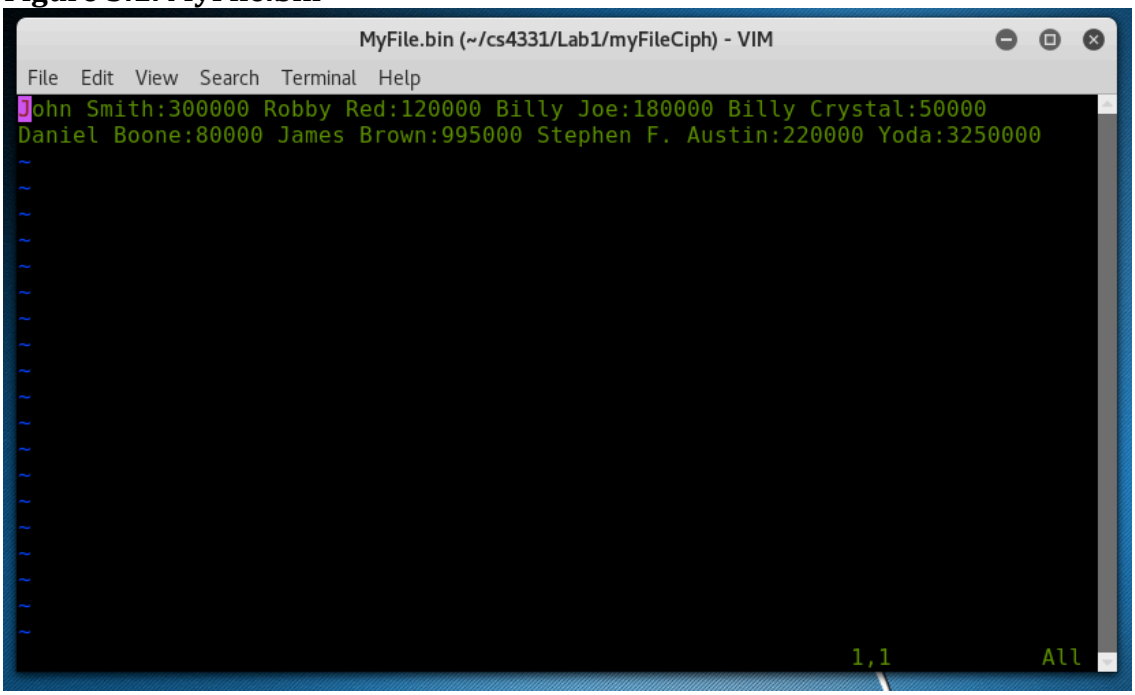
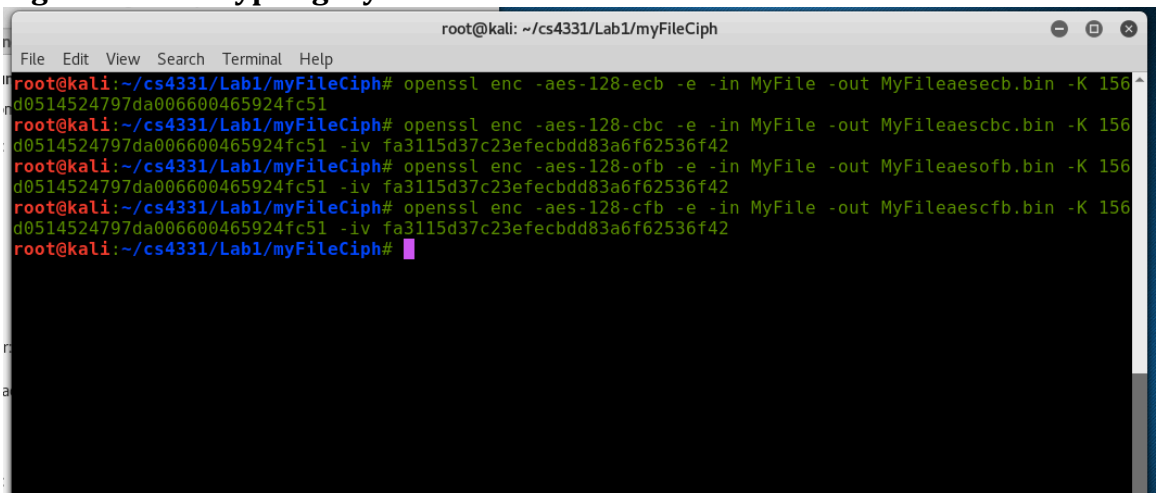


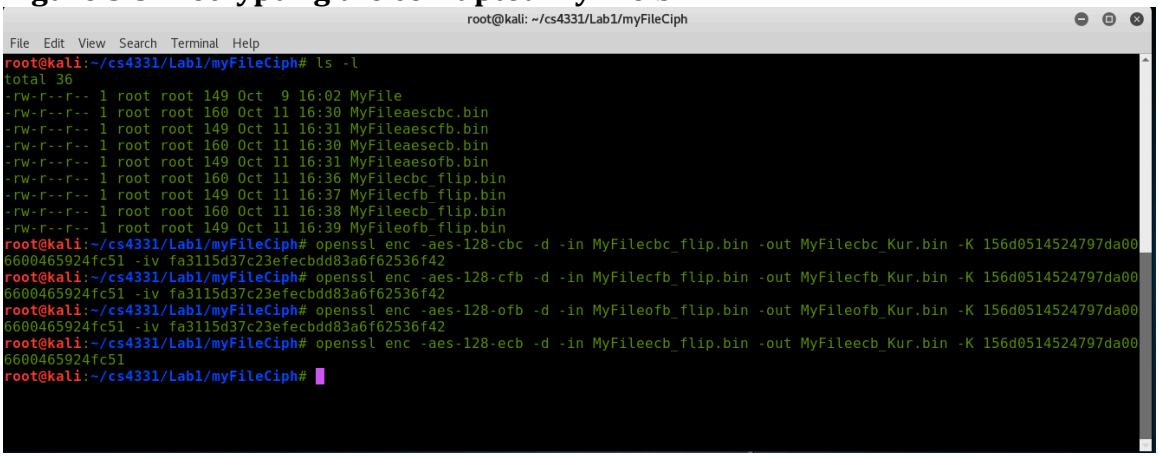
Figure 3.2: Encrypting MyFile.bin



c) After the encryption I then flipped a single bit in the first block of plain text. I used the bless editor to flip the bits.

d) I then decrypted the corrupted encrypted files. See **Figure 3.3** below.

Figure 3.3: Decrypting the corrupted MyFile.bin



e) After the decryption I began the final task of viewing the corrupted files one at a time and comparing them to the original plain text file. I then noticed the differences and how they relate to the modes of operation.
   a. –ecb: The simplest mode where plaintext is handled one block at a time and each block of text is encrypted using the same key.
      i. The issues with –ecb are as follows: if the same block of plaintext appears more than once in the message, it always produces the same cipher text (this can be seen when encrypting the cs4331.bmp file). For big messages the –ecb mode may not be secure, but is ideal for a short amount of data, such as encryption key i.e. transmission of a –des or –aes key securely.
      ii. When decrypting the corrupted MyFile –ecb decrypts all blocks with the same key. Therefore the corrupted file decrypts and destroys the entire block of code with the flipped bit and is inherited by only a few plaintext blocks. See **Figure 3.4a** below
   b. –cbc: Repeated plaintext blocks produce different cipher text blocks and the same key is used for each block. Input encryption function for each plaintext block bears no fixed relationship to the plaintext block. Repeating patterns are not exposed.
      i. Error recovery –cbc. The mode has a self-healing property. If one block of cipher text is altered, the error propagates for at most 2 blocks. See **Figure 3.4b** below.
   c. –cfb: Converts block cipher into a stream cipher. Eliminating the nee d to pad a message to be a integral number of blocks. Also it operates in real time. Thus, each character encrypted and transmitted immediately using a character-oriented stream cipher.
      i. Error recovery –cfb. The mode deals with corruption almost in the same way as –cbc, but because it has the character-oriented stream cipher trait. Only the flipped bit is corrupted in the first block of plaintext. With that being said it also has the chain properties of –cbc, so the following block of plaintext is dependent on the first block. Although it does have a self –

healing property as well, so at most the following 2 blocks will be destroyed. See **Figure 3.4c** below.

d. –ofb: Similar in structure to the –cfb, except that the output of encryption is fed back to become the input encrypting block of the plaintext. It also uses an IV and the IV must be a nonce.
    i. Error recovery –ofb. The mode has a great advantage; transmission errors do not propagate throughout out the plaintext. See **Figure 3.4d** below.

Figure 3.4a: Corrupted Myfile (-aes-128-ecb)

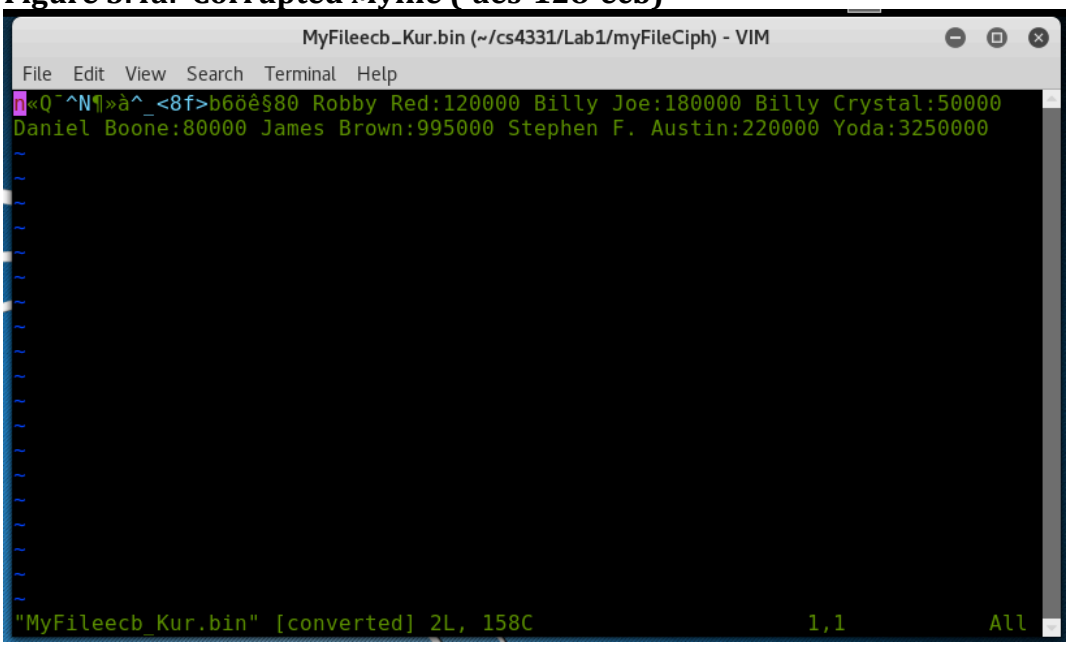


Figure 3.4b: Corrupted Myfile (-aes-128-cbc)

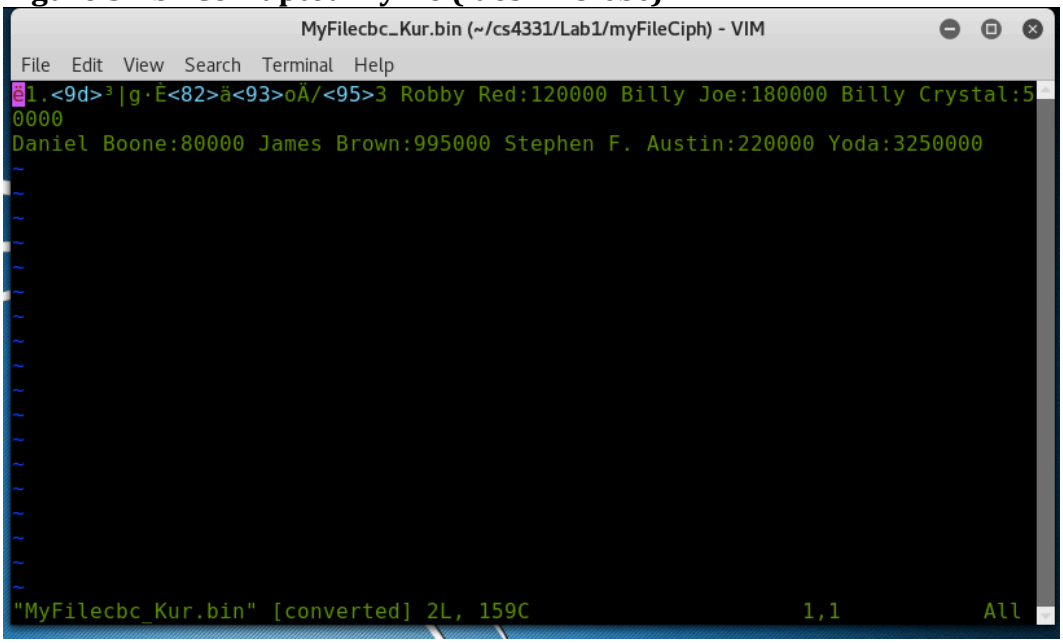


Figure 3.4c: Corrupted Myfile (-aes-128-cfb)

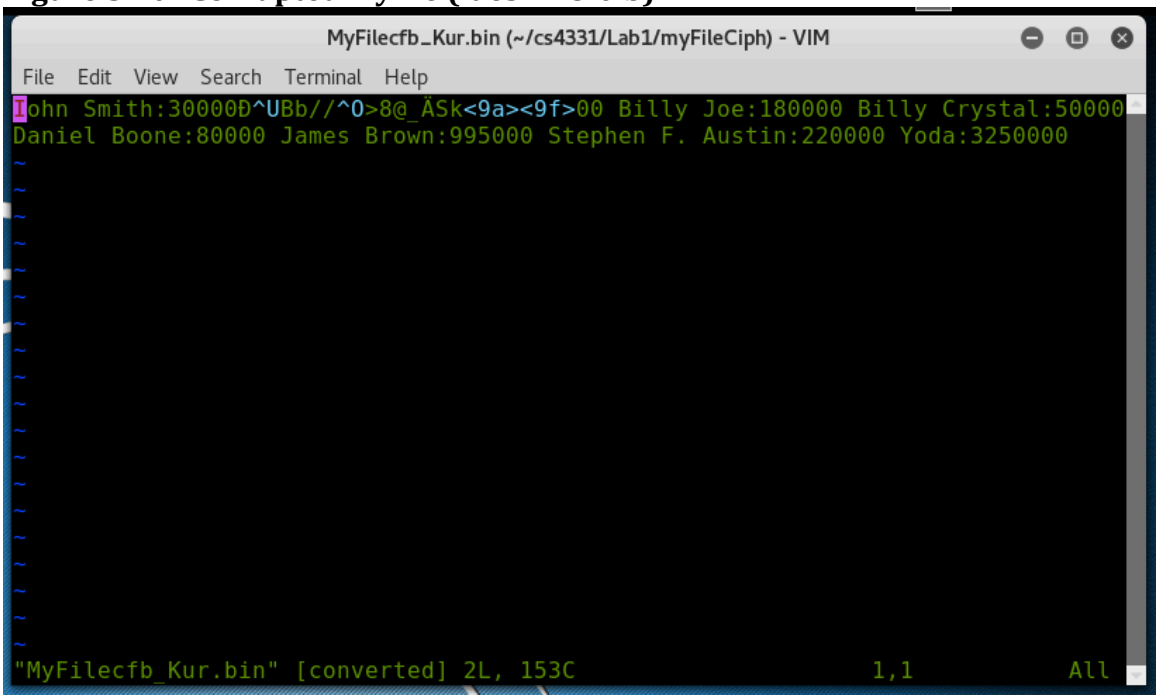


Figure 3.4d: Corrupted Myfile (-aes-128-ofb)