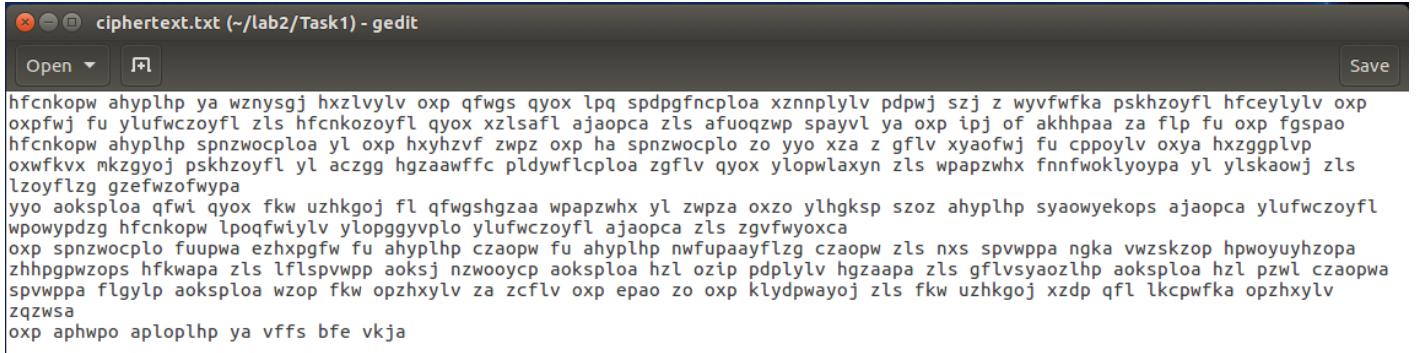


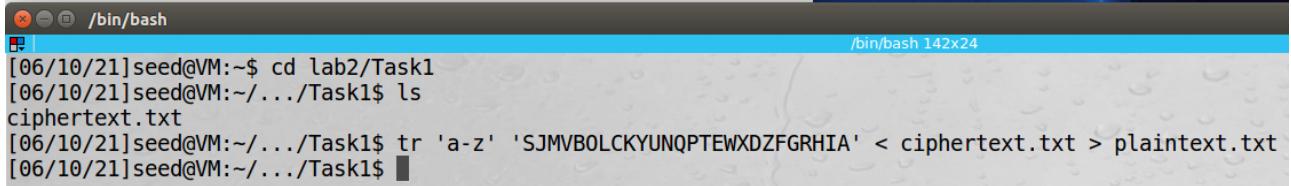
Task 1: Frequency Analysis

The text file content of ciphertext.txt



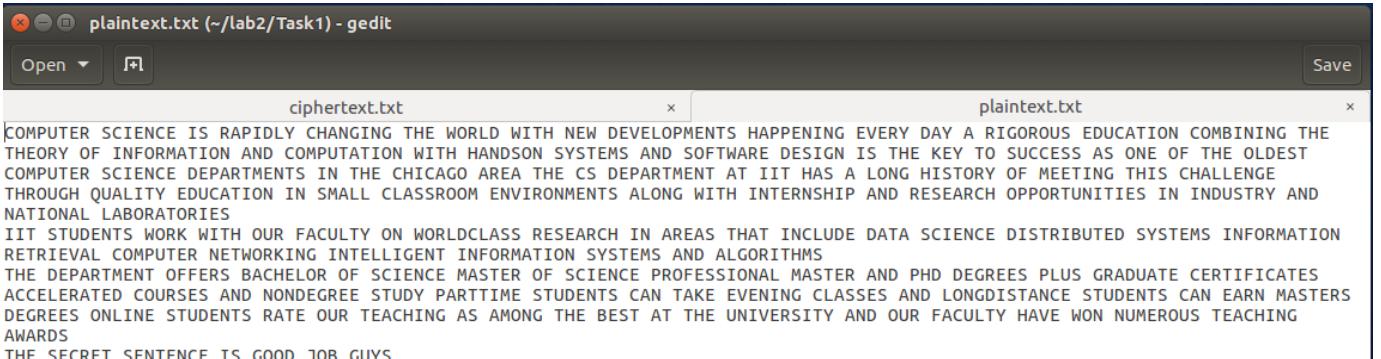
```
hfcnkopw ahyplhp ya wznysgj hxzllylv oxp qfwgs qyox lpq spdpgfnycoploa xznnplylv pdpjw szj z wyvfwfka pskhzoyle fl hfceyllylv oxp  
oxpfwj fu ylufwczoyfl zls hfcnkoyfl qyox xlslsafl ajaopca zls afuoqzwp spayvl ya oxp ipj of akhhpa za flp fu oxp fgspao  
hfcnkopw ahyplhp spnzwcoploa yl oxp hxyhzvf zwpz oxp ha spnzwcoplo zo yyo xza z gflv xyaofwj fu cppoilv oxya hxzggplvp  
oxwfkvx mkzgyoj pskhzoyle fl aczgg hgzaawffc pldywflcploa zgflv qyox ylopwlaxyn zls wpapzwhx fnnfwoklyoypa yl ylskaowj zls  
lzoyflzg gzfewzofywa  
yyo aoksploa qywi qyox fkw uzhkgoj fl qfwgshgzaa wpapzwhx yl zwpza oxzo ylhgksp szoz ahyplhp syaowyekops ajaopca ylufwczoyfl  
wpwopwdzg hfcnkopw lpoqfwiylyl vlopggyvpl o ylufwczoyfl ajaopca zls zgvfywoxca  
oxp spnzwcoplo fuupwa ezhxpgfw fu ahyplhp czaopw fu ahyplhp nwfupaayflzg czaopw zls nxs spvwppa ngka vwszkop hwoyuyhzopa  
zhhpgpwzops hfkwapla zls lflspwp aoksj nwooycp aoksploa hzl oizp pdplyv hgzaapa zls gflvsyaozlp aoksploa hzl pzw1 czaopwa  
spvwppa flgylp aoksploa wzop fkw opzhxyl za zcflv oxp epao zo oxp klydpwayoj zls fkw uzhkgoj xzdp qfl lkcpwfka opzhxyl  
zqzwsa  
oxp aphwpo aploplhp ya vffs bfe vkja
```

In order to decrypt the ciphertext, the following command should be performed:



```
[06/10/21]seed@VM:~$ cd lab2/Task1  
[06/10/21]seed@VM:~/.../Task1$ ls  
ciphertext.txt  
[06/10/21]seed@VM:~/.../Task1$ tr 'a-z' 'SJMVBOLCKYUNQPTEWXDZFGRHIA' < ciphertext.txt > plaintext.txt  
[06/10/21]seed@VM:~/.../Task1$
```

Plaintext



```
plaintext.txt (~/lab2/Task1) - gedit  
Open Save  
ciphertext.txt x plaintext.txt x  
COMPUTER SCIENCE IS RAPIDLY CHANGING THE WORLD WITH NEW DEVELOPMENTS HAPPENING EVERY DAY A RIGOROUS EDUCATION COMBINING THE  
THEORY OF INFORMATION AND COMPUTATION WITH HANDSON SYSTEMS AND SOFTWARE DESIGN IS THE KEY TO SUCCESS AS ONE OF THE OLDEST  
COMPUTER SCIENCE DEPARTMENTS IN THE CHICAGO AREA THE CS DEPARTMENT AT IIT HAS A LONG HISTORY OF MEETING THIS CHALLENGE  
THROUGH QUALITY EDUCATION IN SMALL CLASSROOM ENVIRONMENTS ALONG WITH INTERNSHIP AND RESEARCH OPPORTUNITIES IN INDUSTRY AND  
NATIONAL LABORATORIES  
IIT STUDENTS WORK WITH OUR FACULTY ON WORLDCLASS RESEARCH IN AREAS THAT INCLUDE DATA SCIENCE DISTRIBUTED SYSTEMS INFORMATION  
RETRIEVAL COMPUTER NETWORKING INTELLIGENT INFORMATION SYSTEMS AND ALGORITHMS  
THE DEPARTMENT OFFERS BACHELOR OF SCIENCE MASTER OF SCIENCE PROFESSIONAL MASTER AND PHD DEGREES PLUS GRADUATE CERTIFICATES  
ACCELERATED COURSES AND NONDEGREE STUDY PARTTIME STUDENTS CAN TAKE EVENING CLASSES AND LONGDISTANCE STUDENTS CAN EARN MASTERS  
DEGREES ONLINE STUDENTS RATE OUR TEACHING AS AMONG THE BEST AT THE UNIVERSITY AND OUR FACULTY HAVE WON NUMEROUS TEACHING  
AWARDS  
THE SECRET SENTENCE IS GOOD JOB GUYS
```

Task 2: Encryption using Different Ciphers and Modes

Let's create 3 different files with the same content of "Hello World"

```
[06/10/21]seed@VM:~/.../Task2$ echo "Hello World" > plain_ecb.txt  
[06/10/21]seed@VM:~/.../Task2$ echo "Hello World" > plain_cbc.txt  
[06/10/21]seed@VM:~/.../Task2$ echo "Hello World" > plain_cfb.txt
```

- Let's start with cipher -aes-128 with 3 modes: ECB, CBC, CFB

```
[06/10/21]seed@VM:~/.../Task2$ openssl enc -aes-128-ecb -e -in plain_ecb.txt -out cipher_ecb.bin -K 00010203040506070809aabccddeeff -iv 0a0b0c0d0e0f010203040506070809  
warning: iv not use by this cipher  
[06/10/21]seed@VM:~/.../Task2$ openssl enc -aes-128-cbc -e -in plain_cbc.txt -out cipher_cbc.bin -K 00010203040506070809aabccddeeff -iv 0a0b0c0d0e0f010203040506070809  
[06/10/21]seed@VM:~/.../Task2$ openssl enc -aes-128-cfb -e -in plain_cfb.txt -out cipher_cfb.bin -K 00010203040506070809aabccddeeff -iv 0a0b0c0d0e0f010203040506070809
```

Using “Bless”, we can take a look inside “cipher_ecb.bin”, “cipher_cbc.bin”, and “cipher_cfb.bin”

- ECB



cipher_ecb.bin	cipher_cbc.bin	cipher_cfb.bin
00000000 91 59 B0 1A 83 11 87 41 E1 DB 0C 25 A5 34 C2 6D	Y.....A...%4.m	

- CBC



cipher_ecb.bin	cipher_cbc.bin	cipher_cfb.bin
00000000 A6 94 F6 5D B8 9B FB B4 00 B3 52 01 AC CA F5 1A	[...].....R.....	

- CFB



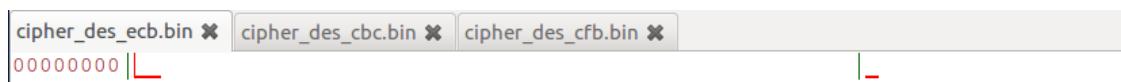
cipher_ecb.bin	cipher_cbc.bin	cipher_cfb.bin
00000000 8D DE E5 99 43 AC 10 70 2C 75 21 26	...C..p,u!&	

- Next, we will take a look at cipher -des with 3 modes: ECB, CBC, CFB

```
[06/10/21]seed@VM:~/.../Task2$ openssl enc -des-ecb -e -in plain_ecb.txt -out cipher_des_ecb.bin -K 00010203040506070809aabccddeff -iv 0a0b0c0d0e0f010203040506070809
warning: iv not use by this cipher
hex string is too long
invalid hex key value
[06/10/21]seed@VM:~/.../Task2$ openssl enc -des-cbc -e -in plain_cbc.txt -out cipher_des_cbc.bin -K 00010203040506070809aabccddeff -iv 0a0b0c0d0e0f010203040506070809
hex string is too long
invalid hex key value
[06/10/21]seed@VM:~/.../Task2$ openssl enc -des-cfb -e -in plain_cfb.txt -out cipher_des_cfb.bin -K 00010203040506070809aabccddeff -iv 0a0b0c0d0e0f010203040506070809
hex string is too long
invalid hex key value
```

Again, we can take a look inside” cipher_des_ecb.bin”, “cipher_des_cbc.bin”, and “cipher_des_cfb.bin” using “Bless”

- ECB



cipher_des_ecb.bin	cipher_des_cbc.bin	cipher_des_cfb.bin
00000000 L	L	

- CBC



cipher_des_ecb.bin	cipher_des_cbc.bin	cipher_des_cfb.bin
00000000 L	L	

- CFB



cipher_des_ecb.bin	cipher_des_cbc.bin	cipher_des_cfb.bin
00000000 L	L	

- Finally, we will take a look at cipher -bf with 3 modes: ECB, CBC, CFB

```
[06/10/21]seed@VM:~/.../Task2$ openssl enc -bf-ecb -e -in plain_ecb.txt -out cipher_bf_ecb.bin -K 00010203040506070809aabbcdddeeff -iv 0a0b0c0d0e0f010203040506070809
warning: iv not use by this cipher
[06/10/21]seed@VM:~/.../Task2$ openssl enc -bf-cbc -e -in plain_cbc.txt -out cipher_bf_cbc.bin -K 00010203040506070809aabbcdddeeff -iv 0a0b0c0d0e0f010203040506070809
[06/10/21]seed@VM:~/.../Task2$ openssl enc -bf-cfb -e -in plain_cfb.txt -out cipher_bf_cfb.bin -K 00010203040506070809aabbcdddeeff -iv 0a0b0c0d0e0f010203040506070809
```

Again, we can take a look inside "cipher_bf_ecb.bin", "cipher_bf_cbc.bin", and "cipher_bf_cfb.bin" using "Bless"

- ECB



- CBC

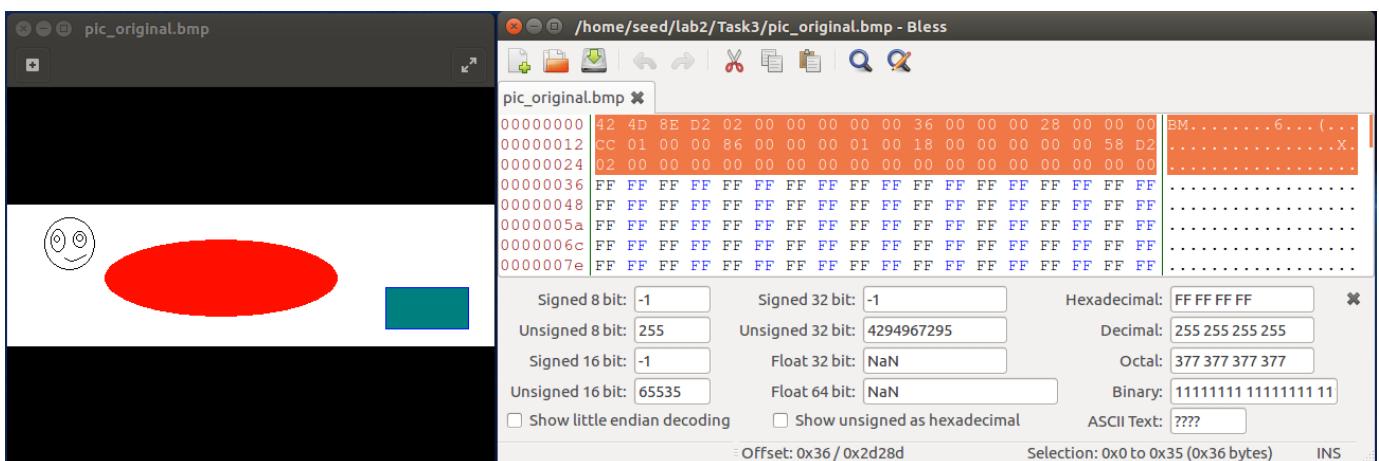


- CFB



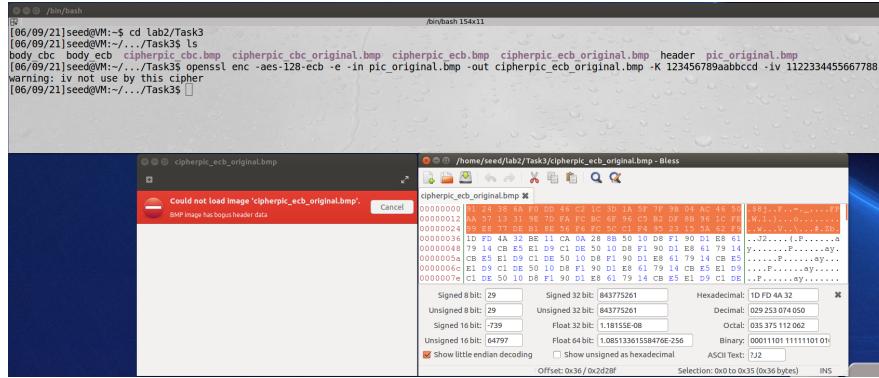
Task 3: Encryption Mode – ECB vs. CBC

The picture below shows the original content of "pic_original.bmp" downloaded from Blackboard. Using Bless, we can see the hex. The highlighted part of the hex shows the first 54 bytes that compose **header** section for both "cbc" and "ecb" modes

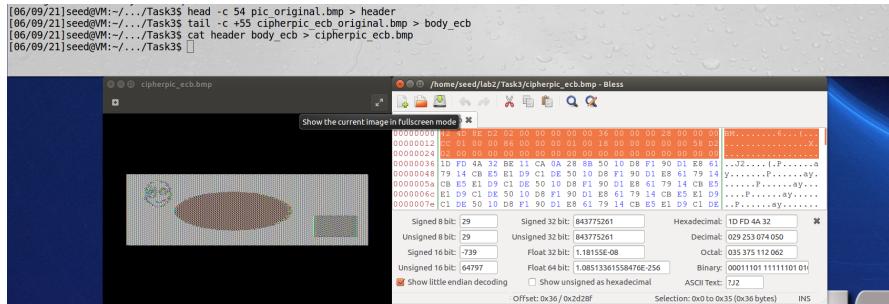


ECB

When we encrypt “pic_original.bmp” using **-aes-128-ecb**, we create a file called “cipherpic_ecb_original.bmp”. When we try to open the file, we get the error message shown below. To solve the problem, the highlighted part of the hex file of “cipherpic_ecb_original.bmp” needs to be replaced with **header** and the tail after the 54th byte should be kept the same.

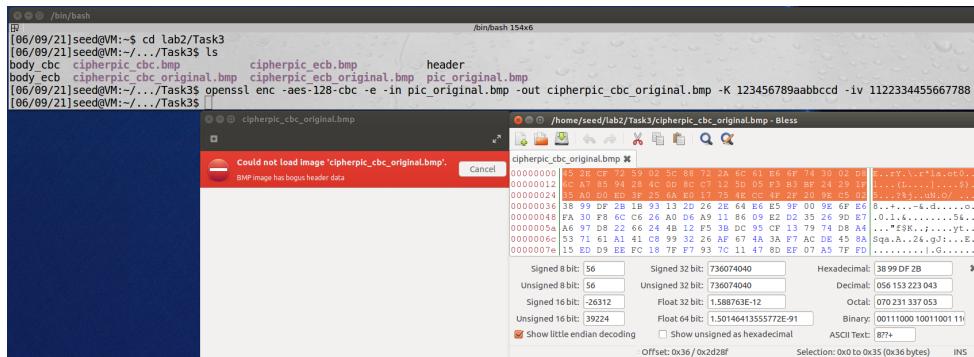


After some manipulations (using the code below), we create a new image that contains a **header** from original “pic_original.bmp” and **body_ecb** (the content of “cipherpic_ecb_original.bmp” that goes from offset 55 to the end of the file), thus resulting with a new file called “**cipherpic_ecb.bmp**”



CBC

When we encrypt “pic_original.bmp” using **-aes-128-cbc**, we create a file called “cipherpic_cbc_original.bmp”. When we try to open the file, we get the error message shown below. To solve the problem, the highlighted part of the hex file of “cipherpic_cbc_original.bmp” needs to be replaced with the already created **header** and the tail after the 54th byte should be kept the same.



After some manipulations (using the code below), we modify **header** from original “pic_original.bmp” and create **body_cbc** (the content of “cipherpic_cbc_original.bmp” that goes from offset 55 to the end of the file), thus resulting with a new file called “**cipherpic_cbc.bmp**”

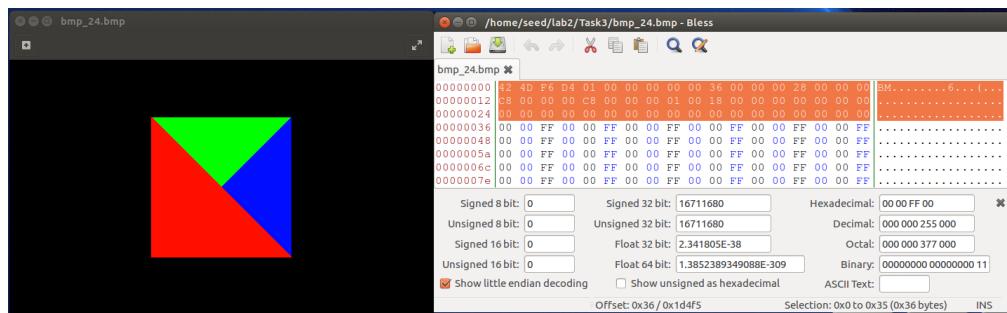
```

# ./Task35 ./pic_original.bmp Cipherpic_cbc_original.bmp -out cipherpic_cbc_original.bmp -K 123456789aabcccd -iv 1122334455667788
[06/09/21]seed@VM:~/.../Task35 [openssl] -aes-128-cbc e -in pic_original.bmp -out cipherpic_cbc_original.bmp -K 123456789aabcccd -iv 1122334455667788
[06/09/21]seed@VM:~/.../Task35 head -c 54 pic_original.bmp > header
[06/09/21]seed@VM:~/.../Task35 tail -c +55 cipherpic_cbc_original.bmp > body_cbc
[06/09/21]seed@VM:~/.../Task35 cat header body_cbc > cipherpic_cbc.bmp
[06/09/21]seed@VM:~/.../Task35

```

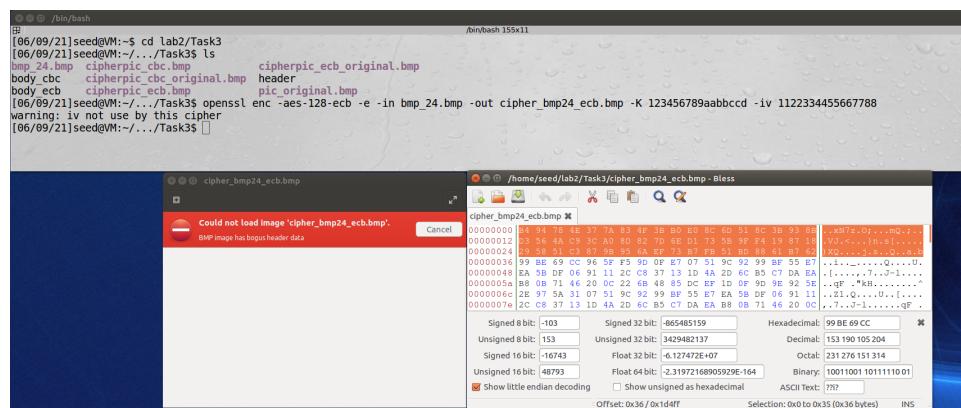
New Image

The picture below shows the original content of “**bmp_24.bmp**” found on the Internet. Using Bless, we can see the hex. The highlighted part of the hex shows the first 54 bytes that compose **header** section for both “cbc” and “ecb” modes

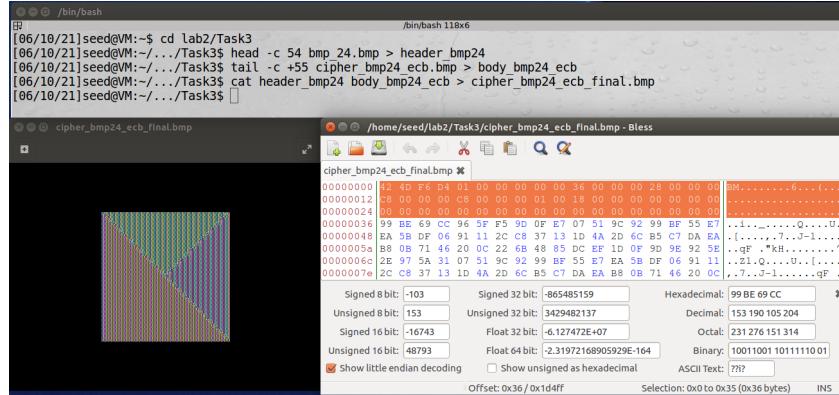


ECB

When we encrypt “**bmp_24.bmp**” using **-aes-128-ecb**, we create a file called “**cipher_bmp24_ecb.bmp**” When we try to open the file, we get the error message shown below. To solve the problem, the highlighted part of the hex file of “**cipher_bmp24_ecb.bmp**” needs to be replaced with **header_bmp24** and the tail after the 54th byte should be kept the same.

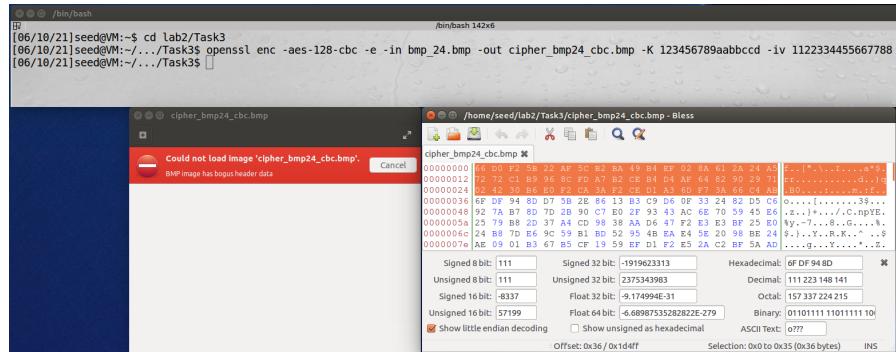


After some manipulations (using the code below), we create a new image that contains a **header_bmp24** from original “**bmp_24.bmp**” and **body_bmp24_ecb** (the content of “**cipher_bmp24_ecb.bmp**” that goes from offset 55 to the end of the file), thus resulting with a new file called “**cipher_bmp24_ecb_final.bmp**”

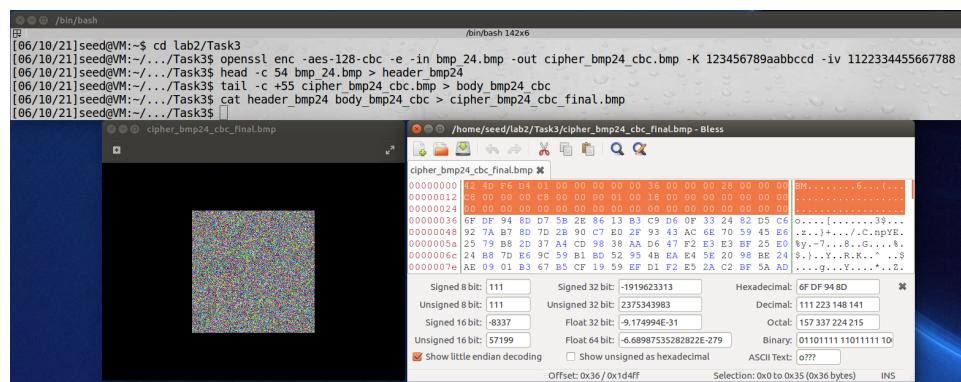


CBC

When we encrypt “**bmp_24.bmp**” using **-aes-128-cbc**, we create a file called “**cipher_bmp24_cbc.bmp**” When we try to open the file, we get the error message shown below. To solve the problem, the highlighted part of the hex file of “**cipher_bmp24_cbc.bmp**” needs to be replaced with already created **header_bmp24** and the tail after the 54th byte should be kept the same.



After some manipulations (using the code below), we modify **header_bmp24** from original “**bmp_24.bmp**” and create **body_bmp24_cbc** (the content of “**cipher_bmp24_cbc.bmp**” that goes from offset 55 to the end of the file), thus resulting with a new file called “**cipher_bmp24_cbc_final.bmp**”

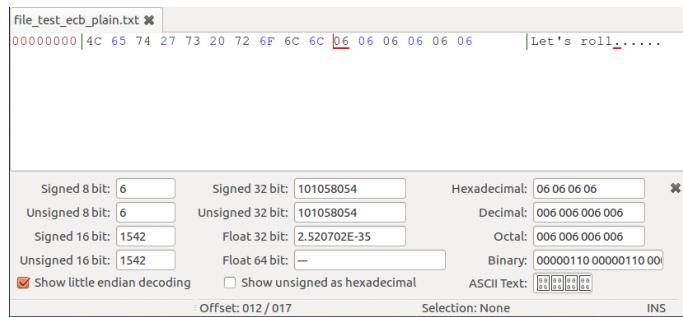


Task 4: Padding

1) Use ECB, CBC, 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.

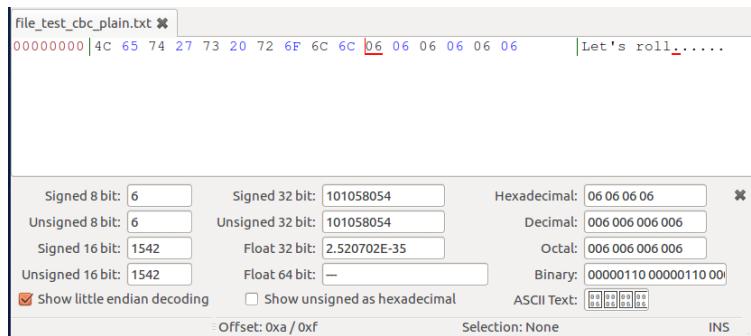
- ECB uses padding (5-bytes of “06” are added)

```
[06/10/21]seed@VM:~/.../Task4$ echo -n "Let's roll" > file_test_ciphers.txt
[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-ecb -e -in file_test_ciphers.txt -out file_test_ecb.txt
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
[06/10/21]seed@VM:~/.../Task4$ cat file_test_ecb.txt
[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-ecb -d -nopad -in file_test_ecb.txt -out file_test_ecb_plain.txt
enter aes-128-ecb decryption password:
[06/10/21]seed@VM:~/.../Task4$ cat file_test_ecb_plain.txt
Let's roll\x06\x06\x06\x06\x06\x06 [Let's roll.....]
00000000 4c 65 74 27 73 20 72 6f 6c 06 06 06 06 06 06 |Let's roll.....|
00000010
[06/10/21]seed@VM:~/.../Task4$ xxd file_test_ecb_plain.txt
00000000: 4c65 7427 7320 726f 6c6c 0606 0606 0606 Let's roll.....
```



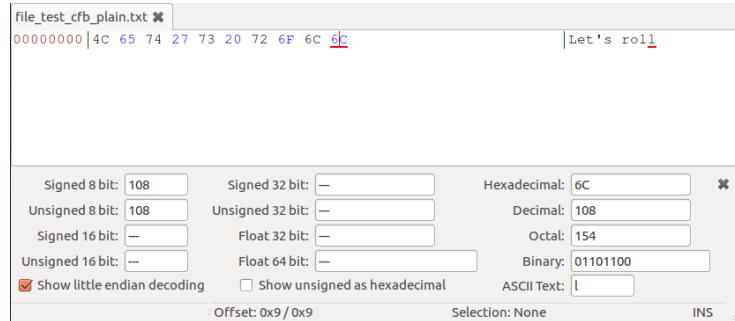
- CBC uses padding (5-bytes of “06” are added)

```
[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -e -in file_test_ciphers.txt -out file_test_cbc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
[06/10/21]seed@VM:~/.../Task4$ cat file_test_cbc.txt
\x06\x06\x06\x06\x06\x06 [06/10/21]se
[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -d -nopad -in file_test_cbc.txt -out file_test_cbc_plain.txt
enter aes-128-cbc decryption password:
[06/10/21]seed@VM:~/.../Task4$ cat file_test_cbc_plain.txt
Let's roll\x06\x06\x06\x06\x06\x06 [Let's roll.....]
[06/10/21]seed@VM:~/.../Task4$ hexdump -C filr_test_cbc_plain.txt
hexdump: filr_test_cbc_plain.txt: No such file or directory
[06/10/21]seed@VM:~/.../Task4$ hexdump -C file_test_cbc_plain.txt
00000000 4c 65 74 27 73 20 72 6f 6c 06 06 06 06 06 |Let's roll.....|
00000010
[06/10/21]seed@VM:~/.../Task4$ xxd file_test_cbc_plain.txt
00000000: 4c65 7427 7320 726f 6c6c 0606 0606 0606 Let's roll.....
```



- CFB

```
[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-cfb -e -in file_test_ciphers.txt -out file_test_cfb.txt
enter aes-128-cfb encryption password:
Verifying - enter aes-128-cfb encryption password:
[06/10/21]seed@VM:~/.../Task4$ cat file_test_cfb.txt
[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-cfb -d -nopad -in file_test_cfb.txt -out file_test_cfb_plain.txt
enter aes-128-cfb decryption password:
[06/10/21]seed@VM:~/.../Task4$ hexdump -C file_test_cfb_plain.txt
00000000 4c 65 74 27 73 20 72 6f 6c 06 06 06 06 06 |Let's roll|
0000000a
[06/10/21]seed@VM:~/.../Task4$ xxd file_test_cfb_plain.txt
00000000: 4c65 7427 7320 726f 6c6c Let's roll
```



As it seen from code and decryption results from above, CFB mode does not perform padding

- OFB

```
[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-ofb -e -in file_test_ciphers.txt -out file_test_ofb.txt
enter aes-128-ofb encryption password:
Verifying - enter aes-128-ofb encryption password:
[06/10/21]seed@VM:~/.../Task4$ cat file_test_ofb.txt
Salted 0000000006/10/2
[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-ofb -d -nopad -in file_test_ofb.txt -out file_test_ofb_plain.txt
enter aes-128-ofb decryption password:
[06/10/21]seed@VM:~/.../Task4$ hexdump -C file_test_ofb_plain.txt
00000000  4c 65 74 27 73 20 72 6f  6c 6c          |Let's roll|
0000000a
[06/10/21]seed@VM:~/.../Task4$ xxd file_test_ofb_plain.txt
00000000: 4c65 7427 7320 726f 6c6c      Let's roll
```

As it seen from code and decryption results from above, OFB mode also does not perform padding

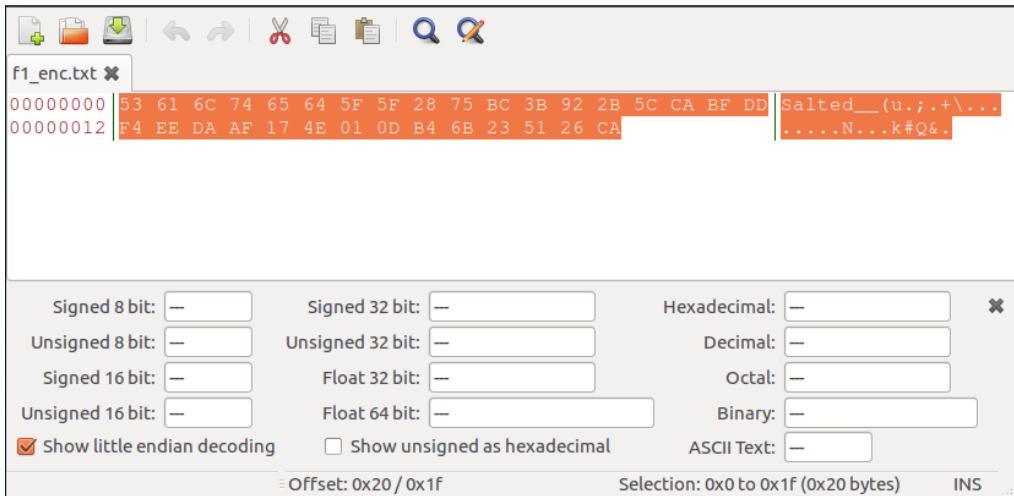
Explanation: CFB and OFB do not need padding because their ciphertexts will have the same size as the plaintexts

2) Let's create a file f1.txt that contains 5 bytes and has a content of "12345"

```
/bin/bash
[06/10/21]seed@VM:~$ cd lab2/Task4
[06/10/21]seed@VM:~/.../Task4$ echo -n "12345" > f1.txt
[06/10/21]seed@VM:~/.../Task4$ 
```

Next, we encrypt the message in a content file named "f1_enc.txt". The encrypted size of the data is 32 bytes.

```
[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -e -in f1.txt -out f1_enc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
```



After that, we create an actual file “f1_enc.txt” and decrypt it to “f1_plain.txt”, using `-nopad` option in order to keep padding in the decryption message

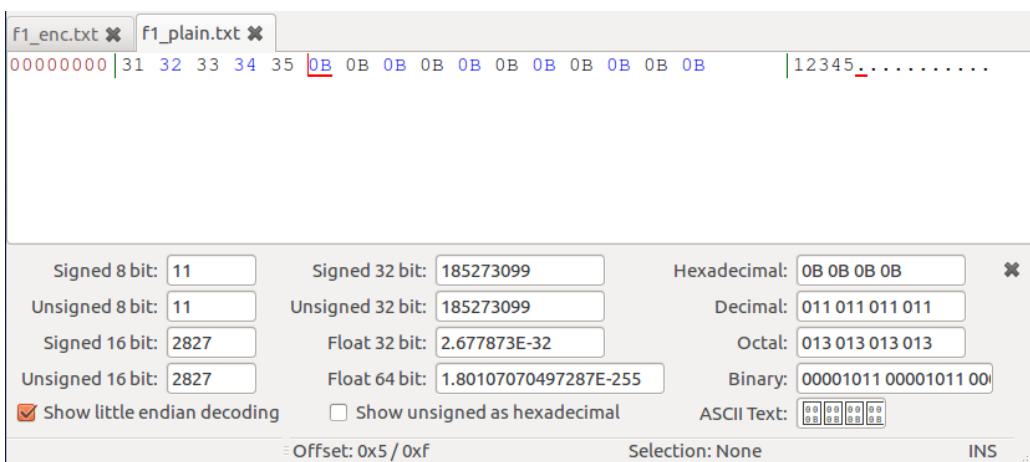
```
[06/10/21]seed@VM:~/.../Task4$ cat f1_enc.txt
0k#Q&@[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -d -nopad -in f1_enc.txt -out f1_plain.txt
enter aes-128-cbc decryption password:
```

Finally, we create “f1_plain.txt” and perform hexdump to see what paddings are added to the file

```
[06/10/21]seed@VM:~/.../Task4$ cat f1_plain.txt
12345

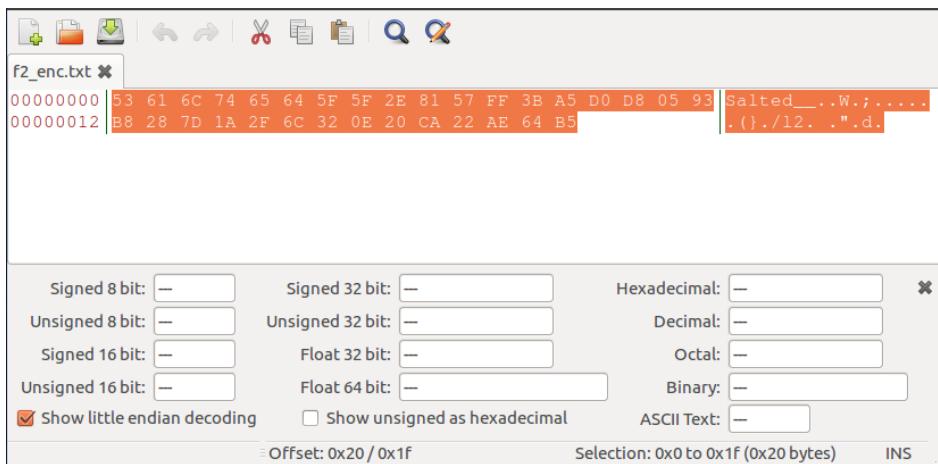
[06/10/21]seed@VM:~/.../Task4$ hexdump -C f1_plain.txt
00000000  31 32 33 34 35 0b |12345.....|
00000010
[06/10/21]seed@VM:~/.../Task4$ xxd f1_plain.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b 12345.....
```

Looking at data of file “f1_plain” using Bless, it shows that there are 11-byte paddings of “0B” in the file



Let's create a file f2.txt that contains 10 bytes and has a content of "1234567890". Next, we encrypt the message in a content file named "f2_enc.txt". The encrypted size of the data is also 32 bytes.

```
[06/10/21]seed@VM:~/.../Task4$ echo -n "1234567890" > f2.txt  
[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -e -in f2.txt -out f2_enc.txt  
enter aes-128-cbc encryption password:  
Verifying - enter aes-128-cbc encryption password:
```



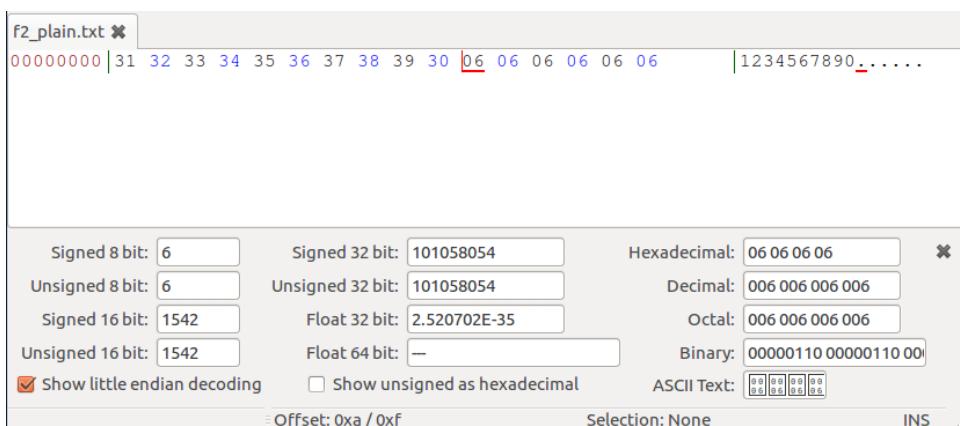
After that, we create an actual file "f2_enc.txt" and decrypt it to "f2_plain.txt", using `-nopad` option in order to keep padding in the decryption message

```
[06/10/21]seed@VM:~/.../Task4$ cat f2_enc.txt  
[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -d -nopad -in f2_enc.txt -out f2_plain.txt  
enter aes-128-cbc decryption password:
```

Finally, we create "f2_plain.txt" and perform hexdump to see what paddings are added to the file

```
[06/10/21]seed@VM:~/.../Task4$ cat f2_plain.txt  
1234567890[06/10/21]seed@VM:~/.../Task4$ hexdump -C f2_plain.txt  
00000000  31 32 33 34 35 36 37 38  39 30 06 06 06 06 06 06 |1234567890.....|  
00000010  
[06/10/21]seed@VM:~/.../Task4$ xxd f2_plain.txt  
00000000: 3132 3334 3536 3738 3930 0606 0606 0606 1234567890.....
```

Looking at data of file "f2_plain" using Bless, it shows that there are 5-byte paddings of "06" in the file



Let's create a file f3.txt that contains 16 bytes and has a content of "1234567890abcdef". Next, we encrypt the message in a content file named "f3_enc.txt". The encrypted size of the data is 48 bytes.

```
[06/10/21]seed@VM:~/.../Task4$ echo -n "1234567890abcdef" > f3.txt
[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -e -in f3.txt -out f3_enc.txt
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
```

The screenshot shows the Bless hex editor interface. The file f3_enc.txt is open, displaying 48 bytes of data. The first 16 bytes are the original message "1234567890abcdef". The next 16 bytes are the salted header, which includes the string "Salted_1....p.,," followed by some random characters. The last 16 bytes are the encrypted message. Below the hex dump, there are conversion tools for various data types (Signed 8 bit, Unsigned 8 bit, etc.) and a checkbox for "Show little endian decoding".

After that, we create an actual file "f3_enc.txt" and decrypt it to "f3_plain.txt", using -nopad option in order to keep padding in the decryption message

```
[06/10/21]seed@VM:~/.../Task4$ cat f3_enc.txt
Salted_01:p0,000"prq
00e#02#03#04#05#06#[06/10/21]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -d -nopad -in f3_enc.txt -out f3_plain.txt
enter aes-128-cbc decryption password:
```

Finally, we create "f3_plain.txt" and perform hexdump to see what paddings are added to the file

```
[06/10/21]seed@VM:~/.../Task4$ cat f3_plain.txt
1234567890abcdef
00000000 31 32 33 34 35 36 37 38 39 30 61 62 63 64 65 66 |1234567890abcdef|
00000010 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 |.....|
00000020
[06/10/21]seed@VM:~/.../Task4$ xxd f3_plain.txt
00000000: 3132 3334 3536 3738 3930 6162 6364 6566 1234567890abcdef
00000010: 1010 1010 1010 1010 1010 1010 1010 1010 .....
```

Looking at data of file "f3_plain" using Bless, it shows that there are 16-byte paddings of "10" in the file

The screenshot shows the Bless hex editor interface. The file f3_plain.txt is open, displaying its contents. The first 16 bytes are the original message "1234567890abcdef". The next 16 bytes are padding, consisting of the byte value 10 repeated 16 times. The last 16 bytes are another padding block. Below the hex dump, there are conversion tools for various data types (Signed 8 bit, Unsigned 8 bit, etc.) and a checkbox for "Show little endian decoding".

Task 5: Error Propagation

- How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively?

Answer:

- In ECB mode, when any problem in a ciphertext happens, only one block is being affected. Each block decrypts independently from one another. However, the corrupted bit of the 55th byte in ciphertext blocks 8 bytes, and this 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 on two blocks.
- In CFB mode, there is a problem with every (n / r) number of blocks. Therefore, the error propagation criterion performs even worse.
- In OFB mode, the feedback is only in the key-generation system. If the single digit of the 55th byte is corrupted, then only that byte or character is oxidized in the plaintext.

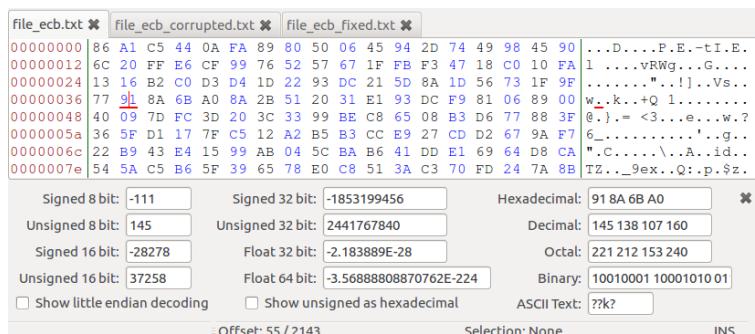
The text file content (from the first paragraph of “The Count Of Monte Cristo”)

```
file_ecb_fixed.txt
On the 24th of February, 1815, the look-out at Notre-Dame de la Garde signalled the three-master, the Pharaon from Smyrna, Trieste, and Naples. As usual, a pilot put off immediately, and rounding the Château d'If, got on board the vessel between Cape Morgiou and Rion island. Immediately, and according to custom, the ramparts of Fort Saint-Jean were covered with spectators; it is always an event at Marseilles for a ship to come into port, especially when this ship, like the Pharaon, has been built, rigged, and laden at the old Phocée docks, and belongs to an owner of the city. The ship drew on and had safely passed the strait, which some volcanic shock has made between the Calasareigne and Jaros islands; had doubled Poméga, and approached the harbor under topsails, jib, and spanker, but so slowly and sedately that the idlers, with that instinct which is the forerunner of evil, asked one another what misfortune could have happened on board. However, those experienced in navigation saw plainly that if any accident had occurred, it was not to the vessel herself, for she bore down with all the evidence of being skilfully handled, the anchor a-cockbill, the jib-boom guys already eased off, and standing by the side of the pilot, who was steering the Pharaon towards the narrow entrance of the inner port, was a young man, who, with activity and vigilant eye, watched every motion of the ship, and repeated each direction of the pilot. The vague disquietude which prevailed among the spectators had so much affected one of the crowd that he did not await the arrival of the vessel in harbor, but jumping into a small skiff, destined to be pulled alongside the Pharaon, which he reached as she rounded into La Réserve basin. When the young man on board saw this person approach, he left his station by the pilot, and, hat in hand, leaned over the ship's bulwarks. He was a fine, tall, slim young fellow of eighteen or twenty, with black eyes, and hair as dark as a raven's wing; and his whole appearance bespoke that calmness and resolution peculiar to men accustomed from their cradle to contend with danger.]
```

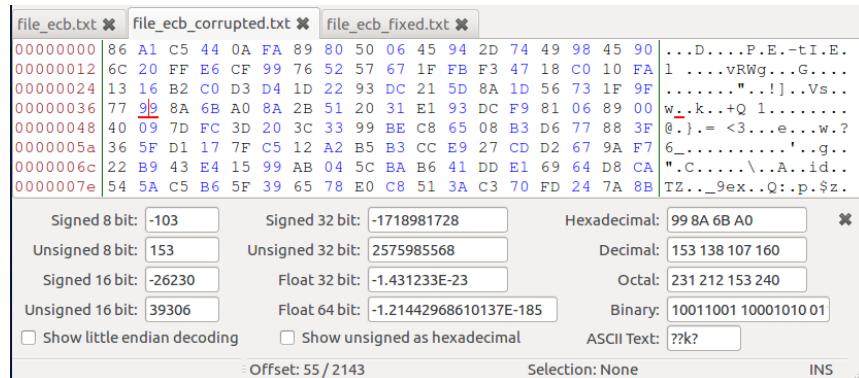
Next, we encrypt the text. Let's use ECB mode first.

```
[06/10/21]seed@VM:~$ cd lab2/Task5
[06/10/21]seed@VM:~/.../Task5$ ls
file.txt
[06/10/21]seed@VM:~/.../Task5$ openssl enc -aes-128-ecb -e -in file.txt -out file_ecb.txt -K 0123456789abcdef
```

The click bar is at offset 55 [00000036 / 91]



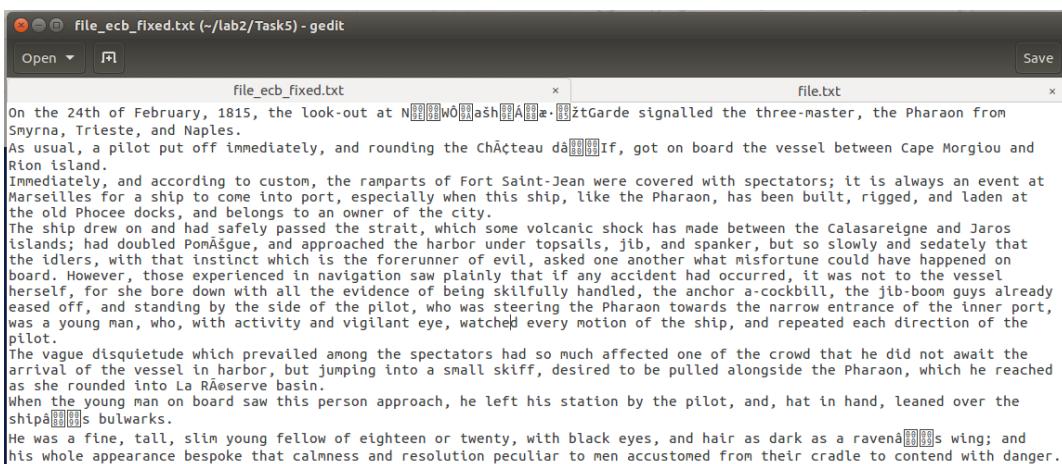
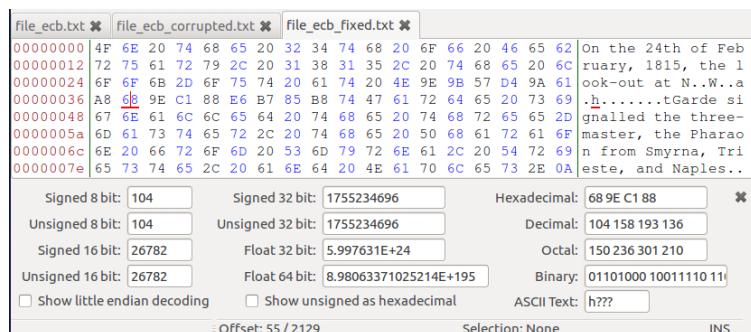
At offset 55 of encrypted file “file_ecb.txt”, let’s place 9 so that new 55th offset becomes “99” and change name of a new file to be “file_ecb_corrupted.txt”



Using the code below, we decrypt “file_ecb_corrupted.txt” into “file_ecb_fixed.txt”

```
[06/10/21]seed@VM:~/.../Task5$ openssl enc -aes-128-ecb -d -in file_ecb_corrupted.txt -out file_ecb_fixed.txt -K 0123456789abcdef
[06/10/21]seed@VM:~/.../Task5$
```

This is the content of “file_ecb_fixed.txt”



Next, let’s use CBC mode.

```
[06/10/21]seed@VM:~/.../Task5$ openssl enc -aes-128-cbc -e -in file.txt -out file_cbc.txt -K 0123456789abcdef -iv 012345678aabccddeff
[06/11/21]seed@VM:~/.../Task5$
```

The click bar is at offset 55 [00000036 / B6]

file_cbc.txt		file_cbc_corrupted.txt	
00000000	95 25 97 99 76 63 AC F0 06 BC 89 AB A3 4E 91 36 CA EC	.%..vc.....N.6..	
00000012	B3 54 B0 FC 43 40 DC 71 82 46 C1 F2 0A 12 6B D9 FD 75	.T..C@.q.F....k..u	
00000024	AB 6A 39 02 A2 6B BB A2 EC 8B F3 17 1F F2 4A 98 F7 53	.j9..k.....J..S	
00000036	59 B6 0C 80 65 19 81 B7 FA F2 4F 9F 35 14 82 49 EC 15	Y...e.....O.5..I..	
00000048	AF 0C 15 8B E7 45 3E 9C 19 94 DD 04 5C A1 E1 90 00 ECE>.....\.....	
0000005a	BF D9 22 0A 76 2F 23 C5 43 88 D8 75 67 C1 7E F2 3F B2	..".v/#.C..ug.~?.	
0000006c	BC 0E 7E DC 8E 16 32 13 EC 06 00 2B 84 57 EB 35 DB E3~.2....+.W.5..	
0000007e	40 29 26 3F 55 15 38 03 12 26 52 79 CA 6A EE B5 E3 7C	(@)&?U.8..&Ry.j...	
Signed 8 bit: -74		Signed 32 bit: -1240694683	Hexadecimal: B6 0C 80 65
Unsigned 8 bit: 182		Unsigned 32 bit: 3054272613	Decimal: 182 012 128 101
Signed 16 bit: -18932		Float 32 bit: -2.093636E-06	Octal: 266 014 200 145
Unsigned 16 bit: 46604		Float 64 bit: -2.43769335678486E-48	Binary: 10110110 00001100 10
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal	ASCII Text: ?B6?e
Offset: 55 / 2143		Selection: None	INS

At offset 55 of encrypted file “file_cbc.txt”, let’s place 9 so that new 55th offset becomes “BB” and change name of a new file to be “file_cbc_corrupted.txt”

file_cbc.txt		file_cbc_corrupted.txt	
00000000	95 25 97 99 76 63 AC F0 06 BC 89 AB A3 4E 91 36 CA EC	.%..vc.....N.6..	
00000012	B3 54 B0 FC 43 40 DC 71 82 46 C1 F2 0A 12 6B D9 FD 75	.T..C@.q.F....k..u	
00000024	AB 6A 39 02 A2 6B BB A2 EC 8B F3 17 1F F2 4A 98 F7 53	.j9..k.....J..S	
00000036	59 BB 0C 80 65 19 81 B7 FA F2 4F 9F 35 14 82 49 EC 15	Y...e.....O.5..I..	
00000048	AF 0C 15 8B E7 45 3E 9C 19 94 DD 04 5C A1 E1 90 00 ECE>.....\.....	
0000005a	BF D9 22 0A 76 2F 23 C5 43 88 D8 75 67 C1 7E F2 3F B2	..".v/#.C..ug.~?.	
0000006c	BC 0E 7E DC 8E 16 32 13 EC 06 00 2B 84 57 EB 35 DB E3~.2....+.W.5..	
0000007e	40 29 26 3F 55 15 38 03 12 26 52 79 CA 6A EE B5 E3 7C	(@)&?U.8..&Ry.j...	
Signed 8 bit: -69		Signed 32 bit: -1156808603	Hexadecimal: BB 0C 80 65
Unsigned 8 bit: 187		Unsigned 32 bit: 3138158693	Decimal: 187 012 128 101
Signed 16 bit: -17652		Float 32 bit: -0.002143883	Octal: 273 014 200 145
Unsigned 16 bit: 47884		Float 64 bit: -2.94699043932027E-24	Binary: 1011011 00001100 10
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal	ASCII Text: ?B6?e
Offset: 55 / 2143		Selection: None	INS

Using the code below, we decrypt “file_ecb_corrupted.txt” into “file_cbc_fixed.txt”

```
[06/11/21]seed@VM:~/.../Task5$ openssl enc -aes-128-cbc -d -in file_cbc_corrupted.txt -out file_cbc_fixed.txt -K 0123456789abcdef -iv 012345678aabccddeff  
[06/11/21]seed@VM:~/.../Task5$
```

This is the content of “file_cbc_fixed.txt”

file_cbc.txt		file_cbc_corrupted.txt		file_cbc_fixed.txt	
00000000	4F 6E 20 74 68 65 20 32 34 74 68 20 6F 66 20 46 65 62	On the 24th of Feb			
00000012	72 75 61 72 79 2C 20 31 38 31 35 2C 20 74 68 65 20 6C	ruary, 1815, the l			
00000024	6F 6F 6B 2D 6F 75 74 20 61 74 20 4E D5 13 52 1C 4E 60	ook-out at N..R.N'			
00000036	87 DC 39 91 57 E3 90 AD 53 13 47 61 72 64 65 20 73 64	..9.W...S.Carde sd			
00000048	67 6E 61 6C 65 64 20 74 68 65 20 74 68 72 65 65 2D	gnailed the three-			
0000005a	6D 61 73 74 65 72 2C 20 74 68 65 20 50 68 61 72 61 6F	master, the Phara			
0000006c	6B 20 66 72 6F 6D 20 53 6D 79 72 6E 61 2C 20 54 72 69	n from Smyrna, Tri			
0000007e	65 73 74 65 2C 20 61 6E 64 20 4E 61 70 6C 65 73 2E 0A	este, and Naples..			
Signed 8 bit: 71		Signed 32 bit: 1197568612	Hexadecimal: 47 61 72 64		
Unsigned 8 bit: 71		Unsigned 32 bit: 1197568612	Decimal: 071 097 114 100		
Signed 16 bit: 18273		Float 32 bit: 57714.39	Octal: 107 141 162 144		
Unsigned 16 bit: 18273		Float 64 bit: 7.24713563227011E+35	Binary: 01000111 01100001 01		
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal	ASCII Text: Gard		
Offset: 64 / 2129		Selection: None	INS		

```

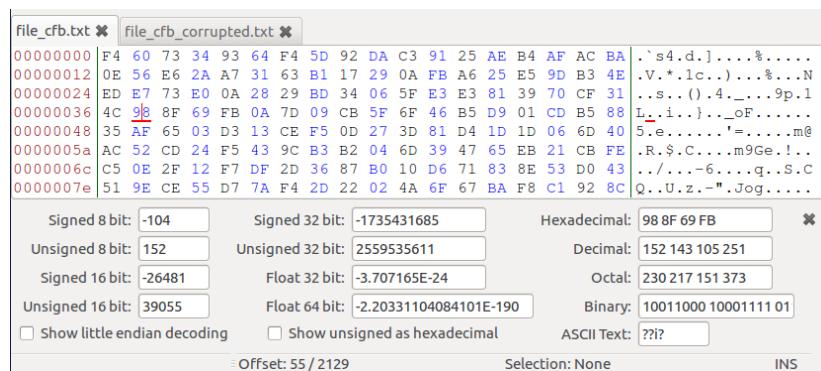
file_cbc_fixed.txt (~/lab2/Task5) - gedit
Save
Open ▾
On the 24th of February, 1815, the look-out at Nô̄r̄n̄`Ü9̄w̄s̄ Garde sdgnalled the three-master, the Pharaon from Smyrna, Trieste, and Naples. As usual, a pilot put off immediately, and rounding the Châ̄teau dâ̄If, got on board the vessel between Cape Morgiou and Rion island. Immediately, and according to custom, the ramparts of Fort Saint-Jean were covered with spectators; it is always an event at Marseilles for a ship to come into port, especially when this ship, like the Pharaon, has been built, rigged, and laden at the old Phocée docks, and belongs to an owner of the city. The ship drew on and had safely passed the strait, which some volcanic shock has made between the Calasareigne and Jaros islands; had doubled Pomâ̄gue, and approached the harbor under topsails, jib, and spanker, but so slowly and sedately that the idlers, with that instinct which is the forerunner of evil, asked one another what misfortune could have happened on board. However, those experienced in navigation saw plainly, that if any accident had occurred, it was not to the vessel herself, for she bore down with all the evidence of being skilfully handled, the anchor a-cockbill, the jib-boom guys already eased off, and standing by the side of the pilot, who was steering the Pharaon towards the narrow entrance of the inner port, was a young man, who, with activity and vigilant eye, watched every motion of the ship, and repeated each direction of the pilot. The vague disquietude which prevailed among the spectators had so much affected one of the crowd that he did not await the arrival of the vessel in harbor, but jumping into a small skiff, destined to be pulled alongside the Pharaon, which he reached as she rounded into La Râ̄serve basin. When the young man on board saw this person approach, he left his station by the pilot, and, hat in hand, leaned over the shipâ̄s bulwarks. He was a fine, tall, slim young fellow of eighteen or twenty, with black eyes, and hair as dark as a ravenâ̄s wing; and his whole appearance bespoke that calmness and resolution peculiar to men accustomed from their cradle to contend with danger.

```

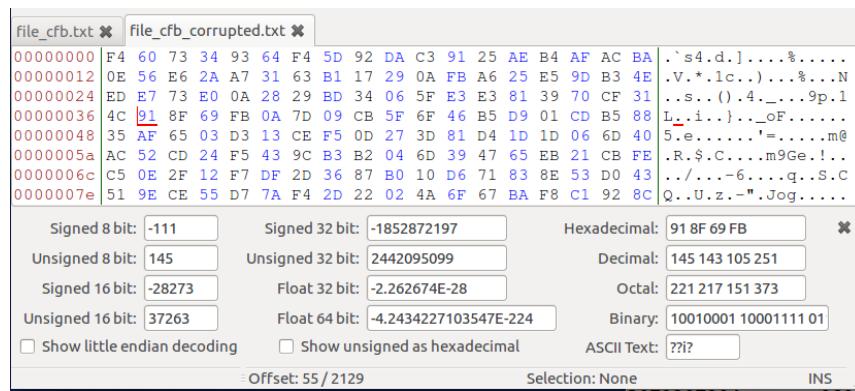
Next, let's use CFB mode.

```
[06/11/21]seed@VM:~/.../Task5$ openssl enc -aes-128-cfb -e -in file.txt -out file_cfb.txt -K 1023456789abcdfe -iv 0123456789aabbccddeeff
[06/11/21]seed@VM:~/.../Task5$
```

The click bar is at offset 55 [00000036 / 98]



At offset 55 of encrypted file “file_cfb.txt”, let's place 1 so that new 55th offset becomes “91” and change name of a new file to be “file_cfb_corrupted.txt”



Using the code below, we decrypt “file_cfb_corrupted.txt” into “file_cfb_fixed.txt”

```
[06/11/21]seed@VM:~/.../Task5$ openssl enc -aes-128-cfb -d -in file_cfb_corrupted.txt -out file_cfb_fixed.txt -K 1023456789abcdfe -iv 0123456789aabbccddeeff
[06/11/21]seed@VM:~/.../Task5$
```

This is the content of “file_cfb_fixed.txt”

The screenshot shows a hex editor window with three tabs: file_cfb.txt, file_cfb_corrupted.txt, and file_cfb_fixed.txt. The file_cfb_fixed.txt tab is active, displaying the following text:

On the 24th of February, 1815, the look-out at Notre-Dame de la Médée, the three-master, the Pharaon from Smyrna, Trieste, and Naples.

As usual, a pilot put off immediately, and rounding the Château d'If, got on board the vessel between Cape Morgiou and Rion island.

Immediately, and according to custom, the ramparts of Fort Saint-Jean were covered with spectators; it is always an event at Marseilles for a ship to come into port, especially when this ship, like the Pharaon, has been built, rigged, and laden at the old Phocée docks, and belongs to an owner of the city.

The ship drew on and had safely passed the strait, which some volcanic shock has made between the Calasareigne and Jaros Islands; had doubled Pomègue, and approached the harbor under topsails, jib, and spanker, but so slowly and sedately that the idlers, with instinct which is the forerunner of evil, asked one another what misfortune could have happened on board. However, those experienced in navigation saw plainly that if any accident had occurred, it was not to the vessel herself, for she bore down with all the evidence of being skilfully handled, the anchor a-cockbill, the jib-boom guys already eased off, and standing by the side of the pilot, who was steering the Pharaon towards the narrow entrance of the inner port, was a young man, who, with activity and vigilant eye, watched every motion of the ship, and repeated each direction of the pilot.

The vague disquietude which prevailed among the spectators had so much affected one of the crowd that he did not await the arrival of the vessel in harbor, but jumping into a small skiff, desired to be pulled alongside the Pharaon, which he reached as she rounded into La Réservé basin.

When the young man on board saw this person approach, he left his station by the pilot, and, hat in hand, leaned over the ship's bulwarks.

He was a fine, tall, slim young fellow of eighteen or twenty, with black eyes, and hair as dark as a raven's wing; and his whole appearance bespoke that calmness and resolution peculiar to men accustomed from their cradle to contend with danger.

Below the text, there is a hex dump of the file. At offset 55, the byte value is 63 (A9 in hex). The status bar shows "Offset: 55 / 2129".

Next, let's use OFB mode.

```
[06/11/21]seed@VM:~/.../Task5$ openssl enc -aes-128-ofb -e -in file.txt -out file_ofb.txt -K 1923456789abcdefabcdef
[06/11/21]seed@VM:~/.../Task5$
```

The click bar is at offset 55 [00000036 / 63]

The screenshot shows a hex editor window with the file_ofb.txt tab active. The text is mostly illegible due to encryption, but the click bar highlights the byte at offset 55, which is 63 (A9 in hex).

Below the text, there is a hex dump of the file. At offset 55, the byte value is 63 (A9 in hex). The status bar shows "Offset: 55 / 2129".

At offset 55 of encrypted file “file_ofb.txt”, let's place 1 so that new 55th offset becomes “91” and change name of a new file to be “file_ofb_corrupted.txt”

file_ofb.txt		file_ofb_corrupted.txt	
00000000	56 13 8A A7 86 BC 46 6C 1B 05 8C C9 02 F8 78 95 B0 78	V.....F1.....x..x	
00000012	59 B6 CB 62 46 0F 85 A7 2C B3 1B F5 CA A8 E8 05 CE 0A	Y..bF...,.....	
00000024	7A E6 2A DA 64 A9 AF 9F 90 1A F9 8E 0E C9 8A 1C CE 64	z.*.d.....d	
00000036	8D 64 1C A1 AD 30 BB 86 44 1B 84 58 28 53 88 72 E0 A6	.d...0..D..X(S.r..	
00000048	6D A1 54 F1 1C 4C 03 E0 85 E3 32 C6 D3 AA 52 CD 40 1F	m.T..L....2..R.@.	
0000005a	8E 7A C7 08 47 C1 8F FD D2 7B 9B 0B 38 57 8D 38 1A E2	.z..G....{..8W.8..	
0000006c	8A 67 85 B1 C4 FB 14 86 46 9F 31 04 89 B5 B8 FB 5D F6	.g.....F.1.....].	
0000007e	5D 1E D1 17 A6 13 5E F8 83 C8 A1 90 94 C1 2E 35 42 FC	l.....^.....5B.	
Signed 8 bit:		Signed 32 bit:	1679597997
Unsigned 8 bit:		Unsigned 32 bit:	1679597997
Signed 16 bit:		Float 32 bit:	1.155737E+22
Unsigned 16 bit:		Float 64 bit:	1.77036312261616E+174
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal	
		ASCII Text: d[bc]??	
Offset: 55 / 2129		Selection: None	
		INS	

Using the code below, we decrypt “file_ofb_corrupted.txt” into “file_ofb_fixed.txt”

```
[06/11/21]seed@VM:~/.../Task5$ openssl enc -aes-128-ofb -d -in file_ofb_corrupted.txt -out file_ofb_fixed.txt -K 192345678abcdfe -iv 1234506789abcdefabcdef
[06/11/21]seed@VM:~/.../Task5$
```

This is the content of “file_ofb_fixed.txt”

file_ofb.txt		file_ofb_corrupted.txt		file_ofb_fixed.txt	
00000000	4F 6E 20 74 68 65 20 32 34 74 68 20 6F 66 20 46 65 62	On the 24th of Feb			
00000012	72 75 61 72 79 2C 20 31 38 31 35 2C 20 74 68 65 20 6C	ruary, 1815, the l			
00000024	6F 6F 6B 2D 6F 75 74 20 61 74 20 4E 6F 74 72 65 2D 44	ook-out at Notre-D			
00000036	61 6A 65 20 64 65 20 6C 61 20 47 61 72 64 65 20 73 69	aje de la Garde si			
00000048	67 6E 61 6C 6C 65 64 20 74 68 65 20 74 68 72 65 65 2D	gnalled the three-			
0000005a	6D 61 73 74 65 72 2C 20 74 68 65 20 50 68 61 72 61 6F	master, the Pharaon			
0000006c	6E 20 66 72 6F 6D 20 53 6D 79 72 6E 61 2C 20 54 72 69	n from Smyrna, Tri			
0000007e	65 73 74 65 2C 20 61 6E 64 20 4E 61 70 6C 65 73 2E 0A	este, and Naples..			
Signed 8 bit:		Signed 32 bit:	1785012324	Hexadecimal:	6A 65 20 64
Unsigned 8 bit:		Unsigned 32 bit:	1785012324	Decimal:	106 101 032 100
Signed 16 bit:		Float 32 bit:	6.924924E+25	Octal:	152 145 040 144
Unsigned 16 bit:		Float 64 bit:	3.31188526620617E+204	Binary:	01101010 01100101 00
<input type="checkbox"/> Show little endian decoding		<input type="checkbox"/> Show unsigned as hexadecimal		ASCII Text: jed	
Offset: 55 / 2129		Selection: None		INS	

file_ofb_fixed.txt (~/lab2/Task5) - gedit

Open Save

On the 24th of February, 1815, the look-out at Notre-Daje de la Garde signalled the three-master, the Pharaon from Smyrna, Trieste, and Naples.

As usual, a pilot put off immediately, and rounding the Château d'If, got on board the vessel between Cape Morgiou and Rion island.

Immediately, and according to custom, the ramparts of Fort Saint-Jean were covered with spectators; it is always an event at Marseilles for a ship to come into port, especially when this ship, like the Pharaon, has been built, rigged, and laden at the old Phocée docks, and belongs to an owner of the city.

The ship drew on and had safely passed the strait, which some volcanic shock has made between the Calasareigne and Jaros islands; had doubled Pomègue, and approached the harbor under topsails, jib, and spanker, but so slowly and sedately that the idlers, with that instinct which is the forerunner of evil, asked one another what misfortune could have happened on board. However, those experienced in navigation saw plainly that if any accident had occurred, it was not to the vessel herself, for she bore down with all the evidence of being skilfully handled, the anchor a-cockbill, the jib-boom guys already eased off, and standing by the side of the pilot, who was steering the Pharaon towards the narrow entrance of the inner port, was a young man, who, with activity and vigilant eye, watched every motion of the ship, and repeated each direction of the pilot. The vague disquietude which prevailed among the spectators had so much affected one of the crowd that he did not await the arrival of the vessel in harbor, but jumping into a small skiff, desired to be pulled alongside the Pharaon, which he reached as she rounded into La Réserve basin.

When the young man on board saw this person approach, he left his station by the pilot, and, hat in hand, leaned over the ship's bulwarks.

He was a fine, tall, slim young fellow of eighteen or twenty, with black eyes, and hair as dark as a raven's wing; and his whole appearance bespoke that calmness and resolution peculiar to men accustomed from their cradle to contend with danger.

Task 6: Initial Vector

- 1) Let's create a plaintext with the following content

Then, we encrypt the plaintext into “ciphertext_cbc_one.txt” using cipher -aes-128-cbc, key 00112233445566778899aabcccddeff, and IV 0123456789abcdef. Using “bless”, we can see the hex content of “ciphertext_cbc_one.txt”

```
[06/10/21]seed@VM:~/.../Task6$ openssl enc -aes-128-cbc -e -in plaintext.txt -out ciphertext_cbc_one.txt -K 00112233445566778899aabcccddeff -iv 0123456789abcdef
```

Let's see what happens if we encrypt the plaintext into another ciphertext called “ciphertext_cbc_two.txt” using the same key and IV as before.

```
[06/10/21]seed@VM:~/.../Task6$ openssl enc -aes-128-cbc -e -in plaintext.txt -out ciphertext_cbc_two.txt -K 00112233445566778899aabcccddeff -iv 0123456789abcdef
```

As can be seen from above, there is no change if we use the same initial vector. Now, we perform what if the initial vectors are different. Let's see the change:

```
[06/10/21]seed@VM:~/.../Task6$ openssl enc -aes-128-cbc -e -in plaintext.txt -out ciphertext_cbc_three.txt -K 00112233445566778899aabcccddeff -iv 0123456789abcdef
```

The third ciphertext looks different from the first one. Based upon this observation, the IV must be unique because an attacker, who is given two ciphertext might have a difficult time to find if the two decrypt to the same plaintext or not. The uniqueness of IVs increases the security of plaintext.

- 2) Let's convert the plaintext, P1, to hex using Ascii table

P1 = This is a known message! => 54 68 69 73 20 69 73 20 61 20 6b 6e 6f 77 6e 20 6d 65 73 73 61 67 65 21 0a

C1 is already in hex, so we can find K, using formula $K = P1 \text{ xor } C1$ (because we encrypt the message)

$K =$

54 68 69 73 20 69 73 20 61 20 6b 6e 6f 77 6e 20 6d 65 73 73 61 67 65 21 0a

xor

a4 69 b1 c5 02 c1 ca b9 66 96 5e 50 42 54 38 e1 bb 1b 5f 90 37 a4 c1 59 13

=

f0 01 d8 b6 22 a8 b9 99 07 b6 35 3e 2d 23 56 c1 d6 7e 2c e3 56 c3 a4 78 19

Since the usage of IV was the same, we assume that $P2 = K \text{ xor } C2$ (because we decrypt the message)

$P2 =$

f0 01 d8 b6 22 a8 b9 99 07 b6 35 3e 2d 23 56 c1 d6 7e 2c e3 56 c3 a4 78 19

xor

bf 73 bc d3 50 92 99 d5 66 c3 5b 5d 45 03 37 e1 bb 17 5f 90 3f af c1 59 13

=

4f 72 64 65 72 3a 20 4c 61 75 6e 63 68 20 61 20 6d 69 73 73 69 6c 65 21 0a

Hex to Ascii in P2 gives result:

“Order: Launch a missile!”

If OFB is replaced with CFB, then only the first block can be revealed from the plaintext OFB. If the IV is reused, the whole key stream will be reproduced again, since the keystream is done by successive encryption of the IV: $K_0 = \text{IV}$, $K_1 = E_k(C_0)$, $K_2 = E_k(C_1)$, ...

- 3) Assume that P1 is “Yes”. According to Ascii table, “Yes” turns to be 59 65 73 in hex