ARVIND SAI DOODA A20553046

Lab 07 - Secret-Key Encryption Lab

1 Overview

Lab Environment. This lab has been tested on our pre-built Ubuntu 20.04 VM, which can be downloaded from the SEED website.

```
[10/07/23]seed@VM:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04.1 LTS
Release:        20.04
Codename:       focal
[10/07/23]seed@VM:~$
```

3 Task 1: Frequency Analysis

It is well-known that monoalphabetic substitution cipher (also known as monoalphabetic cipher) is not secure, because it can be subjected to frequency analysis. In this lab, you are given a cipher-text that is encrypted using a monoalphabetic cipher; namely, each letter in the original text is replaced by another letter, where the replacement does not vary (i.e., a letter is always replaced by the same letter during the encryption). Your job is to find out the original text using frequency analysis. It is known that the original text is an English article.

In the following, we describe how we encrypt the original article, and what simplification we have made. Instructors can use the same method to encrypt an article of their choices, instead of asking students to use the ciphertext made by us.

Step 1 : generate encryption key

```
[10/07/23]seed@VM:~/.../Files$ echo "Task 1"
Task 1
[10/07/23]seed@VM:~/.../Files$ nano randomgen.py
[10/07/23]seed@VM:~/.../Files$ cat randomgen.py
#!/bin/env python3
import random
s = "abcdefghijklmnopqrstuvwxyz"
list = random.sample(s, len(s))
key = ''.join(list)
print(key)
[10/07/23]seed@VM:~/.../Files$ python3 randomgen.py
npubslrtvafjzhgmycqiwdkoex
[10/07/23]seed@VM:~/.../Files$
```

With the above steps given in the description we will be using.

Plaintext

```
[10/09/23]seed@VM:~/.../Files$ tr 'ytn' 'THE' <ciphertext.txt> plaintext.txt
[10/09/23]seed@VM:~/.../Files$ cat plaintext.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
```

In the above screenshot we are using tr command to do the decryption.
After typing the cipher text command we can see the cipher.txt in the below
We are changing the cipher text by using the tr in the below screenshot ,we can see that the alphabets were changing .

Screenshot changing from **ytn → THE**

```
[10/09/23]seed@VM:~/.../Files$ tr 'ytn' 'THE' <ciphertext.txt> plaintext.txt
[10/09/23]seed@VM:~/.../Files$ cat plaintext.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
```

In the below screenshot **ytnv —> THEA**

```
[10/09/23]seed@VM:~/.../Files$ tr 'ytnv' 'THEA' <ciphertext.txt> plaintext.txt
[10/09/23]seed@VM:~/.../Files$ cat plaintext.txt
THE xqaAhq Tzhu  xu qzupAd lHmaH qEEcq AgxzT hmrHT AbTEh THmq ixur qThAurE
AlAhpq Thme THE gArrEh bEEiq imsE A uxuArEuAhmAu Txx

THE AlAhpq hAaE lAq gxxsEupEp gd THE pEcmqE xb HAhfEd lEmuqTEmu AT mTq xzTqET
Aup THE AeeAhEuT mceixqmxu xb Hmq bmic axceAud AT THE Eup Aup mT lAq qHAeEp gd
THE EcEhrEuaE xb cETxx TmcEq ze giAasrxlu eximTmaq AhcaAupd AaTmfmqc Aup
A uATmxuAi axufEhqATmxu Aq ghmEb Aup cAp Aq A bEfEh phEAc AgxzT lHETHEh THEhE
xzrHT Tx gE A ehEqmpEuT lmubhEd THE qEAqxu pmpuT ozqT qEEc EkThA ixur mT lAq
EkThA ixur gEaAzqE THE xqaAhq lEhE cxfEp Tx THE bmhqT lEEsEup mu cAhaH Tx
Afxmp axubimaTmur lmTH THE aixqmur aEhEcxud xb THE lmuTEh xidcemaq THAusq
edExuraHAur
```

changed **up** → **ND**

```
[10/09/23]seed@VM:~/.../Files$ tr 'ytnvup' 'THEAND' <ciphertext.txt> plaintext.t
xt
[10/09/23]seed@VM:~/.../Files$ cat plaintext.txt
THE xqaAhq TzhN  xN qzNDAd lHmaH qEEcq AgxzT hmrHT AbTEh THmq ixNr qThANrE
AlAhDq Thme THE gArrEh bEEiq imsE A NxNArENAhmAN Txx
```

Changed **Xb —> OF**

```
[10/09/23]seed@VM:~/.../Files$ tr 'ytnvupxb' 'THEANDOF' <ciphertext.txt> plainte
xt.txt
[10/09/23]seed@VM:~/.../Files$ cat plaintext.txt
THE OqaAhq TzhN  ON qzNDAd lHmaH qEEcq AgOzT hmrHT AFTEh THmq iONr qThANrE
AlAhDq Thme THE gArrEh FEEiq imsE A NONArENAhmAN TOO

THE AlAhDq hAaE lAq gOOsENDED gd THE DEcmqE OF HAhfEd lEmNqTEmN AT mTq OzTqET
AND THE AeeAhENT mceiOqmON OF Hmq Fmic aOceANd AT THE END AND mT lAq qHAeED gd
THE EcEhrENaE OF cETOO TmcEq ze giAasrOlN eOimTmaq AhcaANDd AaTmfmqc AND
A NATmONAi aONfEhqATmON Aq ghmEF AND cAD Aq A FEfEh DhEAc AgOzT lHETHEh THEhE
```

Changed **Hgl —> RBW**

```
[10/09/23]seed@VM:~/.../Files$ tr 'ytnvupxbhgl' 'THEANDOFRBW' <ciphertext.txt> p
laintext.txt
[10/09/23]seed@VM:~/.../Files$ cat plaintext.txt
THE OqaARq TzRN  ON qzNDAd WHmaH qEEcq ABOzT RmrHT AFTER THmq iONr qTRANrE
AWARDq TRme THE BArrER FEEiq imsE A NONArENARmAN TOO

THE AWARDq RAaE WAq BOOsENDED Bd THE DEcmqE OF HARfEd WEmNqTEmN AT mTq OzTqET
AND THE AeeARENT mceiOqmON OF Hmq Fmic aOceANd AT THE END AND mT WAq qHAeED Bd
THE EcERrENaE OF cETOO TmcEq ze BiAasrOWN eOimTmaq ARcaANDd AaTmfmqc AND
A NATmONAi aONfERqATmON Aq BRmEF AND cAD Aq A FEfER DREAc ABOzT WHETHER THERE
OzrHT TO BE A eREqmDENT WmNFREd THE qEAqON DmDNT ozqT qEEc EkTRA iONr mT WAq
EkTRA iONr BEaAzqE THE OqaARq WERE cOfED TO THE FmRqT WEEsEND mN cARaH TO
AfOmD aONFimaTmNr WmTH THE aiOqmNr aEREcONd OF THE WmNTER Oidcemaq THANsq
edEONraHANr
```

Changed **lzq —> WUS**

```
[10/09/23]seed@VM:~/.../Files$ tr 'ytnvupxbhglzq' 'THEANDOFRBWUS' <ciphertext.tx
t> plaintext.txt
[10/09/23]seed@VM:~/.../Files$ cat plaintext.txt
THE OSaARS TURN  ON SUNDAd WHmaH SEEcS ABOUT RmrHT AFTER THmS iONr STRANrE
AWARDS TRme THE BArrER FEEiS imsE A NONArENARmAN TOO
```

Likewise we will be using same technique

```
[10/09/23]seed@VM:~/.../Files$ tr 'ytnvupxbhglzqcamr' 'THEANDOFRBWUSMCIG' <ciphe
rtext.txt> plaintext.txt
[10/09/23]seed@VM:~/.../Files$ tr 'ytnvupxbhglzqcamriefd' 'THEANDOFRBWUSMCIGLPVY
' <ciphertext.txt> plaintext.txt
[10/09/23]seed@VM:~/.../Files$ tr 'ytnvupxbhglzqcamriefdsk' 'THEANDOFRBWUSMCIGLP
VYKX' <ciphertext.txt> plaintext.txt
[10/09/23]seed@VM:~/.../Files$ tr 'ytnvupxbhglzqcamriefdskjw' 'THEANDOFRBWUSMCIG
LPVYKXYK' <ciphertext.txt> plaintext.txt
[10/09/23]seed@VM:~/.../Files$ tr 'ytnvupxbhglzqcamriefdskj' 'THEANDOFRBWUSMCIGL
PVYKX' <ciphertext.txt> plaintext.txt
[10/09/23]seed@VM:~/.../Files$ cat plaintext.txt
THE OSCARS TURN  ON SUNDAY WHICH SEEMS ABOUT RIGHT AFTER THIS LONG STRANGE
AWARDS TRIP THE BAGGER FEELS LIKE A NONAGENARIAN TOO
```

Frequency of file.

After running the ./freq.py

```
[10/07/23]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
```

After changing all , we get the paragraph.

```
[10/09/23]seed@VM:~/.../Files$ cat plaintext.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 OUST 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 OUESTION SURROUNDING THIS YEARS ACADEMY AWARDS IS HOW OR IF THE
CEREMONY WILL ADDRESS METOO ESPECIALLY AFTER THE GOLDEN GLOBES WHICH BECAME
A OUBILANT 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 INEOUITY AFTER
ITS FORMER ANCHOR CATT SADLER OUIT 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 OUST 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 OUDD 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

THE WAY THE ACADEMY TABULATES THE BIG WINNER DOESNT HELP IN EVERY OTHER
CATEGORY THE NOMINEE WITH THE MOST VOTES WINS BUT IN THE BEST PICTURE
CATEGORY VOTERS ARE ASKED TO LIST THEIR TOP MOVIES IN PREFERENTIAL ORDER IF A
MOVIE GETS MORE THAN  PERCENT OF THE FIRSTPLACE VOTES IT WINS WHEN NO
MOVIE MANAGES THAT THE ONE WITH THE FEWEST FIRSTPLACE VOTES IS ELIMINATED AND
```

Task 2: Encryption using Different Ciphers and Modes

In this task, we will play with various encryption algorithms and modes. You can use the following `openssl enc` command to encrypt/decrypt a file. To see the manuals, you can type `man openssl` and `man enc`.

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

Please replace the `ciphertype` with a specific cipher type, such as `-aes-128-cbc`, `-bf-cbc`, `-aes-128-cfb`, etc. In this task, you should try at least 3 different ciphers. You can find the meaning of the command-line options and all the supported cipher types by typing "`man enc`". We include some common options for the `openssl enc` command in the following:

```
-in <file>     input file
-out <file>    output file
-e             encrypt
-d             decrypt
-K/-iv         key/iv in hex is the next argument
-[pP]          print the iv/key (then exit if -P)
```

Here we are using different ciphers and modes

1. AES Cipher - CBC Mode

1. AES is a Symmetric-key encryption algorithm, utilizing a 128-bit key for both encryption and decryption.
2. In Cipher Block Chaining (CBC) mode, the plaintext is divided into fixed 128-bit blocks, and each block is XORed with the previous ciphertext block before encryption, enhancing security through chaining.

Commands used :

openssl enc -aes-128-cbc -e -in plain.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

Replaced the cipher type with a specific cipher type as -aes-128-cbc

```
[10/08/23]seed@VM:~/.../Files$ echo " Task2"
 Task2
[10/08/23]seed@VM:~/.../Files$ cat plain.txt
On the other hand, we denounce with righteous indignation and dislike men who are so beguiled and demoralized by the ch
arms of pleasure of the moment, so blinded by desire, that they cannot foresee the pain and trouble that are bound to e
nsue; and equal blame belongs to those who fail in their duty through weakness of will, which is the same as saying thr
ough shrinking from toil and pain. These cases are perfectly simple and easy to distinguish. In a free hour, when our p
ower of choice is untrammelled and when nothing prevents our being able to do what we like best, every pleasure is to b
e welcomed and every pain avoided. But in certain circumstances and owing to the claims of duty or the obligations of b
usiness it will frequently occur that pleasures have to be repudiated and annoyances accepted. The wise man therefore a
lways holds in these matters to this principle of selection: he rejects pleasures to secure other greater pleasures, or
 else he endures pains to avoid worse pains.
[10/08/23]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher.bin -K 00112233445566778889aabbccd
deeff -iv 0102030405060708
```

In the Above screenshot the plain text given some paragraph took from somewhere

.

Hexdump cipher.bin

```
[10/08/23]seed@VM:~/.../Files$ hexdump cipher.bin
0000000 4557 43a4 9535 60dc 79b1 b0ce 2bf1 d5bf
0000010 03d5 5515 3571 e75e 02d5 ce4e c5bd b59f
0000020 c07b 8a3f ea01 231c ad99 0b5a cb79 3b76
0000030 d652 38f2 8d42 cd8b 83c9 cac5 3dd8 e19c
0000040 1cb4 bc83 b56a 310c 2149 37a9 7571 5d7c
0000050 78a9 af82 4571 fc42 57a5 44ae 9def 74a0
0000060 9c8f 8532 22b1 9bd8 f7cb 23fd 403b 1a11
0000070 8544 4e94 8dcf beb0 4b9c 6d9e 7a58 5b0f
0000080 e865 8717 f2ad 634b 5912 c36c ad18 4ee6
0000090 8511 e56b e627 f670 aa6c 5998 e286 87a5
00000a0 f677 7c3d 8798 88c7 e769 ac56 e9e4 3b16
00000b0 33d5 57b1 64f7 852b d546 be01 5d4b e079
00000c0 5b3b ed6e 5037 98ef d489 3aa1 d020 826b
00000d0 bb1c 89b4 71c4 a59c f218 df73 c878 eb55
00000e0 b0d4 cf40 6097 d3e8 e36c dfb7 72a6 3b0f
00000f0 0693 c5eb 31d4 f814 6ead 4edc 812d 63f7
0000100 cbe5 8eb6 5200 efac 62df 855f 3fd8 180c
0000110 49fc 8086 03bc d2f2 37b7 639e afe9 9e29
0000120 5ad4 9071 d252 bcc7 19c1 0b61 3c65 5aff
0000130 6e66 70cd e00d 3ec7 00ae 39b7 53a3 8192
0000140 85c6 44aa 5923 b0cb c044 19e7 63e4 27de
0000150 4377 e488 ada5 97b6 4926 1908 7033 8fd4
```

We can see the alphanumerics.

2.. AES Cipher - Cfb Mode

1. AES Cipher in CFB mode allows for the encryption of individual units smaller than the block size, typically 8 bits (1 byte) at a time.
2. In CFB mode, the previous ciphertext segment is encrypted and then XORed with the plaintext to produce the current ciphertext, creating a self-synchronizing stream cipher.

Command used :

openssl enc -aes-128-cfb -e -in plain.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

Below screenshot is about the running the command

```
[10/08/23]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in plain.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 01020304050
60708
```

After typing hexdump cipher.bin

```
[10/08/23]seed@VM:~/.../Files$ hexdump cipher.bin
0000000 e8c8 51af 5aa9 be9a f6db 6fda 7855 033d
0000010 7d4d 1bcd abd8 8ad0 b103 70e5 7dc0 ea83
0000020 d2d4 2816 bca8 3886 3460 3289 0a72 964c
0000030 05c3 e9dc 773c 3650 5462 9313 05a7 e223
0000040 0211 5e1d 5e2a fdab d7e9 ff64 5597 d6a5
0000050 9ec8 cb5b 7049 1940 f955 f21f c6c4 6264
0000060 1e78 2902 8355 4b42 c0e1 8c0f 44db d66d
0000070 2476 f6da 805e fa98 e9f8 cae5 8498 f729
0000080 a6c6 2fe2 94c3 99ac 29d9 4a78 26a7 ee66
0000090 1f49 75e5 05ae 577c 51e4 77be 45d2 4182
00000a0 678f 9002 d92a f6cb c4f9 16d5 6950 a310
00000b0 4b5f 4a30 db9f 3dfa c317 8da5 3470 80e3
00000c0 8eca 6f6a f47a 8231 0471 a1f9 fcb5 0d05
00000d0 57df 3235 9786 05eb 7d2b f526 8f08 cfdb
00000e0 dbc7 b71e 3ae4 8b74 a148 b91b 5e9b d3c2
00000f0 7fbe d6bc 20bd 82bd ea29 5f4a 0a10 9948
0000100 0ade 8b4a 7ba3 9b13 ccc1 00bd 6a9e 2298
0000110 9df5 1eef 5ef7 5f16 f629 2daf ff76 d5c5
0000120 0879 b97d 4e0e 925b 7cbd 73df 73ee a103
0000130 65e7 8dea c502 ea2a 21b3 de4d 0dc1 6e95
0000140 dfc8 8222 f541 df6a a9cb 38a8 48ee 5a0c
0000150 d146 0339 2c5d cb9b 28dc 0476 1485 cbc4
```

We can see the change of numbers for each cipher mode

3. Cipher -bf-cbc

- Cipher Block Feedback (BF-CBC) mode is a cryptographic mode that operates on data in fixed-size blocks, typically 64 bits, and uses a feedback mechanism to create a chain effect.
- In BF-CBC mode, each ciphertext block is XORed with the previous ciphertext block before encryption, enhancing security through feedback

Command used :

openssl enc -bf-cbc -e -in plain.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

```
[10/08/23]seed@VM:~/.../Files$ openssl enc -bf-cbc -e -in plain.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 010203
0405060708
[10/08/23]seed@VM:~/.../Files$ hexdump cipher.bin
0000000 fb27 52be 8ead bc52 6ff5 a0c0 d02c 5a5f
0000010 d030 5858 3d22 5754 6a3c d90f 4f7a 922a
0000020 df69 0ec4 3421 1234 4eb3 fdf4 2085 0a7c
0000030 ef16 0a03 840a 6801 ec65 3a15 ca85 9d7f
0000040 5ef9 9977 0173 117a aa33 d002 876b 3def
0000050 8271 65e3 598c 5783 7046 0425 1c61 1835
0000060 0726 fd28 ef5f 3173 eed6 5ac7 85bc cf10
0000070 ab3d 60b0 3c37 9cb1 db74 336d 60c5 1678
0000080 94b9 9609 c080 4737 e927 05e6 663a 5ff9
0000090 87e9 b7fb 58cf 7912 f3aa 7937 fba0 f026
00000a0 e035 e077 d127 60e8 2002 b8b9 a7d0 b311
00000b0 1e93 9c8d 5c3a 1351 a552 79db c5a8 ff0d
00000c0 5c1f 64cb 8ae6 c27b 2656 9511 026c 6710
00000d0 2cf2 e0ae c53b e286 5820 6d02 9413 9366
00000e0 e24c 6a8f 7871 a59c f1ea 4a5a 2c8b ca5d
00000f0 cae6 5b9f de9e b89b d63c 5d27 8b8d bac5
0000100 a2b9 195c 34e8 ea46 8cee 43a6 acbf 16f9
0000110 3f15 6611 9a20 5406 6dab 6f39 96f2 1f50
0000120 fd90 0c5f 1583 196f 2220 6c6a 3f12 8aa5
0000130 dc3f 49b5 07f3 f944 4870 97ea 0a3c b965
0000140 c5ac 09fe 6c94 3860 c373 2e5e 7486 9a57
0000150 6cfd d705 dcb0 e33b 8cc4 e966 c63b 1e09
```

In the above screen shot we can see the encrypted code and hexdump cipher.bin

Task 3: Encryption Mode – ECB vs. CBC

Image which was given in the lab setup file.

Encryption Mode – ECB

1. Electronic Codebook (ECB) mode is a basic encryption mode used for encrypting data.
2. In ECB mode, each plaintext block is independently encrypted with the same encryption key, which means identical plaintext blocks will produce identical ciphertext blocks. This mode is not suitable for encrypting large amounts of data with patterns, as it does not provide the same security as other modes like CBC or GCM.

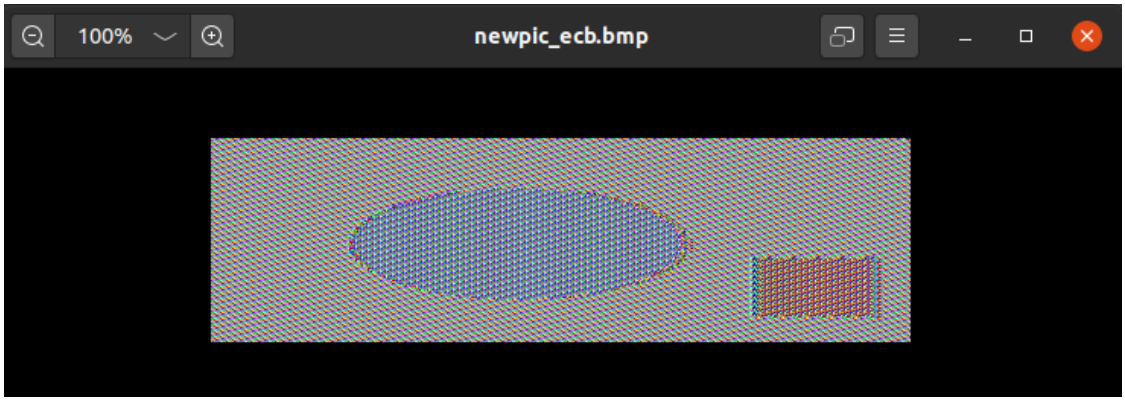
Command used :

openssl enc aes-128-ebc -e -in pic_original.bmp -out pic_encrypted.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708

```
[10/08/23]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out pic_ecb.bmp -K 00112233445566778889aabbccddee
ff
[10/08/23]seed@VM:~/.../Files$ head -c 54 pic_original.bmp > header
[10/08/23]seed@VM:~/.../Files$ tail -c +55 pic_ecb.bmp > body
[10/08/23]seed@VM:~/.../Files$ cat header body_ecb > newpic_ecb.bmp
[10/08/23]seed@VM:~/.../Files$
[10/08/23]seed@VM:~/.../Files$ eog newpic_ecb.bmp
```

In the above screenshot we are using Electronic Codebook (ECB) mode

The visibility of the image indicates that this encryption mode is not effective.



Observation :
Even after encryption, the shapes and hue colors of the images in the encrypted picture remain strikingly similar to the original, creating an outline image that closely resembles the original. As a result, the encryption does not appear to provide significant security, as the content can still be interpreted from the encrypted image itself.

Encryption Mode – CBC (Cipher Block Chaining)

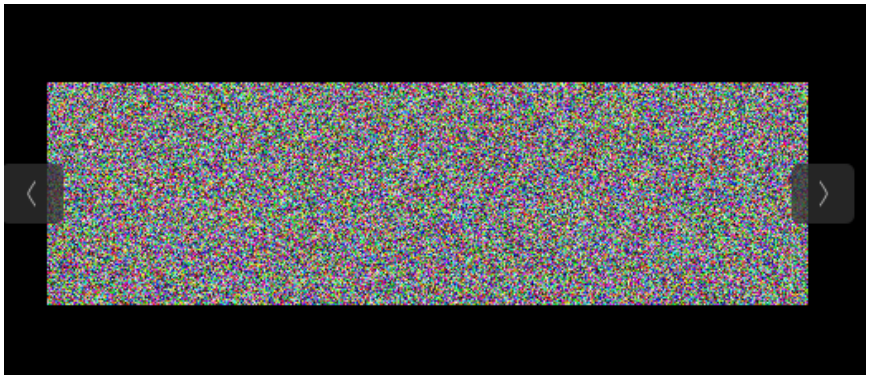
CBC (Cipher Block Chaining) encryption mode is commonly used for enhancing the security of data transmission and storage.

Command used :

openssl enc -bf-cbc -e -in pic_original.bmp -out pic_encrypted.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708

```
[10/08/23]seed@VM:~/.../Files$ openssl enc -bf-cbc -e -in pic_original.bmp -out pic_cbc.bmp -K 00112233445566778889aabbccddeeff -i
v 0102030405060708
[10/08/23]seed@VM:~/.../Files$ eog pic_cbc.bmp

(eog:5281): EOG-WARNING **: 22:49:37.781: Thumbnail creation failed
[10/08/23]seed@VM:~/.../Files$ head -c 54 pic_cbc.bmp > header
[10/08/23]seed@VM:~/.../Files$ tail -c +55 pic_cbc.bmp > body
[10/08/23]seed@VM:~/.../Files$ cat header cbcbody > new.bmp
[10/08/23]seed@VM:~/.../Files$ ego new.bmp

Command 'ego' not found, but can be installed with:

sudo snap install ego-dev

[10/08/23]seed@VM:~/.../Files$ eog new.bmp
```

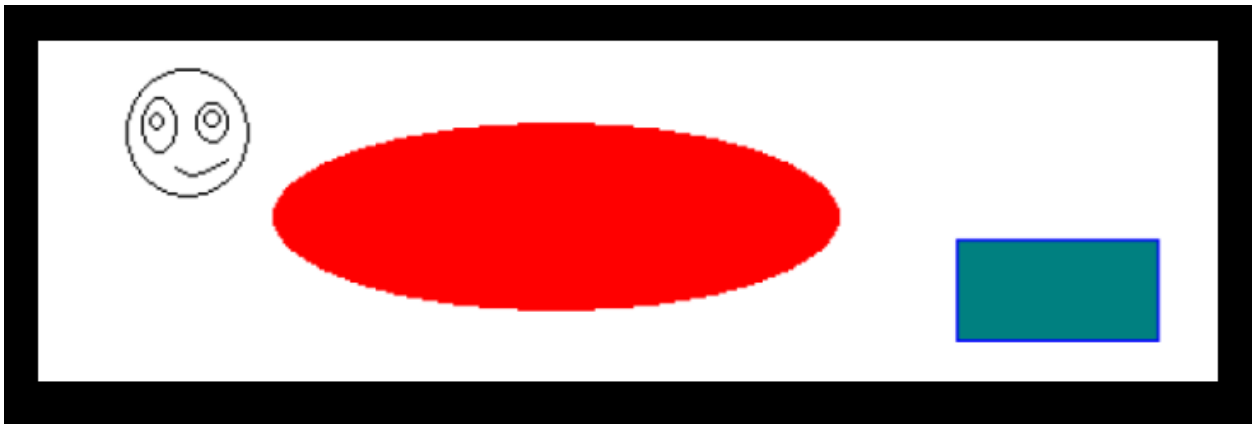
In the above screenshot we are using Encryption Mode – CBC (Cipher Block Chaining)



In this mode , we can't see the image properly not even visible.

2. Display the encrypted picture using a picture viewing program (we have installed an image viewer program called `eog` on our VM). Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations.

Select a picture of your choice, repeat the experiment above, and report your observations.

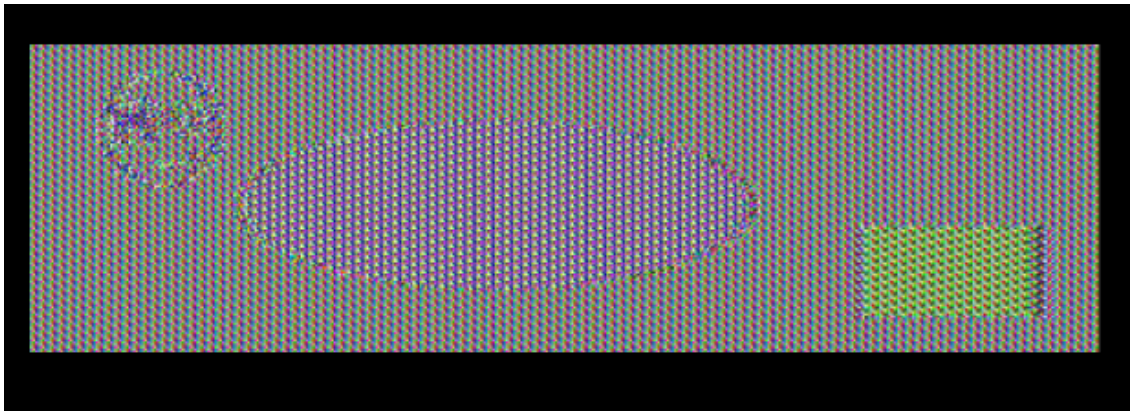


There was a slight difference in the image I made and i runned the experiment.

Encryption Mode – ECB

```
[10/08/23]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in pic_org.bmp -out pic_ecb.bmp -K 00112233445566778889aabbccddeeff
[10/08/23]seed@VM:~/.../Files$ head -c 54 pic_org.bmp > header
[10/08/23]seed@VM:~/.../Files$ tail -c +55 pic_ecb.bmp > body
[10/08/23]seed@VM:~/.../Files$ cat header body_ecb > newpic_ecb.bmp
[10/08/23]seed@VM:~/.../Files$ eog newpic_ecb.bmp
```

Image after encryption



Observation :
Even after encryption, the shapes and hue colors of the images in the encrypted picture remain strikingly similar to the original, creating an outline image that closely resembles the original. As a result, the encryption does not appear to provide significant security, as the content can still be interpreted from the encrypted image itself.

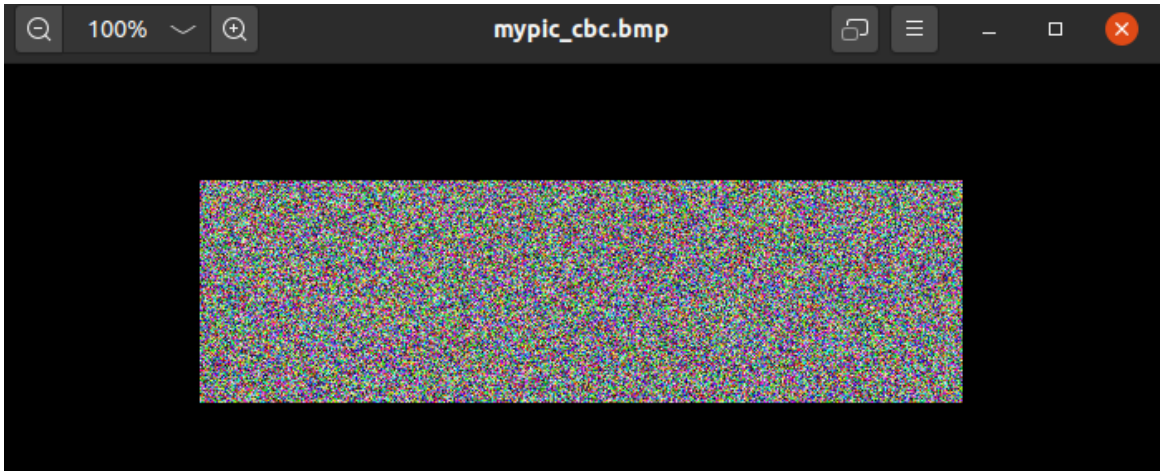
Encrypt file using cbc (Cipher Block Chaining)

```
[10/08/23]seed@VM:~/.../Files$ openssl enc -sm4-cbc -e -in img.bmp -out mypic_cbc.bmp -K 00112233445566778889aabbccddeeff -iv 0102
030405060708
hex string is too short, padding with zero bytes to length
[10/08/23]seed@VM:~/.../Files$ tail -c +55 mypic_cbc.bmp > body2
[10/08/23]seed@VM:~/.../Files$ cat header1 body2 > mypic_cbc.bmp
```

After typing the command to open eog i used for opening file

```
[10/08/23]seed@VM:~/.../Files$ eog mypic_cbc.bmp
```

In the below screenshot we can see the image was blur



Task 4: Padding

For block ciphers, when the size of a plaintext is not a multiple of the block size, padding may be required. The PKCS#5 padding scheme is widely used by many block ciphers (see Chapter 21.4 of the SEED book for details). We will conduct the following experiments to understand how this type of padding works:

1. Use ECB, CBC, CFB, and OFB modes to encrypt a file (you can pick any cipher). Please report which modes have paddings and which ones do not. For those that do not need paddings, please explain why.

2. Let us create three files, which contain 5 bytes, 10 bytes, and 16 bytes, respectively. We can use the following "`echo -n`" command to create such files. The following example creates a file `f1.txt` with length 5 (without the `-n` option, the length will be 6, because a newline character will be added by `echo`):

```
$ echo -n "12345" > f1.txt
```

As described in the lab descriptionf1, f2, f3 files were created

```
[10/09/23]seed@VM:~/.../Files$ echo -n "12345" > f1.txt
[10/09/23]seed@VM:~/.../Files$ echo -n "1234567" > f2.txt
[10/09/23]seed@VM:~/.../Files$ echo -n "1234567890" > f3.txt
[10/09/23]seed@VM:~/.../Files$ ls -l f*
-rw-rw-r-- 1 seed seed   5 Oct  9 00:04 f1.txt
-rw-rw-r-- 1 seed seed   7 Oct  9 00:04 f2.txt
-rw-rw-r-- 1 seed seed  10 Oct  9 00:04 f3.txt
-rwxrwxr-x 1 seed seed 786 Nov  7  2021 freq.py
[10/09/23]seed@VM:~/.../Files$
```

Cbc AES - Padded Mode

Purpose:

CBC AES with padding mode is commonly used for encrypting data, especially when the data size is not a multiple of the block size (128 bits for AES). This mode serves the following purposes

In the below screenshot we have used commands for encryption

```
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f1.txt -nosalt -out cipherf1.txt -K 00112233445566778889aabbccddeef
f -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f2.txt -nosalt -out cipherf2.txt -K 00112233445566778889aabbccddeef
f -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f3.txt -nosalt -out cipherf3.txt -K 00112233445566778889aabbccddeef
f -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ ls -l cipherf*
-rw-rw-r-- 1 seed seed 16 Oct  9 00:10 cipherf1.txt
-rw-rw-r-- 1 seed seed 16 Oct  9 00:10 cipherf2.txt
-rw-rw-r-- 1 seed seed 16 Oct  9 00:11 cipherf3.txt
[10/09/23]seed@VM:~/.../Files$
```

Decryption :

```
[10/09/23]seed@VM:~/.../Files$ ls -l cipherf*
-rw-rw-r-- 1 seed seed 16 Oct  9 00:10 cipherf1.txt
-rw-rw-r-- 1 seed seed 16 Oct  9 00:10 cipherf2.txt
-rw-rw-r-- 1 seed seed 16 Oct  9 00:11 cipherf3.txt
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ echo "DEcryption"
DEcryption
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -nopad -in cipherf1.txt -out padf1.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -nopad -in cipherf2.txt -out padf2.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -nopad -in cipherf3.txt -out padf3.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$
```

Hex dump : aes -cbc hexdump

```
[10/09/23]seed@VM:~/.../Files$ hexdump -C padf1.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345...........|
00000010
[10/09/23]seed@VM:~/.../Files$ xxd padf1.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345...........
[10/09/23]seed@VM:~/.../Files$ hexdump -C padf2.txt
00000000  31 32 33 34 35 36 37 09  09 09 09 09 09 09 09 09  |1234567.........|
00000010
[10/09/23]seed@VM:~/.../Files$ xxd padf2.txt
00000000: 3132 3334 3536 3709 0909 0909 0909 0909  1234567.........
[10/09/23]seed@VM:~/.../Files$ hexdump -C padf3.txt
00000000  31 32 33 34 35 36 37 38  39 30 06 06 06 06 06 06  |1234567890......|
00000010
[10/09/23]seed@VM:~/.../Files$ xxd padf3.txt
00000000: 3132 3334 3536 3738 3930 0606 0606 0606  1234567890......
[10/09/23]seed@VM:~/.../Files$
```

2. ECB - PADDED MODE

ECB (Electronic Codebook) mode with padding is used for encrypting data, particularly when confidentiality is the primary concern, and the data size is a multiple of the block size (128 bits for AES)

Below is the screenshot of encryption

```
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in f1.txt -out aesecbcipherf3.bin -K 00112233445566778889aabbccddeeff
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in f1.txt -out aesecbcipherf1.bin -K 00112233445566778889aabbccddeeff
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in f2.txt -out aesecbcipherf2.bin -K 00112233445566778889aabbccddeeff
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in f3.txt -out aesecbcipherf3.bin -K 00112233445566778889aabbccddeeff
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ ls -l aesecbcipherf*
-rw-rw-r-- 1 seed seed 16 Oct  9 00:30 aesecbcipherf1.bin
-rw-rw-r-- 1 seed seed 16 Oct  9 00:30 aesecbcipherf2.bin
-rw-rw-r-- 1 seed seed 16 Oct  9 00:31 aesecbcipherf3.bin
[10/09/23]seed@VM:~/.../Files$
```

Below screenshot is the Decryption.and hexdump also we can the values

```
[10/09/23]seed@VM:~/.../Files$ echo "Decryption"
Decryption
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -d -nopad -in aesebccipherf1.bin -out aesecbf1.txt -K 00112233445566778889aabbccddeeff
Can't open aesebccipherf1.bin for reading, No such file or directory
140462918194496:error:02001002:system library:fopen:No such file or directory:crypto/bio/bss_file.c:69:fopen('aesebccipherf1.bin','rb')
140462918194496:error:2006D080:BIO routines:BIO_new_file:no such file:crypto/bio/bss_file.c:76:
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -d -nopad -in aesecbcipherf1.bin -out aesecbf1.txt -K 00112233445566778889aabbccddeeff
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -d -nopad -in aesecbcipherf2.bin -out aesecbf2.txt -K 00112233445566778889aabbccddeeff
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -d -nopad -in aesecbcipherf3.bin -out aesecbf3.txt -K 00112233445566778889aabbccddeeff
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ ls -l aesecbf*
-rw-rw-r-- 1 seed seed 16 Oct  9 00:36 aesecbf1.txt
-rw-rw-r-- 1 seed seed 16 Oct  9 00:36 aesecbf2.txt
-rw-rw-r-- 1 seed seed 16 Oct  9 00:36 aesecbf3.txt
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ hexdump -C aesecbf1.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345...........|
00000010
[10/09/23]seed@VM:~/.../Files$ xxd aesecbf1.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345...........
[10/09/23]seed@VM:~/.../Files$ hexdump -C aesecbf2.txt
00000000  31 32 33 34 35 36 37 09  09 09 09 09 09 09 09 09  |1234567.........|
00000010
[10/09/23]seed@VM:~/.../Files$ xxd aesecbf2.txt
00000000: 3132 3334 3536 3709 0909 0909 0909 0909  1234567.........
[10/09/23]seed@VM:~/.../Files$ hexdump -C aesecbf3.txt
00000000  31 32 33 34 35 36 37 38  39 30 06 06 06 06 06 06  |1234567890......|
00000010
[10/09/23]seed@VM:~/.../Files$ xxd aesecbf3.txt
00000000: 3132 3334 3536 3738 3930 0606 0606 0606  1234567890......
[10/09/23]seed@VM:~/.../Files$
```

3. Ofb - non padded mode

OFB (Output Feedback) mode in non-padded form is used for encrypting data in situations where confidentiality and data integrity are the primary concerns, and data sizes don't necessarily align with block boundaries.

encryption

```
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in f1.txt -out ofccipherf1.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in f2.txt -out ofccipherf2.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in f3.txt -out ofccipherf3.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$
```

Decryption and hexdump.

```
[10/09/23]seed@VM:~/.../Files$ echo "decryption"
decryption
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in ofccipherf1.bin -out ofcpf1.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in ofccipherf2.bin -out ofcpf2.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in ofccipherf3.bin -out ofcpf3.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ ls -l ofccpf*
ls: cannot access 'ofccpf*': No such file or directory
[10/09/23]seed@VM:~/.../Files$ ls -l ofccpf f*
ls: cannot access 'ofccpf': No such file or directory
-rw-rw-r-- 1 seed seed   5 Oct  9 00:04 f1.txt
-rw-rw-r-- 1 seed seed   7 Oct  9 00:04 f2.txt
-rw-rw-r-- 1 seed seed  10 Oct  9 00:04 f3.txt
-rwxrwxr-x 1 seed seed 786 Nov  7  2021 freq.py
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ hexdump -C ofcpf1.txt
00000000  31 32 33 34 35                                    |12345|
00000005
[10/09/23]seed@VM:~/.../Files$ xxd ofcpf1.txt
00000000: 3132 3334 35                             12345
[10/09/23]seed@VM:~/.../Files$ hexdump -C ofcpf2.txt
00000000  31 32 33 34 35 36 37                              |1234567|
00000007
[10/09/23]seed@VM:~/.../Files$ xxd ofcpf2.txt
00000000: 3132 3334 3536 37                        1234567
[10/09/23]seed@VM:~/.../Files$ hexdump -C ofcpf3.txt
00000000  31 32 33 34 35 36 37 38  39 30                    |1234567890|
0000000a
[10/09/23]seed@VM:~/.../Files$ xxd ofcpf3.txt
00000000: 3132 3334 3536 3738 3930                 1234567890
[10/09/23]seed@VM:~/.../Files$
```

4. Cfb - non padded Mode

Ciphertext Feedback (CFB) mode in non-padded form is primarily used for encrypting data in scenarios where confidentiality is the primary concern, and data sizes may not align with block boundaries.

Encryption

```
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in f1.txt -out cfbcipherf1.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in f2.txt -out cfbcipherf2.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in f3.txt -out cfbcipherf3.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ ech0 "Decrypt"
```

Decrypt

```
[10/09/23]seed@VM:~/.../Files$ echo "Decrypt"
Decrypt
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in cfbcipherf1.bin -out cfbpf1.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in cfbcipherf2.bin -out cfbpf2.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in cfbcipherf3.bin -out cfbpf3.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ ls -l cfbpff*
ls: cannot access 'cfbpff*': No such file or directory
[10/09/23]seed@VM:~/.../Files$ ls -l cfbpf*
-rw-rw-r-- 1 seed seed  5 Oct  9 00:59 cfbpf1.txt
-rw-rw-r-- 1 seed seed  7 Oct  9 01:00 cfbpf2.txt
-rw-rw-r-- 1 seed seed 10 Oct  9 01:00 cfbpf3.txt
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ hexdump -C cfbpf1.txt
00000000  31 32 33 34 35                                    |12345|
00000005
[10/09/23]seed@VM:~/.../Files$ xxd cfbpf1.txt
00000000: 3132 3334 35                             12345
[10/09/23]seed@VM:~/.../Files$ hexdump -C cfbpf2.txt
00000000  31 32 33 34 35 36 37                             |1234567|
00000007
[10/09/23]seed@VM:~/.../Files$ xxd cfbpf2.txt
00000000: 3132 3334 3536 37                        1234567
[10/09/23]seed@VM:~/.../Files$ hexdump -C cfbpf3.txt
00000000  31 32 33 34 35 36 37 38  39 30                   |1234567890|
0000000a
[10/09/23]seed@VM:~/.../Files$ xxd cfbpf3.txt
00000000: 3132 3334 3536 3738 3930                 1234567890
[10/09/23]seed@VM:~/.../Files$
```

Observation :

Cbc and ecb are padded modes where the cfb and ofb are non padded

Because of the reasons below

1. CFB and OFB operate at the bit or byte level, allowing individual plaintext bits or bytes to be encrypted without the need for padding. Data alignment to block sizes is unnecessary.
2. These modes use a feedback mechanism that generates a keystream. By XORing this keystream with the plaintext, every bit or byte of plaintext is processed separately, eliminating the need for padding.

3. CFB and OFB effectively transform block ciphers into stream ciphers, designed to encrypt data continuously as it arrives. This inherent capability accommodates data of any length without padding.
4. Ideal for real-time encryption and data transmission scenarios where data flows continuously. CFB and OFB can encrypt data as it's generated or transmitted, without requiring padding for block alignment.

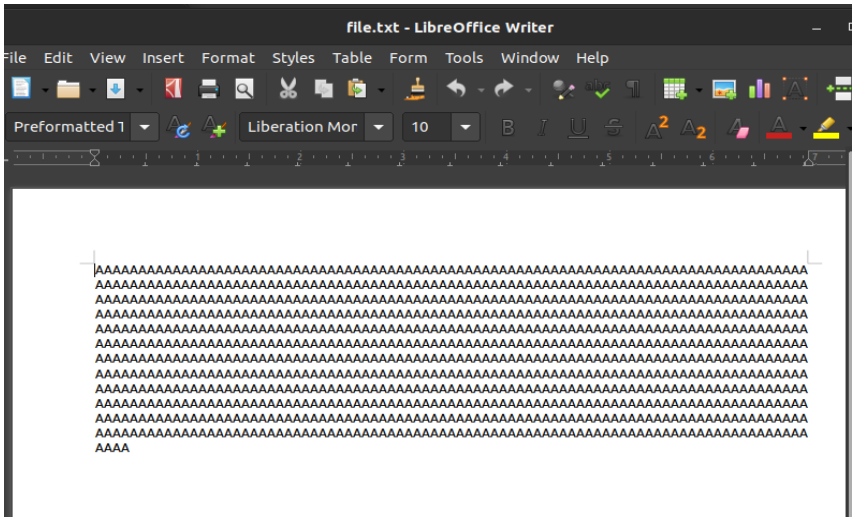
Task 5: Error Propagation – Corrupted Cipher Text

1. Create a text file that is at least 1000 bytes long.
2. Encrypt the file using the AES-128 cipher.
3. Unfortunately, a single bit of the 55th byte in the encrypted file got corrupted. You can achieve this corruption using the `bless` hex editor.
4. Decrypt the corrupted ciphertext file using the correct key and IV.

Please answer the following question: How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task. Please provide justification.

In the below screen shot we were creating the file.txt consists of “A “ 1000 bytes

```
[10/09/23]seed@VM:~/.../Files$ perl -E 'say "A" x 1000' > file.txt
[10/09/23]seed@VM:~/.../Files$ ls -l file.txt
-rw-rw-r-- 1 seed seed 1001 Oct  9 01:10 file.txt
```



Above screenshot is the file.txt.

1. ECB ERROR PROPAGATION

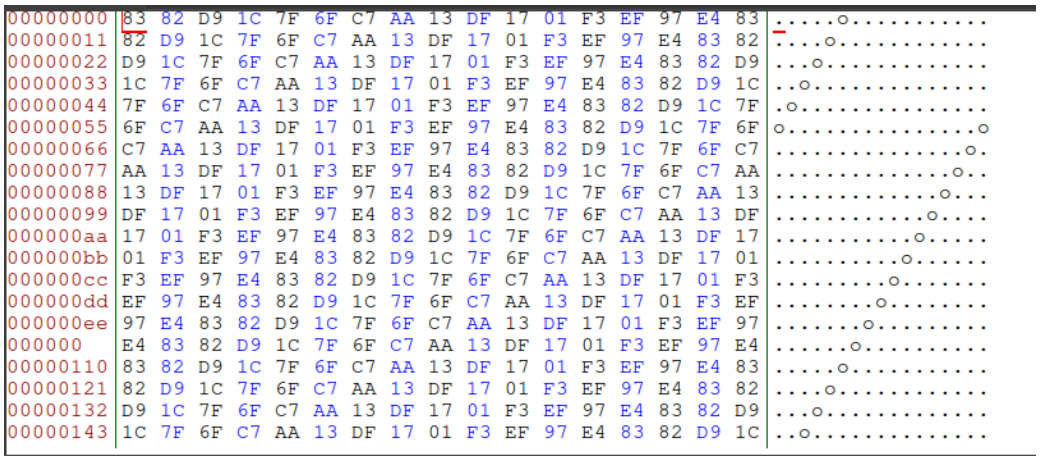
In ECB mode, corruption is limited to the single affected block, and all other blocks can still be recovered.Error propagation is minimal as each block of plaintext is encrypted independently in ECB mode.

Below screenshot is the command used for ecb

```
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in file.txt -out fileecb.bin -K 00112233445566778889aabbccddeeff
```

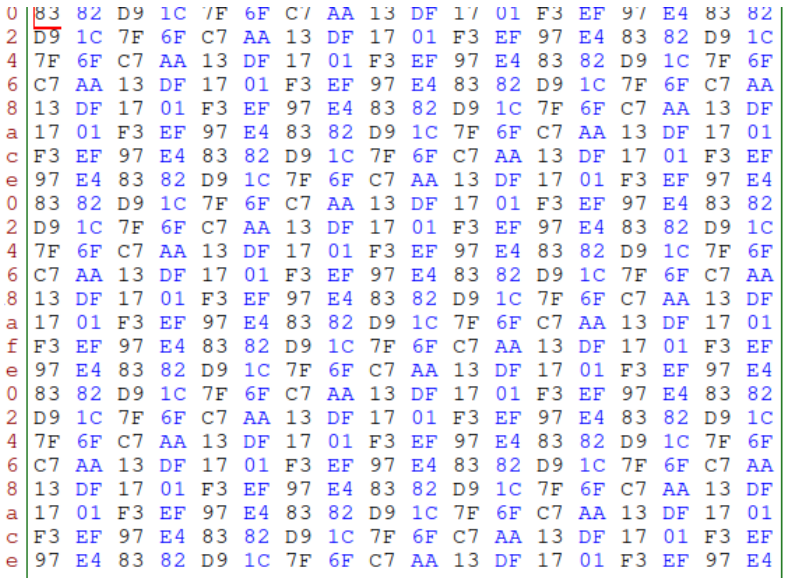
ECB Encryption

Changing 55th bit from A1 to A 2



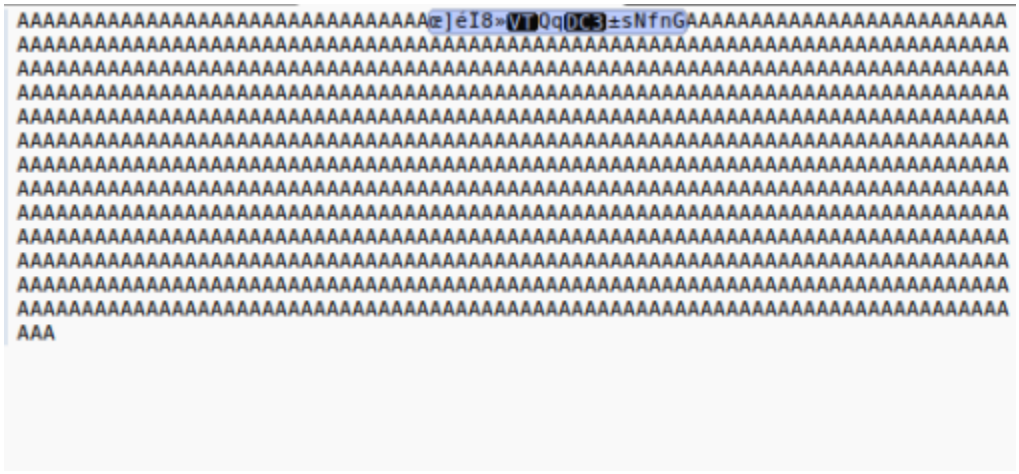
```
00000000 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 .....o...........
00000011 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 ....o...........
00000022 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 ...o.............
00000033 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C ..o..............
00000044 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F .o...............
00000055 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F o...............o
00000066 C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 ...............o.
00000077 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA ...............o..
00000088 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 ...............o...
00000099 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF ...............o....
000000aa 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 ...........o.....
000000bb 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 ...........o......
000000cc F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 .........o.......
000000dd EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF ........o........
000000ee 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 .......o.........
000000   E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 ......o..........
00000110 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 .....o...........
00000121 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 ....o............
00000132 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 ...o.............
00000143 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C ..o..............
```

Decrypted file



```
0 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82
2 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C
4 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F
6 C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA
8 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF
a 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01
c F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF
e 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4
0 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82
2 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C
4 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F
6 C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA
8 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF
a 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01
f F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF
e 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4
0 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82
2 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C
4 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F
6 C7 AA 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA
8 13 DF 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF
a 17 01 F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01
c F3 EF 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF
e 97 E4 83 82 D9 1C 7F 6F C7 AA 13 DF 17 01 F3 EF 97 E4
```

On decryption, one block of data (16 bytes) appears to be corrupted, with 16 characters that differ from the original input.

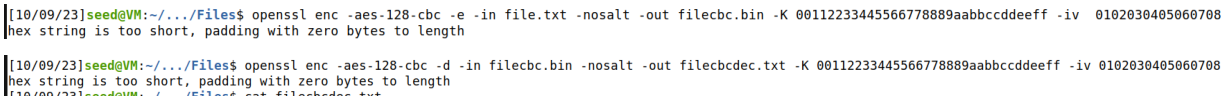


Above screen shot is the text file

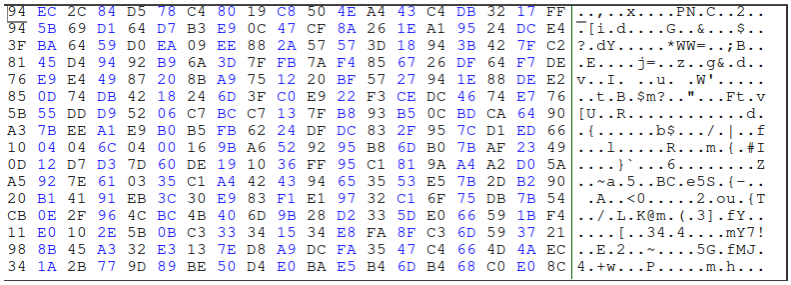
2.CBC ERROR PROPAGATION:

CBC (Cipher Block Chaining) is a mode of operation for block ciphers in cryptography. One of its important characteristics is its handling of error propagation. Error propagation refers to how errors or changes in the ciphertext affect the decryption process and the resulting plaintext.

Cbc encryption
Below commands used for encryption



Decrypted screenshot were attached



Changing 55th bite

```
[10/09/23]seed@VM:~/.../Files$ cat filecbcdec.txt
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA>?91?m??tAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Text file

3. CFB ERROR PROPAGATION

CFB (Cipher Feedback) mode is a mode of operation for block ciphers in cryptography. It has its own characteristics when it comes to error propagation

Encrypt file :

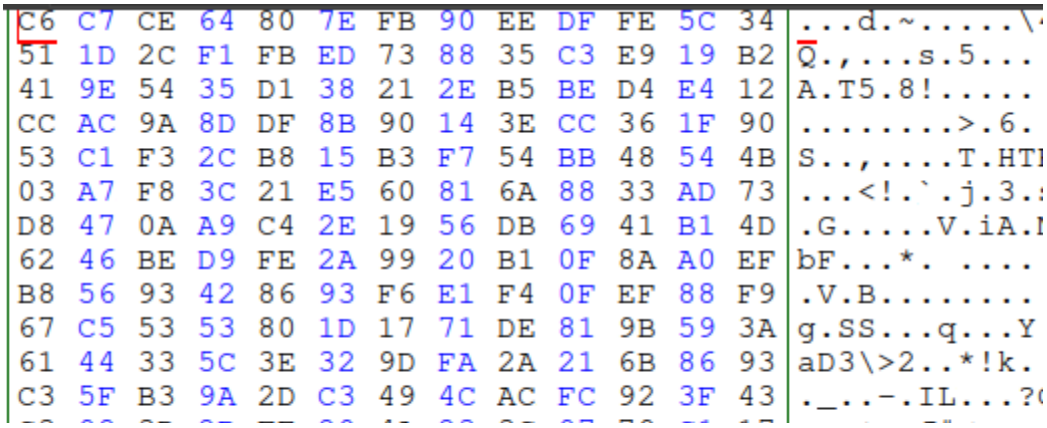
Below commands are for both encryption and decryption

```
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in file.txt -out filecfb.bin -K 00112233445566778889aabbccddeeff -iv  0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in filecfb.bin -out filecfbdec.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
```

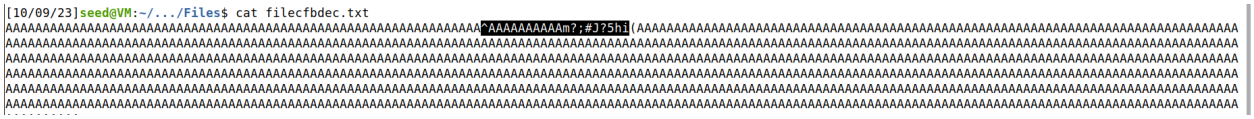
Encryption Changed the 55th bite

```
00000000 C6 C7 CE 64 80 7E FB 90 EE DF FE 5C 34 ...d.~.....\4
0000000d 51 1D 2C F1 FB ED 73 88 35 C3 E9 19 B2 Q.,...s.5....
0000001a 41 9E 54 35 D1 38 21 2E B5 BE D4 E4 12 A.T5.8!......
00000027 CC AC 9A 8D DF 8B 90 14 3E CC 36 1F 90 ........>.6..
00000034 53 C1 F3 2C B8 15 B3 F7 54 BB 48 54 4B S..,....T.HTK
00000041 03 A7 F8 3C 21 E5 60 81 6A 88 33 AD 73 ...<!.`.j.3.s
0000004e D8 47 0A A9 C4 2E 19 56 DB 69 41 B1 4D .G.....V.iA.M
0000005b 62 46 BE D9 FE 2A 99 20 B1 0F 8A A0 EF bF...*. .....
00000068 B8 56 93 42 86 93 F6 E1 F4 0F EF 88 F9 .V.B.........
00000075 67 C5 53 53 80 1D 17 71 DE 81 9B 59 3A g.SS...q...Y:
00000082 61 44 33 5C 3E 32 9D FA 2A 21 6B 86 93 aD3\>2..*!k..
0000008f C3 5F B3 9A 2D C3 49 4C AC FC 92 3F 43 ._..-.IL...?C
0000009c C3 98 3D 2B FE 20 4A 23 3C 97 79 C1 17 ..=+. J#<.y..
000000a9 A7 BD 89 D6 13 F8 2C 02 1F 72 57 C3 DC ......,..rW..
000000b6 82 5F 84 09 16 42 7D 7C 1A 4E BE 8E E8 ._...B}|.N...
000000c3 E7 65 6A 66 B5 96 75 2B A5 B2 96 45 25 .ejf..u+...E%
```

Decryption



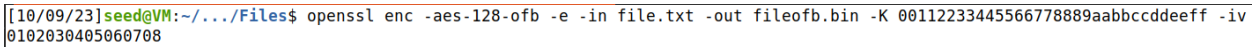
Text file

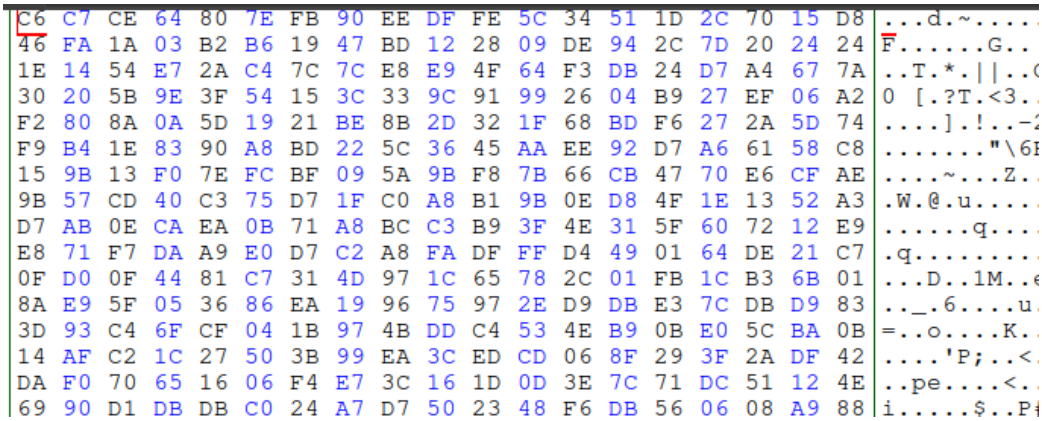


4. OFB ERROR PROPAGATION

Output Feedback (OFB) mode is a mode of operation for block ciphers in cryptography. It has its own characteristics when it comes to error propagation

Encryption command





Decryption command

```
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in fileofb.bin -out fileofbdec.txt -K 0011223344556677
8889aabbccddeeff -iv 0102030405060708
```

```
C6 C7 CE 64 80 7E FB 90 EE DF FE 5C 34 51 1D 2C 70 15 ...d.~.....\4Q.,p.
D8 46 FA 1A 03 B2 B6 19 47 BD 12 28 09 DE 94 2C 7D 20 .F......G..(...,}
24 24 1E 14 54 E7 2A C4 7C 7C E8 E9 4F 64 F3 DB 24 D7 $$..T.*.||..Od..$.
A4 67 7A 30 20 5B 9E 3F 54 15 3C 33 9C 91 99 26 04 B9 .gz0 [.?T.<3...&..
27 EF 06 A2 F2 80 8A 0A 5D 19 21 BE 8B 2D 32 1F 68 BD '.......].!..-2.h.
F6 27 2A 5D 74 F9 B4 1E 83 90 A8 BD 22 5C 36 45 AA EE .'*]t......."\6E..
92 D7 A6 61 58 C8 15 9B 13 F0 7E FC BF 09 5A 9B F8 7B ...aX.....~...Z..{
66 CB 47 70 E6 CF AE 9B 57 CD 40 C3 75 D7 1F C0 A8 B1 f.Gp....W.@.u.....
9B 0E D8 4F 1E 13 52 A3 D7 AB 0E CA EA 0B 71 A8 BC C3 ...O..R.......q...
B9 3F 4E 31 5F 60 72 12 E9 E8 71 F7 DA A9 E0 D7 C2 A8 .?N1_`r...q.......
FA DF FF D4 49 01 64 DE 21 C7 0F D0 0F 44 81 C7 31 4D ....I.d.!....D..1M
97 1C 65 78 2C 01 FB 1C B3 6B 01 8A E9 5F 05 36 86 EA ..ex,....k..._.6..
19 96 75 97 2E D9 DB E3 7C DB D9 83 3D 93 C4 6F CF 04 ..u.....|...=..o..
1B 97 4B DD C4 53 4E B9 0B E0 5C BA 0B 14 AF C2 1C 27 ..K..SN...\......'
50 3B 99 EA 3C ED CD 06 8F 29 3F 2A DF 42 DA F0 70 65 P;..<....)?*.B..pe
16 06 F4 E7 3C 16 1D 0D 3E 7C 71 DC 51 12 4E 69 90 D1 ....<...>|q.Q.Ni..
```

Below c6 we will see the changes.

The decrypted file we can see S

```
[10/09/23]seed@VM:~/.../Files$ cat fileofbdec.txt
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAASAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

> Please answer the following question: How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task. Please provide justification.

Justification :

ECB Mode:Each block of plaintext is encrypted independently. Only one block is corrupted in case of an error.

CBC Mode: Decryption of one block depends on the ciphertext of the previous block. Error propagation affects two blocks in this mode.

CFB Mode: Decryption of one block requires the ciphertext of the previous block and the current block. Error propagation affects two blocks in this mode.

OFB Mode: Decryption of one block requires only the ciphertext of the current block. Error propagation is limited to one block in this mode.

——————————————————————————justified ——————————————————————————

Task 6: Initial Vector (IV) and Common Mistakes

Task 6.1. IV Experiment

In below screenshot we are creating plaintext

```
[10/09/23]seed@VM:~/.../Files$ echo "Task 6"
Task 6
[10/09/23]seed@VM:~/.../Files$ nano task6plain.txt
[10/09/23]seed@VM:~/.../Files$ cat task6plain.txt
A basic requirement for IV is uniqueness, which means that no IV may be reused under the same key
[10/09/23]seed@VM:~/.../Files$
```

In below screenshot we encryption using **same iv**

```
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in task6plain.txt -nosalt -out task6cipher.txt -K 00112233445566778889aabbccddee
ff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in task6plain.txt -nosalt -out task6cipher1.txt -K 00112233445566778889aabbccdde
eff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ xxd task6cipher.txt
00000000: c6a6 ed44 b256 d9f1 ddfb ce68 1c62 3900  ...D.V.....h.b9.
00000010: 543a ed27 dd34 30d3 be0e 2695 2049 3df1  T:.'.40...&. I=.
00000020: bc1c 4904 0b00 2c26 3986 1ced 545e c188  ..I...,&9...T^..
00000030: 6340 d3f4 16b6 914e 5a05 4174 b05e 5c02  c@.....NZ.At.^\.
00000040: 5d1f bca9 f805 20d8 14cb 3290 d6a5 eb3e  ]..... ...2....>
00000050: 723c 058d ea18 1b3b 098f d60b 0e3c 5edd  r<.....;.....<^.
00000060: 8c55                                     .U
[10/09/23]seed@VM:~/.../Files$ xxd task6cipher1.txt
00000000: c6a6 ed44 b256 d9f1 ddfb ce68 1c62 3900  ...D.V.....h.b9.
00000010: 543a ed27 dd34 30d3 be0e 2695 2049 3df1  T:.'.40...&. I=.
00000020: bc1c 4904 0b00 2c26 3986 1ced 545e c188  ..I...,&9...T^..
00000030: 6340 d3f4 16b6 914e 5a05 4174 b05e 5c02  c@.....NZ.At.^\.
00000040: 5d1f bca9 f805 20d8 14cb 3290 d6a5 eb3e  ]..... ...2....>
00000050: 723c 058d ea18 1b3b 098f d60b 0e3c 5edd  r<.....;.....<^.
00000060: 8c55                                     .U
[10/09/23]seed@VM:~/.../Files$
```

In the below screenshot we encryption using **Diff iv.**

```
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in task6plain.txt -nosalt -out task6cipher2.txt -K 00112233445566778889aabbccdde
eff -iv 0102030405060708090
hex string is too short, padding with zero bytes to length
[10/09/23]seed@VM:~/.../Files$ xxd task6cipher2.txt
00000000: 40a2 1aa3 9c92 908c c023 679f be57 82a8  @........#g..W..
00000010: 55d5 3766 32f6 f9c4 8fd8 b250 0166 3839  U.7f2......P.f89
00000020: bcd0 c4db c078 be37 b274 176b 906e 3391  .....x.7.t.k.n3.
00000030: e2c3 e97e 5be6 88a6 e72d 88f6 7fd0 e3ce  ...~[....-......
00000040: 036b e4c0 75c8 0582 6ddb ad54 5dff 64c9  .k..u...m..T].d.
00000050: f9c3 5482 692d 61a1 df4d 8e9a 5b3b 6b1c  ..T.i-a..M..[;k.
00000060: 903f                                     .?
[10/09/23]seed@VM:~/.../Files$
```

Explain why iv needs to be unique ?

Changing the IV significantly alters the ciphertext, even with the same plaintext. This highlights the necessity for unique IVs. Using the same IV can result in identical ciphertexts, potentially leading to information leakage.

8.2 Task 6.2. Common Mistake: Use the Same IV

Screenshot of creating text files.

```
[10/09/23]seed@VM:~/.../Files$ echo -n  ‘ a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159 ‘ > c1.txt
[10/09/23]seed@VM:~/.../Files$ echo -n  ‘ bf73bcd3509299d566c35b5d450337e1bb175f903fafc159 ‘ > c2.txt
[10/09/23]seed@VM:~/.../Files$ xxd -p p1.txt
20546869732069732061206b6e6f776e206d65737361676521
```

Below we can see the text which we were given, which we can watch by seeing cat commands.

```
[10/09/23]seed@VM:~/.../Files$ cat c1.txt
a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159
[10/09/23]seed@VM:~/.../Files$ cat c2.txt
bf73bcd3509299d566c35b5d450337e1bb175f903fafc159
[10/09/23]seed@VM:~/.../Files$ ls -l c1.txt c2.txt p1.txt
-rw-rw-r-- 1 seed seed 49 Oct  9 02:43 c1.txt
-rw-rw-r-- 1 seed seed 49 Oct  9 02:44 c2.txt
-rw-rw-r-- 1 seed seed 25 Oct  9 02:39 p1.txt
[10/09/23]seed@VM:~/.../Files$
```

python3 xorhex.py 546869732069732061206b6e6f776e206d65737361676521
a469b1c502c1cab966965e50425438e1bb1b5f9037a4c15913

python3 xorhex.py f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478
bf73bcd3509299d566c35b5d450337e1bb175f903fafc15913

The above commands were runned we can get the values by the running
Xxd -p -p1

```
[10/09/23]seed@VM:~/.../Files$ chmod a+x xord.py
[10/09/23]seed@VM:~/.../Files$ python3 xord.py 5468697320697320612061206b6e6f776e206d65737361676521 a469
b1
f001d8
[10/09/23]seed@VM:~/.../Files$ python3 xord.py 5468697320697320612061206b6e6f776e206d65737361676521 a469
b1c502c1cab966965e50425438e1bb1b5f9037a4c15913
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478
[10/09/23]seed@VM:~/.../Files$ python3 xord.py f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478 bf73
bcd3509299d566c35b5d450337e1bb175f903fafc15913
4f726465723a204c61756e63682061206d697373696c6521
[10/09/23]seed@VM:~/.../Files$ echo -n "4f726465723a204c61756e63682061206d697373696c6521" | xxd - p
-r  > p2.txt
Usage:
       xxd [options] [infile [outfile]]
    or
       xxd -r [-s [-]offset] [-c cols] [-ps] [infile [outfile]]
Options:
    -a          toggle autoskip: A single '*' replaces nul-lines. Default off.
    -b          binary digit dump (incompatible with -ps,-i,-r). Default hex.
    -C          capitalize variable names in C include file style (-i).
    -c cols     format <cols> octets per line. Default 16 (-i: 12, -ps: 30).
    -E          show characters in EBCDIC. Default ASCII.
    -e          little-endian dump (incompatible with -ps,-i,-r).
    -g          number of octets per group in normal output. Default 2 (-e: 4).
    -h          print this summary.
    -i          output in C include file style.
    -l len      stop after <len> octets.
    -o off      add <off> to the displayed file position.
    -ps         output in postscript plain hexdump style.
    -r          reverse operation: convert (or patch) hexdump into binary.
```

In the above screenshot we can see that it was working fine

```
[10/09/23]seed@VM:~/.../Files$ echo -n "4f726465723a204c61756e63682061206d697373696c6521" | xxd -r -
p > p2.txt
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ cat p2.txt
[10/09/23]seed@VM:~/.../Files$ cat p2.txt
Order: Launch a missile![10/09/23]seed@VM:~/.../Files$
```

If we replace OFB in this experiment with CFB (Cipher Feedback), how much of P2 can be revealed? You only need to answer the question; there is no need to demonstrate that.

The attack used in this experiment is called the *known-plaintext attack*, which is an attack model for cryptanalysis where the attacker has access to both the plaintext and its encrypted version (ciphertext). If this can lead to the revealing of further secret information, the encryption scheme is not considered as secure.

Explanation : In the context of encryption, if the IV remains constant, it becomes vulnerable to known-plaintext attacks. Let's consider two plaintexts, P1 and P2, along with their corresponding ciphertexts, C1 and C2. With access to P1 and the knowledge that P1 and P2 were encrypted using the same IV, an attacker can XOR P1 with C1 to obtain the output stream. Subsequently, XORing this output stream with C2 will reveal P2.However, if we switch from OFB to CFB mode, we can only expose the first block of the plaintext. In CFB, the decryption of the next block relies on the ciphertext of the first block, unlike OFB, where the next block's decryption depends solely on the output stream and not the ciphertext

8.3 Task 6.3. Common Mistake: Use a Predictable IV

Below screenshot we are creating a file with name of p1

```
[10/09/23]seed@VM:~/.../Files$ echo "Task 6.3"
Task 6.3
[10/09/23]seed@VM:~/.../Files$ echo -n "Yes............." > p1
[10/09/23]seed@VM:~/.../Files$ cat p1
Yes.............[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ xxd -p p1
5965732e2e2e2e2e2e2e2e2e2e2e2e
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in P1 -nosalt -out C1 -K 001122334455667
78899aabbccddeeff -iv 31323334353637383930313233343536
Can't open P1 for reading, No such file or directory
140710566196544:error:02001002:system library:fopen:No such file or directory:crypto/bio/bss_file.c:
69:fopen('P1','rb')
140710566196544:error:2006D080:BIO routines:BIO_new_file:no such file:crypto/bio/bss_file.c:76:
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in p1 -nosalt -out c1 -K 001122334455667
78899aabbccddeeff -iv 31323334353637383930313233343536
[10/09/23]seed@VM:~/.../Files$ xxd -p c1
cc99821bb09af95228e7763836a90791763f1ecda46d2824f8dd2086c53b
e7fa
[10/09/23]seed@VM:~/.../Files$
```

```
[10/09/23]seed@VM:~/.../Files$ echo -n "Yes............." > p2
[10/09/23]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in p2 -nosalt -out c2 -K 001122334455667
78899aabbccddeeff -iv 31323334353637383930313233343536
[10/09/23]seed@VM:~/.../Files$ xxd -p c2
cc99821bb09af95228e7763836a90791763f1ecda46d2824f8dd2086c53b
e7fa
[10/09/23]seed@VM:~/.../Files$ xxd -p c1
cc99821bb09af95228e7763836a90791763f1ecda46d2824f8dd2086c53b
e7fa
[10/09/23]seed@VM:~/.../Files$
```

In the given scenario, we made an initial guess, "Yes," and subsequently employed the known IV and ciphertext to verify the accuracy of our guess.

> After showing you the next IV, the oracle will ask you to input a plaintext message (as a hex string). The oracle will encrypt the message with the next IV, and outputs the new ciphertext. You can try different plaintexts, but keep in mind that every time, the IV will change, but it is predictable. To simply your job, we let the oracle print out the next IV. To exit from the interaction, press Ctrl+C.

In the described scenario, we can initially assume the word is "Yes." It stores this word as P2 and performs a hexadecimal XOR operation with the known IV (initial). Afterward, XORs the output with a new IV and copies this to P2. Bob encrypts P2 and provides the output as C2 . it then compares C2 with C1,and observes that C1 and C2 are identical, confirming the accuracy of our guess.

Task 7: Programming using the Crypto Library

Your goal is to write a program to find out the encryption key. You can download a English word list from the Internet. We have also included one in the `Labsetup.zip` file. The plaintext, ciphertext, and IV are listed in the following:

Below the screenshot we can find the key was visible.

```
[10/09/23]seed@VM:~/.../Lab7-encryption$ cd Files
[10/09/23]seed@VM:~/.../Files$ python3 task7finderkey.py
  File "task7finderkey.py", line 7
    ciphertext_hex = "764aa26b55a4da654df6b19e4bce00f4
                                                      ^
SyntaxError: EOL while scanning string literal
[10/09/23]seed@VM:~/.../Files$ python3 task7finderkey.py
cipher text     :  b'vJ\xa2kU\xa4\xdaeM\xf6\xb1\x9eK\xce\x00\xf4\xed\x05\xe0\x93
F\xfb\x0ev%\x83\xcb}\xa2\xac\x93\xa2'
iv              :  b'\xaa\xbb\xcc\xdd\xee\xff\x00\x99\x88wfUD3"\x11'
Key found       : Syracuse
Successful Decryption!!!
[10/09/23]seed@VM:~/.../Files$
```

Code snippets were attached below

Open — task7finderkey.py — ~/Desktop/Seedlab/Lab7-encryption/Files

```python
from Crypto.Cipher import AES
from Crypto.Util import Padding
import binascii
from Crypto.Util.Padding import unpad

# --------- Hexadecimal ciphertext and IV ---------
ciphertext_hex = "764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2"
iv_hex = " aabbccddeeff00998877665544332211"

# ----- Converting ciphertext and IV to bytes -----
#ciphertext = binascii.unhexlify(ciphertext_hex)
#iv = binascii.unhexlify(iv_hex)
ciphertext= bytes.fromhex(ciphertext_hex)
iv = bytes.fromhex(iv_hex)

print("cipher text      : ",ciphertext)
print("iv               : ",iv)

# ---------- Reading word.txt file to get key list ----------
with open("words.txt", "r") as word_file:
    potential_keys = [line.strip() for line in word_file]

# Decrypting ciphertext with every potential key

for validkey in potential_keys:
    #print('\t',validkey)

    if len(validkey) > 16:  #                          --- Ignoring key length > 16 ---
        continue
    #print('\t',validkey)

    # ----------- Encoding and padding the key -----------
    padsize = 16 - len(validkey)
```

```
    # ------------ Encoding and padding the key ------------
    padsize = 16 - len(validkey)
    paddedkey = validkey.encode("utf-8") + b"#" * padsize
    key = bytes.fromhex(paddedkey.hex())


    cipher = AES.new(key, AES.MODE_CBC, iv) #   --- Creating cipher object with key &
    #print("cipher -- ",cipher)

    # ------------ Decryption using cipher object ------------
    try:
        plaintext = unpad(cipher.decrypt(ciphertext), 16).decode("utf-8")
        if plaintext == "This is a top secret.":
            #print("Hex PadKey          : ",key)
            print("Key found    : {0}".format(validkey))
            break  # Stop when a valid key is found
    except Exception as e:
        continue  # Try the next key if decryption fails

print("Successful Decryption!!!")
```

Code Explanation (Working process)

1. Hexadecimal Ciphertext and IV - It defines the ciphertext and IV as hexadecimal strings. These are the values that need to be decrypted.

2. Converting Hexadecimal to Bytes - The code converts the hexadecimal ciphertext and IV into bytes, which are suitable for cryptographic operations. This is done using the bytes.fromhex() method.

3. It reads potential encryption keys from a file called "words.txt." These keys will be used to attempt decryption of the ciphertext.

4. Decrypting with Potential Keys: The code iterates through each potential key and attempts to decrypt the ciphertext using AES encryption in CBC (Cipher Block Chaining) mode. It ensures that the key length is less than or equal to 16 bytes.

5. Key Padding: If the key length is less than 16 bytes, it pads the key with '#' characters to make it exactly 16 bytes long.

6. Decryption Attempt: The code decrypts the ciphertext with the current key and checks if the resulting plaintext matches the expected value, which is "This is a top secret." If a valid key is found, it prints the key and breaks out of the loop.

7. Handling Exceptions: Exception handling is used to continue trying the next potential key if decryption with the current key fails.

8. Successful Decryption: If a valid key is found and decryption is successful, it prints "Successful Decryption!!!" to indicate that the code successfully decrypted the ciphertext.

Python code actually performs a **brute-force search** for the **correct encryption key** by trying a list of potential keys until it successfully decrypts the ciphertext to the expected plaintext.

We can find the key (syracuse) in the words.txt given in the lab setup.