Crypto Lab I – Symmetric-Key Encryption

IMPORTANT NOTES: Study lecture materials at least 1 hour and attempt the tasks in Section 2 prior to the lab session. Prepared questions will be discussed in the lab session.

1 Overview

The learning objective of this lab is for students to get familiar with the concepts in the secret-key (symmetric key) encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms, avalanche effect, and modes of operation.

2 Lab Environment

Create the plaintext file. Run the command `echo "Say the year is the year of the phoenix" > plain.txt` will create a file in current directory or use a text editor such as `nano` to create the text file.

OpenSSL. In this lab, 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 enc`.


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


Please replace the `ciphertype` with a specific cipher type, such as `-aes-128-cbc`, or `-aes-128-ecb`. 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)
```

3 Lab Task: Encryption Mode – ECB vs. CBC

The file `art.bmp` in the `/srv/fit2093files/fit2093lab` folder contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture.

1. Copy the `art.bmp` file to the home directory and encrypt the `art.bmp` file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes.

```
cp /srv/fit2093files/fit2093lab/art.bmp ~
cd
```

To encrypt the file using the ECB run the following command:

```
openssl enc -aes-128-ecb -K  00112233445566778899aabbccddeeff -in art.bmp -out art-ecb.bin
```

The ECB mode of operation does not require an IV. Repeat the steps for the CBC mode of operation.

```
openssl enc -aes-128-cbc -K  00112233445566778899aabbccddeeff -iv 0102030405060708 -in art.bmp
-out art-cbc.bin
```

The CBC mode require an IV of size 128-bit which in the above case the provided IV is 64 bits. The second half of the IV will be set to zero.

2. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. However, 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 a hex editor tool (e.g. `bless`) to directly modify binary files. Your tutor will demonstrate how to do this.

   To edit the first 54 byte of the ECB ciphertext file (so it would be considered a valid bitmap file) first run the command `bless art.bmp` and copy the first 54 bytes (each row is 18 byte) then open the ciphertext file `bless art-ecb.bin` and select the first 54 byte of this file and then paste the copied hex value and save the file. Change the extension of the file to bmp using `mv art-ecb.bin art-ecb.bmp` command.

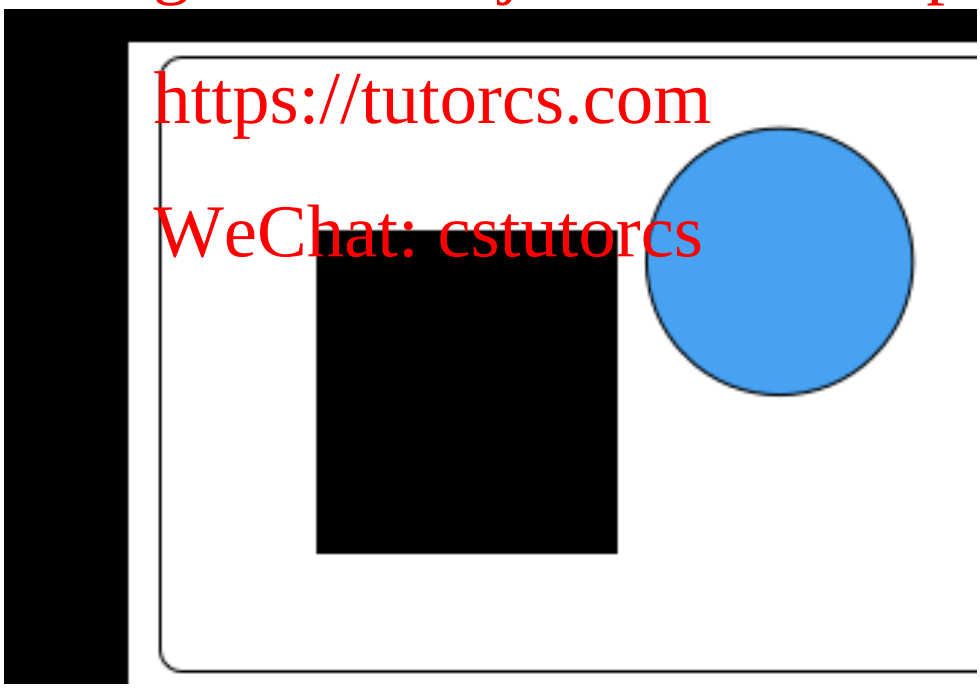

Figure 1: The bitmap image

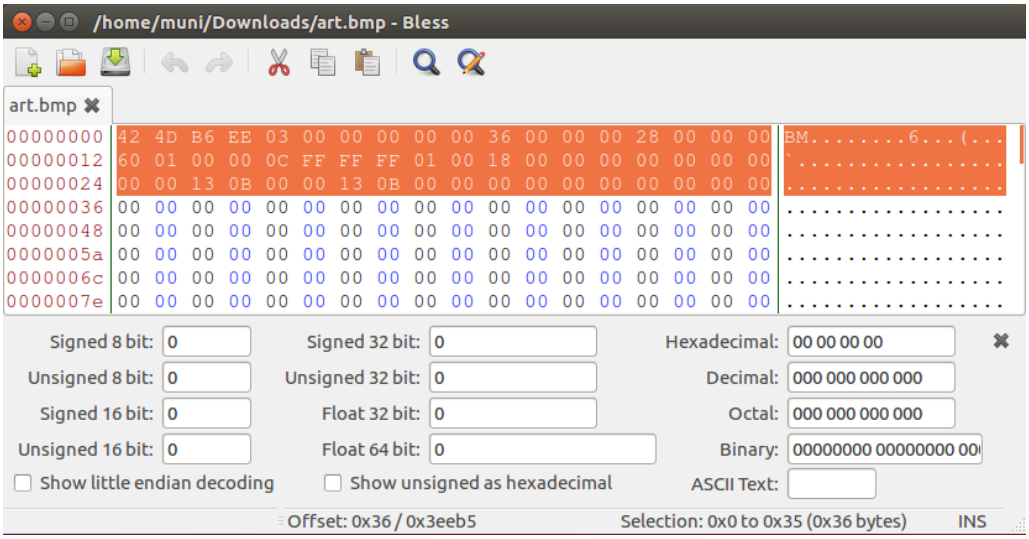

Figure 2: Copying the first 54 byte of the art.bmp file using bless

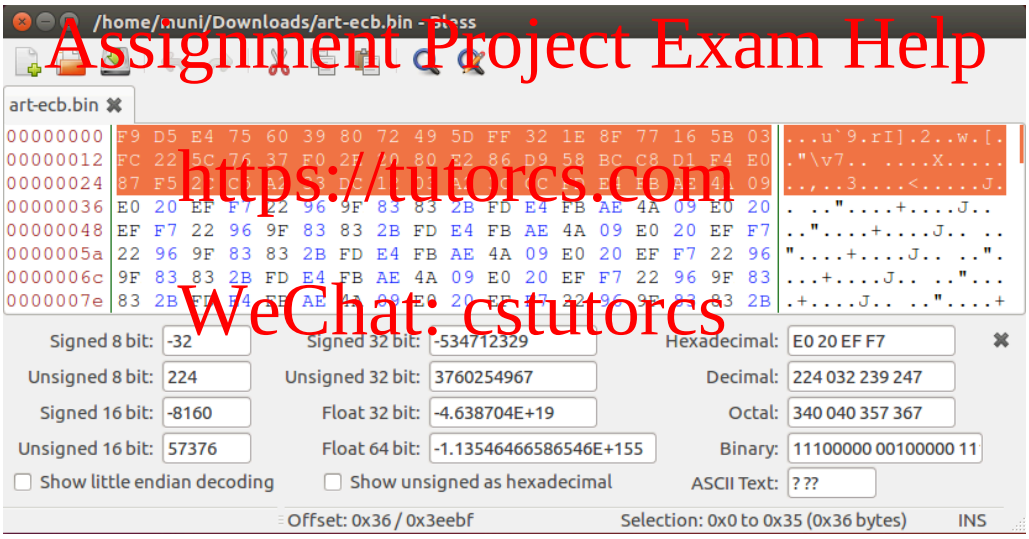

Figure 3: Selecting the first 54 byte of the art-ecb.bin file opened in bless

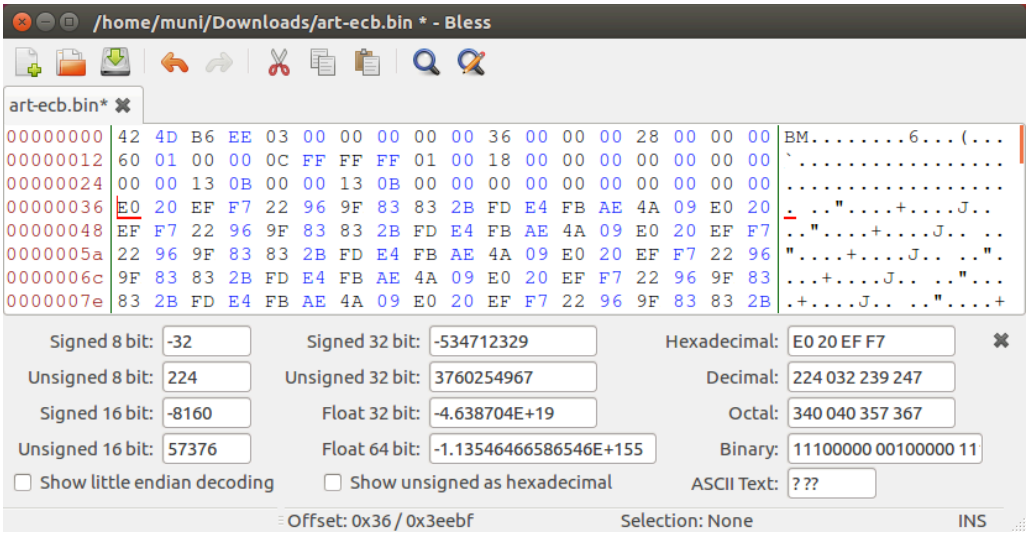

Figure 4: Pasting the copied 54 byte from art.bmp in art-ecb.bin

For the CBC ciphertext, similarly copy the first 54 byte of the `art.bmp` and paste over the first 54 byte of the file `art-cbc.bin` and save the file. Change the extension of the binary file to bitmap and then open the bitmap file.

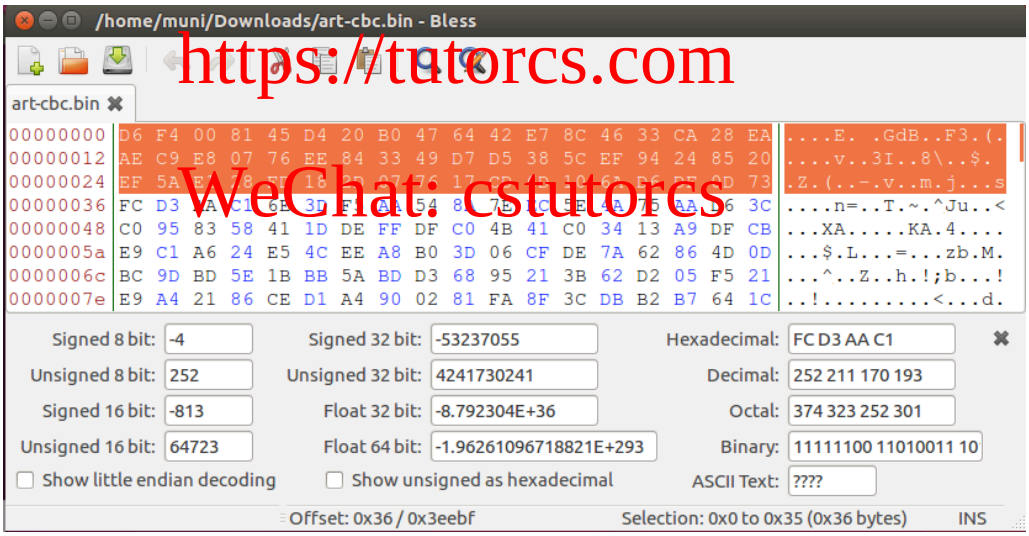

Figure 5: Selecting the first 54 byte of art-cbc.bin

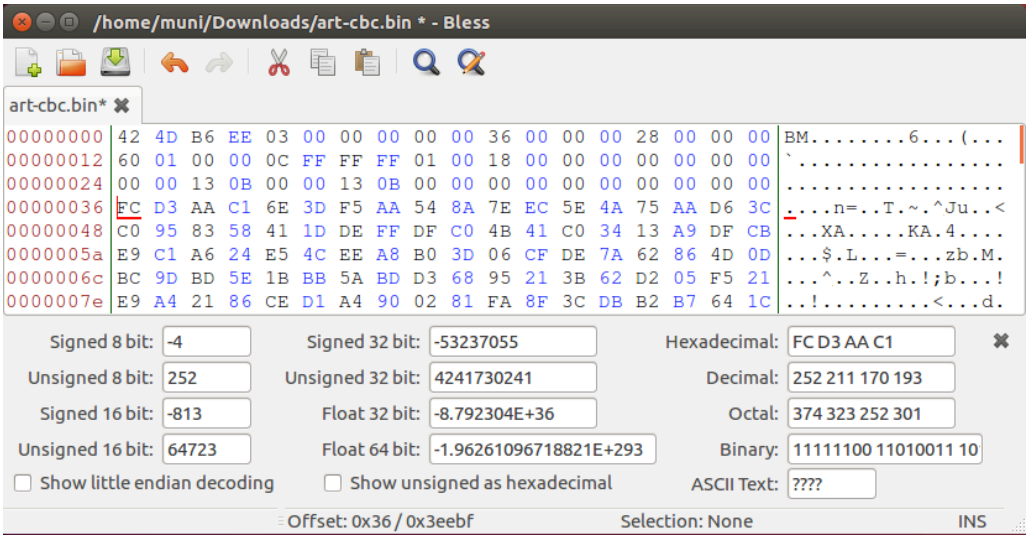

Figure 6: Pasting the copied first 54 bytes of art.bmp over the first 54 bytes of art-cbc.bin using bless

3. Display the encrypted picture using any picture viewing software. Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations.

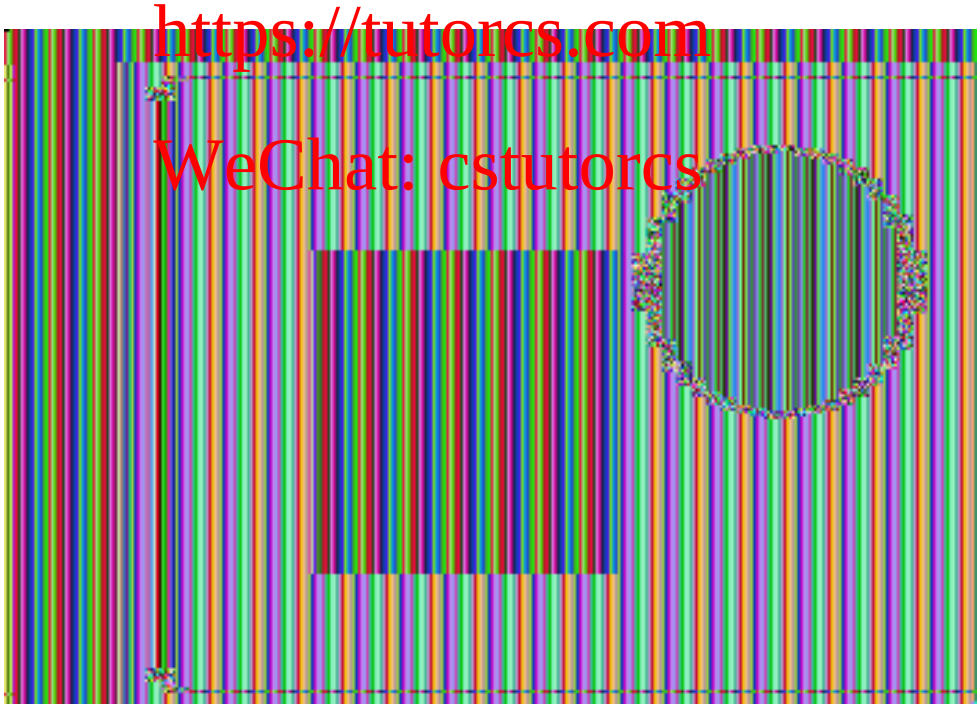

Figure 7: The encrypted file using ECB mode of operation treated as a bitmap file

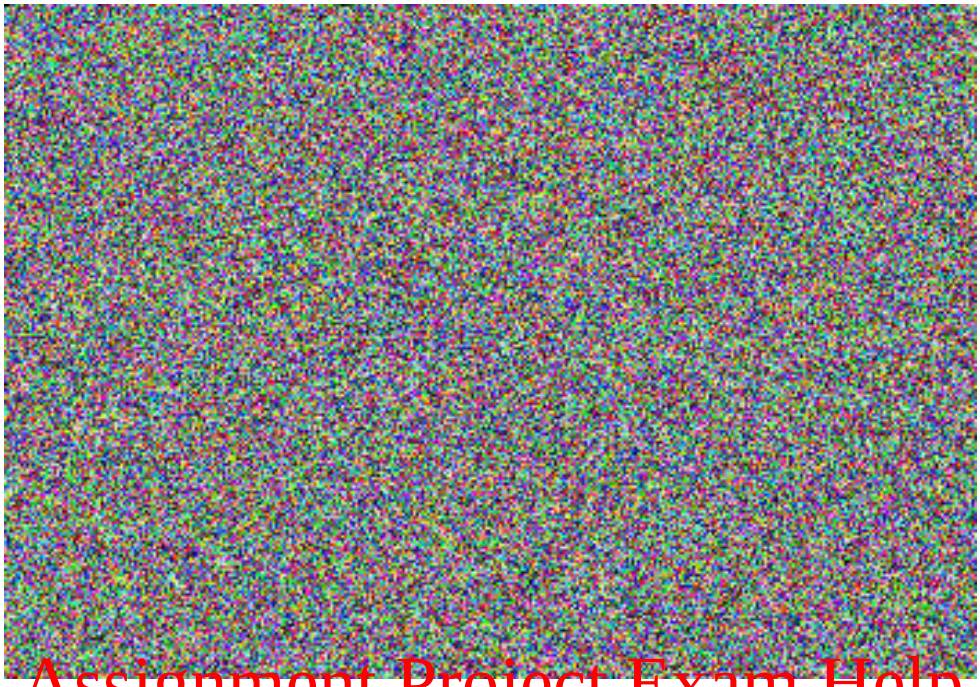

Figure 8: The encrypted file using CBC mode of operation treated as a bitmap file

4 Optional Exercises for Further Exploration

4.1 Avalanche effect

1. Encrypt the plaintext file using `-aes128` algorithm using a key provided in command line with `-K` and IV using `-iv`.

   Use the `-nosalt` for the lab exercise but not in any real use of the tool as the salt provides protection against pre-computation tables for off-line attacks. The command:

   ```
   openssl enc -aes128 -in plain.txt -out c1.bin -K 00112233445566778899aabbccddeeff
      -iv ffeeddccbbaa99887766554433221100 -nosalt
   ```

   See the hex of the ciphertext using `xxd -ps c1.bin`

2. Change a single bit of the key and encrypt the file again (choose a different name for the output file), and compare the content of the two files. You can do this by changing the provided key with the option `-K`.

   ```
   openssl enc -aes128 -in plain.txt -out c2.bin -K 10112233445566778899aabbccddeeff
      -iv ffeeddccbbaa99887766554433221100 -nosalt
   ```

   To see a side by side comparison of differences between the hex use:

   ```
   diff -y <(xxd -ps c1.bin) <(xxd -ps c2.bin)
   ```

The `-aes128` is an alias for CBC mode of operation and the result would be equivalent to `-aes-128-cbc` option. As the key is used in encryption of all blocks regardless of the mode of operation a change in the key will affect all the output blocks. Hence the avalanche effect will be observed as the first block changes (C_1) and then this will change the input to the next block ($P_2 \bigoplus C_1$) in addition to the change in the key.

To see the difference between the CBC and ECB you can use the same key with `-aes-128-ecb` for ECB mode of operation.

3. Change a single bit of the plaintext and encrypt the file with the key and IV used in the first step and compare the two encrypted files. You can do this using the `bless` hex editor, or a text editor and change for instance the first letter from `S` to `R` (hex 52).

   Using bless:

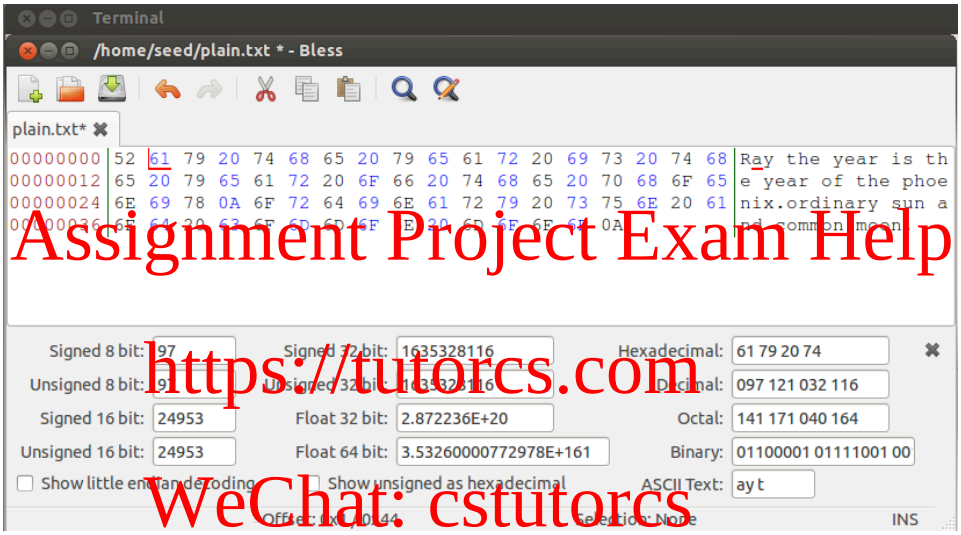

Figure 9: Using bless hex editor to change the first byte from 53 hex to 52 hex (0011 to 0010) of plain.txt

```
openssl enc -aes128 -in plain.txt -out c3.bin -K 00112233445566778899aabbccddeeff
  -iv ffeeddccbbaa99887766554433221100 -nosalt
```

and to compare:

```
diff -y <(xxd -ps c1.bin) <(xxd -ps c3.bin)
```

This will change the output of the first ciphertext block C_1 which is used as the input to the next stage: $P_2 \bigoplus C_1$. The change will propagate to all of ciphertext blocks.

If ECB mode is used the change will stop at C_1.

The avalanche effect however can be seen for the single block in ECB.

4. Change a single bit in the first encrypted file and decrypt it and compare the recovered file with the plaintext file.

   Open the first encrypted file using bless `bless c1.bin` and change a single bit (e.g. the first pair of hex digits).

   Then decrypting the file:

```
openssl enc -d -aes128 -in c1.bin -out p1.bin -K 00112233445566778899aabbccddeeff
  -iv ffeeddccbbaa99887766554433221100 -nosalt
```

and to compare:

```
diff -y <(xxd -ps p1.bin) <(xxd -ps plain.txt)
```

Since the change affects C_1 it will corrupt P_1 in the recovered plaintext after decryption. C_1 is also used in recovering P_2 as $P_2 = D(K, C_2) \bigoplus C_1$ and a corrupted C_1 will affect recovered P_2.

The avalanche effect can be seen as the changes in a single block of recovered plaintext P_1.

4.2 Encryption Modes – Error Propagation

To understand the properties of various modes of operation, we would like to do the following exercises:

1. Encrypt `plain.txt` file using the AES-128 cipher in ECB, CBC, CFB, OFB, CTR modes.

   The CBC, CFB, CTR, and OFB modes require an IV whereas ECB does not. For ease of comparison we use the `-nosalt` option however in practice salt must always be used.

   To encrypt we can use the following commands:


```
openssl enc -aes-128-ecb -K  00112233445566778899aabbccddeeff -in plain.txt -out ecb.bin -nosalt

openssl enc -aes-128-ctr -K  00112233445566778899aabbccddeeff -iv 01020304050607080 90a0b0c0d0e0f
-in plain.txt -out ctr.bin -nosalt
```



```
openssl enc -aes-128-cbc -K  00112233445566778899aabbccddeeff -iv 01020304050607080 90a0b0c0d0e0f
-in plain.txt -out cbc.bin -nosalt
```



```
openssl enc -aes-128-cfb -K  00112233445566778899aabbccddeeff -iv 01020304050607080 90a0b0c0d0e0f
-in plain.txt -out cfb.bin -nosalt

openssl enc -aes-128-ofb -K  00112233445566778899aabbccddeeff -iv 01020304050607080 90a0b0c0d0e0f
-in plain.txt -out ofb.bin -nosalt
```

   The CFB mode using `-aes-128-cfb` will use all 128-bit of the output to XOR with the plaintext blocks. To use 1-bit or 8-bit block sizes (of the output of the block cipher to XOR with plaintext block of 1-bit or 8-bit) the options `-aes-128-cfb1` and `-aes-128-cfb8` must be used.

2. Unfortunately, the 4^{th} byte in the received encrypted file was corrupted. You can simulate this corruption (which would in practice happen due to communication reception errors) using a hex editor. Decrypt the corrupted file using the correct key and IV and determine how many plaintext blocks are recovered correctly and how many are corrupted in each of the above modes of operation.

   Change the 4^{th} byte of all ciphertext files and decrypt the corrupted files and observe the propagation of error in recovered plaintext. To magnify the effect of the corruption of a single byte we change the 4^{th} byte to `FF` hex (unless the byte is also `FF` which we need to use another value). When the changed bits would only change the corresponding bits in XOR we would like to see a few bits to change (`FF` will flip all the bits when XORed with).

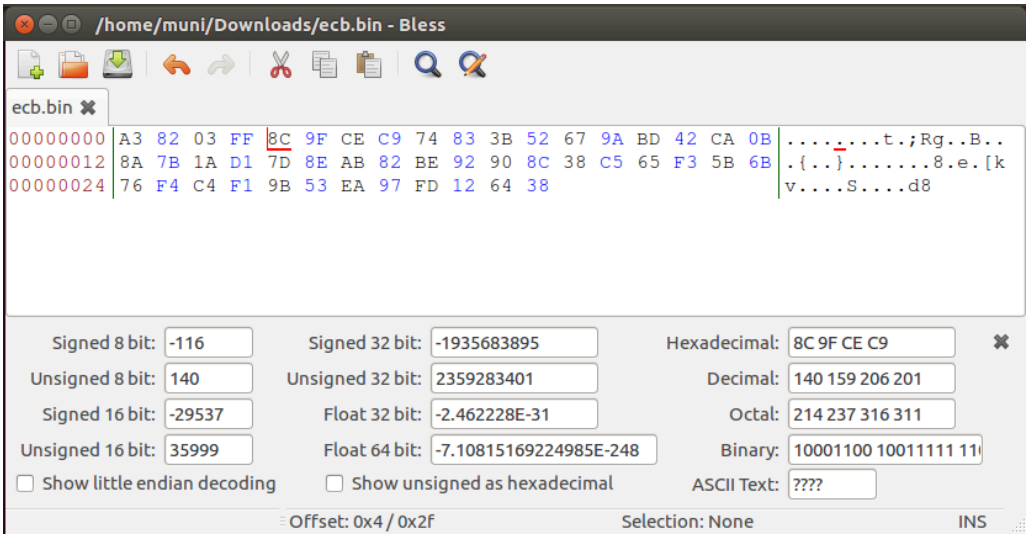

Figure 10: Changing the 4th byte of the ecb.bin

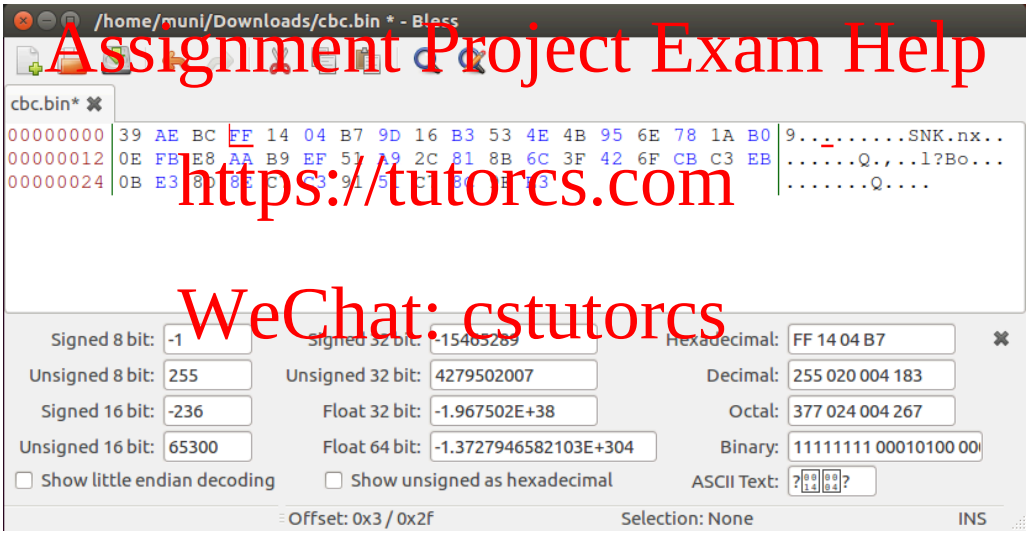

Figure 11: Changing the 4th byte of the cbc.bin

```
openssl enc -aes-128-ecb -d -K  00112233445566778899aabbccddeeff -in ecb.bin -out ecb.txt -nosalt

openssl enc -aes-128-ctr -d -K  00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
-in ctr.bin -out ctr.txt -nosalt

openssl enc -aes-128-cbc -d -K  00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
-in cbc.bin -out cbc.txt -nosalt

openssl enc -aes-128-cfb -d -K  00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
-in cfb.bin -out cfb.txt -nosalt

openssl enc -aes-128-ofb -d -K  00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
-in ofb.bin -out ofb.txt -nosalt
```