Laboratory Report

Encryption

The laboratory prompt for this was provided by SEED Security Labs. SEED Security Labs is a project focused on enhancing cybersecurity education through hands-on laboratory exercises. Visit them at https://seedsecuritylabs.org/.

Ramnick Francis P. Ramos
+63 960 277 1720
ramnickfrancisramos@gmail.com

Cybersecurity Portfolio
September 26, 2025

Encryption

Ramnick Francis P. Ramos
ramnickfrancisramos@gmail.com

Table of Contents

Introduction

Cryptography's primary objective is to make it difficult for unauthorized users to decipher a communication message between two parties (IBM, n.d.). This laboratory report aims to explore how to use encryption in different file types, such as text and BMP files, and also study its different modes and strategies to make its use more secure (Du, 2018).

Specifically, this laboratory report aims to understand the following:

- Secret-key encryption;
- Substitution cipher and frequency analysis;
- Encryption modes, IV, and paddings;
- Common mistakes in using encryption algorithms; and
- Programming using the crypto library.

Environment Setup

This lab was tested on the SEED Ubuntu 20.04 VM using Oracle VirtualBox. The prebuilt image for the virtual machine was obtained from CMSC 191: Cybersecurity's Google Classroom, but it can also be downloaded directly from the SEED website. The virtual machine ran locally, and no cloud server was used for this lab exercise.

Container Setup and Commands

Docker was also used to make the lab environment for this exercise. This exercise requires the use of a container for task 6.

For the commands, the following aliases were used: dcbuild for building the container; dcup for running the container; and dcdown for closing the containers. Seen in the figures below are sample runs of the Docker container.

```
[09/25/25]seed@VM:~/.../Labsetup$ dcbuild
Building oracle-server
Step 1/8 : FROM handsonsecurity/seed-ubuntu:dev AS builder
dev: Pulling from handsonsecurity/seed-ubuntu
da7391352a9b: Already exists
14428a6d4bcd: Already exists
2c2d948710f2: Already exists
5d39fdfbe330: Pulling fs layer
56b236c9d9da: Downloading [==========>                                        ]
56b236c9d9da: Downloading [==================================================>]
5d39fdfbe330: Downloading [>                                                  ]
5d39fdfbe330: Downloading [=>                                                 ]
5d39fdfbe330: Downloading [===>                                               ]
5d39fdfbe330: Downloading [=====>                                             ]
5d39fdfbe330: Downloading [=======>                                           ]
5d39fdfbe330: Downloading [==========>                                        ]
5d39fdfbe330: Downloading [============>                                      ]
5d39fdfbe330: Downloading [===============>                                   ]
5d39fdfbe330: Pull complete
56b236c9d9da: Pull complete
```

Figure 1. Building the container.

```
[09/25/25]seed@VM:~/.../Labsetup$ dcup
WARNING: Found orphan containers (victim-10.9.0.80) for this project. If you rem
oved or renamed this service in your compose file, you can run this command with
 the --remove-orphans flag to clean it up.
Creating oracle-10.9.0.80 ... done
Attaching to oracle-10.9.0.80
oracle-10.9.0.80 | Server listening on 3000 for known_iv
```

Figure 2. Making the container available.

```
[09/25/25]seed@VM:~/.../Labsetup$ dockps
429ef463d5dc  oracle-10.9.0.80
[09/25/25]seed@VM:~/.../Labsetup$
```

Figure 3. The Docker container

Laboratory Tasks and Execution

Frequency Analysis

For this laboratory exercise task, a ciphertext that was encrypted using monoalphabetic encryption was given. This task aims to decipher this ciphertext using frequency analysis.

The goal then is to determine the original article through comprehending the text.

```
[09/22/25]seed@VM:~/.../Files$ chmod +x freq.py
[09/22/25]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
```

Figure 4. Frequency Table

From the frequency table, we can see that the most frequent letters are n, y, v, and x.

From the laboratory prompt, as well, an example was shown where the following substitution was used for creating the ciphertext (See Figure 5). This example was used as a guiding hint for the analysis of the cipher text..

```
$ tr 'abcdefghijklmnopqrstuvwxyz' 'sxtrwinqbedpvgkfmalhyuojzc' \
      < plaintext.txt > ciphertext.txt
```

Figure 5. Translation used.

```
[09/25/25]seed@VM:~/.../Files$ cat ciphertext.txt
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
```

Figure 6. Analyzing the Cipher

First, I attempted to analyze the cipher as is (See Figure 6), independently of the given translation. From the text, we can see that the first word is “ytn”. The first natural substitution for me for a document for a three-letter word is the word “the”, so I guessed that y->t, t->h, and n->e. Attempting to use the translation from the exercise then gave me this (See Figure 7):

```
[09/25/25]seed@VM:~/.../Files$ tr 'ytn' 'the' < ciphertext.txt > deciphered.txt
[09/25/25]seed@VM:~/.../Files$ cat deciphered.txt
the xqavhq tzhu  xu qzupvd lhmah qeecq vgxzt hmrht vbteh thmq ixur qthvure
vlvhpq thme the gvrreh beeiq imse v uxuvreuvhmvu txx

the vlvhpq hvae lvq gxxseupep gd the pecmqe xb hvhfed lemuqtemu vt mtq xztqet
vup the veevheut mceixqmxu xb hmq bmic axcevud vt the eup vup mt lvq qhveep gd
the ecehreuae xb cetxx tmceq ze givasrxlu eximtmaq vhcavupd vatmfmqc vup
v uvtmxuvi axufehqvtmxu vq ghmeb vup cvp vq v befeh phevc vgxzt lhetheh thehe
xzrht tx ge v eheqmpeut lmubhed the qevqxu pmput ozqt qeec ekthv ixur mt lvq
ekthv ixur geavzqe the xqavhq lehe cxfep tx the bmhqt leeseup mu cvhah tx
vfxmp axubimatmur lmth the aixqmur aehecxud xb the lmuteh xidcemaq thvusq
edexurahvur

xue gmr jzeqtmxu qzhhxzupmur thmq devhq vavpecd vlvhpq mq hxl xh mb the
aehecxud lmii vppheqq cetxx eqeeamviid vbteh the rxipeu rixgeq lhmah geavce
v ozgmivut axcmurxzt evhtd bxh tmceq ze the cxfeceut qeevhhevpep gd
exlehbzi hxiidlxxp lxceu lhx heieep hvmqe cmiimxuq xb pxiivhq tx bmrht qekzvi
hvhvqqceut vhxzup the axzuthd
```

Figure 7. Attempting ytn -> the.

```
[09/25/25]seed@VM:~/.../Files$ tr 'ytnvup' 'THEAND' < ciphertext.txt > deciphered.txt
 cat deciphered.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
```

Figure 8. Attempting the substitution of an AND and using uppercase

```
[09/25/25]seed@VM:~/.../Files$ tr 'ytnvupxbhzemqcri' 'THEANDIFROPISVGL' < ciphertext.txt > deciphered.txt; cat deciphered.txt
THE ISaARS TORN  IN SONDAd lHIaH SEEVS AgIOT RIGHT AFTER THIS LING STRANGE
AlARDS TRIP THE gAGGER FEELS LIsE A NINAGENARIAN TII

THE AlARDS RAaE lAS gIIsENDED gd THE DEVISE IF HARfEd lEINSTEIN AT ITS IOTSET
AND THE APPARENT IVPLISIIN IF HIS FILV aIVPANd AT THE END AND IT lAS SHAPED gd
THE EVERGENaE IF VETII TIVES OP gLAasGIlN PILITIaS ARVaANDd AaTIfISV AND
A NATIINAL aINfERSATIIN AS gRIEF AND VAD AS A FEfER DREAV AgIOT lHETHER THERE
IOGHT TI gE A PRESIDENT lINFREd THE SEASIN DIDNT oOST SEEV EkTRA LING IT lAS
EkTRA LING gEaAOSE THE ISaARS lERE VIfED TI THE FIRST lEEsEND IN VARaH TI
AfIID aINFLIaTING lITH THE aLISING aEREVINd IF THE lINTER ILdVPIaS THANsS
PdEINGaHANG

INE gIG jOESTIIN SORRIONDING THIS dEARS AaADEVd AlARDS IS HIl IR IF THE
aEREVINd lILL ADDRESS VETII ESPEaIALLd AFTER THE GILDEN GLIgES lHIaH gEaAVE
A oOgILANT aIVINGIOT PARTd FIR TIVES OP THE VIfEVENT SPEARHEADED gd
PIlERFOL HILLdlIID lIVEN lHI HELPED RAISE VILLIINS IF DILLARS TI FIGHT SEkOAL
HARASSVENT ARIOND THE aIONTRd

SIGNALING THEIR SOPPIRT GILDEN GLIgES ATTENDEES SlATHED THEVSELfES IN gLAas
SPIRTED LAPEL PINS AND SIONDED IFF AgIOT SEkIST PIlER IVgALANaES FRIV THE RED
aARPET AND THE STAGE IN THE AIR E lAS aALLED IOT AgIOT PAd INEjOITd AFTER
ITS FIRVER ANaHIR aATT SADLER jOIT INaE SHE LEARNED THAT SHE lAS VAsING FAR
LESS THAN A VALE aIHIST AND DORING THE aEREVINd NATALIE PIRTVAN TIIs A gLONT
AND SATISFdING DIG AT THE ALLVALE RISTER IF NIVINATED DIREaTIRS HIl aIOLD
THAT gE TIPPED
```

Figure 9. Some other attempts.

I then continued on making more guesses using context clues on what is naturally occurring words (See Figur 8 and 9).

```
[09/25/25]seed@VM:~/.../Files$ tr 'ytnxqavhlp' 'THEOSCARWD' < ciphertext.txt > deciphered.txt; cat deciphered.tx
THE OSCARS TzRu  Ou SzuDAd WHmCH SEEcS AgOzT RmrHT AbTER THmS iOur STRAurE
AWARDS TRme THE gArrER bEEiS imsE A uOuArEuARmAu TOO

THE AWARDS RACE WAS gOOsEuDED gd THE DEcmSE Ob HARfEd WEmuSTEmu AT mTS OzTSET
AuD THE AeeAREuT mceiOSmOu Ob HmS bmic COceAud AT THE EuD AuD mT WAS SHAeED gd
THE EcERrEuCE Ob cETOO TmcES ze giACsrOWu eOimTmCS ARcCAuDd ACTmfmSc AuD
A uATmOuAi COufERSATmOu AS gRmEb AuD cAD AS A bEfER DREAc AgOzT WHETHER THERE
OzrHT TO gE A eRESmDEuT WmubREd THE SEASOu DmDuT ozST SEEc EkTRA iOur mT WAS
EkTRA iOur gECAzSE THE OSCARS WERE cOfED TO THE bmRST WEEsEuD mu cARCH TO
AfOmD COubimCTmur WmTH THE CiOSmur CEREcOud Ob THE WmuTER OidcemCS THAusS
edEOurCHAur

OuE gmr jzESTmOu SzRROzuDmur THmS dEARS ACADEcd AWARDS mS HOW OR mb THE
CEREcOud Wmii ADDRESS cETOO ESeECmAiid AbTER THE rOiDEu riOgES WHmCH gECAcE
A ozgmiAuT COcmurOzT eARTd bOR TmcES ze THE cOfEcEuT SeEARHEADED gd
eOWERbzi HOiidWOOD WOcEu WHO HEieED RAmSE cmiimOuS Ob DOiiARS TO bmrHT SEkzAi
HARASScEuT AROzuD THE COzuTRd

SmruAimur THEmR SzeeORT rOiDEu riOgES ATTEuDEES SWATHED THEcSEifES mu giACs
SeORTED iAeEi emuS AuD SOzuDED Obb AgOzT SEkmST eOWER mcgAiAuCES bROc THE RED
CAReET AuD THE STArE Ou THE AmR E WAS CAiiED OzT AgOzT eAd muEjzmTd AbTER
mTS bORcER AuCHOR CATT SADiER jzmT OuCE SHE iEARuED THAT SHE WAS cAsmur bAR
iESS THAu A cAiE COHOST AuD DzRmur THE CEREcOud uATAimE eORTcAu TOOs A gizuT
AuD SATmSbdmur Dmr AT THE AiicAiE ROSTER Ob uOcmuATED DmRECTORS HOW COziD
THAT gE TOeeED
```

Figure 10. Understanding that ISaARS might be OSCARS; revamping guesses

As seen in Figure 10, I then realized that the excerpt might contain words like OSCARS and AWARDS. This might be about some awards shows in Hollywood.

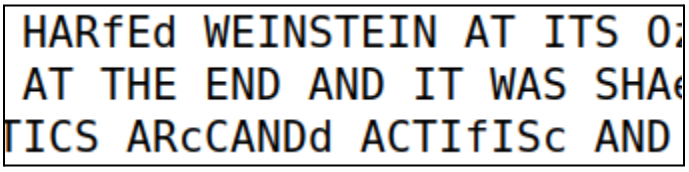

Figure 11. Guessing HARVEY WEINSTEIN

I then noticed how HARVEY WEINSTEIN might be part of the script (See Figure 11).

```
[09/25/25]seed@VM:~/.../Files$ tr 'ytnxqavhlpumbrisfegjzdcok' 'THEOSCARWDNIFGLKVPBQUPMJX' < ciphertext.txt > deciphered.txt; cat deciphe
xt
THE OSCARS TURN  ON SUNDAP WHICH SEEMS ABOUT RIGHT AFTER THIS LONG STRANGE
AWARDS TRIP THE BAGGER FEELS LIKE A NONAGENARIAN TOO

THE AWARDS RACE WAS BOOKENDED BP THE DEMISE OF HARVEP WEINSTEIN AT ITS OUTSET
AND THE APPARENT IMPLOSION OF HIS FILM COMPANP AT THE END AND IT WAS SHAPED BP
THE EMERGENCE OF METOO TIMES UP BLACKGOWN POLITICS ARMCANDP ACTIVISM AND
A NATIONAL CONVERSATION AS BRIEF AND MAD AS A FEVER DREAM ABOUT WHETHER THERE
OUGHT TO BE A PRESIDENT WINFREP 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 CEREMONP OF THE WINTER OLPMPICS THANKS
PPEONGCHANG

ONE BIG QUESTION SURROUNDING THIS PEARS ACADEMP AWARDS IS HOW OR IF THE
CEREMONP WILL ADDRESS METOO ESPECIALLP AFTER THE GOLDEN GLOBES WHICH BECAME
A JUBILANT COMINGOUT PARTP FOR TIMES UP THE MOVEMENT SPEARHEADED BP
POWERFUL HOLLPWOOD WOMEN WHO HELPED RAISE MILLIONS OF DOLLARS TO FIGHT SEXUAL
HARASSMENT AROUND THE COUNTRP

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 PAP INEQUITP AFTER
ITS FORMER ANCHOR CATT SADLER QUIT ONCE SHE LEARNED THAT SHE WAS MAKING FAR
LESS THAN A MALE COHOST AND DURING THE CEREMONP NATALIE PORTMAN TOOK A BLUNT
AND SATISFPING DIG AT THE ALLMALE ROSTER OF NOMINATED DIRECTORS HOW COULD
THAT BE TOPPED
```

Figure 12. Final deciphered text.

Seen in Figure 12 is the final attempt at deciphering the cipher text. It appears that this is a feature article on what happened at the Oscars on Sunday. Furthermore,there appears to be a conflict with letters P and Y in the cipher, which could be attributed to the encryption done.

Encryption using Different Ciphers and Modes

This task aims to have an exploration in the different modes of different ciphers, namely -aes-128-cbc,-bf-cbc,-aes-128-cfb.

```
[09/25/25]seed@VM:~/.../Files$ cat plain.txt
United States of America

The Star Spangled Banner

Oh, say! can you see by the dawn's early light
What so proudly we hailed at the twilight's last gleaming;
Whose broad stripes and bright stars, through the perilous fight,
O'er the ramparts we watched were so gallantly streaming?
And the rocket's red glare, the bombs bursting in air,
Gave proof through the night that our flag was still there:
Oh, say! does that star-spangled banner yet wave
O'er the land of the free and the home of the brave?

On the shore, dimly seen through the mists of the deep,
Where the foe's haughty host in dread silence reposes,
What is that which the breeze, o'er the towering steep,
As it fitfully blows, half conceals, half discloses?
Now it catches the gleam of the morning's first beam,
In fully glory reflected now shines in the stream:
'Tis the star-spangled banner! Oh, long may it wave
O'er the land of the free and the home of the brave!

And where is that band who so vauntingly swore
That the havoc of war and the battle's confusion
A home and a country should leave us no more?
Their blood has washed out their foul footsteps' pollution!
No refuge could save the hireling and slave
From the terror of flight or the gloom of the grave:
And the star-spangled banner in triumph doth wave
```

Figure 13. Plain Text Used

For this task on exploring the use of encryption, the national anthem of the United States of America was chosen to be used (See Figure 13).

```
[09/22/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher_aes_cbc.bin \
>   -K 00112233445566778889aabbccddeeff \
>   -iv 0102030405060708090a0b0c0d0e0f10
[09/22/25]seed@VM:~/.../Files$ openssl enc -bf-cbc -e -in plain.txt -out cipher_bf_cbc.bin \
>   -K 00112233445566778889aabbccddeeff \
>   -iv 0102030405060708
[09/22/25]seed@VM:~/.../Files$
[09/22/25]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in plain.txt -out cipher_aes_cfb.bin \
>   -K 00112233445566778889aabbccddeeff \
>   -iv 0102030405060708090a0b0c0d0e0f10
[09/22/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in cipher_aes_cbc.bin -out decrypted.txt \
>   -K 00112233445566778889aabbccddeeff \
>   -iv 0102030405060708090a0b0c0d0e0f10
[09/22/25]seed@VM:~/.../Files$
[09/22/25]seed@VM:~/.../Files$ cat decrypted.txt
United States of America

The Star Spangled Banner

Oh, say! can you see by the dawn's early light
What so proudly we hailed at the twilight's last gleaming;
Whose broad stripes and bright stars, through the perilous fight,
O'er the ramparts we watched were so gallantly streaming?
And the rocket's red glare, the bombs bursting in air,
Gave proof through the night that our flag was still there:
Oh, say! does that star-spangled banner yet wave
O'er the land of the free and the home of the brave?

On the shore, dimly seen through the mists of the deep,
```

Figure 14. Trying out -aes-128-cbc,-bf-cbc,-aes-128-cfb and doing a Decryption Sample for -aes-128-cbc

As shown in Figure 14, we experimented with several encryption techniques before fianlly attempting to decipher back from the ciphered the aes_cbc encrypted file.

Encryption Mode – ECB vs. CBC

In this task, the goal is to compare the ECB and CBC encryption on an image file. The main difference between ECB (Electronic Code Block) and CBC (Cipher Block Chaining) encryption modes is in how ECB directly encrypts blocks of inputs, while CBC utilizes another layer of XOR operations in the chain of encryption it makes (Geeks for Geeks, 2025).

For this task, we are given the file pic_original.bmp as our test file (See Figure 15).

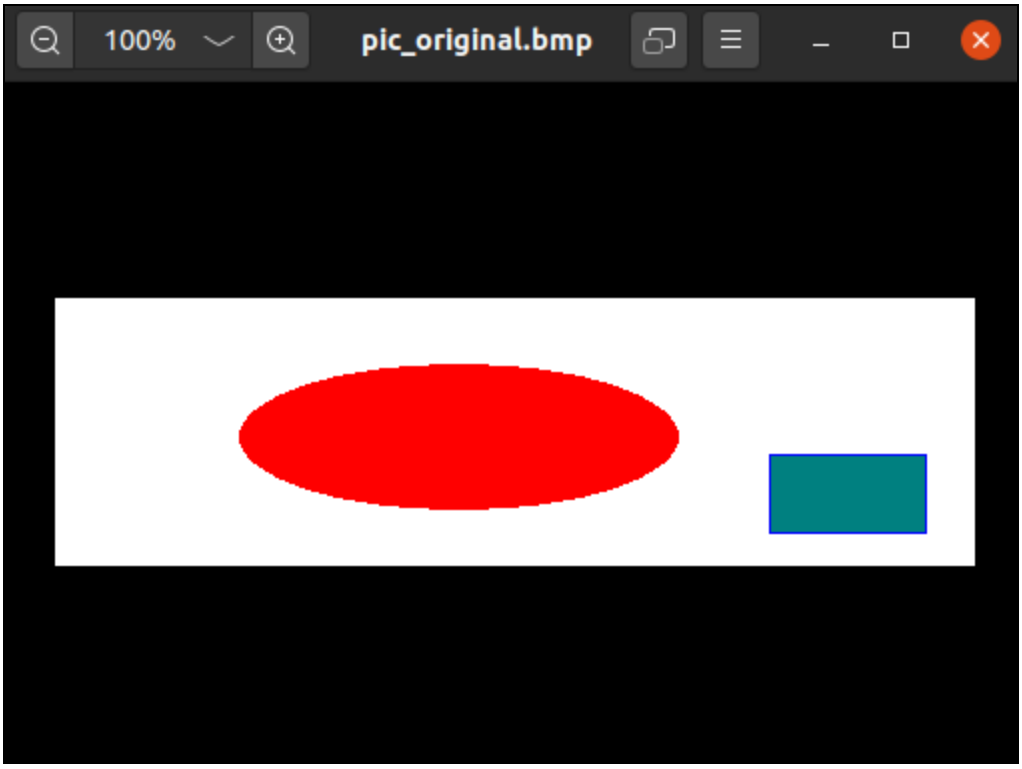

Figure 15. pic _original.bmp

```
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -in pic_original.bmp -ou
t pic_original_ecb.bmp
enter aes-128-ecb encryption password:
Verifying - enter aes-128-ecb encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -in pic_original.bmp -ou
t pic_original_cbc.bmp
enter aes-128-cbc encryption password:
Verifying - enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
```

Figure 16. Performing the encryption on the pic_original.bmp using ECB and CBC modes

We then performed ECB and CBC modes of encryption to the same bmp file (See Figure 16).

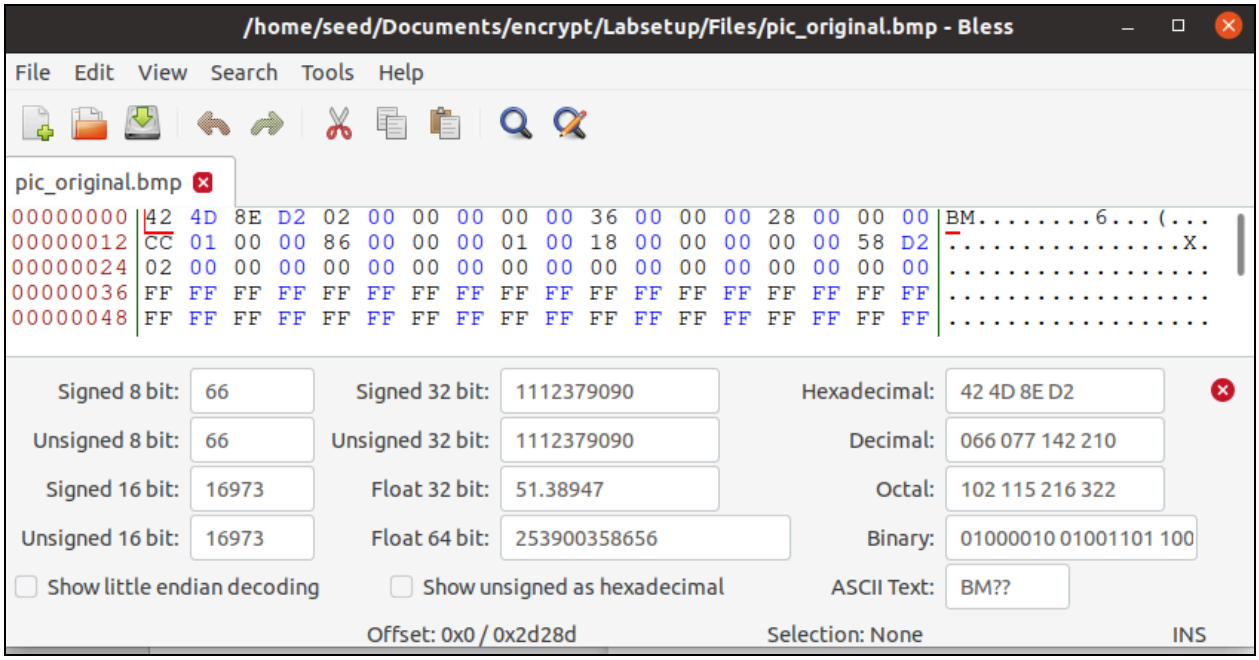

Figure 17. BLESS HEXEDITOR for the original image

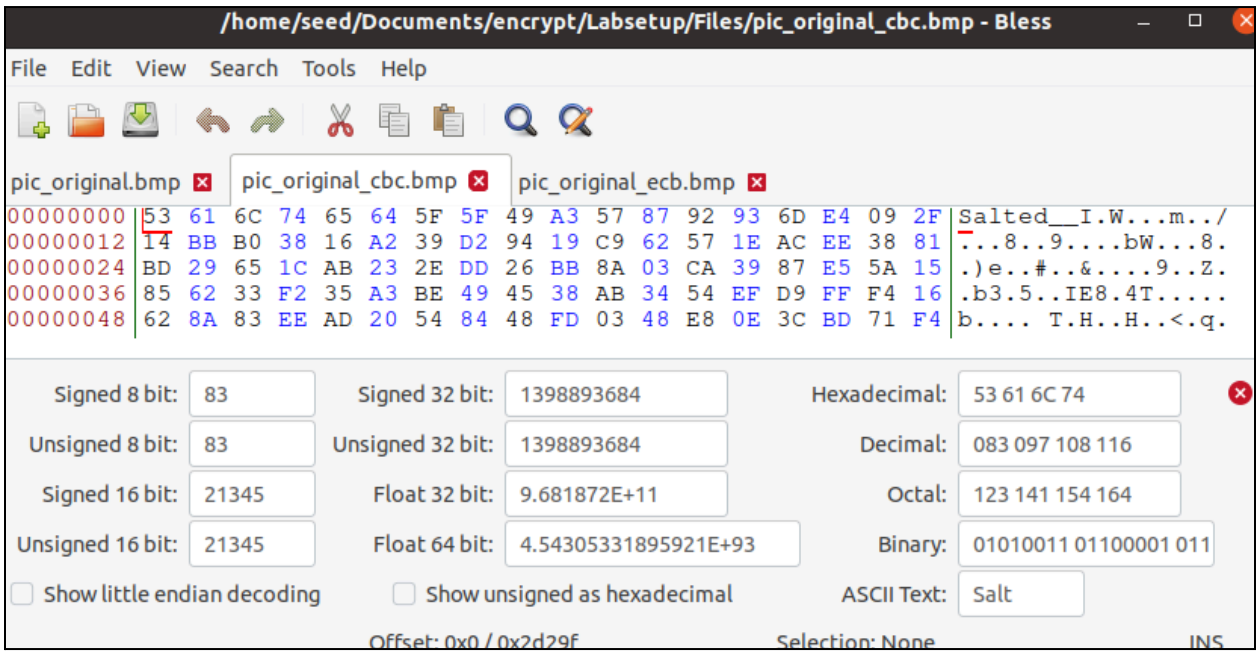

Figure 18. BLESS HEXEDITOR for the cbc encrypted

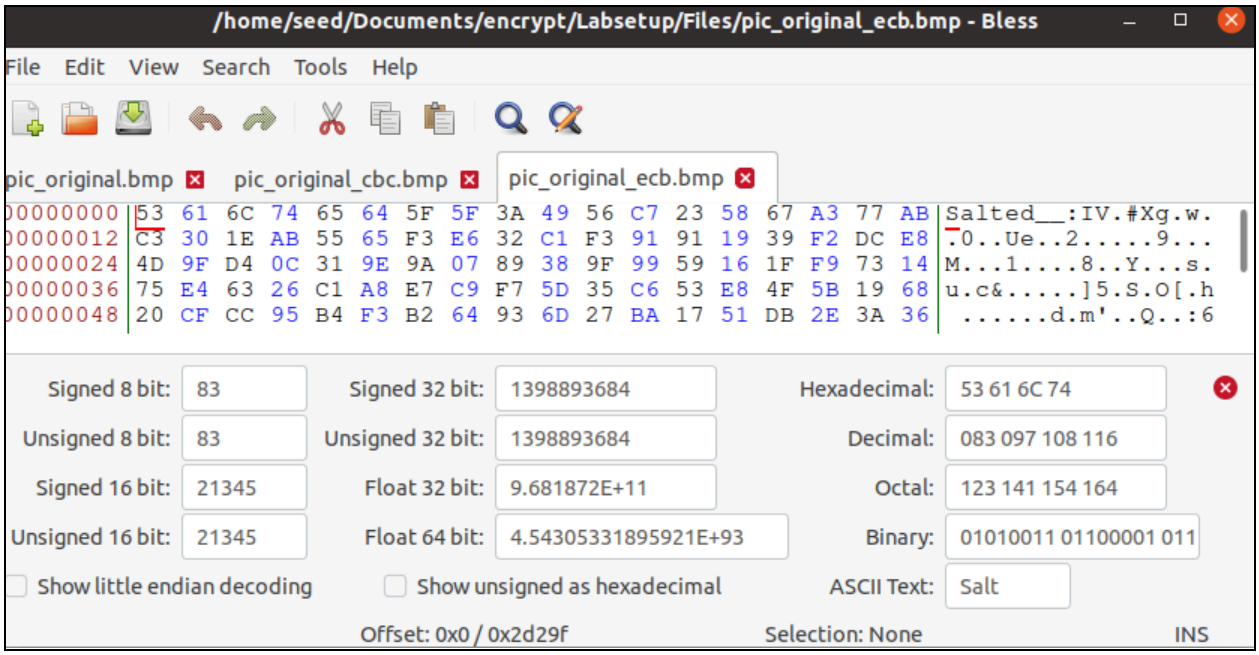

Figure 19. BLESS HEXEDITOR for the ECB encrypted

Seen in Figure 17, 18, and 19 are the bitmaps of these files viewed using BLESS.

```
[09/26/25]seed@VM:~/.../Files$ head -c 54 pic_original.bmp > header.bmp
[09/26/25]seed@VM:~/.../Files$ tail -c +55 pic_original_ecb.bmp > body_ecb.bmp
[09/26/25]seed@VM:~/.../Files$ tail -c +55 pic_original_cbc.bmp > body_cbc.bmp
[09/26/25]seed@VM:~/.../Files$ cat header.bmp body_ecb.bmp > new_ecb.bmp
[09/26/25]seed@VM:~/.../Files$ cat header.bmp body_cbc.bmp > new_cbc.bmp
[09/26/25]seed@VM:~/.../Files$
```

Figure 20. Performing the fixup needed for the header

In Figure 20, it can be seen that we made a fix on the header of the encrypted bmp file to make it viewable in the image viewer of the homelab.

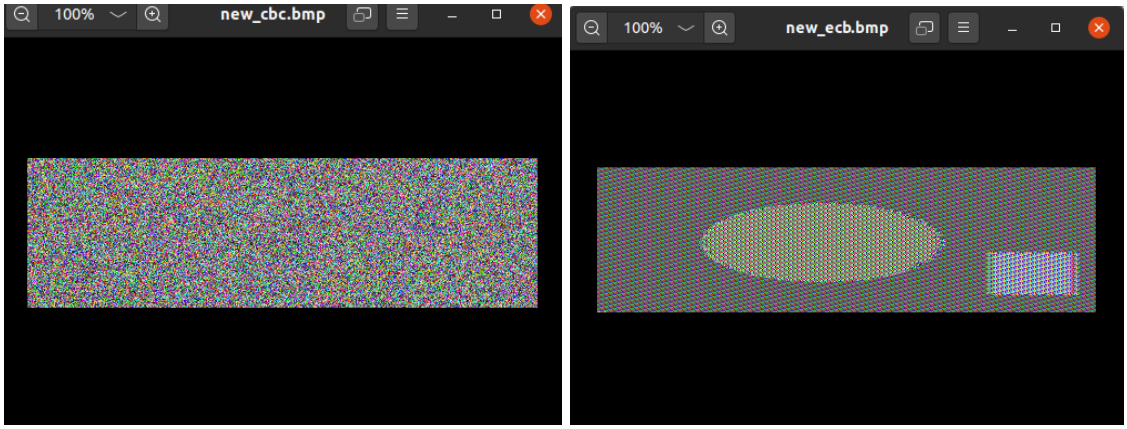

Figure 21. CBC and EBC encrypted file for pic_original.bmp.

After performing the necessary fixup on the header of the encrypted files for the ECB and CBC, we can see that the encrypted image outputted by CBC appears to be more unintelligible than ECB. This

implies that CBC may be more appropriate in encrypting images as it makes images almost like it is just random noise. This can be attributed to how its encryption is chained with XOR operations.

Replicating the Encryption on an Image of a Nightsky

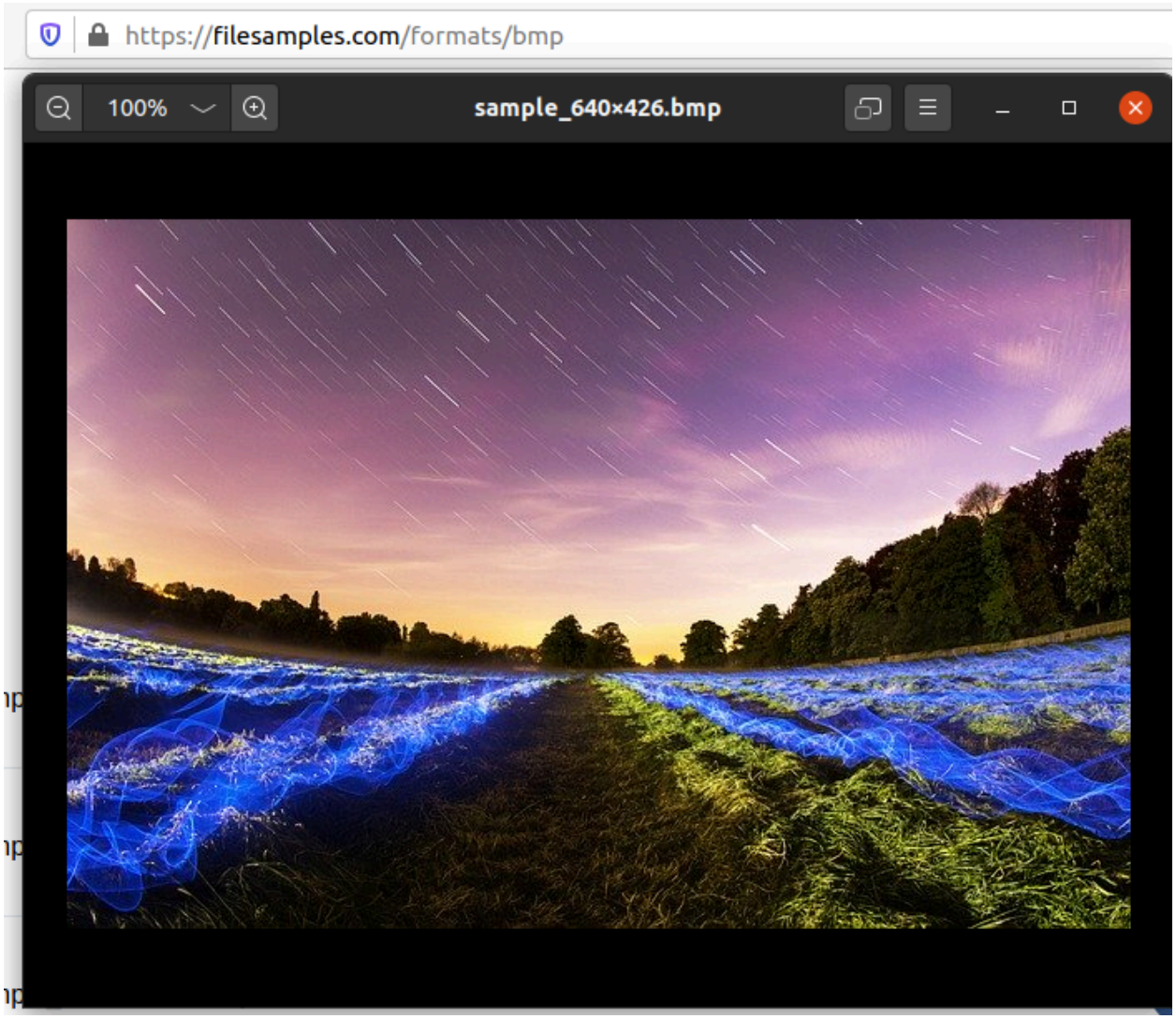

Figure 22. Sample bmp file from the internet

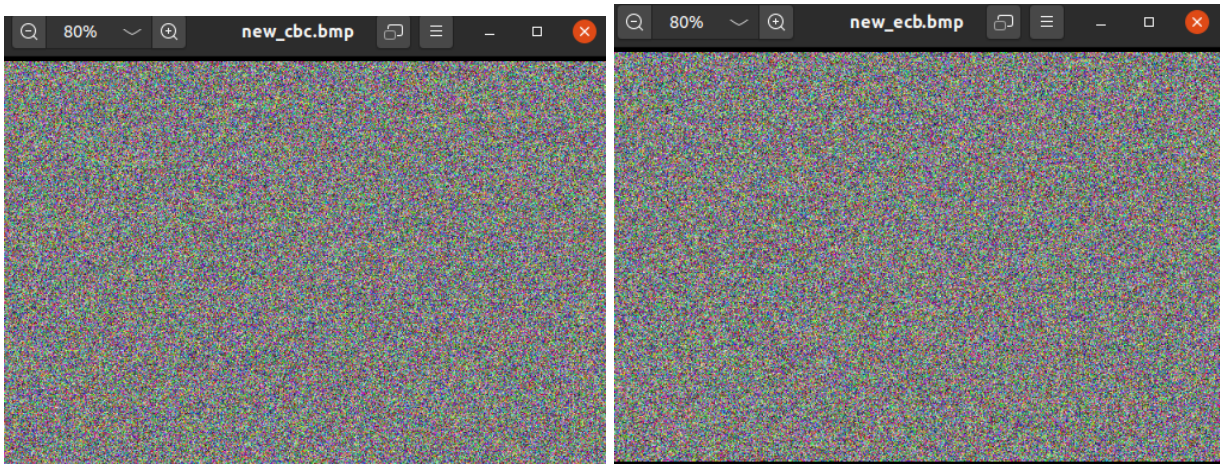

Figure 23. Outputted CBC and ECB encrypted image for the sample file

For the replication of the image encryption, I chose an image file that is naturally noisy (night image) from the internet. As seen in Figure 23, we can see that the outputted encrypted image are both unintelligible for naturally noisy images, which implies that in scenarios such as this, using ECB encryption mode would suffice.

Padding

This task aims to explore how padding affects the encryption in different modes. Block Chaining Encryption modes, like CBC and ECB, is an encryption process made using blocks of cipher (Geek for Geeks, 2025). On the other hand, OFB(Output Feedback Mode) and CFB(Cipher Feedback Mode) use feedback for the next block to be ciphered; furthermore, it utilizes an initial vector for the stream of feedbacks it will create for the encryption.

```
[09/26/25]seed@VM:~/.../Files$ echo -n "12345" > f1.txt
[09/26/25]seed@VM:~/.../Files$ echo -n "1234567890" > f2.txt
[09/26/25]seed@VM:~/.../Files$ echo -n "1234567890ABCDEF" > f3.txt
```

```
[09/26/25]seed@VM:~/.../Files$ ls -l f1.txt
-rw-rw-r-- 1 seed seed 5 Sep 26 02:16 f1.txt
[09/26/25]seed@VM:~/.../Files$ ls -l f2.txt
-rw-rw-r-- 1 seed seed 10 Sep 26 02:16 f2.txt
[09/26/25]seed@VM:~/.../Files$ ls -l f3.txt
-rw-rw-r-- 1 seed seed 16 Sep 26 02:16 f3.txt
```

Figure 24. Creating the Test Files and Noting the sizes of the files for the original test files

As seen in Figure 24, this task first started with creating three different text files with characters of 5, 10, and 16 sizes.

```
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -K 00112233445566778899aabbccddeeff -in f1.txt -out f1.ecb.enc
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -K 00112233445566778899aabbccddeeff -in f2.txt -out f2.ecb.enc
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -K 00112233445566778899aabbccddeeff -in f3.txt -out f3.ecb.enc
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -K 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10 -in f1.txt -out f1.cbc.enc
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -K 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10 -in f2.txt -out f2.cbc.enc
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -K 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10 -in f3.txt -out f3.cbc.enc
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -K 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10 -in f1.txt -out f1.cfb.enc
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -K 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10 -in f2.txt -out f2.cfb.enc
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -K 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10 -in f3.txt -out f3.cfb.enc
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -K 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10 -in f1.txt -out f1.ofb.enc
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -K 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10 -in f2.txt -out f2.ofb.enc
[09/26/25]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -K 00112233445566778899aabbccddeeff \
> -iv 0102030405060708090a0b0c0d0e0f10 -in f3.txt -out f3.ofb.enc
```

Figure 25. Encrypting the files using ECB, CBC, CFB, and OFB

```
[09/26/25]seed@VM:~/.../Files$ ls -l f*.enc
-rw-rw-r-- 1 seed seed 16 Sep 26 02:29 f1.cbc.enc
-rw-rw-r-- 1 seed seed  5 Sep 26 02:29 f1.cfb.enc
-rw-rw-r-- 1 seed seed 16 Sep 26 02:28 f1.ecb.enc
-rw-rw-r-- 1 seed seed  5 Sep 26 02:29 f1.ofb.enc
-rw-rw-r-- 1 seed seed 16 Sep 26 02:29 f2.cbc.enc
-rw-rw-r-- 1 seed seed 10 Sep 26 02:29 f2.cfb.enc
-rw-rw-r-- 1 seed seed 16 Sep 26 02:28 f2.ecb.enc
-rw-rw-r-- 1 seed seed 10 Sep 26 02:29 f2.ofb.enc
-rw-rw-r-- 1 seed seed 32 Sep 26 02:29 f3.cbc.enc
-rw-rw-r-- 1 seed seed 16 Sep 26 02:29 f3.cfb.enc
-rw-rw-r-- 1 seed seed 32 Sep 26 02:28 f3.ecb.enc
-rw-rw-r-- 1 seed seed 16 Sep 26 02:29 f3.ofb.enc
```

Figure 26. Comparing the sizes of the encrypted files

These are then encrypted using ECB, CBC, CFB, and OFB (See Figure 25). Figure 26 shows the file sizes of the encrypted files. It is interesting to note how the block chaining encryptions are always larger than their feedback output encryption modes counterpart.

```
[09/26/25]seed@VM:~/.../Files$ openssl enc -d -aes-128-ecb -K 00112233445566778899aabbccddeeff -in f1.ecb.enc -out f1.ecb.dec
[09/26/25]seed@VM:~/.../Files$ openssl enc -d -aes-128-cbc -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f1
0 -in f1.cbc.enc -out f1.cbc.dec
[09/26/25]seed@VM:~/.../Files$ ls -l f1.ecb.dec
-rw-rw-r-- 1 seed seed 5 Sep 26 02:37 f1.ecb.dec
[09/26/25]seed@VM:~/.../Files$ ls -l f1.cbc.dec
-rw-rw-r-- 1 seed seed 5 Sep 26 02:37 f1.cbc.dec
```

Figure 27. Sizes of the Decrypted File for CBC and ECB

Figure 27, on the other hand, shows how, despite the largening of the file when encrypted using cipher block modes, the size returns to the original when decrypted.

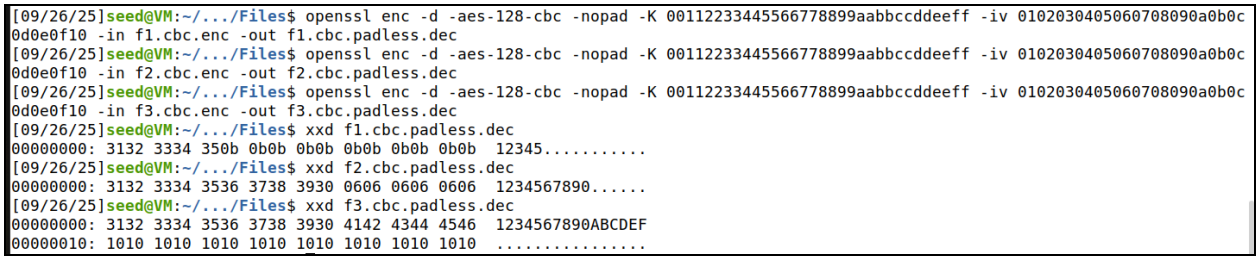

Figure 28. Using -nopad for the decryption on CBC

In Figure 28, -nopad option was added when decrypting CBC-encrypted files. -nopad, as per its definition, is used to show the padding used by OpenSSL so that when it is returned, the output is not just the plain text of the decipher. From Figure 28 then, we can see that paddings like “0b0b” were used for f1, “0606” was used for f2, and “1010” for f3.

Error Propagation – Corrupted Cipher Text

This task aims to analyze how a corrupted bit in the encrypted file affects the decryption.

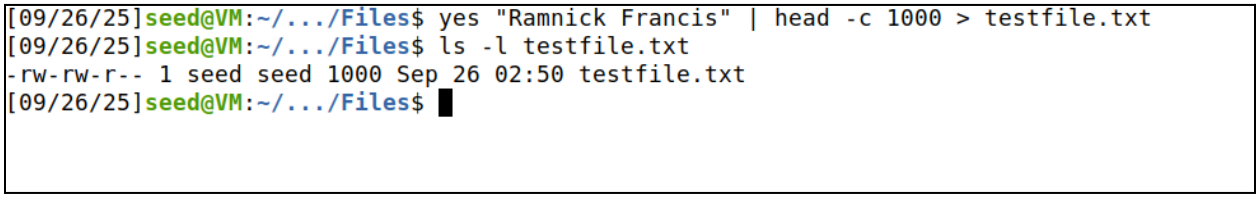

Figure 29. Making 1000-character Test file

For this task, we created a 1000-character test file that just prints “Ramnick Francis” repeatedly.

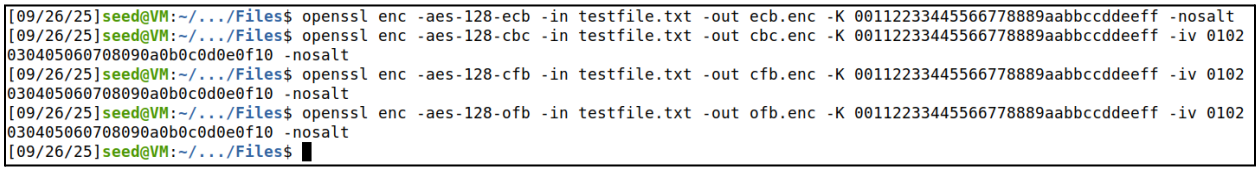

Figure 30. Generating the encrypted versions for ECB, CBC, CFB, and OFB

On Figure 30, we then used ECB, CBC, CFB, and OFB to encrypt the file. We added the option -nosalt so that the input of a password will not be necessary anymore.

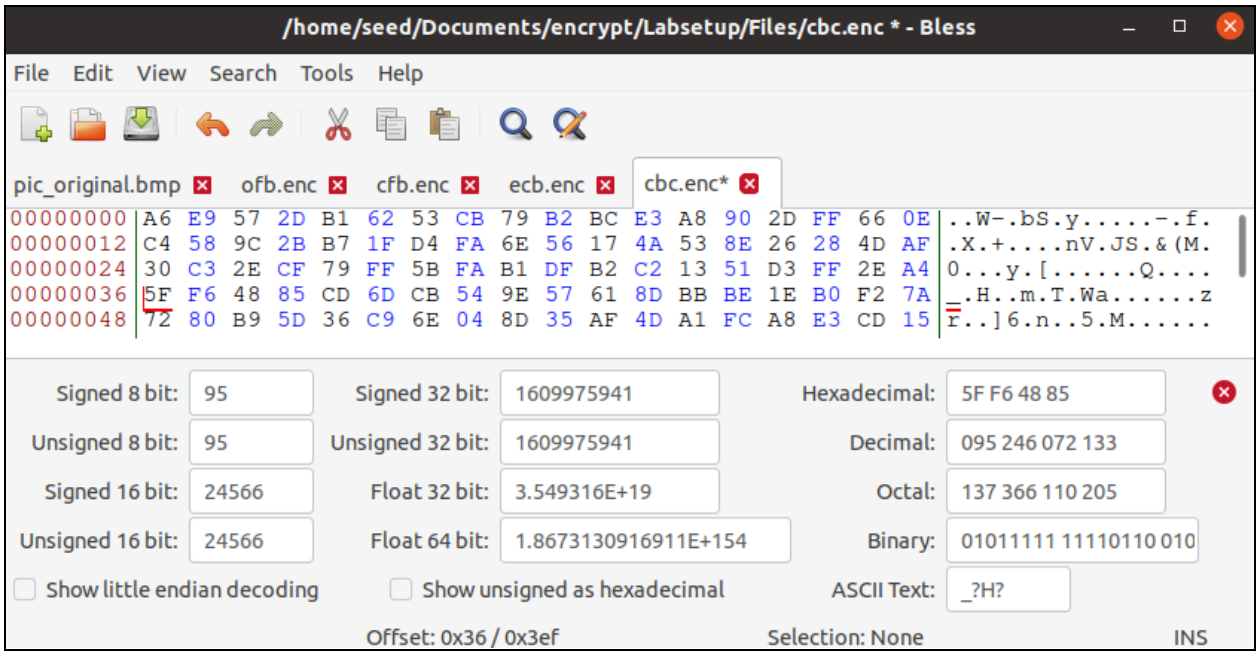

Figure 31. Example corrupting of CBC enc file by changing the 55th byte. Similar is implemented to all other enc files

We then corrupted the 55th bit of all these encrypted files using BLESS. An example is shown in Figure 31 where the 55th bit of cbc.enc was changed.

```
[09/26/25]seed@VM:~/.../Files$ xxd -s 54 -l 1 ecb.enc
00000036: 89                                       .
[09/26/25]seed@VM:~/.../Files$ xxd -s 54 -l 1 cfb.enc
00000036: ab                                       .
[09/26/25]seed@VM:~/.../Files$ xxd -s 54 -l 1 ofb.enc
00000036: fb                                       .
[09/26/25]seed@VM:~/.../Files$ xxd -s 54 -l 1 cbc.enc
00000036: 5d                                       ]
[09/26/25]seed@VM:~/.../Files$ xxd -s 54 -l 1 ecb.enc
00000036: 8f                                       .
[09/26/25]seed@VM:~/.../Files$ xxd -s 54 -l 1 cfb.enc
00000036: af                                       .
[09/26/25]seed@VM:~/.../Files$ xxd -s 54 -l 1 cbc.enc
00000036: 5d                                       ]
[09/26/25]seed@VM:~/.../Files$ xxd -s 54 -l 1 ofb.enc
00000036: ff                                       .
[09/26/25]seed@VM:~/.../Files$
```

Figure 32. Before and After corrupting 55th byte

Figure 32 shows how the 55th bit are changed. Mostly, the last “character” was just changed into an F.

```
[09/26/25]seed@VM:~/.../Files$ openssl enc -d -aes-128-ecb -in ecb.enc -out ecb_corrupt.dec -K 00112233445566778889aabbccddeeff -n
osalt
[09/26/25]seed@VM:~/.../Files$ openssl enc -d -aes-128-cbc -in cbc.enc -out cbc_corrupt.dec -K 00112233445566778889aabbccddeeff -i
v 01020304050607080 90a0b0c0d0e0f10 -nosalt
[09/26/25]seed@VM:~/.../Files$ openssl enc -d -aes-128-cfb -in cfb.enc -out cfb_corrupt.dec -K 00112233445566778889aabbccddeeff -i
v 01020304050607080 90a0b0c0d0e0f10 -nosalt
[09/26/25]seed@VM:~/.../Files$ openssl enc -d -aes-128-ofb -in ofb.enc -out ofb_corrupt.dec -K 00112233445566778889aabbccddeeff -i
v 01020304050607080 90a0b0c0d0e0f10 -nosalt
[09/26/25]seed@VM:~/.../Files$
```

Figure 32. Decrypting the corrupted encryption

We then decrypted them back to see what will happen to them.

```
[09/26/25]seed@VM:~/.../Files$ head -c 100 *_corrupt.dec
==> cbc_corrupt.dec <==
Ramnick Francis
Ramnick Francis
Ramnick Francis
Ramnick Francis
Ramnick Francis
Ramnick Francis
Ramn
==> cfb_corrupt.dec <==
Ramnick Francis
Ramnick Francis
Ramnick Francis
Ramnico Francis
[bo gc)-M<Ramnick Francis
Ramn
==> ecb_corrupt.dec <==
Ramnick Francis
Ramnick Francis
Ramnick Francis
K5Az Ramnick Francis
Ramnick Francis
Ramn
==> ofb_corrupt.dec <==
Ramnick Francis
Ramnick Francis
Ramnick Francis
Ramnico Francis
Ramnick Francis
Ramnick Francis
Ramn[09/26/25]seed@VM:~/.../Files$
```

Figure 33. Actual Decrypted Files (for the first 100 characters, inclusive of the supposed corrupted byte)

My guess was that what will be unrecoverable was the Feedback Output ciphers OFB and CFB since they use feedbacks for the encryption. I also guessed that CBC will be problematic when corrupted

since it usesa layer of XOR operations. And, from Figure 33, we can see that the fully recoverable was CBC and ECB. They are the only ones who did not alter the string “Ramnick Francis” into “Ramnico…”. This means that the corrupted byte only affected its block.

Initial Vector (IV) and Common Mistakes

The aim of this task is to understand the importance of choosing the correct initial vector.

Initial Vector (IV) Experiment

```
[09/26/25]seed@VM:~/.../Files$ cd experiment/
[09/26/25]seed@VM:~/.../experiment$ touch trialfile.txt
[09/26/25]seed@VM:~/.../experiment$ cat trialfile.txt
trial file by ramnick
[09/26/25]seed@VM:~/.../experiment$
```

Figure 34. Making Trial File

```
[09/26/25]seed@VM:~/.../experiment$ openssl rand 16 > rand_key.bin
[09/26/25]seed@VM:~/.../experiment$ openssl enc -aes-128-cbc -K $(xxd -p rand_key.bin) -iv $(openssl rand -hex 16) -in trialfile.t
xt -out c1.bin
[09/26/25]seed@VM:~/.../experiment$ openssl enc -aes-128-cbc -K $(xxd -p rand_key.bin) -iv $(openssl rand -hex 16) -in trialfile.t
xt -out c2.bin
[09/26/25]seed@VM:~/.../experiment$ IV=$(openssl rand -hex 16)
[09/26/25]seed@VM:~/.../experiment$ openssl enc -aes-128-cbc -K $(xxd -p rand_key.bin) -iv $IV -in trialfile.txt -out c3.bin
[09/26/25]seed@VM:~/.../experiment$ openssl enc -aes-128-cbc -K $(xxd -p rand_key.bin) -iv $IV -in trialfile.txt -out c4.bin
```

Figure 35. Making Randomized key and Using for two different IV (c1 and c2) the same for another two (c3 and c4)

```
[09/26/25]seed@VM:~/.../experiment$ xxd c1.bin | head
00000000: 2c97 7c8a 15fd 8e7e d521 cfe6 dc16 b523  ,.|....~.!.....#
00000010: a409 1d7f 9e98 c197 87c9 e7c2 f0a7 dfd5  ................
[09/26/25]seed@VM:~/.../experiment$ xxd c2.bin | head
00000000: 1a53 2f9a 5e62 216e 02da ee82 c3ee 5431  .S/.^b!n......T1
00000010: cbb0 0e94 7f97 0f85 0d07 7b4c d0f7 c3db  ..........{L....
[09/26/25]seed@VM:~/.../experiment$ xxd c3.bin | head
00000000: 59ab a9c2 9e2d 7514 4445 ae57 b29a eb29  Y....-u.DE.W...)
00000010: a30d b434 0248 ee24 70ad 4496 b93d a9b4  ...4.H.$p.D..=..
[09/26/25]seed@VM:~/.../experiment$ xxd c4.bin | head
00000000: 59ab a9c2 9e2d 7514 4445 ae57 b29a eb29  Y....-u.DE.W...)
00000010: a30d b434 0248 ee24 70ad 4496 b93d a9b4  ...4.H.$p.D..=..
```

Figure 36. Comparing the outputted cipher texts

From Figure 35, we can see that using the same IV (c3 and c4) would output the same ciphered version of a plain text, making it not secure. Therefore, different versions of IV should be used

Common Mistake: Use the Same IV

This task aims to show how using the same initial vector makes the decryption of a message easier.

Open

sample_code.py
~/Documents/encrypt/Labsetup/Files

```
#!/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(D2, D3)
r3 = xor(D2, D2)

print("Recovered:")
print(r1.hex())

print("\n")
print(r2.hex())
print(r3.hex())

guess = xor(D3, r1)

print("\n\nAnswer for Guess:", guess)
```

Figure 37. Revised sample_code.py to try the given sample on the laboratory prompt

```
[09/26/25]seed@VM:~/.../Files$ python3 sample_code.py
Recovered:
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478


1b1a0d165253536c0055050d07570f00000c0000080b0000
00000000000000000000000000000000000000000000000


Answer for Guess: bytearray(b'Order: Launch a missile!')
[09/26/25]seed@VM:~/.../Files$
```

Figure 38. Recovered message: Order: Launch a missile!

The secret message for this task is Launch a missile. This was done by simply editting the sample_code.py (See Figure 37), knowing that they use the same IV.

On the question " how much of P2 can be revealed? When OFB is used", no parts of the message will be revealed. Since the main feature of CFB here that was exploited was its use of XOR command, the script used will not reveal any messages in the cipher. Furthermore, the keystream for OFB is independent to each other, unlike CFB (Geek for Geeks, 2025).

Common Mistake: Use a Predictable IV

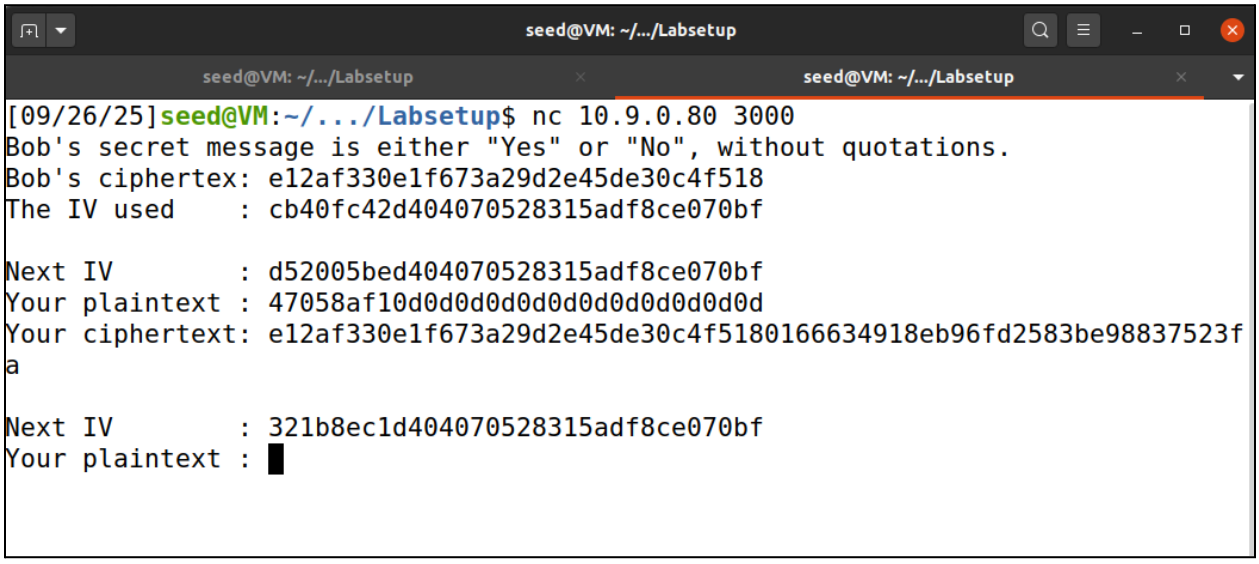

Figure 40. Encrypted Exchanges between Bob and The user

To find Bob's response to the flow of messages, we exploited the fact that block cipher modes like CBC and OFB perform XOR operations between plaintext, ciphertext, and keystreams. **It is known that XOR operations are reversible.** By taking the XOR of the known plaintext (P1) and its ciphertext (C1),

the value (47058af10d0d0d0d...) was derived for an answer of yes. This represents a masked intermediate value in this operation flow. We then applied this to Bob's cipher text, this is how we figured out if Bob's response was "yes" since the next ciphertext matches Bob's first ciphertext – through analyzing the XOR Operation.

Challenges and Troubleshooting

The main challenge really was on utilizing context clues when deciphering ciphers texts. I think really understanding how messages are naturally written was the solution. I also struggled with using the syntax for OpenSSL command, but searching up documentation solved it.

Discussion

Cryptography and its development had truly come a long way from the era of the enigma during World War I. Its application today, with the endless amount of messages exchange in the digital age, is truly endless. However, with the advancement of technology and the hacking culture, it became imperative to be mindful in the modes of encryption to be used. All of these modes expounded in thai laboratory report has both pros and cons, be it in the heaviness of the operations or the costs of security. It is up to the security officers to deliberate and understand what is needed by the assets being protected.

References

Geek for Geeks. (2025, July 11). *Block Cipher modes of Operation*. GeeksforGeeks. Retrieved September 26, 2025, from https://www.geeksforgeeks.org/ethical-hacking/block-cipher-modes-of-operation/

Geeks for Geeks. (2025, July 23). *ECB Mode vs CBC Mode in Cryptography*. https://www.geeksforgeeks.org/computer-networks/ecb-mode-vs-cbc-mode-in-cryptography/. https://www.geeksforgeeks.org/computer-networks/ecb-mode-vs-cbc-mode-in-cryptography/

IBM. (n.d.). *What Is Cryptography?* IBM. Retrieved September 26, 2025, from https://www.ibm.com/think/topics/cryptography