CSE510 Lab 1

Name: Shreyas Ramesh
Email: ramesh3@buffalo.edu
UBID: ramesh3
UB Number: 50540974

Before You Start:

Please write a detailed lab report, with **screenshots**, to describe what you have **done** and what you have **observed**. You also need to provide **explanation** to the observations that you noticed. Please also show the important **code snippets** followed by explanation. Simply attaching code without any explanation will NOT receive credits.
After you finish, export this report as a **PDF** file and submit it on UBLearns.

Academic Integrity Statement:
I, _Shreyas Ramesh____, have read and understood the course academic integrity policy.
(Your report will not be graded without filling your name in the above AI statement)

Task 1: Frequency Analysis

I ran the freq.py python script in the lab setup file and checked the frequency of the n-grams of the cipher text. Running it gives:

```
[02/13/24]seed@VM:~/.../Files$ ./freq.py
----------------------------------
1-gram (top 20):
n: 488
y: 373
v: 348
x: 291
u: 280
q: 276
m: 264
h: 235
t: 183
i: 166
p: 156
a: 116
c: 104
z: 95
l: 90
g: 83
b: 83
r: 82
e: 76
d: 59
```

```
-----------------------------------
2-gram (top 20):
yt: 115
tn: 89
mu: 74
nh: 58
vh: 57
hn: 57
vu: 56
nq: 53
xu: 52
up: 46
xh: 45
yn: 44
np: 44
vy: 44
nu: 42
qy: 39
vq: 33
vi: 32
gn: 32
av: 31
```

```
-----------------------------------
3-gram (top 20):
ytn: 78
vup: 30
mur: 20
ynh: 18
xzy: 16
mxu: 14
gnq: 14
ytv: 13
nqy: 13
vii: 13
bxh: 13
lvq: 12
nuy: 12
vyn: 12
uvy: 11
lmu: 11
nvh: 11
cmu: 11
tmq: 10
vhp: 10
[02/13/24]seed@VM:~/.../Files$
```

After looking at the n-gram information, I tried to play around with different combinations and kept changing different cipher text variations with different alphabets using the ‘tr’ command. I used the resource https://en.wikipedia.org/wiki/Trigram and saw that the words with most frequency in the English vernacular is ‘and’ and ‘the’. So, I swapped the words ‘ytn’ and ‘vup’ with ‘the’ and ‘and’.

```
[02/13/24]seed@VM:~/.../Files$ tr 'ytnvup' 'THEAND' <ciphertext.txt > out.txt
[02/13/24]seed@VM:~/.../Files$ ls
ciphertext.txt  freq.py  out.txt  pic_original.bmp  sample_code.py  words.txt
[02/13/24]seed@VM:~/.../Files$ cat out.txt
THE xqaAhq TzhN  xN qzNDAd lHmaH qEEcq AgxzT hmrHT AbTEh THmq ixNr qThANrE
AlAhDq Thme THE gArrEh bEEiq imsE A NxNArENAhmAN Txx

THE AlAhDq hAaE lAq gxxsENDED gd THE DEcmqE xb HAhfEd lEmNqTEmN AT mTq xzTqET
AND THE AeeAhENT mceixqmxN xb Hmq bmic axceANd AT THE END AND mT lAq qHAeED gd
THE EcEhrENaE xb cETxx TmcEq ze giAasrxlN eximTmaq AhcaANDd AaTmfmqc AND
A NATmxNAi axNfEhqATmxN Aq ghmEb AND cAD Aq A bEfEh DhEAc AgxzT lHETHEh THEhE
xzrHT Tx gE A ehEqmDENT lmNbhEd THE qEAqxN DmDNT ozqT qEEc EkThA ixNr mT lAq
EkThA ixNr gEaAzqE THE xqaAhq lEhE cxfED Tx THE bmhqT lEEsEND mN cAhaH Tx
AfxmD axNbimaTmNr lmTH THE aixqmNr aEhEcxNd xb THE lmNTEh xidcemaq THANsq
edExNraHANr

xNE gmr jzEqTmxN qzhhxzNDmNr THmq dEAhq AaADEcd AlAhDq mq Hxl xh mb THE
aEhEcxNd lmii ADDhEqq cETxx EqeEamAiid AbTEh THE rxiDEN rixgEq lHmaH gEaAcE
A ozgmiANT axcmNrxzT eAhTd bxh TmcEq ze THE cxfEcENT qeEAhHEADED gd
exlEhbzi HxiidlxxD lxcEN lHx HEieED hAmqE cmiimxNq xb DxiiAhq Tx bmrHT qEkzAi
HAhAqqcENT AhxzND THE axzNThd
```

The ciphertext started making more sense now and I kept making adjustments to the words I swapped.

```
[02/13/24]seed@VM:~/.../Files$ tr 'ytnvupxcmfglh' 'THEANDOMIVBWR' <ciphertext.txt > out.txt
[02/13/24]seed@VM:~/.../Files$ cat out.txt
THE OqaARq TzRN  ON qzNDAd WHIaH qEEMq ABOzT RIrHT AbTER THIq iONr qTRANrE
AWARDq TRIe THE BArrER bEEiq iIsE A NONArENARIAN TOO

THE AWARDq RAaE WAq BOOsENDED Bd THE DEMIqE Ob HARVEd WEINqTEIN AT ITq OzTqET
AND THE AeeARENT IMeiOqION Ob HIq bIiM aOMeANd AT THE END AND IT WAq qHAeED Bd
THE EMERrENaE Ob METOO TIMEq ze BiAasrOWN eOiITIaq ARMaANDd AaTIVIqM AND
A NATIONAi aONVERqATION Aq BRIEb AND MAD Aq A bEVER DREAM ABOzT WHETHER THERE
OzrHT TO BE A eREqIDENT WINbREd THE qEAqON DIDNT ozqT qEEM EkTRA iONr IT WAq
EkTRA iONr BEaAzqE THE OqaARq WERE MOVED TO THE bIRqT WEEsEND IN MARaH TO
AVOID aONbiIaTINr WITH THE aiOqINr aEREMONd Ob THE WINTER OidMeIaq THANsq
edEONraHANr

ONE BIr jzEqTION qzRROzNDINr THIq dEARq AaADEMd AWARDq Iq HOW OR Ib THE
aEREMONd WIii ADDREqq METOO EqeEaIAiid AbTER THE rOiDEN riOBEq WHIaH BEaAME
A ozBIiANT aOMINrOzT eARTd bOR TIMEq ze THE MOVEMENT qeEARHEADED Bd
eOWERbzi HOiidWOOD WOMEN WHO HEieED RAIqE MIiiIONq Ob DOiiARq TO bIrHT qEkzAi
HARAqqMENT AROzND THE aOzNTRd

qIrNAiINr THEIR qzeeORT rOiDEN riOBEq ATTENDEEq qWATHED THEMqEiVEq IN BiAas
qeORTED iAeEi eINq AND qOzNDED Obb ABOzT qEkIqT eOWER IMBAiANaEq bROM THE RED
aAReET AND THE qTArE ON THE AIR E WAq aAiiED OzT ABOzT eAd INEjzITd AbTER
ITq bORMER ANaHOR aATT qADiER jzIT ONaE qHE iEARNED THAT qHE WAq MAsINr bAR
iEqq THAN A MAiE aOHOqT AND DzRINr THE aEREMONd NATAiIE eORTMAN TOOs A BizNT
AND qATIqbdINr DIr AT THE AiiMAiE ROqTER Ob NOMINATED DIREaTORq HOW aOziD
THAT BE TOeeED
```

After lots more changes, I was able to crack the ciphertext with the key: ‘ytnvupxcmfglhaqrbiedokswj’.

```
[02/13/24]seed@VM:~/.../Files$ tr 'ytnvupxcmfglhaqzrbiedokswj' 'THEANDOMIVBWRCSUGFLPYJXKZQ' <ciphertext.txt > out.txt
[02/13/24]seed@VM:~/.../Files$ cat out.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
```

Task 2: Encryption using Different Ciphers and Modes

In this task, we used multiple ciphers and modes for encryption and decryption of the text taken into consideration. We use ‘openssl’ library for encryption and decryption of a given file. I have used rc2-cbc, blowfish and aes-128-cfb.

1) RC2-CBC:

I used the openssl library to encrypt and decrypt the plaintexts. The command I used for encryption is:

Openssl enc -rc2-cbc -e -in demo.txt -out -cipher1.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

I then printed out the contents of the encrypted file to observe its contents using the cat command. I then decrypted the encrypted ciphertext using the above command with few small modifications such as changing the encrypting flag (-e) to decryption (-d) and changed the input and output files.

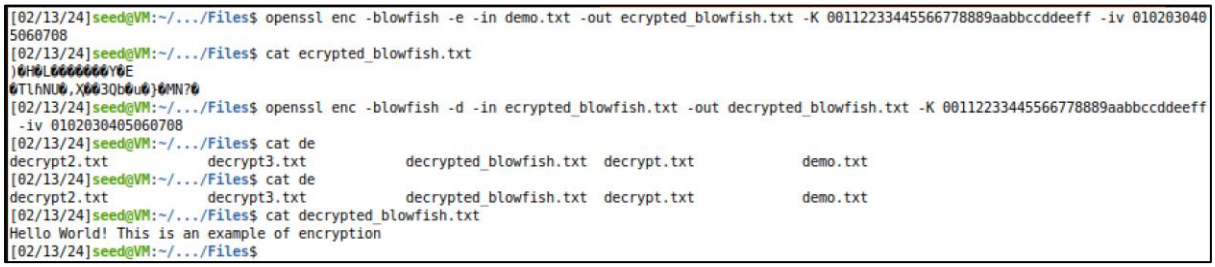
```
[02/13/24]seed@VM:~/.../Files$ echo "Hello World! This is an example of encryption" > demo.txt
[02/13/24]seed@VM:~/.../Files$ cat demo.txt
Hello World! This is an example of encryption
[02/13/24]seed@VM:~/.../Files$ openssl enc -ciphertype rc2-cbc -e -in demo.txt -out cipher1.txt
enc: Unrecognized flag ciphertype
enc: Use -help for summary.
[02/13/24]seed@VM:~/.../Files$ openssl enc -rc2-cbc -e -in demo.txt -out cipher1.txt
enter rc2-cbc encryption password:
Verifying - enter rc2-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[02/13/24]seed@VM:~/.../Files$ ls
cipher1.txt  ciphertext.txt  demo.txt  freq.py  out.txt  pic_original.bmp  sample_code.py  words.txt
[02/13/24]seed@VM:~/.../Files$ cat cipher1.txt
Salted__d>&[illegible]I[illegible]fw-[illegible]Tw[illegible]$[illegible]N[illegible]r[illegible]qs[illegible]y [illegible]jQi[illegible]*[illegible][02/13/24]seed@VM:~/.../Files$ openssl enc -rc2-cbc -e -in demo.txt -out cipher1.txt -K 001
12233445566778889aabbccddeeff -iv 0102030405060708
[02/13/24]seed@VM:~/.../Files$ cat cipher1.txt
V[illegible]q[illegible]S[illegible]s#[illegible]
          [illegible]L![illegible]p[illegible]n
                    [illegible]7[illegible]s[illegible]iXr[illegible]F[02/13/24]seed@VM:~/.../Files$ openssl enc -rc2-cbc -e -in demo.txt -out cipher1.txt -K 00112233445566778889aa
cat cipher1.txt ^C
[02/13/24]seed@VM:~/.../Files$ openssl enc -rc2-cbc -d -in cipher1.txt -out decrypt.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[02/13/24]seed@VM:~/.../Files$ cat decrypt.txt
Hello World! This is an example of encryption
[02/13/24]seed@VM:~/.../Files$
```

2) Blowfish:

The second encryption cipher I used was Blowfish. Its command syntax is the same as above and I only had to change the cipher flag. The command I used is:

Openssl enc -blowfish -e -in demo.txt -out encrypted_blowfish.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

I then repeated the process of printing out the file contents to observe the encrypted file. Later, I decrypted the file using the '-d' flag.

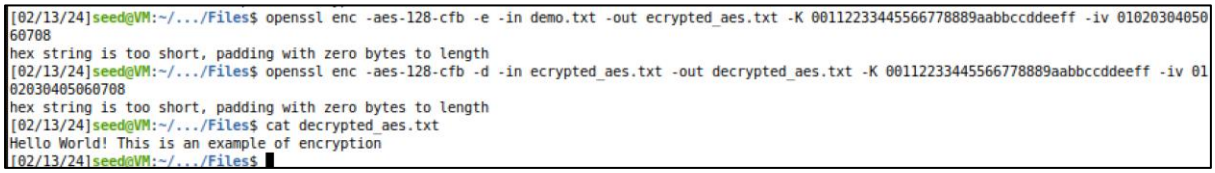


3) AES-128-cfb:

The second encryption cipher I used was AES. Its command syntax is the same as above and I only had to change the cipher flag. The command I used is:

Openssl enc -aes-128-cfb -e -in demo.txt -out encrypted_aes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

I then repeated the process of printing out the file contents to observe the encrypted file. Later, I decrypted the file using the '-d' flag.

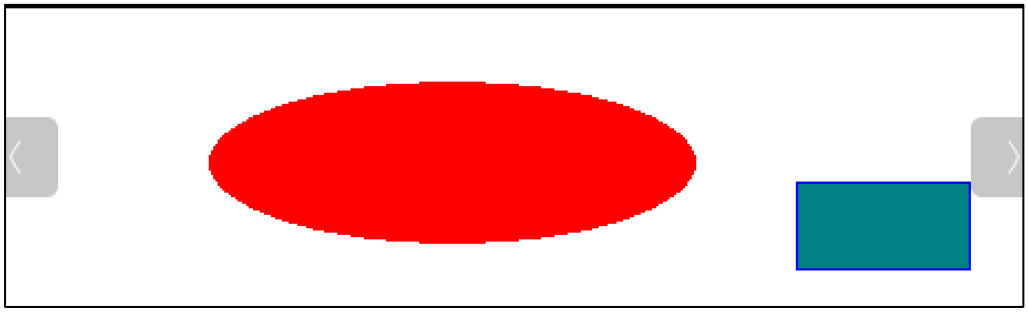


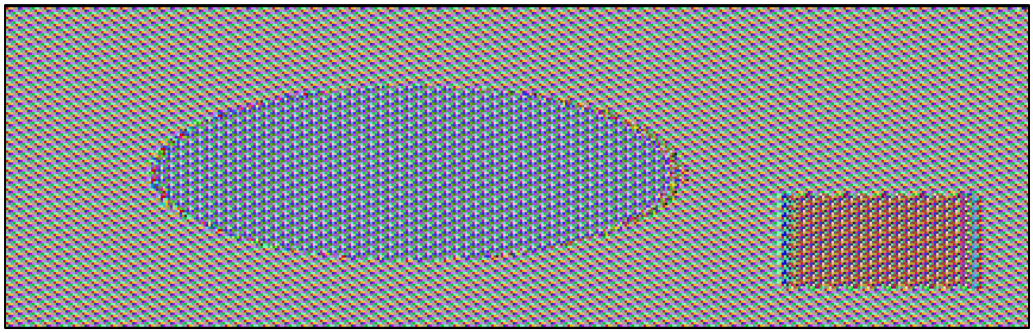
Task 3: Encryption Mode – ECB vs. CBC

I have encrypted the original_pic.bmp with both modes ECB and CBC as shown in the below image.

```
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out encrypted_pic.bmp -K 00112233445566778889aabbccddeeff -iv 01
02030405060708
warning: iv not used by this cipher
[02/13/24]seed@VM:~/.../Files$ head -c 54 pic_original.bmp  > header
[02/13/24]seed@VM:~/.../Files$ tail -c +55 encrypted_pic.bmp > body
[02/13/24]seed@VM:~/.../Files$ cat header body > new_ecb.bmp
[02/13/24]seed@VM:~/.../Files$ eog new_ecb.bmp
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in pic_original.bmp -out encrypted_pic1.bmp -K 00112233445566778889aabbccddeeff -iv 0
102030405060708
hex string is too short, padding with zero bytes to length
[02/13/24]seed@VM:~/.../Files$ head -c 54 pic_original.bmp  > header
[02/13/24]seed@VM:~/.../Files$ tail -c +55 encrypted_pic1.bmp > body
[02/13/24]seed@VM:~/.../Files$ cat header body > new_cbc.bmp
[02/13/24]seed@VM:~/.../Files$ eog new_cbc.bmp
[02/13/24]seed@VM:~/.../Files$
```

And the comparison of the resultant outputs is presented below :
First pic : pic_original.bmp
Second pic : new_ecb.bmp
Third pic : new_cbc.bmp





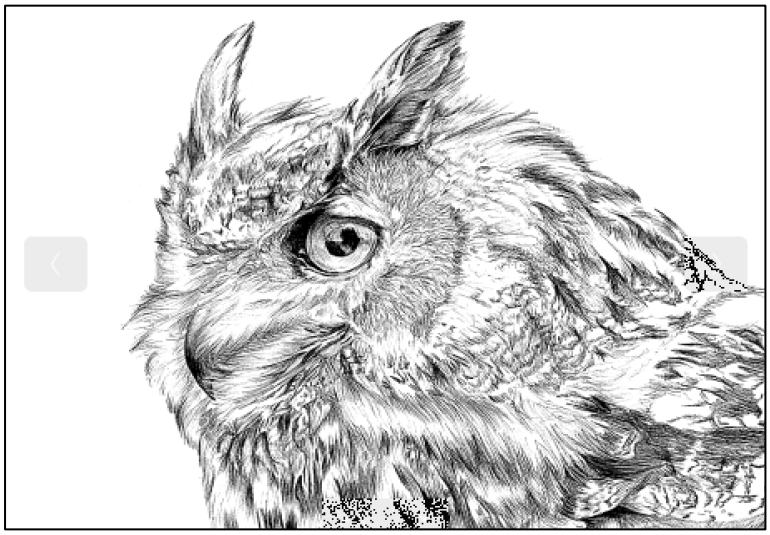
We can see the cbc image is completely converted to noise whereas ecb image we can interpret some data related to pic_original.bmp.

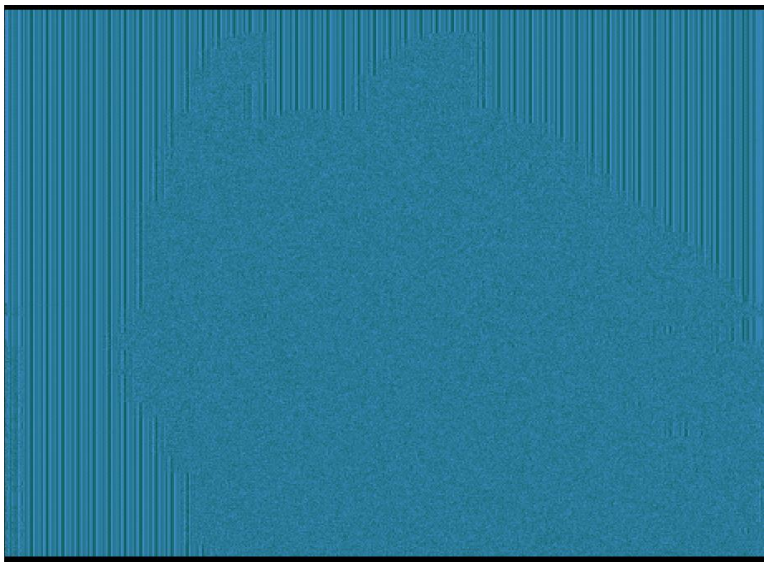
Own image:

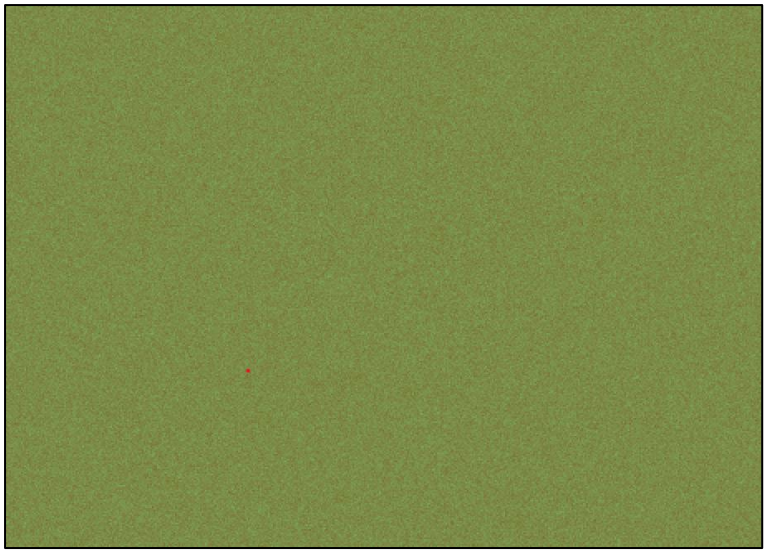
I downloaded a bmp image from the internet and encrypted that using the two modes using the command below.

```
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in example2.bmp -out encrypted2_pic.bmp -K 00112233445566778889aabbccddeeff -iv 01020
30405060708
warning: iv not used by this cipher
[02/13/24]seed@VM:~/.../Files$ head -c 54 example2.bmp  > header
[02/13/24]seed@VM:~/.../Files$ tail -c +55 encrypted2_pic.bmp > body
[02/13/24]seed@VM:~/.../Files$ cat header body > example2_ecb.bmp
[02/13/24]seed@VM:~/.../Files$ eog example2.ecb.bmp
[02/13/24]seed@VM:~/.../Files$ eog example2_ecb.bmp
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in example2.bmp -out encrypted3_pic.bmp -K 00112233445566778889aabbccddeeff -iv 01020
30405060708
hex string is too short, padding with zero bytes to length
[02/13/24]seed@VM:~/.../Files$ head -c 54 example2.bmp  > header
[02/13/24]seed@VM:~/.../Files$ tail -c +55 encrypted3_pic.bmp > body
[02/13/24]seed@VM:~/.../Files$ cat header body > example2_cbc.bmp
[02/13/24]seed@VM:~/.../Files$ eog example2_cbc.bmp
```

And the comparison of the resultant outputs is presented below :
First pic: example2.bmp
Second pic (ecb): encrypted2_pic.bmp
Third pic (cbc): encrypted2_pic.bmp



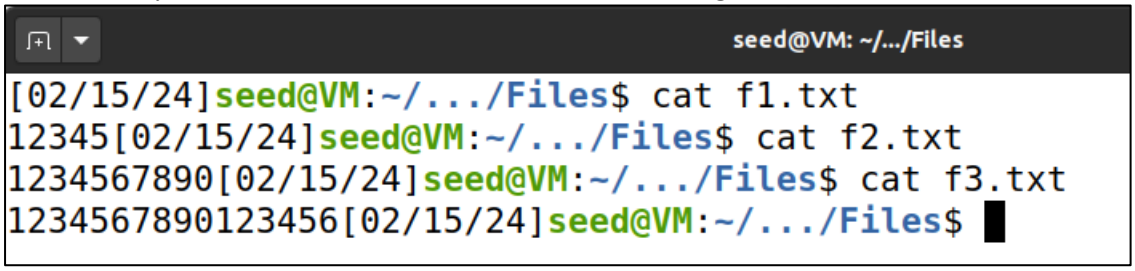




From the output pictures we can infer with ecb we got the outline of the deer. But in cbc we are getting noise. In conclusion, we can see from the above image that the ecb encryption is not that secure as it encrypts block by block whereas cbc encrypts in dependence with previous block.

Task 4: Padding

I have initially created three files as shown in the below image.



I have encrypted all three files using cbc, ecb, cfb and ofb modes and observed the padding in each of those modes.

1) OFB:

```
[02/15/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in f1.txt -out f1_ofb.txt -K 00112233445566778889aabbccddeeff -iv 010203040506070
8
hex string is too short, padding with zero bytes to length
[02/15/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in f1_ofb.txt -out f1_ofb_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
hex string is too short, padding with zero bytes to length
[02/15/24]seed@VM:~/.../Files$ cat f1_ofb_dec.txt
12345[02/15/24]seed@VM:~/.../Files$ hexdump -C f1_ofb_dec.txt
00000000  31 32 33 34 35                                    |12345|
00000005
[02/15/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in f2.txt -out f2_ofb.txt -K 00112233445566778889aabbccddeeff -iv 010203040506070
8
hex string is too short, padding with zero bytes to length
[02/15/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in f2_ofb.txt -out f2_ofb_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
hex string is too short, padding with zero bytes to length
[02/15/24]seed@VM:~/.../Files$ hexdump -C f2_ofb_dec.txt
00000000  31 32 33 34 35 36 37 38  39 30                    |1234567890|
0000000a
[02/15/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in f3.txt -out f3_ofb.txt -K 00112233445566778889aabbccddeeff -iv 010203040506070
8
hex string is too short, padding with zero bytes to length
[02/15/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in f3_ofb.txt -out f3_ofb_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
hex string is too short, padding with zero bytes to length
[02/15/24]seed@VM:~/.../Files$ hexdump -C f3_ofb_dec.txt
00000000  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36  |1234567890123456|
00000010
[02/15/24]seed@VM:~/.../Files$
```

2) CFB:

```
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in f1.txt -out f1_cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in f1_cfb.txt -out decrypt_f1_cfb.txt -nopad -K 00112233445566778889aabbccddeeff -iv
0102030405060708
hex string is too short, padding with zero bytes to length
[02/13/24]seed@VM:~/.../Files$ hexdump -C decrypt_f1_cfb.txt
00000000  31 32 33 34 35                                    |12345|
00000005
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in f2.txt -out f2_cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in f2_cfb.txt -out decrypt_f2_cfb.txt -nopad -K 00112233445566778889aabbccddeeff -iv
0102030405060708
hex string is too short, padding with zero bytes to length
[02/13/24]seed@VM:~/.../Files$ hexdump -C decrypt_f2_cfb.txt
00000000  31 32 33 34 35 36 37 38  39 30                    |1234567890|
0000000a
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in f3.txt -out f3_cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in f3_cfb.txt -out decrypt_f3_cfb.txt -nopad -K 00112233445566778889aabbccddeeff -iv
0102030405060708
hex string is too short, padding with zero bytes to length
[02/13/24]seed@VM:~/.../Files$ hexdump -C decrypt_f3_cfb.txt
00000000  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36  |1234567890123456|
00000010
[02/13/24]seed@VM:~/.../Files$
```

3) CBC:

```
    [02/13/24]seed@VM:~/.../Files$ hexdump -C decrypt_f1_cbc.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345...........|
00000010
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f2.txt -out f2_cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in f2_cbc.txt -out decrypt_f2_cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
hex string is too short, padding with zero bytes to length
[02/13/24]seed@VM:~/.../Files$ hexdump -C decrypt_f2_cbc.txt
00000000  31 32 33 34 35 36 37 38  39 30                    |1234567890|
0000000a
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f3.txt -out f3_cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in f3_cbc.txt -out decrypt_f3_cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030
405060708
hex string is too short, padding with zero bytes to length
[02/13/24]seed@VM:~/.../Files$ hexdump -C decrypt_f3_cbc.txt
00000000  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36  |1234567890123456|
00000010
[02/13/24]seed@VM:~/.../Files$
```

4) ECB:

```
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in f1.txt -out f1_ecb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
warning: iv not used by this cipher
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -d -in f1_ecb.txt -out decrypt_f1_ecb.txt -nopad -K 00112233445566778889aabbccddeeff -iv
0102030405060708
warning: iv not used by this cipher
[02/13/24]seed@VM:~/.../Files$ hexdump -C decrypt_f1_ecb.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345...........|
00000010
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in f2.txt -out f2_ecb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
warning: iv not used by this cipher
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -d -in f2_ecb.txt -out decrypt_f2_ecb.txt -nopad -K 00112233445566778889aabbccddeeff -iv
0102030405060708
warning: iv not used by this cipher
[02/13/24]seed@VM:~/.../Files$ hexdump -C decrypt_f2_ecb.txt
00000000  31 32 33 34 35 36 37 38  39 30 06 06 06 06 06 06  |1234567890......|
00000010
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in f3.txt -out f3_ecb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
warning: iv not used by this cipher
[02/13/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -d -in f3_ecb.txt -out decrypt_f3_ecb.txt -nopad -K 00112233445566778889aabbccddeeff -iv
0102030405060708
warning: iv not used by this cipher
[02/13/24]seed@VM:~/.../Files$ hexdump -C decrypt_f3_ecb.txt
00000000  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36  |1234567890123456|
00000010  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  |................|
00000020
```

From the above code we have observed that ECB and CBC have same behavior i.e. padding is getting appended for f1.txt and f2.txt files it is padded till 16 bytes and for f3.txt it is padded till 32 bytes. There is no padding for OFB and CFB modes when decrypting the files.

Task 5: Error Propagation – Corrupted Cipher Text

The purpose of this task is to understand the significance on decryption if a single encrypted character gets corrupted due to any reason. To achieve this, we encrypt the file with a cipher and corrupt a character using a hex editor (I used bless) and then decrypted this file. To corrupt the file, I deleted the 55th character (0x37 hex offset) and added a random character in its place. I repeated this exercise using the AES cipher with the CBC, ECB, CFB and OFB modes.

1) ECB:

```
[02/14/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -d -in enc_word_ecb.txt -out dec_word_ecb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
warning: iv not used by this cipher
[02/14/24]seed@VM:~/.../Files$ cat dec_word_ecb.txt
luctus accumsan tortor posuere ac ut consequat s?��     a簪 ��O%m libero justo laoreet sit amet cursus sit amet dictum sit amet justo donec enim diam vulputate ut pharetra sit amet aliquam id diam maecenas ultricies mi eget mauris pharetra et ultrices neque ornare aenean euismod elementum nisi quis eleifend quam adipiscing vitae proin sagittis nisl rhoncus mattis rhoncus urna neque viverra justo nec ultrices dui sapien eget mi proin sed libero enim sed faucibus turpis in eu mi bibendum neque egestas congue quisque egestas diam in arcu cursus euismod quis viverra nibh cras pulvinar mattis nunc sed blandit libero volutpat
```

Observation: After decryption, I observed that 16 bytes were corrupted in total.

2) CFB:

```
[02/14/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in enc_word_cfb.txt -out dec_word_cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[02/14/24]seed@VM:~/.../Files$ cat dec_word_cfb.txt
luctus accumsan tortor posuere ac ut consequat semper ��verra naB�/DX��p�V���aoreet sit amet cursus sit amet dictum sit amet justo donec enim diam vulputate ut pharetra sit amet aliquam id diam maecenas ultricies mi eget mauris pharetra et ultrices neque ornare aenean euismod elementum nisi quis eleifend quam adipiscing vitae proin sagittis nisl rhoncus mattis rhoncus urna neque viverra justo nec ultrices dui sapien eget mi proin sed libero enim sed faucibus turpis in eu mi bibendum neque egestas congue quisque egestas diam in arcu cursus euismod quis viverra nibh cras pulvinar mattis nunc sed blandit libero volutpat
[02/14/24]seed@VM:~/.../Files$
```

Observation: After decryption, I observed that 17 bytes were corrupted in total.

3) CBC:

```
[02/14/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in enc_word1.txt -out dec_word1.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[02/14/24]seed@VM:~/.../Files$ cat dec_word1.txt
��f�`x�g$S�fIp��tortor�posuere ac ut consequat semper viverra nam libero justo laoreet sit amet cursus sit amet dictum sit amet justo donec enim diam vulputate ut pharetra sit amet aliquam id diam maecenas ultricies mi eget mauris pharetra et ultrices neque ornare aenean euismod elementum nisi quis eleifend quam adipiscing vitae proin sagittis nisl rhoncus mattis rhoncus urna neque viverra justo nec ultrices dui sapien eget mi proin sed libero enim sed faucibus turpis in eu mi bibendum neque
[02/14/24]seed@VM:~/.../Files$
```

Observation: After decryption, I observed that 17 bytes were corrupted in total.

4) OFB:

```
[02/14/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in enc_word_ofb.txt -out dec_word_ofb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[02/14/24]seed@VM:~/.../Files$ cat dec_word_ofb.txt
luctus accumsan tortor posuere ac ut consequat semper ��verra nam libero justo laoreet sit amet cursus sit amet dictum sit amet justo donec enim diam vulputate ut pharetra sit amet aliquam id diam maecenas ultricies mi eget mauris pharetra et ultrices neque ornare aenean euismod elementum nisi quis eleifend quam adipiscing vitae proin sagittis nisl rhoncus mattis rhoncus urna neque viverra justo nec ultrices dui sapien eget mi proin sed libero enim sed faucibus turpis in eu mi bibendum neque egestas congue quisque egestas diam in arcu cursus euismod quis viverra nibh cras pulvinar mattis nunc sed blandit libero volutpat
[02/14/24]seed@VM:~/.../Files$
```

Observation: After decryption, I observed that 1 byte was corrupted in total.

Overall Observation:
ECB: 16 bytes was corrupted
CFB: 17 bytes was corrupted
CBC: 17 bytes was corrupted
OFB: 1 byte was corrupted

Task 6: Initial Vector (IV) and Common Mistakes

Task 6.1. IV Experimented

I experimented encryption with the same IV and different IV's and printed out the contents of the encrypted file.

Observation: We can see that for the same IV we are getting the same ciphered text but when the iv is changed the ciphered text got updated. So, it is better to create new iv every time one encrypts a new message.

Task 6.2. Common Mistake: Use the Same IV:

I edited the sample_code.py to the given messages and ciphers.

```
 GNU nano 4.8                    sample_code.py
#!/usr/bin/python3

# XOR two bytearrays
def xor(first, second):
   return bytearray(x^y for x,y in zip(first, second))

MSG   = "This is a known message!"
HEX_1 = "a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159"
HEX_2 = "bf73bcd3509299d566c35b5d450337e1bb175f903fafc159"

# Convert ascii string to bytearray
D1 = bytes(MSG, 'utf-8')

# Convert hex string to bytearray
D2 = bytearray.fromhex(HEX_1)
D3 = bytearray.fromhex(HEX_2)

r1 = xor(D1, D2)
r2 = xor(r1, D3)
print(bytes.fromhex(r2.hex()).decode('utf-8'))
```

Output:

```
[02/14/24]seed@VM:~/.../Files$ nano sample_code.py
[02/14/24]seed@VM:~/.../Files$ python3 sample_code.py
Order: Launch a missile!
[02/14/24]seed@VM:~/.../Files$
```

So from the given text we inferred that the cipher text maps to : “Order: Launch a missile!”

Observations: From the above images we can infer that if the iv is not changed during encrypting we can decipher the data if we know the original text. So generating new iv for every new encryptions lead to a safer encrypting practice.

CFB using instead of OFB:
I have used CFB instead of OFB and I have only got the initial part of the text.

Task 6.3. Common Mistake: Use a Predictable IV

I have built the docker container using docker-compose build and docker-compose up commands.
I used the netcat command to talk to the docker container. I got Bob’s cipher text IV and the next IV.

```
[02/14/24]seed@VM:~/.../Files$ nc 10.9.0.80 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: 027af6f5ac1fb10f12f6bd347f5c25c1
The IV used    : b5841390bcc864f8059c07d8b20ae5a0

Next IV        : f8227de9bcc864f8059c07d8b20ae5a0
```

I used the sample_code.py as a base and edited it to match the requirements of the question as shown in the picture below.

```
 GNU nano 4.8                                                    task.py
#!/usr/bin/python3

def xor(first, second):
        return bytearray(x^y for x,y in zip(first, second))

initialvec1 = "b5841390bcc864f8059c07d8b20ae5a0"
initialvec2 = "e160c54ebdc864f8059c07d8b20ae5a0"

#bit array conversion
converted = bytearray("Yes", "utf-8")
converted.extend([16-len(converted)%16]*(16-len(converted)%16))
ivconverted1 = bytearray.fromhex(initialvec1)
ivconverted2 = bytearray.fromhex(initialvec2)

res1 = xor(converted, ivconverted1)
res2 = xor(res1, ivconverted2)
print(res2.hex())
```

After running this code, I got a hex value which I sent to Bob's server. This returned a 32 byte value in which the first 16 bytes of the ciphered value is equal to Bob's ciphertext.

```
[02/14/24]seed@VM:~/.../Files$ nc 10.9.0.80 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: 027af6f5ac1fb10f12f6bd347f5c25c1
The IV used    : b5841390bcc864f8059c07d8b20ae5a0

Next IV        : f8227de9bcc864f8059c07d8b20ae5a0
Your plaintext : 14c31d740d0d0d0d0d0d0d0d0d0d0d0d
Your ciphertext: 027af6f5ac1fb10f12f6bd347f5c25c1648f88342f6fdb3918a1fb6617e85652

Next IV        : e160c54ebdc864f8059c07d8b20ae5a0
Your plaintext : 0d81a5d30c0d0d0d0d0d0d0d0d0d0d0d
Your ciphertext: 027af6f5ac1fb10f12f6bd347f5c25c1648f88342f6fdb3918a1fb6617e85652

Next IV        : bf5e97a7bdc864f8059c07d8b20ae5a0
Your plaintext :
```

By placing this value in Bob's server we will be getting the ciphertext. Inferring this data to be correct I am concluding that Bob has sent yes as the ciphertext.

Task 7: Programming using the Crypto Library

I have initialized a python program to find out the key. I used the words.txt file in the labsetup folder for iterating through the list. I encrypted the secret_key with each word in the list and compared it to the ciphertext. When a match is found, the loop is broken and key is printed out. I used cryptography library to achieve this outcome.

I used this as a reference: https://www.openssl.org/docs/man1.1.1/man3/EVP_CipherInit.html

Code breakdown:

1) Imported the required libraries
2) Converted hex values to bytearrays of iv and cipher_text
3) Opened the words.txt file and iterated through every word in the list
4) Padded each word with '#'
5) Initialized cipher with a key
6) Encrypted the top_secret with each word in words.txt after initialization of the cipher.
7) Checked weather encrypted text is matched with the cipher_text constant.
8) If a match is found, loop is broken and key is printed out.

```
 GNU nano 4.8                                    task7.py
from cryptography.hazmat.primitives.ciphers import Cipher,algorithms,modes
from cryptography.hazmat.backends import default_backend

initialvec = bytes.fromhex('aabbccddeeff00998877665544332211');
cipher_text = bytes.fromhex('764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2');
top_secret = b"This is a top secret."
file = open("words.txt", "r")
for key in file:
        key=key.strip()
        if len(key)<16:
                key+=('#'*(16-len(key)))
        cipher = Cipher(algorithms.AES(bytes(key[:16], 'utf-8')), modes.CBC(initialvec),backend = default_backend())
        encryptor = cipher.encryptor()
        encrypted_text = encryptor.update(top_secret)
        if encrypted_text in cipher_text:
                break
print(key)
```

Output:

```
[02/14/24]seed@VM:~/.../Files$ nano task7.py
[02/14/24]seed@VM:~/.../Files$ python3 task7.py
Syracuse#########
[02/14/24]seed@VM:~/.../Files$
```

The key is Syracuse.