LAB 8: Secret-Key Encryption Lab

TASK 1: Frequency Analysis Against Monoalphabetic Substitution Cipher

```
ytn xqavhq yzhu  xu qzupvd ltmat qnncq vgxzy hmrty vbynh ytmq ixur qyhvurn
vlvhpq yhme ytn gvrrnh bnniq imsn v uxuvrnuvhmvu yxx

ytn vlvhpq hvan lvq gxxsnupnp gd ytn pncmqn xb tvhfnd lnmuqynmu vy myq xzyqny
vup ytn veevhnuy mceixqmxu xb tmq bmic axcevud vy ytn nup vup my lvq qtvenp gd
ytn ncnhrnuan xb cnyxx ymcnq ze givasrxlu eximymaq vhcavupd vaymfmqc vup
v uvymxuvi axufnhqvymxu vq ghmnb vup cvp vq v bnfnh phnvc vgxzy ltnytnh ytnhn
xzrty yx gn v ehnqmpnuy lmubhnd ytn qnvqxu pmpuy ozqy qnnc nkyhv ixur my lvq
nkyhv ixur gnavzqn ytn xqavhq lnhn cxfnp yx ytn bmhqy lnnsnup mu cvhat yx
vfxmp axubimaymur lmyt ytn aixqmur anhncxud xb ytn lmuynh xidcemaq ytvusq
ednxuratvur

xun gmr jznqymxu qzhhxzupmur ytmq dnvhq vavpncd vlvhpq mq txl xh mb ytn
anhncxud lmii vpphnqq cnyxx nqenamviid vbynh ytn rxipnu rixgnq ltmat gnavcn
v ozgmivuy axcmurxzy evhyd bxh ymcnq ze ytn cxfncnuy qenvhtnvpnp gd
exlnhbzi txiidlxxp lxcnu ltx tnienp hvmqn cmiimxuq xb pxiivhq yx bmrty qnkzvi
tvhvqqcnuy vhxzup ytn axzuyhd

qmruvimur ytnmh qzeexhy rxipnu rixgnq vyynupnnq qlvytnp ytncqnifnq mu givas
qexhynp iveni emuq vup qxzupnp xbb vgxzy qnkmqy exlnh mcgvivuanq bhxc ytn hnp
avheny vup ytn qyvrn xu ytn vmh n lvq aviinp xzy vgxzy evd munjzmyd vbynh
myq bxhcnh vuatxh avyy qvpinh jzmy xuan qtn invhunp ytvy qtn lvq cvsmur bvh
inqq ytvu v cvin axtxqy vup pzhmur ytn anhncxud uvyvimn exhycvu yxxs v gizuy
vup qvymqbdmur pmr vy ytn viicvin hxqynh xb uxcmuvynp pmhnayxhq txl axzip
ytvy gn yxeenp

vq my yzhuq xzy vy invqy mu ynhcq xb ytn xqavhq my ehxgvgid lxuy gn

lxcnu mufxifnp mu ymcnq ze qvmp ytvy viytxzrt ytn rixgnq qmrumbmnp ytn
mumymvymfnq ivzuat ytnd unfnh muynupnp my yx gn ozqy vu vlvhpq qnvqxu
avcevmru xh xun ytvy gnavcn vqqxamvynp xuid lmyt hnpavheny vaymxuq muqynvp
v qexsnqlxcvu qvmp ytn rhxze mq lxhsmur gntmup aixqnp pxxhq vup tvq qmuan
vcvqqnp  cmiimxu bxh myq inrvi pnbnuqn bzup ltmat vbynh ytn rixgnq lvq
bixxpnp lmyt ytxzqvupq xb pxuvymxuq xb  xh inqq bhxc enxein mu qxcn
axzuyhmnq


ux avii yx lnvh givas rxluq lnuy xzy mu vpfvuan xb ytn xqavhq ytxzrt ytn
cxfncnuy lmii vicxqy anhyvmuid gn hnbnhnuanp gnbxhn vup pzhmur ytn anhncxud
nqenamviid qmuan fxavi cnyxx qzeexhynhq imsn vqtind ozpp ivzhv pnhu vup
umaxin smpcvu vhn qatnpzinp ehnqnuynhq

vuxytnh bnvyzhn xb ytmq qnvqxu ux xun hnviid suxlq ltx mq rxmur yx lmu gnqy
emayzhn vhrzvgid ytmq tveenuq v ixy xb ytn ymcn muvhrzvgid ytn uvmigmynh
uvhhvymfn xuid qnhfnq ytn vlvhpq tden cvatmun gzy xbynu ytn enxein bxhnavqymur
ytn hvan qxaviinp xqavhxixrmqyq avu cvsn xuid npzavynp rznqqnq
```

```
[04/04/19]\Shenava@VM:~$ cd Desktop/Encryption/
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'y' 'T' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'yt' 'TH' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytn' 'THE' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnb' 'THEF' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbx' 'THEFO' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxv' 'THEFOA' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvh' 'THEFOAR' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvha' 'THEFOARC' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvhaq' 'THEFOARCS' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvhaqu' 'THEFOARCSN' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvhaquc' 'THEFOARCSNM' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvhaqucl' 'THEFOARCSNMW' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvhaquclr' 'THEFOARCSNMWG' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvhaquclrd' 'THEFOARCSNMWGY' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvhaquclrde' 'THEFOARCSNMWGYP' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvhaquclrdei' 'THEFOARCSNMWGYPL' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvhaquclrdeik' 'THEFOARCSNMWGYPLX' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvhaquclrdeikf' 'THEFOARCSNMWGYPLXV' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvhaquclrdeikfp' 'THEFOARCSNMWGYPLXVD' < ciphertext.txt > decipher.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ tr 'ytnbxvhaquclrdeikfpsgjmozw' 'THEFOARCSNMWGYPLXVDKBQIJUZ' < ciphertext.txt > deci
pher.txt
```

```
THE OSCARS TURN  ON SUNDAY WHICH SEEMS ABOUT RIGHT AFTER THIS LONG STRANGE
AWARDS TRIP THE BAGGER FEELS LIKE A NONAGENARIAN TOO

THE AWARDS RACE WAS BOOKENDED BY THE DEMISE OF HARVEY WEINSTEIN AT ITS OUTSET
AND THE APPARENT IMPLOSION OF HIS FILM COMPANY AT THE END AND IT WAS SHAPED BY
THE EMERGENCE OF METOO TIMES UP BLACKGOWN POLITICS ARMCANDY ACTIVISM AND
A NATIONAL CONVERSATION AS BRIEF AND MAD AS A FEVER DREAM ABOUT WHETHER THERE
OUGHT TO BE A PRESIDENT WINFREY THE SEASON DIDNT JUST SEEM EXTRA LONG IT WAS
EXTRA LONG BECAUSE THE OSCARS WERE MOVED TO THE FIRST WEEKEND IN MARCH TO
AVOID CONFLICTING WITH THE CLOSING CEREMONY OF THE WINTER OLYMPICS THANKS
PYEONGCHANG

ONE BIG QUESTION SURROUNDING THIS YEARS ACADEMY AWARDS IS HOW OR IF THE
CEREMONY WILL ADDRESS METOO ESPECIALLY AFTER THE GOLDEN GLOBES WHICH BECAME
A JUBILANT COMINGOUT PARTY FOR TIMES UP THE MOVEMENT SPEARHEADED BY
POWERFUL HOLLYWOOD WOMEN WHO HELPED RAISE MILLIONS OF DOLLARS TO FIGHT SEXUAL
HARASSMENT AROUND THE COUNTRY

SIGNALING THEIR SUPPORT GOLDEN GLOBES ATTENDEES SWATHED THEMSELVES IN BLACK
SPORTED LAPEL PINS AND SOUNDED OFF ABOUT SEXIST POWER IMBALANCES FROM THE RED
CARPET AND THE STAGE ON THE AIR E WAS CALLED OUT ABOUT PAY INEQUITY AFTER
ITS FORMER ANCHOR CATT SADLER QUIT ONCE SHE LEARNED THAT SHE WAS MAKING FAR
LESS THAN A MALE COHOST AND DURING THE CEREMONY NATALIE PORTMAN TOOK A BLUNT
AND SATISFYING DIG AT THE ALLMALE ROSTER OF NOMINATED DIRECTORS HOW COULD
THAT BE TOPPED

AS IT TURNS OUT AT LEAST IN TERMS OF THE OSCARS IT PROBABLY WONT BE

WOMEN INVOLVED IN TIMES UP SAID THAT ALTHOUGH THE GLOBES SIGNIFIED THE
INITIATIVES LAUNCH THEY NEVER INTENDED IT TO BE JUST AN AWARDS SEASON
CAMPAIGN OR ONE THAT BECAME ASSOCIATED ONLY WITH REDCARPET ACTIONS INSTEAD
A SPOKESWOMAN SAID THE GROUP IS WORKING BEHIND CLOSED DOORS AND HAS SINCE
AMASSED  MILLION FOR ITS LEGAL DEFENSE FUND WHICH AFTER THE GLOBES WAS
FLOODED WITH THOUSANDS OF DONATIONS OF  OR LESS FROM PEOPLE IN SOME
COUNTRIES


NO CALL TO WEAR BLACK GOWNS WENT OUT IN ADVANCE OF THE OSCARS THOUGH THE
MOVEMENT WILL ALMOST CERTAINLY BE REFERENCED BEFORE AND DURING THE CEREMONY
ESPECIALLY SINCE VOCAL METOO SUPPORTERS LIKE ASHLEY JUDD LAURA DERN AND
NICOLE KIDMAN ARE SCHEDULED PRESENTERS

ANOTHER FEATURE OF THIS SEASON NO ONE REALLY KNOWS WHO IS GOING TO WIN BEST
PICTURE ARGUABLY THIS HAPPENS A LOT OF THE TIME INARGUABLY THE NAILBITER
NARRATIVE ONLY SERVES THE AWARDS HYPE MACHINE BUT OFTEN THE PEOPLE FORECASTING
THE RACE SOCALLED OSCAROLOGISTS CAN MAKE ONLY EDUCATED GUESSES
```

We do frequency analysis of single alphabets, paired alphabets and grouped alphabets and mark them according to their chance of being used the most and frequency is taken. We initially convert all the uppercase to lowercase and avoid the spacing. Then once we figure out the alphabets to change, we use the tr command to replace the cipher text to original text.

TASK 2: Encryption using different ciphers and modes

```
[04/04/19]\Shenava@VM:~/.../Encryption$ gedit plain.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ cat plain.txt
Hi Mrudhula here doing Cybersecurity
[04/04/19]\Shenava@VM:~/.../Encryption$
```

The original plain text, plain.txt.

1. -aes-128-cbc

```
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -e -in plain.tx
t -out cipheraes-128-cbc.bin -K 00112233445566778889aabbccddeeff -iv 01020304050
60708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -d -in cipherae
s-128-cbc.bin -out plainaes-128-cbc.txt -K 00112233445566778889aabbccddeeff -iv
0102030405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ cat plainaes-128-cbc.txt
Hi Mrudhula here doing Cybersecurity
[04/04/19]\Shenava@VM:~/.../Encryption$
```

Above is the screenshot of the -aes-128-cbc mode encryption and decryption. The decrypted cipher text, that is, plainaes-128-cbc.txt is same as the original plaintext, plain.txt.

2. -aes-128-ofb

```
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -e -in plain.tx
t -out cipheraes-128-ofb.bin -K 00112233445566778889aabbccddeeff -iv 01020304050
60708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -d -in cipherae
s-128-ofb.bin -out plainaes-128-ofb.txt -K 00112233445566778889aabbccddeeff -iv
0102030405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ cat plainaes-128-ofb.txt
Hi Mrudhula here doing Cybersecurity
[04/04/19]\Shenava@VM:~/.../Encryption$
```

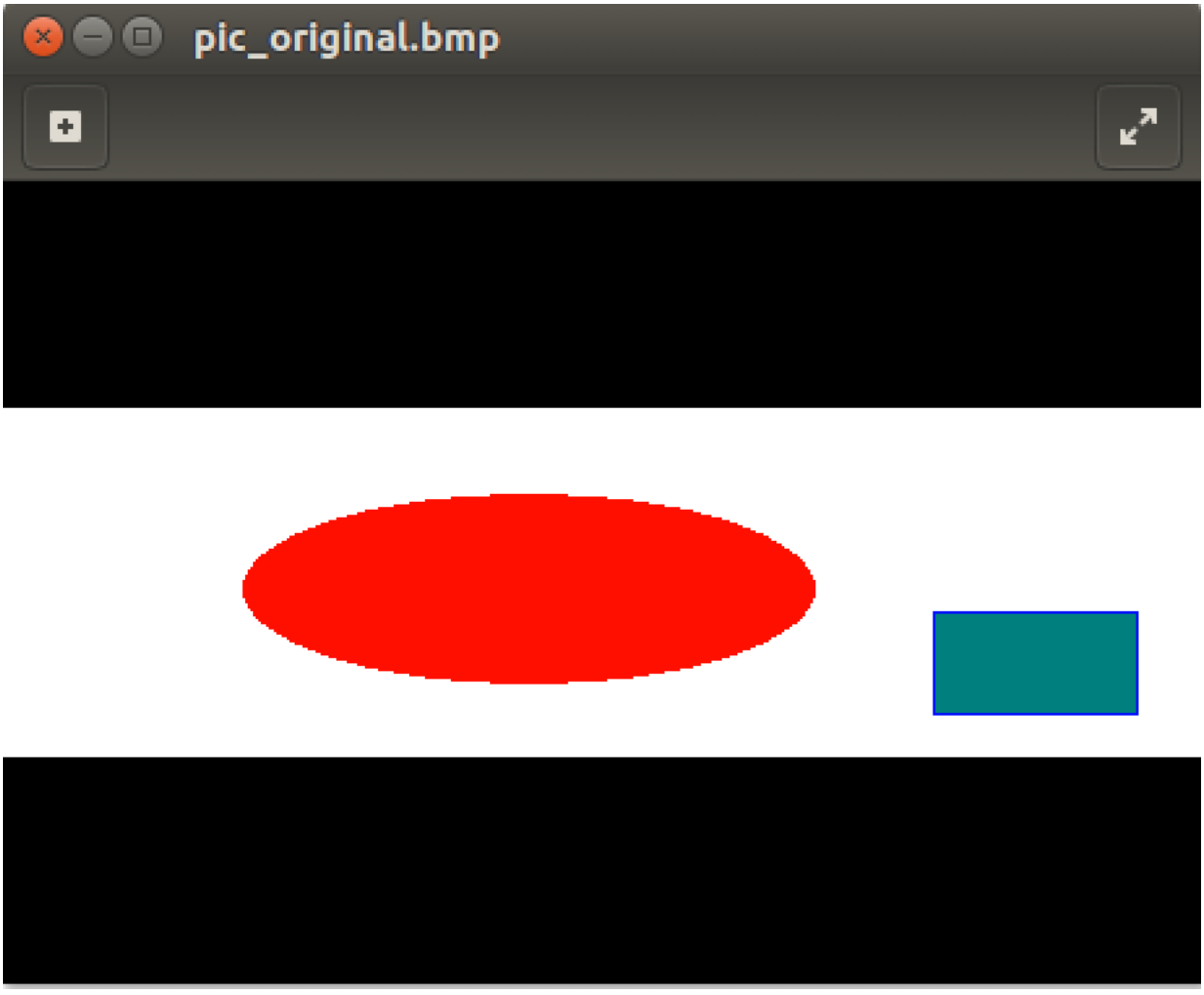
Above is the screenshot of the -aes-128-ofb mode encryption and decryption. The decrypted cipher text, that is, plainaes-128-ofb.txt is same as the original plaintext, plain.txt.

3. -bf-cbc

```
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -bf-cfb -e -in plain.txt -ou
t cipherbf-cfb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -bf-cfb -d -in cipherbf-cfb.
bin -out plainbf-cfb.txt -K 00112233445566778889aabbccddeeff -iv 010203040506070
8
[04/04/19]\Shenava@VM:~/.../Encryption$ cat plainbf-cfb.txt
Hi Mrudhula here doing Cybersecurity
[04/04/19]\Shenava@VM:~/.../Encryption$
```

Above is the screenshot of the -bf-cbc mode encryption and decryption. The decrypted cipher text, that is, plainbf-cbc.txt is same as the original plaintext, plain.txt.

TASK 3: Encryption Mode – EBC vs. CBC

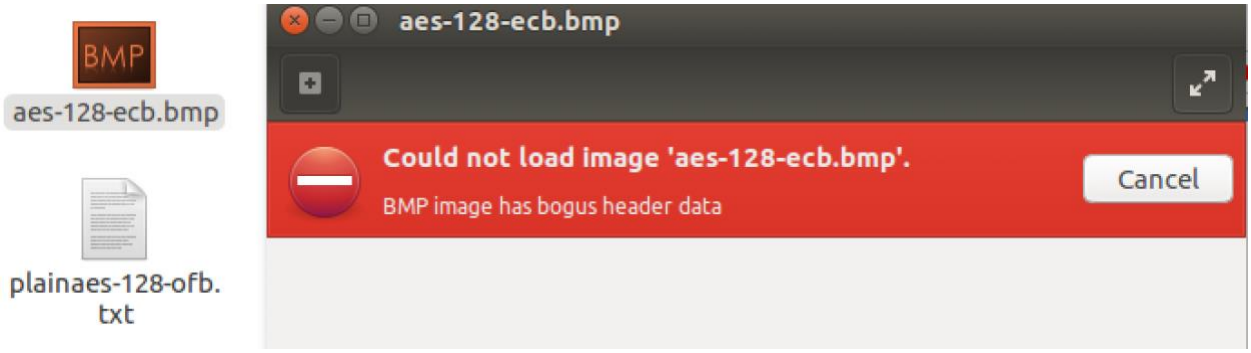


The original picture

We now encrypt the original picture using aes-128 bit ECB and CBC mode of encryption. After we do the encryption, when we try to open the image file using an image viewer it throws an error of the image containing bogus header data.

```
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ecb -e -in pic_orig
inal.bmp -out aes-128-ecb.bmp -K 00112233445566778889aabbccddeeff
```

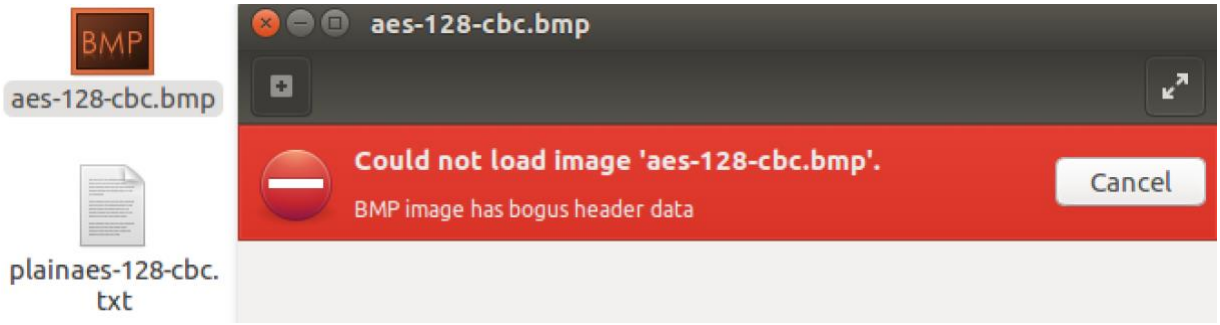
-aes-128-ecb encryption model



when we open after we do the -aes-128-ecb encryption the image cannot be viewed. The header gets encrypted.

```
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -e -in pic_orig
inal.bmp -out aes-128-cbc.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405
060708
[04/04/19]\Shenava@VM:~/.../Encryption$
```

-aes-128-cbc encryption model

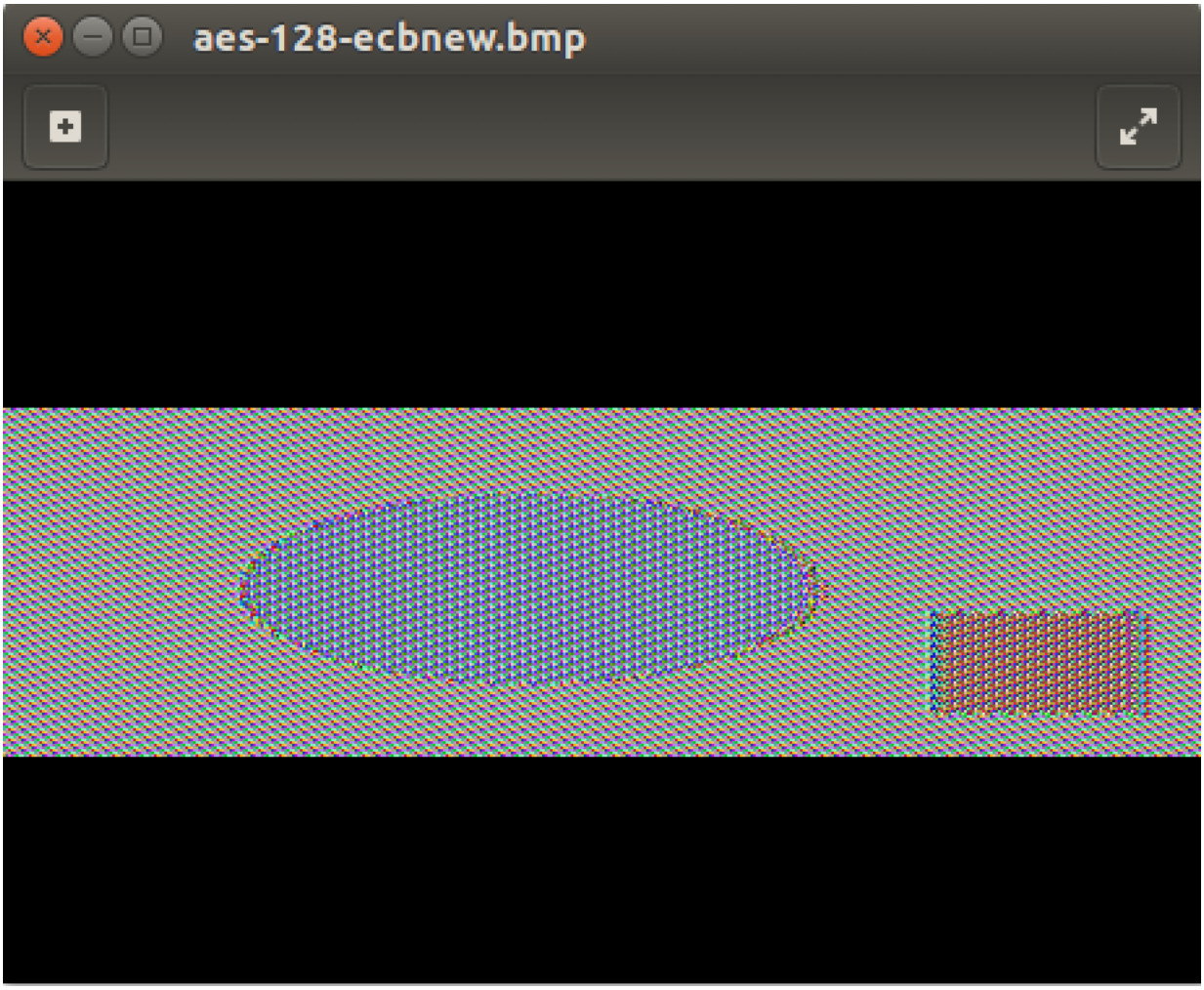


when we open after we do the -aes-128-cbc encryption the image cannot be viewed. The header gets encrypted.

```
[04/04/19]\Shenava@VM:~/.../Encryption$ head -c 54 pic_original.bmp > header
[04/04/19]\Shenava@VM:~/.../Encryption$ tail -c +55 aes-128-ecb.bmp > body
[04/04/19]\Shenava@VM:~/.../Encryption$ cat header body > aes-128-ecbnew.bmp
```

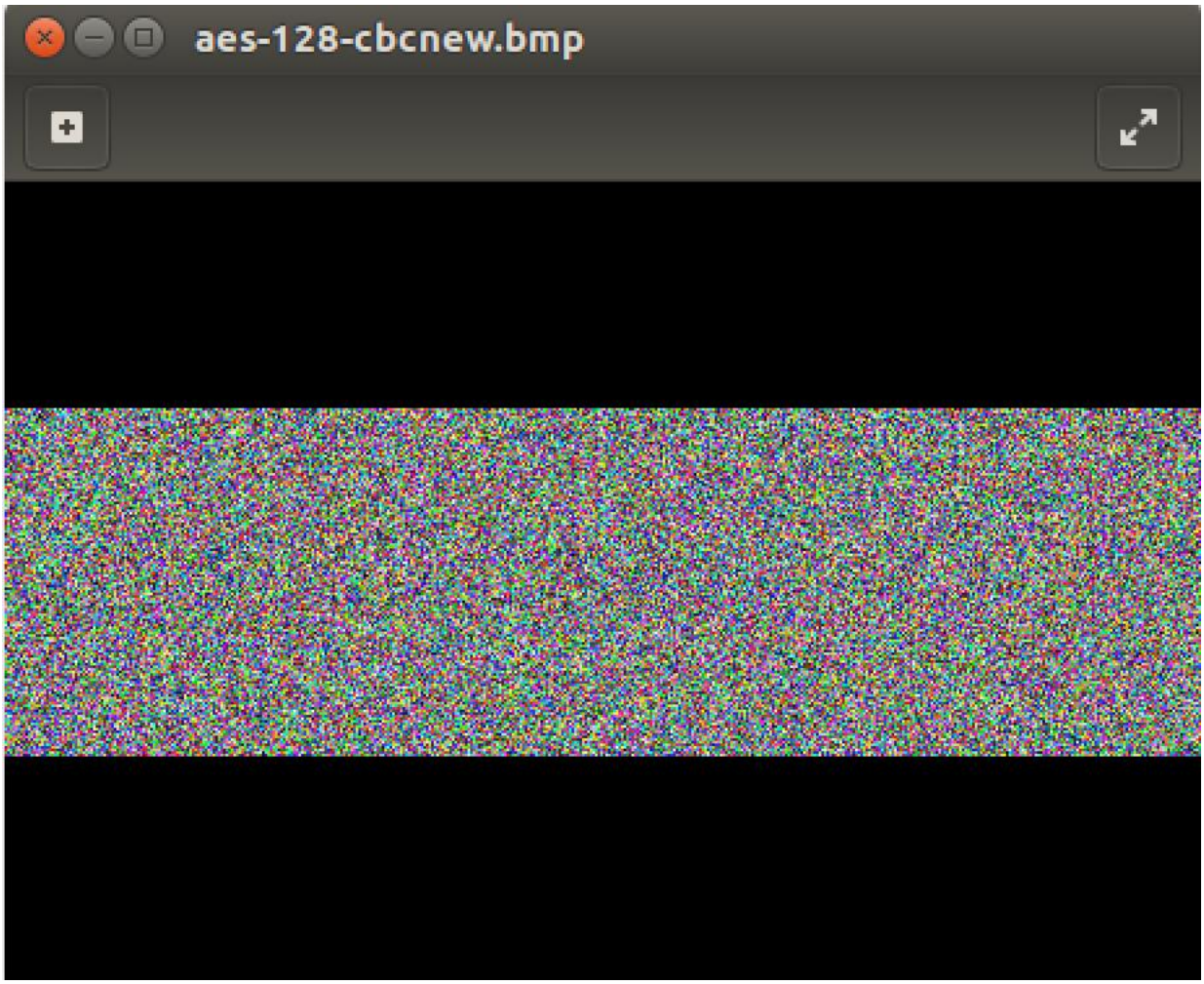
Then we open the original image and both the encrypted images using GHex and notice that the 54 bytes of data is different. We correct the header file of the encrypted images according to the original image by replacing the 54 bytes and make it similar to the original image as shown in above command line screenshot.

We get the below shown result when we open the encrypted images.



When we do ECB encryption the chance of useful information to be derived is likely.

```
[04/04/19]\Shenava@VM:~/.../Encryption$ head -c 54 pic_original.bmp > header
[04/04/19]\Shenava@VM:~/.../Encryption$ tail -c +55 aes-128-cbc.bmp > body
[04/04/19]\Shenava@VM:~/.../Encryption$ cat header body > aes-128-cbcnew.bmp
```



When we do CBC encryption useful information cannot be derived.
The ECB mode of encryption is not secure and CBC mode of encryption is more secure as seen from the above screenshots. ECB mode gives an idea of what is there in the file, whereas CBC mode doesn't reveal any information to the eavesdropper.

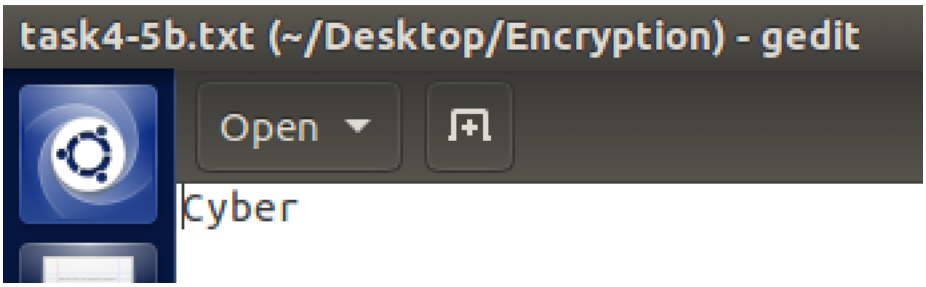
The ECB mode applies the encryption to each block independently. Hence, if the same block is repeated, then the cipher texts of those blocks will also be the same. This reveals information to the attacker or eavesdropper and hence it is not secure for messages where content is repeated.

The CBC mode applies the encryption to each block, but each plaintext block is XORed with the cipher text of previous block to form the new cipher text for the current block. So even if words are repeated, the IV input is different for each block and hence no information is revealed.

TASK 4: Padding

For this task we create two plaintext of lengths 5 bytes, 10 bytes and 16 bytes each.
First, we start off with 5 byte length plaintext named task4-5b.txt

```
[04/04/19]\Shenava@VM:~/.../Encryption$ echo -n "Cyber" > task4-5b.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ gedit task4-5b.txt
```



Original text file.

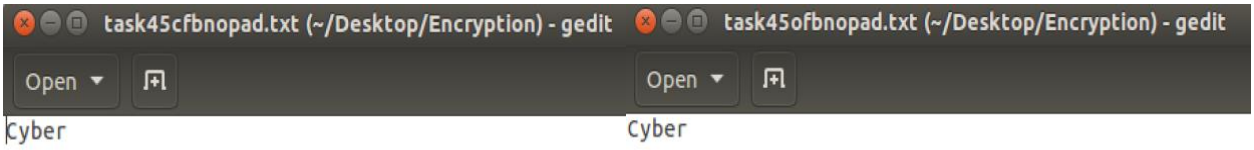
```
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ecb -e -in task4-5b
.txt -out task4cipher5ecb.bin -K 00112233445566778889aabbccddeeff
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -e -in task4-5b
.txt -out task4cipher5cbc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405
060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -e -in task4-5b
.txt -out task4cipher5cfb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405
060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -e -in task4-5b
.txt -out task4cipher5ofb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405
060708
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task4cipher
-rw-rw-r-- 1 seed seed     16 Apr  4 20:47 task4cipher5cbc.bin
-rw-rw-r-- 1 seed seed      5 Apr  4 20:47 task4cipher5cfb.bin
-rw-rw-r-- 1 seed seed     16 Apr  4 20:47 task4cipher5ecb.bin
-rw-rw-r-- 1 seed seed      5 Apr  4 20:47 task4cipher5ofb.bin
[04/04/19]\Shenava@VM:~/.../Encryption$
```

We encrypt the 5 byte plaintext with different modes of AES algorithm. The 5 byte plaintext is padded with 11 bytes so that it occupies the entire block for ECB and CBC modes since they require that the input to be an exact multiple of the block size.

```
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ecb -d -in task4cip
her5ecb.bin -out task45ecb.txt -K 00112233445566778889aabbccddeeff
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ecb -d -in task4cip
her5ecb.bin -out task45ecbnopad.txt -nopad -K 00112233445566778889aabbccddeeff
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task45ecb
-rw-rw-r-- 1 seed seed     16 Apr  4 20:52 task45ecbnopad.txt
-rw-rw-r-- 1 seed seed      5 Apr  4 20:51 task45ecb.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -d -in task4cip
her5cbc.bin -out task45cbc.txt -K 00112233445566778889aabbccddeeff -iv 010203040
5060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -d -in task4cip
her5cbc.bin -out task45cbcnopad.txt -nopad -K 00112233445566778889aabbccddeeff -
iv 0102030405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task45cbc
-rw-rw-r-- 1 seed seed     16 Apr  4 20:54 task45cbcnopad.txt
-rw-rw-r-- 1 seed seed      5 Apr  4 20:53 task45cbc.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -d -in task4cip
her5cfb.bin -out task45cfb.txt -K 00112233445566778889aabbccddeeff -iv 010203040
5060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -d -in task4cip
her5cfb.bin -out task45cfbnopad.txt -nopad -K 00112233445566778889aabbccddeeff -
iv 0102030405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task45cfb
-rw-rw-r-- 1 seed seed      5 Apr  4 21:04 task45cfbnopad.txt
-rw-rw-r-- 1 seed seed      5 Apr  4 21:04 task45cfb.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -d -in task4cip
her5ofb.bin -out task45ofb.txt -K 00112233445566778889aabbccddeeff -iv 010203040
5060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -d -in task4cip
her5ofb.bin -out task45ofbnopad.txt -nopad -K 00112233445566778889aabbccddeeff -
iv 0102030405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task45ofb
-rw-rw-r-- 1 seed seed      5 Apr  4 21:07 task45ofbnopad.txt
-rw-rw-r-- 1 seed seed      5 Apr  4 21:05 task45ofb.txt
[04/04/19]\Shenava@VM:~/.../Encryption$
```

We see that padding is added to ECB and CBC modes by decrypting with the nopad option. Hence, padding is not removed. The other 2 modes have no padding so it is not affected.

Padding is added to the file and it is not removed when we use the nopad option while decrypting. Above we shown the screenshot of task45ecbnopad.txt and task45cbcnopad.txt each having 11 bytes of padding.

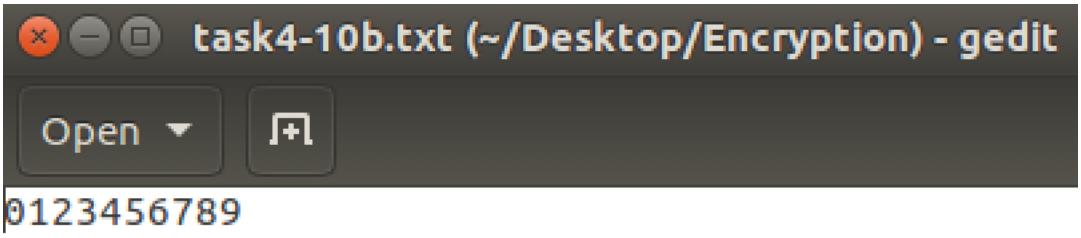


Since CFB and OFB do not have padding it does not get affected. In the above screenshot of task45cfbnopad.txt and task45ofbnopad.txt we notice that there is no padding taking place.

Block ciphers use PKC#5 padding. So, it requires that the plaintext be a multiple of the block size so that encryption can take place for ECB and CBC modes. For this reason, padding is added while encryption. If plaintext doesn't add padding, then it waits for the data to come so that the block size if full and this causes a delay. Hence, padding is added so that blocks are sent in real time. CFB and OFB don't require padding because these modes use stream input and block input.

Second, we try on the 10 byte length plaintext namely task4-10b.txt

```
[04/04/19]\Shenava@VM:~/.../Encryption$ echo -n "0123456789" > task4-10b.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ gedit task4-10b.txt
```



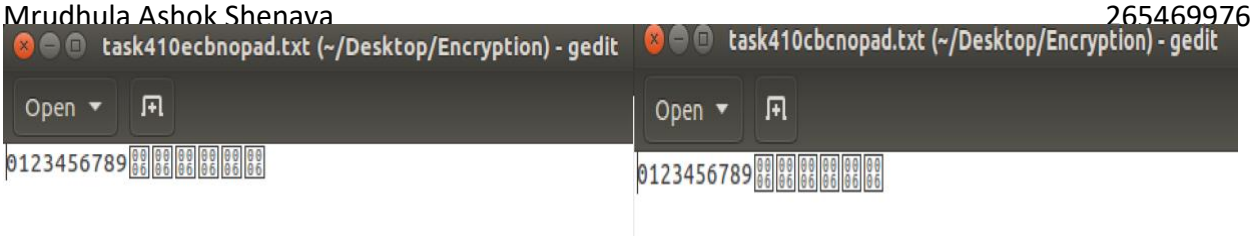
Original text file.

```
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ecb -e -in task4-10
b.txt -out task4cipher10ecb.bin -K 00112233445566778889aabbccddeeff
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -e -in task4-10
b.txt -out task4cipher10cbc.bin -K 00112233445566778889aabbccddeeff -iv 01020304
05060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -e -in task4-10
b.txt -out task4cipher10cfb.bin -K 00112233445566778889aabbccddeeff -iv 01020304
05060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -e -in task4-10
b.txt -out task4cipher10ofb.bin -K 00112233445566778889aabbccddeeff -iv 01020304
05060708
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task4cipher10
-rw-rw-r-- 1 seed seed     16 Apr  4 21:20 task4cipher10cbc.bin
-rw-rw-r-- 1 seed seed     10 Apr  4 21:22 task4cipher10cfb.bin
-rw-rw-r-- 1 seed seed     16 Apr  4 21:18 task4cipher10ecb.bin
-rw-rw-r-- 1 seed seed     10 Apr  4 21:24 task4cipher10ofb.bin
[04/04/19]\Shenava@VM:~/.../Encryption$
```

We encrypt the 10 byte plaintext with different modes of AES algorithm. The 10 byte plaintext is padded with 6 bytes so that it occupies the entire block for ECB and CBC modes since they require that the input to be an exact multiple of the block size.

```
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ecb -d -in task4cip
her10ecb.bin -out task410ecb.txt -K 00112233445566778889aabbccddeeff
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ecb -d -in task4cip
her10ecb.bin -out task410ecbnopad.txt -nopad -K 00112233445566778889aabbccddeeff
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task410ecb
-rw-rw-r-- 1 seed seed     16 Apr  4 21:37 task410ecbnopad.txt
-rw-rw-r-- 1 seed seed     10 Apr  4 21:37 task410ecb.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -d -in task4cip
her10cbc.bin -out task410cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -d -in task4cip
her10cbc.bin -out task410cbcnopad.txt -nopad -K 00112233445566778889aabbccddeeff
 -iv 0102030405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task410cbc
-rw-rw-r-- 1 seed seed     16 Apr  4 21:37 task410cbcnopad.txt
-rw-rw-r-- 1 seed seed     10 Apr  4 21:37 task410cbc.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -d -in task4cip
her10cfb.bin -out task410cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -d -in task4cip
her10cfb.bin -out task410cfbnopad.txt -nopad -K 00112233445566778889aabbccddeeff
 -iv 0102030405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task410cfb
-rw-rw-r-- 1 seed seed     10 Apr  4 21:41 task410cfbnopad.txt
-rw-rw-r-- 1 seed seed     10 Apr  4 21:40 task410cfb.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -d -in task4cip
her10ofb.bin -out task410ofb.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -d -in task4cip
her10ofb.bin -out task410ofbnopad.txt -nopad -K 00112233445566778889aabbccddeeff
 -iv 0102030405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task410ofb
-rw-rw-r-- 1 seed seed     10 Apr  4 21:44 task410ofbnopad.txt
-rw-rw-r-- 1 seed seed     10 Apr  4 21:44 task410ofb.txt
[04/04/19]\Shenava@VM:~/.../Encryption$
```

We see that padding is added to ECB and CBC modes by decrypting with the nopad option. Hence, padding is not removed. The other 2 modes have no padding so it is not affected.

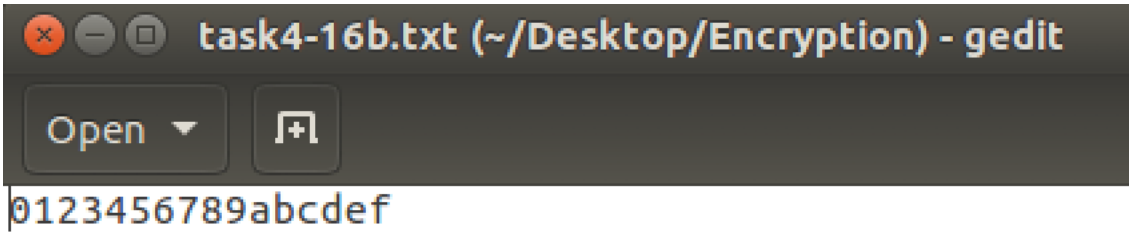


Padding is added to the file and it is not removed when we use the nopad option while decrypting. Above we shown the screenshot of task410ecbnopad.txt and task410cbcnopad.txt each having 6 bytes of padding.

Since CFB and OFB do not have padding it does not get affected. In the above screenshot of task410cfbnopad.txt and task410ofbnopad.txt we notice that there is no padding taking place.

Now we try on the 16 byte length plaintext namely task4-16b.txt

```
[04/04/19]\Shenava@VM:~/.../Encryption$ gedit task4-16b.txt
```



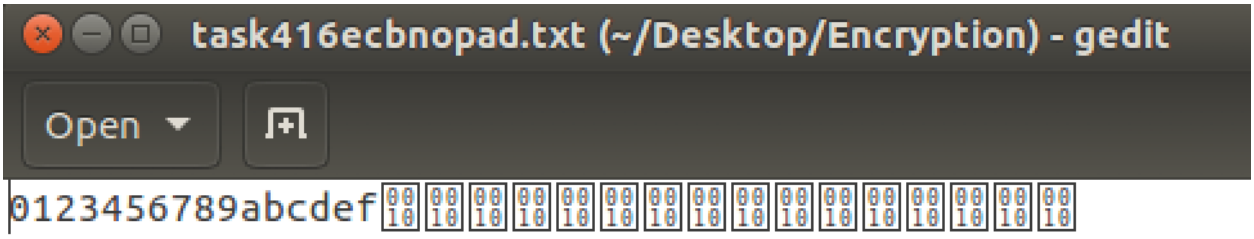
Original text file.

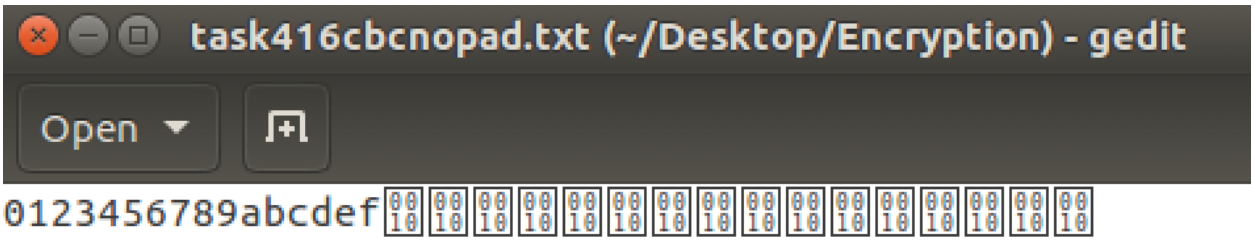
```
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ecb -e -in task4-16
b.txt -out task4cipher16ecb.bin -K 00112233445566778889aabbccddeeff
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -e -in task4-16
b.txt -out task4cipher16cbc.bin -K 00112233445566778889aabbccddeeff -iv 01020304
05060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -e -in task4-16
b.txt -out task4cipher16cfb.bin -K 00112233445566778889aabbccddeeff -iv 01020304
05060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -e -in task4-16
b.txt -out task4cipher16ofb.bin -K 00112233445566778889aabbccddeeff -iv 01020304
05060708
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task4cipher16
-rw-rw-r-- 1 seed seed     32 Apr  4 21:57 task4cipher16cbc.bin
-rw-rw-r-- 1 seed seed     16 Apr  4 21:57 task4cipher16cfb.bin
-rw-rw-r-- 1 seed seed     32 Apr  4 21:56 task4cipher16ecb.bin
-rw-rw-r-- 1 seed seed     16 Apr  4 21:58 task4cipher16ofb.bin
[04/04/19]\Shenava@VM:~/.../Encryption$
```

We encrypt the 16 byte plaintext with different modes of AES algorithm. The 16 byte plaintext is padded with 16 bytes so that it occupies the entire block for ECB and CBC modes since they require that the input to be an exact multiple of the block size.

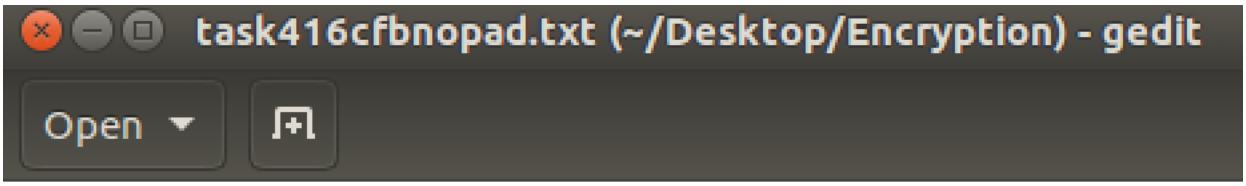
```
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ecb -d -in task4cip
her16ecb.bin -out task416ecb.txt -K 00112233445566778889aabbccddeeff
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ecb -d -in task4cip
her16ecb.bin -out task416ecbnopad.txt -nopad -K 00112233445566778889aabbccddeeff

[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task416ecb
-rw-rw-r-- 1 seed seed     32 Apr  4 21:59 task416ecbnopad.txt
-rw-rw-r-- 1 seed seed     16 Apr  4 21:58 task416ecb.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -d -in task4cip
her16cbc.bin -out task416cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -d -in task4cip
her16cbc.bin -out task416cbcnopad.txt -nopad -K 00112233445566778889aabbccddeeff
 -iv 0102030405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task416cbc
-rw-rw-r-- 1 seed seed     32 Apr  4 22:00 task416cbcnopad.txt
-rw-rw-r-- 1 seed seed     16 Apr  4 21:59 task416cbc.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -d -in task4cip
her16cfb.bin -out task416cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -d -in task4cip
her16cfb.bin -out task416cfbnopad.txt -nopad -K 00112233445566778889aabbccddeeff
 -iv 0102030405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task416cfb
-rw-rw-r-- 1 seed seed     16 Apr  4 22:01 task416cfbnopad.txt
-rw-rw-r-- 1 seed seed     16 Apr  4 22:00 task416cfb.txt
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -d -in task4cip
her16ofb.bin -out task416ofb.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -d -in task4cip
her16ofb.bin -out task416ofbnopad.txt -nopad -K 00112233445566778889aabbccddeeff
 -iv 0102030405060708
[04/04/19]\Shenava@VM:~/.../Encryption$ ls -l | grep task416ofb
-rw-rw-r-- 1 seed seed     16 Apr  4 22:02 task416ofbnopad.txt
-rw-rw-r-- 1 seed seed     16 Apr  4 22:02 task416ofb.txt
[04/04/19]\Shenava@VM:~/.../Encryption$
```

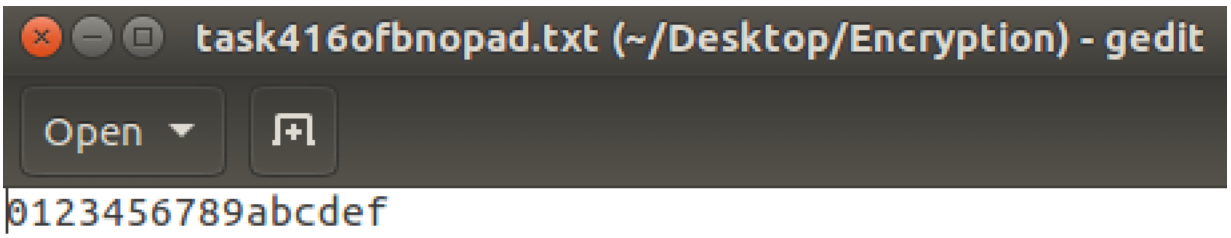
We see that padding is added to ECB and CBC modes by decrypting with the nopad option. Hence, padding is not removed. The other 2 modes have no padding so it is not affected





Padding is added to the file and it is not removed when we use the nopad option while decrypting. Above we shown the screenshot of task416ecbnopad.txt and task416cbcnopad.txt each having 16 bytes of padding.



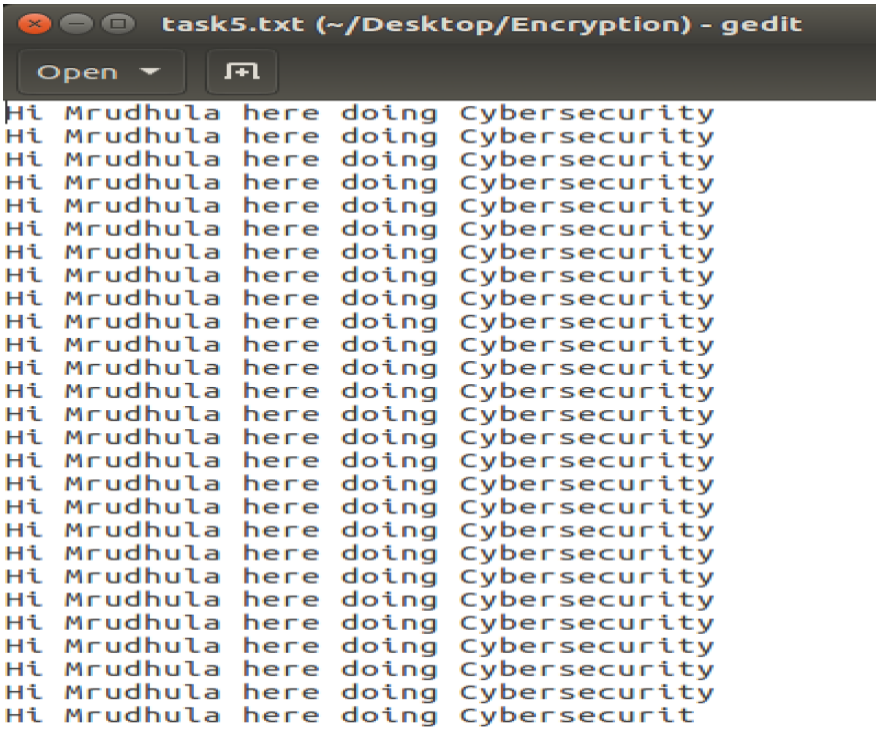


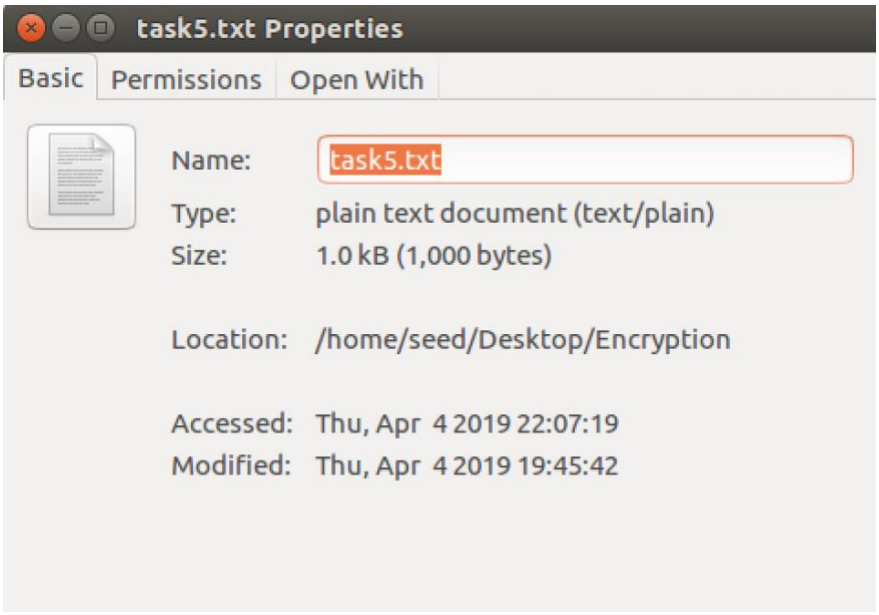
Since CFB and OFB do not have padding it does not get affected. In the above screenshot of task416cfbnopad.txt and task416ofbnopad.txt we notice that there is no padding taking place.

TASK 5: Error Propagation – Corrupted Cipher Text

We create a text file which is at least 1000 bytes long as shown below

```
[04/04/19]\Shenava@VM:~/.../Encryption$ gedit task5.txt
```



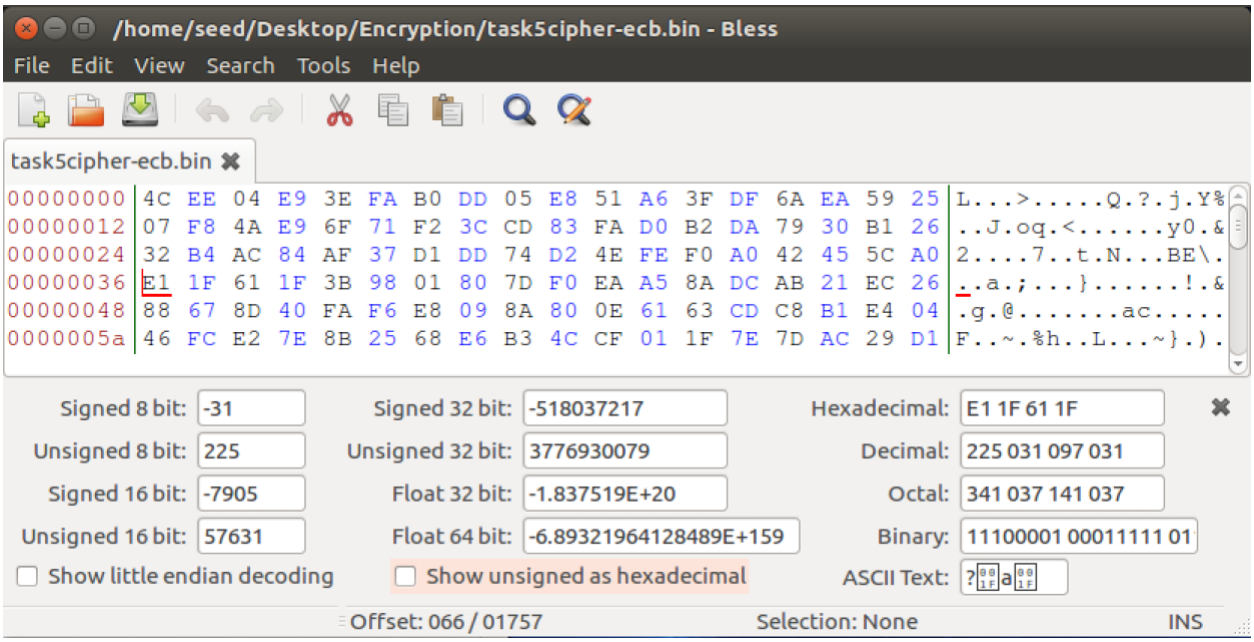


1. ECB encryption model

Encrypted file

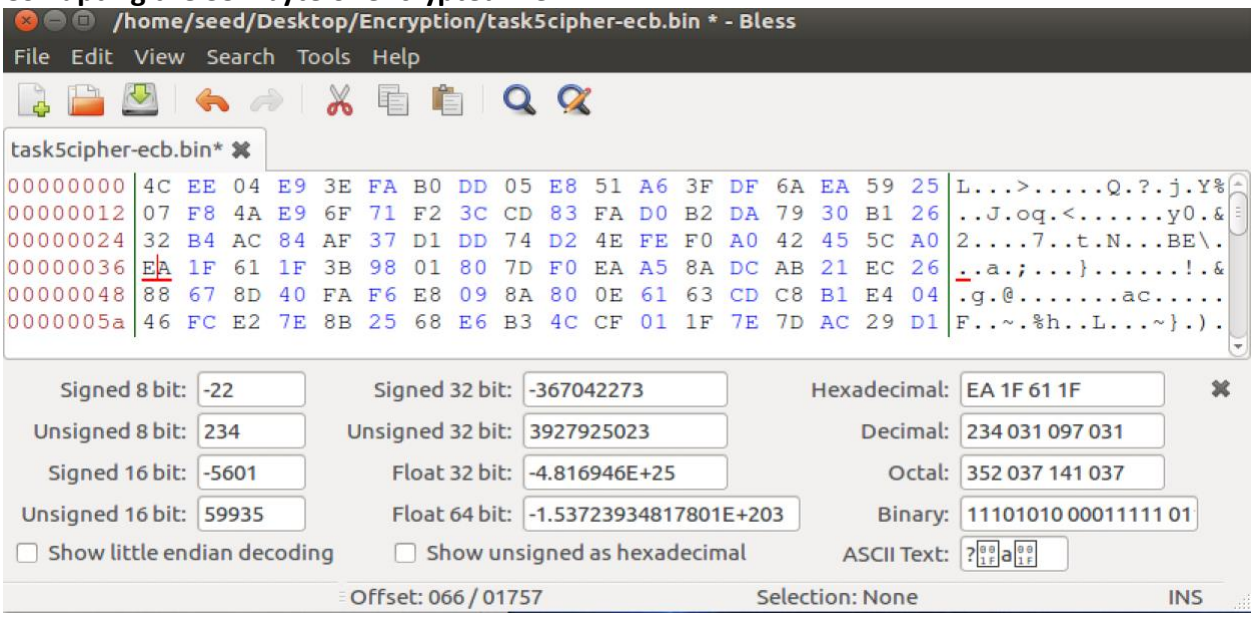
```
[04/04/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ecb -e -in task5.tx
t -out task5cipher-ecb.bin -K 00112233445566778889aabbccddeeff
[04/04/19]\Shenava@VM:~/.../Encryption$
```

-aes-128-ecb encryption mode on task5.txt



The original file in hex after encrypting

Corrupting the 55th byte of encrypted file



We corrupt the 55th byte of the cipher text by changing the value to some other. Since the 30th byte is part of the 4th block, 4th block will be corrupted after the decryption.

Decrypted file

```
[04/05/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ecb -d -in task5cip
her-ecb.bin -out task5-ecb.txt -K 00112233445566778889aabbccddeeff
[04/05/19]\Shenava@VM:~/.../Encryption$ cat task5-ecb.txt
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula##0S0-0FHe0V0Ersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurit
```

We decrypt the encrypted file that we corrupted. We notice the corrupted cipher text in the above screenshot.

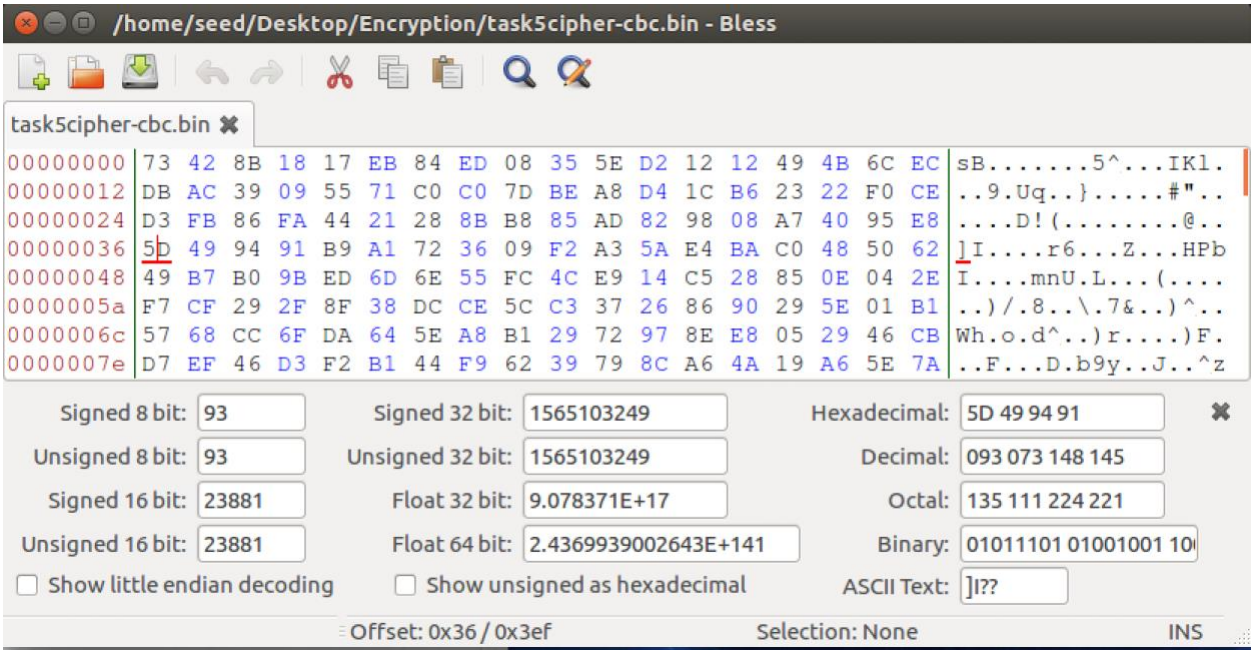
Since ECB mode happens block by block independently, that is why corruption of the cipher text of that particular block will yield in the corruption of the decrypted plain text of that particular block.

2. CBC encryption mode

Encrypted file

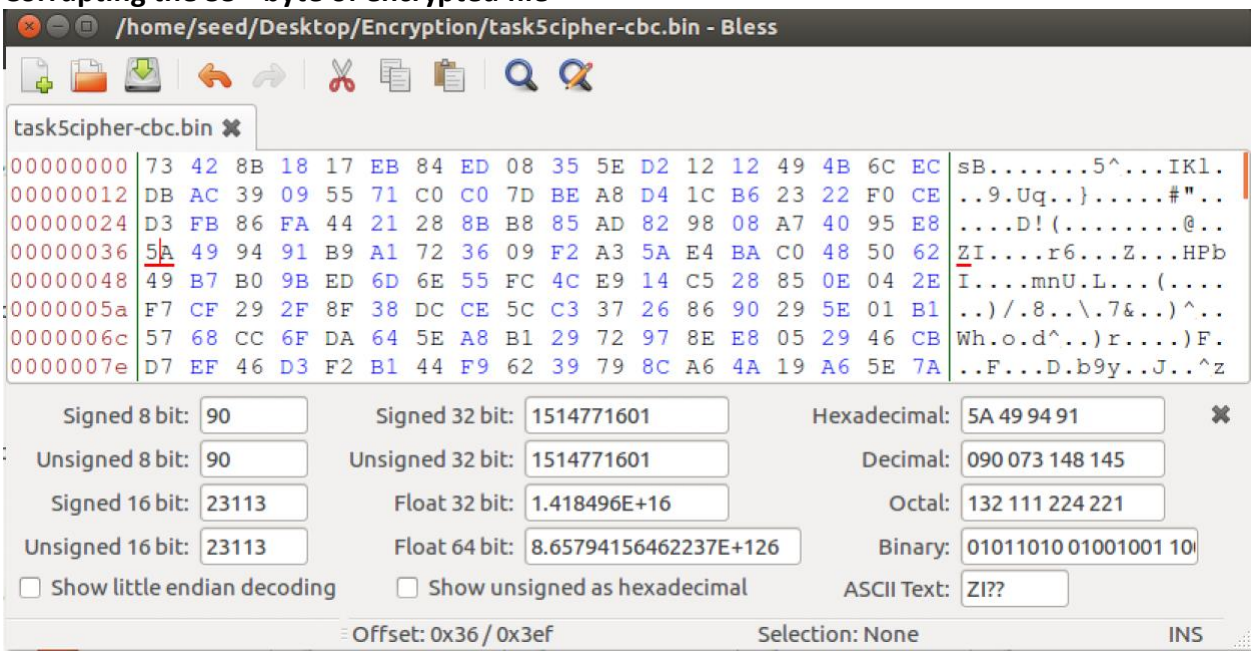
```
[04/05/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -e -in task5.tx
t -out task5cipher-cbc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060
708
[04/05/19]\Shenava@VM:~/.../Encryption$
```

-aes-128-cbc encryption mode on task5.txt



The original file in hex after encrypting

Corrupting the 55th byte of encrypted file



We corrupt the 55th byte of the cipher text by changing the value to some other. Since the 30th byte is part of the 4th block, 4th block will be corrupted along with some part of the 5th block which will be the 14th byte of the 5th block after the decryption.

Decrypted file

```
[04/05/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -d -in task5cip
her-cbc.bin -out task5-cbc.txt -K 00112233445566778889aabbccddeeff -iv 010203040
5060708
[04/05/19]\Shenava@VM:~/.../Encryption$ cat task5-cbc.txt
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula[illegible]D=A[illegible]J\
                      r[illegible]rsecurnty
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurit
```

We decrypt the encrypted file that we corrupted. We notice the corrupted cipher text in the above screenshot.

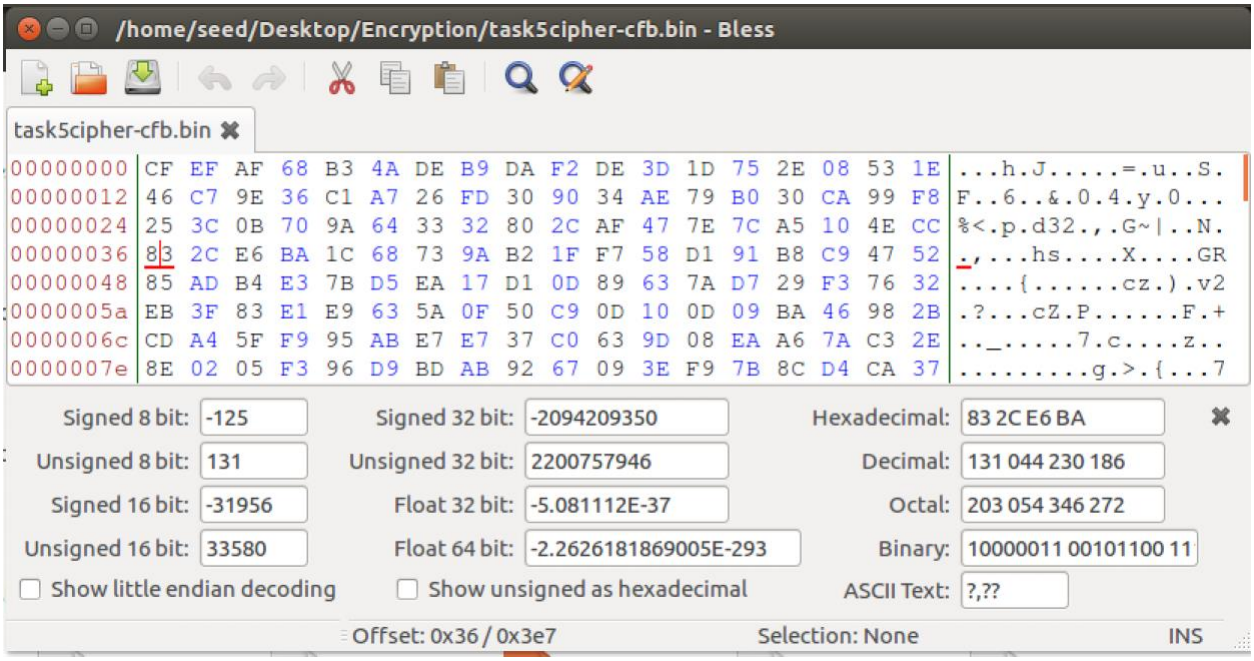
In CBC mode, cipher text is given as input to the next block for encryption, that is why corruption of the cipher text of that particular block will yield in the corruption of the decrypted plain text of that particular block along with the corresponding byte of the next block as well.

3. CFB encryption mode

Encrypted file

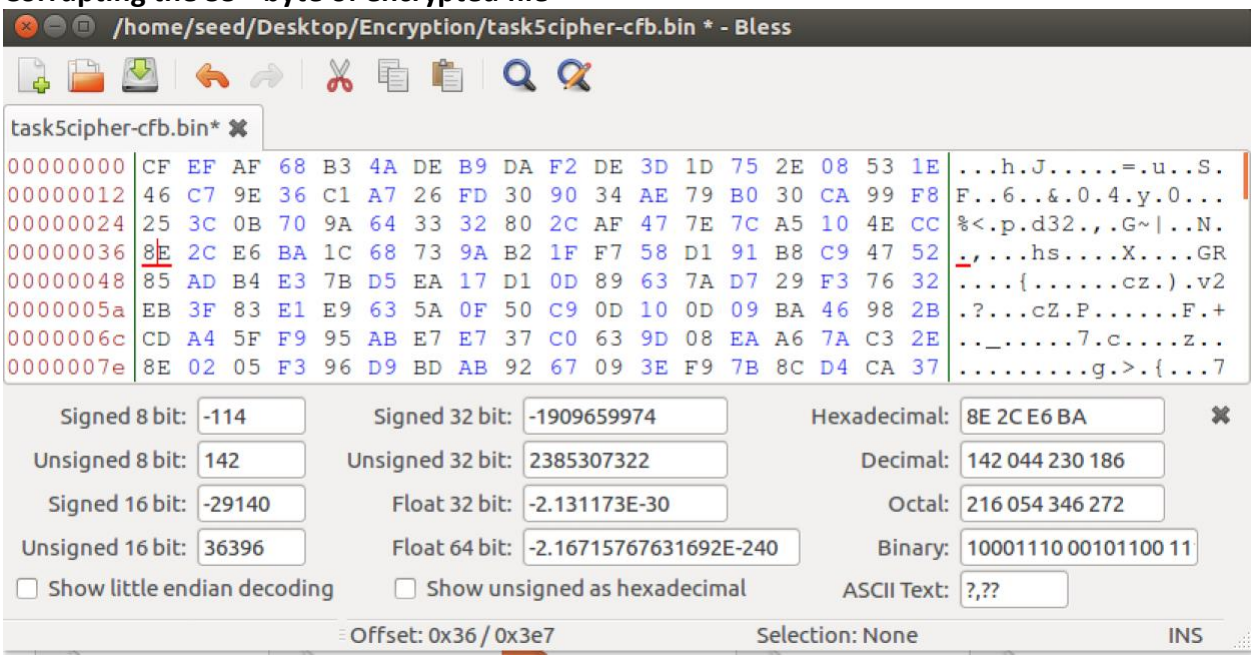
```
[04/05/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -e -in task5.tx
t -out task5cipher-cfb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060
708
[04/05/19]\Shenava@VM:~/.../Encryption$
```

-aes-128-cfb encryption mode on task5.txt



The original file in hex after encrypting

Corrupting the 55th byte of encrypted file



We corrupt the 55th byte of the cipher text by changing the value to some other. Since the 55th byte is part of the 4th block, 4th block will not be corrupted after the decryption, only the 55th byte will get corrupted. The following bloc, that is, the 5th block gets corrupted in CFB mode.

Decrypted file

```
[04/05/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -d -in task5cip
her-cfb.bin -out task5-cfb.txt -K 00112233445566778889aabbccddeeff -iv 010203040
5060708
[04/05/19]\Shenava@VM:~/.../Encryption$ cat task5-cfb.txt
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here ioing Cybe??Q??Y????u?dhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurit
```

We decrypt the encrypted file that we corrupted. We notice the corrupted cipher text in the above screenshot.

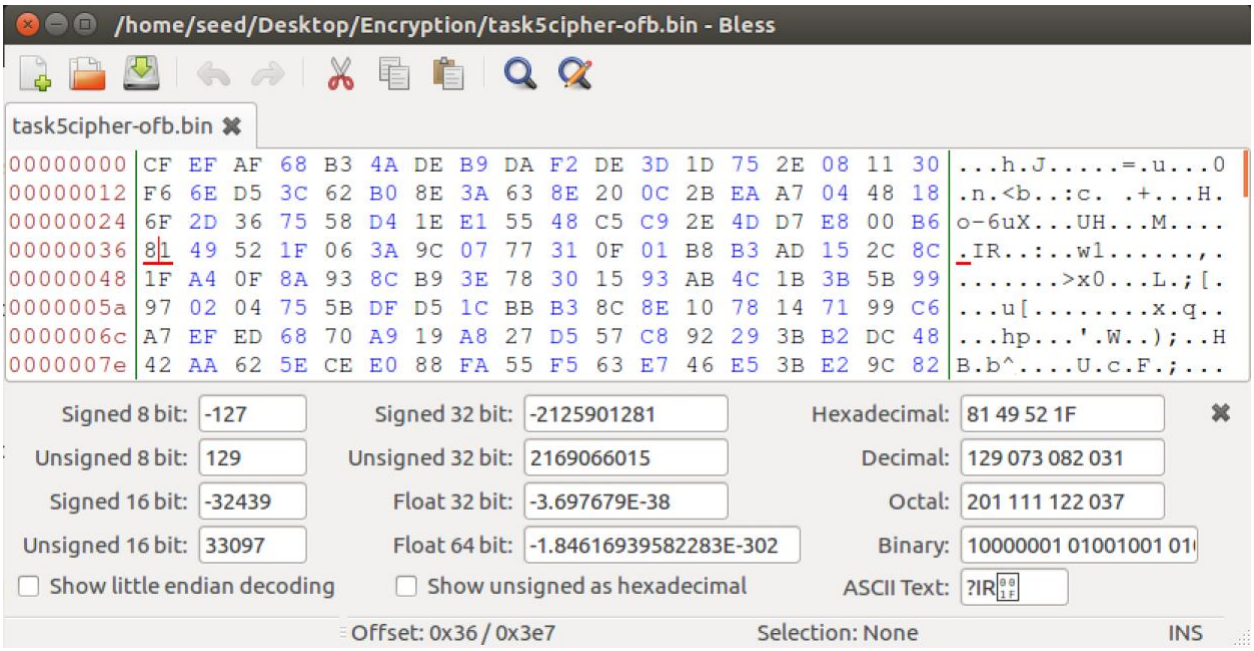
In CFB mode, cipher text is given as input to the next block for encryption, that is why corruption of the cipher text of that particular block will yield in the corruption of the decrypted plain text of the next block and the current byte of that current block.

4. OFB encryption mode

Encrypted file

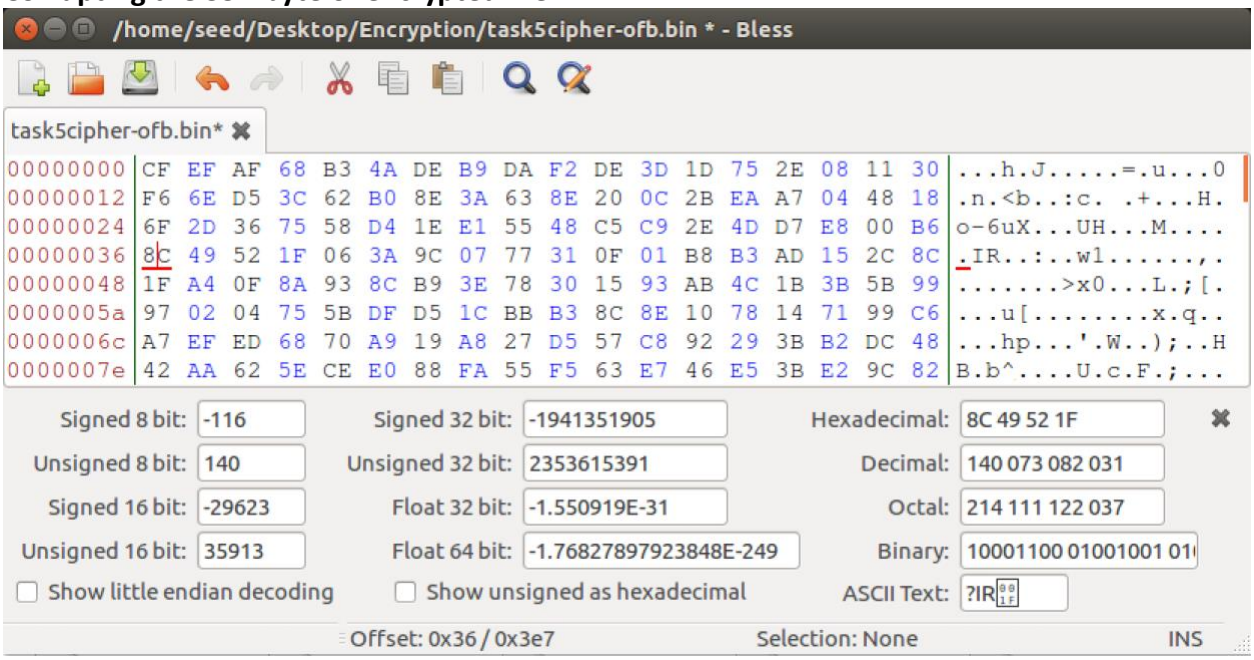
```
[04/05/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -e -in task5.tx
t -out task5cipher-ofb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060
708
[04/05/19]\Shenava@VM:~/.../Encryption$
```

-aes-128-ofb encryption mode on task5.txt



The original file in hex after encrypting

Corrupting the 55th byte of encrypted file



We corrupt the 55th byte of the cipher text by changing the value to some other. Since the 55th byte is part of the 4th block, 4th block will not be corrupted after the decryption, only the 55th byte will get corrupted.

Decrypted file

```
[04/05/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-ofb -d -in task5cip
her-ofb.bin -out task5-ofb.txt -K 00112233445566778889aabbccddeeff -iv 010203040
5060708
[04/05/19]\Shenava@VM:~/.../Encryption$ cat task5-ofb.txt
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here ioing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurity
Hi Mrudhula here doing Cybersecurit
```

We decrypt the encrypted file that we corrupted. We notice the corrupted cipher text in the above screenshot.

In OFB mode, cipher text is not given as input to the next block for encryption, that is why corruption of the cipher text of that particular block will not yield in the corruption of the decrypted plain text of the next block. Only the current byte of the current block is corrupted in this mode.

TASK 6: Initial Vector (IV)

TASK 6.1:

```
[04/05/19]\Shenava@VM:~/.../Encryption$ cat task6.txt
Hi Mrudhula here doing Cybersecurity
```

Original text file.

Different IV's

```
[04/05/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -e -in task6.tx
t -out task6encrypted.txt -K 00112233445566778889aabbccddeeff -iv 01020304050607
08
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd task6encrypted.txt
00000000: cfef af68 b34a deb9 daf2 de3d 1d75 2e08  ...h.J.....=.u..
00000010: 531e 46c7 9e36 c1a7 26fd 3090 34ae 79b0  S.F..6..&.0.4.y.
00000020: 30ca 99f8 25                             0...%
[04/05/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -e -in task6.tx
t -out task6encrypted1.txt -K 00112233445566778889aabbccddeeff -iv 0807060504030
201
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd task6encrypted1.txt
00000000: 7e60 9f0d 06c9 6c0a 1352 0da7 701d 26e4  ~`....l..R..p.&.
00000010: 8e47 d3e2 9f87 a6c2 65c3 5ac0 7448 39ae  .G......e.Z.tH9.
00000020: 6133 0778 ae                             a3.x.
[04/05/19]\Shenava@VM:~/.../Encryption$
```

We notice that when the text is encrypted using different IV's we tend to get different encrypted values and they are not same.

Same IV's

```
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd task6encryptsame.txt
00000000: cfef af68 b34a deb9 daf2 de3d 1d75 2e08  ...h.J.....=.u..
00000010: 531e 46c7 9e36 c1a7 26fd 3090 34ae 79b0  S.F..6..&.0.4.y.
00000020: 30ca 99f8 25                             0...%
[04/05/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cfb -e -in task6.tx
t -out task6encryptsame1.txt -K 00112233445566778889aabbccddeeff -iv 01020304050
60708
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd task6encryptsame1.txt
00000000: cfef af68 b34a deb9 daf2 de3d 1d75 2e08  ...h.J.....=.u..
00000010: 531e 46c7 9e36 c1a7 26fd 3090 34ae 79b0  S.F..6..&.0.4.y.
00000020: 30ca 99f8 25                             0...%
[04/05/19]\Shenava@VM:~/.../Encryption$
```

We notice that when the text is encrypted using same IV's we get same encrypted result.

TASK 6.2:

```
[04/05/19]\Shenava@VM:~/.../Encryption$ echo -n "This is a known message!" > p1
[04/05/19]\Shenava@VM:~/.../Encryption$ echo -n "a469b1c502c1cab966965e50425438e
1bb1b5f9037a4c159" > c1
[04/05/19]\Shenava@VM:~/.../Encryption$ echo -n "bf73bcd3509299d566c35b5d450337e
1bb175f903fafc159" > c2
[04/05/19]\Shenava@VM:~/.../Encryption$
```

We first create the files.

```
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd -p p1
54686973206973732061206b6e6f776e206d65737361676521
[04/05/19]\Shenava@VM:~/.../Encryption$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print hex(0x54686973206973732061206b6e6f776e206d65737361676521 ^ 0xa469b1c502c
1cab966965e50425438e1bb1b5f9037a4c159)
0xf001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478L
```

We get the hex value of p1. Then we XOR hex of p1 and c1 to achieve the IV.

```
>>> print hex(0xf001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478L ^ 0xbf73bcd350
9299d566c35b5d450337e1bb175f903fafc159)
0x4f726465723a204c61756e63682061206d697373696c6521L
```

Once we get the IV, we will next XOR IV and c2.

```
[04/05/19]\Shenava@VM:~/.../Encryption$ echo -n "4f726465723a204c61756e636820612
06d697373696c6521L" | xxd -r -p
Order: Launch a missile![04/05/19]\Shenava@VM:~/.../Encryption$
```

We then see the readable form of generated hex and see that we get a plaintext even if IV's are different

TASK 6.3:

```
[04/05/19]\Shenava@VM:~/.../Encryption$ echo -n "Yes" > P1.txt
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd -p P1.txt
596573
[04/05/19]\Shenava@VM:~/.../Encryption$ echo -n "5965730d0d0d0d0d0d0d0d0d0d0d0d0
d" > P1hex.txt
[04/05/19]\Shenava@VM:~/.../Encryption$ echo -n "3132333435363738393031323334353
6" > IV1hex.txt
[04/05/19]\Shenava@VM:~/.../Encryption$ echo -n "3132333435363738393031323334353
7" > IV2hex.txt
[04/05/19]\Shenava@VM:~/.../Encryption$
```

We first create the files. We generate hex vale of Yes and we see its of 3 bytes and do padding of 13 bytes.

```
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd -r -p P1hex.txt > P1hex.bin
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd -r -p IV1hex.txt > IV1hex.bin
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd -r -p IV2hex.txt > IV2hex.bin
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd -p P1hex.bin
5965730d0d0d0d0d0d0d0d0d0d0d0d0d
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd -p IV1hex.bin
31323334353637383930313233343536
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd -p IV2hex.bin
31323334353637383930313233343537
[04/05/19]\Shenava@VM:~/.../Encryption$
```

We convert the text file to hex file and save in bin file.

```
[04/05/19]\Shenava@VM:~/.../Encryption$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print hex(0x5965730d0d0d0d0d0d0d0d0d0d0d0d0d ^ 0x3132333435363738393031323334
43536 ^ 0x31323334353637383930313233343537)
0x5965730d0d0d0d0d0d0d0d0d0d0d0d0cL
>>>
```

We then XOR P1 with IV1

```
[04/05/19]\Shenava@VM:~/.../Encryption$ echo -n "5965730d0d0d0d0d0d0d0d0d0d0d0d0
cL" > P2.txt
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd -r -p P2.txt > P2.bin
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd -p P2.bin
5965730d0d0d0d0d0d0d0d0d0d0d0d0c
[04/05/19]\Shenava@VM:~/.../Encryption$
```

Then we save the result to a file.

```
[04/05/19]\Shenava@VM:~/.../Encryption$ openssl enc -aes-128-cbc -e -in P2.bin -
out ans.bin -K 00112233445566778899aabbccddeeff -iv 3132333435363738393031323334
3537
[04/05/19]\Shenava@VM:~/.../Encryption$ xxd ans.bin
00000000: bef6 5565 572c cee2 a9f9 5531 54ed 9498  ..UeW,....U1T...
00000010: 3402 de3f 0dd1 6ce7 89e5 4757 79ac a405  4..?..l...GWy...
[04/05/19]\Shenava@VM:~/.../Encryption$
```

We will use openssl to check and we encrypt the result with given key and IV2 and we see that the answer matches the cipher text given. Hence, we consider the answer is yes.

TASK 7: Programming using the Crypto Library

```
#!/usr/bin/python3
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
#Assigning Hex values of IV and Cipher Text
ivhex='aabbccddeeff00998877665544332211'
c1hex="764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2"
#Converting Hex to Binary
iv=bytes.fromhex(ivhex)
c1=bytes.fromhex(c1hex)
#Assigning plaintext in binary form.
p1=b"This is a top secret."
#Creating a empty key
key=""
# Reading the english wordlist into a list
fp=open("words.txt","r")
wordlist_ip=fp.readlines()
fp.close()
# removing the whitespaces and \n from the words and storing the words in a new dictionary
wordlist=[]
for word in wordlist_ip:
        word=word.replace("\n","")
        word=word.strip()
        wordlist.append(word)
# applying bruteforce on the plaintext with keys from the english wordlist for word in wordlist:
        # padding the key with pound size to make it 128 bits
        # # is 0x23 in hex
        if len(word) <= 16:
                n=16-len(word)
                #Creating random keys using the words from word list and padding (if necessary)
                key_bin=word.encode("ascii")+b"\x23"*n
# Initializing AES Cipher
        cipher=AES.new(key_bin,AES.MODE_CBC,iv)
# encrypt the plaintext with padding with block size set as 16 bytes
        c1_new = cipher.encrypt(pad(p1,16))
        if c1 == c1_new:
                key=word
                break
print("The key is :",key)
```

The code.

```
[04/06/19]\Shenava@VM:~/.../Encryption$ sudo python3 mycod
e.py
The key is : Syracuse
[04/06/19]\Shenava@VM:~/.../Encryption$
```

We are already given with the plaintext, cipher text and key. We compare to see if we find the matching key in the given wordlist.

We compile using gcc and also include -lcrypto flag since our code needs crypto library.

OpenSSL provides an API called EVP, which is a high-level interface to cryptographic functions. Although OpenSSL also has direct interfaces for each individual encryption algorithm, the EVP

library provides a common interface for various encryption algorithms. To ask EVP to use a specific algorithm, we simply need to pass our choice to the EVP interface.