



Faculty of Engineering and Technology  
Electrical & Computer Engineering Department

ENCS4320

Applied Cryptography

## Cryptography Lab Report

---

Prepared by:

Al-Ayham Maree 1191408

Sara Ammar 1191052

Supervised by:

Dr. Ahmad Alsadeh

Section:

2

11-1-2023

## Contents

Abstract.....	4
Task 1: Frequency Analysis:.....	5
Step 1: .....	5
Step 2: .....	6
Step 3: .....	6
Task 2: Encryption using different ciphers and modes .....	9
Task 3: Encryption mode ECB vs CBC.....	10
Task 4: Padding.....	12
Task 5: Error Propagation – Corrupted Cipher Text.....	15
Task 6: Initial Vector (IV) and Common Mistakes .....	20
6.1. IV Experiment.....	20
6.2. Common Mistake: Use the Same IV .....	22
6.3. Common Mistake: Use a Predictable IV .....	23
Task 7: Programming using the Crypto Library .....	24
Conclusion.....	25
Appendix.....	26

## Table of Figures:

Figure 1: Generating key using python code .....	5
Figure 2: The mono-alphabetic substitution cipher .....	5
Figure 3: - Converting to lowercase and removing non-alphabetic characters.....	6
Figure 4: the encryption results.....	6
Figure 5: The frequencies of n-gram using freq.py.....	7
Figure 6: The decryption result.....	8
Figure 7: encryption using different cipher modes .....	9
Figure 8: Encryption result and the cipher formats using different modes .....	9
Figure 9: Plaintext used .....	9
Figure 10: Encryption use ECB .....	10
Figure 11: Encryption result using ECB .....	10
Figure 12: Encryption using CBC with IV .....	10
Figure 13: Encryption Result using CBC with IV .....	11
Figure 14: Encryption using AES-128 CBC.....	11
Figure 15: Encryption result of AES-128 CBC.....	11
Figure 16: Creating three files .....	12
Figure 17: Encryption files using ECB.....	12
Figure 18: encrypted files result using ECB.....	12
Figure 19: Encryption files using CBC.....	13
Figure 20: Encryption result of files using CBC and padding files .....	13
Figure 21: Encryption files using CFB .....	14
Figure 22: Encryption result using CFB .....	14
Figure 23: Encryption files using OFB .....	14
Figure 24: Encryption result using OFB.....	15
Figure 25: Creating file with 1.1 Kb size.....	15
Figure 26: encryption and decryption using AES-128 ECB.....	16
Figure 27: Edit 55th byte of the cipher text with ECB .....	16
Figure 28: Encryption and decryption using AES-128 CBC .....	17
Figure 29: Edit 55th byte of the cipher text with CBC .....	17
Figure 30: Encryption and decryption using AES-128 CFB.....	18
Figure 31: Edit 55th byte of the cipher text with CFB.....	18
Figure 32: Encryption and decryption using AES-128 OFB .....	19
Figure 33: Edit 55th byte of the cipher text with OFB .....	19
Figure 34: Encrypting the same plaintext using two different IVs.....	20
Figure 35: The resulting ciphertexts from using two different IVs .....	20
Figure 36: Encrypting the same plaintext using the same IV.....	21
Figure 37: The resulting ciphertexts from using the same IV .....	21
Figure 38: Sample_code to find P2 content .....	22
Figure 39: P2 Content .....	23
Figure 40: P2 Bob message.....	23
Figure 41: The result of the key .....	24

## Abstract

This lab was intended to familiarize users with many concepts in secret-key encryption and how an attacker can break the security of encryption, as well as to gain experience with encryption algorithms and encryption modes, padding, initial vector (IV), and the tasks in this lab helping users in being able to use the tools and write programs to encryption and decryption messages in different ways and modes.

## Task 1: Frequency Analysis:

### Step 1:

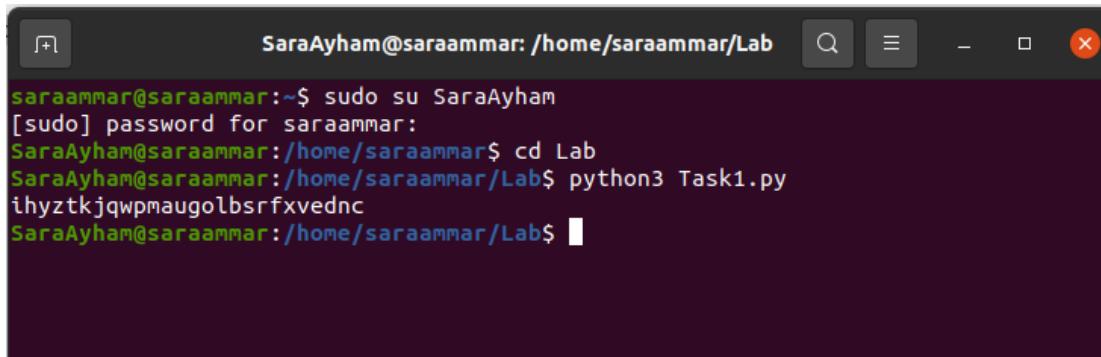
The encryption key was generated using permutation alphabet from (a-z), and the permuted alphabet was used as a key, using python code as shown below



```
1 #!/bin/env python3
2 import random
3
4 s = "abcdefghijklmnopqrstuvwxyz"
5 list = random.sample(s, len(s))
6 key = ''.join(list)
7 print(key)
8
9
10
11 |
```

Figure 1: Generating key using python code

After encryption, the mono-alphabetic substitution cipher was produced as shown below



```
SaraAyham@saraammar:~/home/saraammar/Lab$ sudo su SaraAyham
[sudo] password for saraammar:
SaraAyham@saraammar:/home/saraammar$ cd Lab
SaraAyham@saraammar:/home/saraammar/Lab$ python3 Task1.py
ihyztkjqpmaugolbsrfxvednc
SaraAyham@saraammar:/home/saraammar/Lab$
```

Figure 2: The mono-alphabetic substitution cipher

## Step 2:

In the figure below, the plaintext (message) has been converted to lower case, and the non-alphabetic characters has been removed before producing the mono-alphabetic cipher

```
saraammar@saraammar:~/Lab$ tr [:upper:] [:lower:] < article.txt > lowercase.txt
saraammar@saraammar:~/Lab$ tr -cd '[a-z][\n][space:]' < lowercase.txt > plaintext.txt
saraammar@saraammar:~/Lab$ cat article.txt
WELCOME TO Secret-Key Encryption Lab, We are Sara Ammar and Ayham Maree. :) :)
saraammar@saraammar:~/Lab$ cat lowercase.txt
welcome to secret-key encryption lab, we are sara ammar and ayham maree. :) :)
saraammar@saraammar:~/Lab$ cat plaintext.txt
welcome to secretkey encryption lab we are sara ammar and ayham maree
```

Figure 3: - Converting to lowercase and removing non-alphabetic characters

## Step 3:

In this step, the encryption has been worked using the “tr” command, and the letters has been encrypted only, and the space and return characters has been left alone, as shown below:

```
saraammar@saraammar:~/Lab$ tr 'abcdefghijklmnopqrstuvwxyz' 'sxtrwinqbcdpvkfmalyuojzc' < plaintext.txt > ciphertext.txt
saraammar@saraammar:~/Lab$ cat ciphertext.txt
owptkvw hk lwtawhdwz wgtazfhbk g psx ow saw lsas svvs a sgr szqsv vsaww
saraammar@saraammar:~/Lab$
```

Figure 4: the encryption results

The “ciphertext.txt” has been read and the statistics of n-gram was produced, including the single letter frequencies, bigram frequencies, and trigram frequencies using the “freq.py” python program inside the “LabSetup/Files” folder, as shown in figures below:

```
saraammar@saraammar:~/Lab$ python3 freq.py
-----
1-gram (top 20):
n: 488
y: 373
v: 348
x: 291
u: 280
q: 276
r: 266
m: 235
t: 183
l: 166
p: 159
s: 116
c: 104
z: 95
e: 90
g: 83
b: 83
r: 82
f: 76
d: 59
-----
2-gram (top 20):
yt: 145
tn: 89
mu: 74
nh: 58
vk: 57
hn: 57
vu: 56
nq: 53
ws: 52
up: 46
xh: 45
yn: 44
rp: 44
vy: 44
nu: 42
qy: 39
vq: 33
vt: 32
gn: 32
av: 31
-----
3-gram (top 20):
ytn: 78
vip: 39
mari: 20
```

3-gram (top 20):	
ytn:	78
vup:	30
mur:	20
ynh:	18
xzy:	16
mxu:	14
gnq:	14
ytv:	13
nay:	13
vit:	13
bkh:	13
lvq:	12
nuy:	12
vyn:	12
uvy:	11
lnu:	11
nvh:	11
cnu:	11
tng:	10
vhp:	10

Figure 5: The frequencies of n-gram using freq.py

In this task, the cipher.txt was converted to plain.txt using the frequency analysis of letters, and we utilized Python to perform the letter conversion in order to get the original message that was sent (You can find it in the appendix1). First, the message ‘THE’ was produced from decryption the cipher ‘ytn’ from English, so every Y in the cipher is a T, every T is an H, and every N is an E.

The cipher was 'n,' and the most repeating letter in English is an 'E,' therefore that suggests we are on the right track to decryption. Alternatively, each letter that repeated numerous times was counted. We currently know the cipher's solution for the first three letters, and when we altered those letters and searched for another word, we discovered the following:

1. Since just the letter "v" was found in the cipher, only the letter "A" in English may be found alone. As a result, every "v" in the cipher was altered to an "A," and we then complied again.
2. What else might the word 'Tx' (capital letters indicate converted English letters) be if not 'TO'
3. The cipher contained the words "uO" and "Ou," and we therefore predicted that every "u" is an "N," leading to the creation of the letters "NO" and "ON."
4. What else could the word "ANp" be if not "AND"? Therefore, every "p" is a "D."
5. Since every letter has been altered except for the letter "g," the sentence "TO gE A" has been found. However, given the meaning of the sentence, we had expected that it would be "TO BE A," thus each "g" is now a "B."
6. Since the word "HOI" was found, every "l" is a "W," making it impossible for it to be anything else.
7. Every "h" is an "R" since the word "OTHEh" was discovered; what else could it be.
8. The word "Aii" was found; what else could it be except "ALL," thus each "I" is a "L."
9. The word "mT" was found, but since "A" was already present, it couldn't be "AT"; instead, it could only be "IT," meaning that every "m" is an "I."
10. Since we found a replacement for the letter "R," the word "WAq" couldn't be "WAR" and could only be "WAS," which means that every "q" is a "S."

11. When the word "RIrHT" was found, it could only mean "RIGHT," hence each r stands for "G."
12. Every 'z' is a 'U' because the word 'ABOzT' was found; what else could it be except 'ABOUT'.
13. When the word "AbTER" was found, it could only mean "AFTER," hence every "b" is a "F."
14. Since the word "SUNDAd" was found to be "SUNDAY," every "d" is a "Y."
15. Since the word "THANsS" was found, it could only mean "THANKS," hence each s represents a "K."
16. When the word "DREAc" was found, it could only mean "DREAM," hence each c stands for "M."
17. The word "RAaE" was found; hence, every "a" is a "C," and it could only be "RACE."
18. When the word "TRIe" was found, it could only be "TRIP," therefore each 'e' is a 'P'.
19. When the word "AfOID" was found, it could only mean "AVOID," hence each f stands for "V."
20. The word "oUST" was found; since it could only be "JUST," every "o" is a "J."
21. Since the word "EkSTRA" was found, every "k" is an "X," and there is no other possible meaning for it.
22. When the word "EjUALLY" was found, it could only mean "EQUALLY," therefore every "j" is a "Q."

The key found is therefore "VGAPNBRTMOSICUXEJHQYZFLKDW"

Following the decryption, the plain.txt can be found here (you can also find it in the appendix as text):

```
saraammar@saraammar:~/Lab$ python3 Task1.py
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 FEW DREAM ABOUT WHETHER THERE OUGHT TO BE A PRESIDENT WINFREY THE SEASON DIDNT JUST SEEM EXTRA LONG IT WAS EXTRA LONG BECAUSE THE OSCARS WERE MOVED TO THE FIRST WEEKEND IN MARCH TO AVOID CONFLICTING WITH THE SUPER BOWL CEREMONY WHICH IS ANOTHER STORY THE METOO MOVEMENT HAS BEEN A LONG TIME COMING BUT IT HAS FINALLY ARRIVED WITH A SWELLING CONTINGENT 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 SPOTTED LAPEL PINS AND SOUNDED OFF ABOUT SEXIST POWER IMBALANCES FROM THE RED CARPET AND THE STAGE ON THE AIR IT WAS CALLED OUT ABOUT PAY INEQUALITY AFTER ITS FORMER ANCHOR CATT SAULER QUIT ONCE SHE LEARNED THAT SHE WAS MAKING FAR LESS THAN A MALE COHOST AND DURING THE CEREMONY NATALIE PORTMAN TOOK A BLUNT AND SATISFYING DIG AT THE ALLMALE ROSTER OF NOMINATED DIRECTORS HOW COULD THAT BE TOPLESS SHE TURNED OUT AT LEAST IN TERMS OF THE OSCARS SHE PROBABLY WOULD NOT HAVE BEEN INVOLVED IN THE METOO MOVEMENT PORTMAN TOLD THE NEW YORK TIMES SHE HAD DONATED OVER $100,000 TO THE #METOO FUND LAST SEASON AND SHE HAS BEEN INVOLVED SINCE THEN THE ASSOCIATION ONLY WITH SIGNIFICANT ACTIONS INSTEAD A SPOKESMAN FOR THE METOO MOVEMENT WHICH CLOSED DOORS AND HAS SINCE ASSUED MILLION FOR ITS LEGAL DEFENSE FUND WHICH AFTER THE GLOBES WAS FLOODED WITH THOUSANDS OF DONATIONS OF OR LESS FROM PEOPLE IN SOME COUNTRIES NO CALL TO WEAR BLACK GOWNS WENT OUT IN ADVANCE OF THE OSCARS THOUGH THE MOVEMENT WILL ALMOST CERTAINLY BE REFERENCED BEFORE AND DURING THE CEREMONY ESPECIALLY SINCE VOCAL METOO SUPPORTERS LIKE ASHLEY JUDD LAURA DERN AND NICOLE KIDMAN ARE SCHEDULED PRESENTERS AN OTHER 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 NAUTHER NARRATIVE ONLY SERVES THE AWARDS VOTE MACHINE BUT OFTEN THE VOTERS AREN T THE ONLY ONES CALLED UP TO THE METOO LINEUP AS IN THE CASE OF THE BEST SUPPORTING ACTRESS NOMINEE REBECCA FIERMAN WHO IS A METOO ACTIVIST AND A FEMINIST 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 VOTE IS ELIMINATED AND ITS VOTES ARE REDISTRIBUTED TO THE MOVIES THAT CARRIED THE ELIMINATED BALLOTS SECONDPLACE VOTE AND THIS CONTINUES UNTIL A WINNER EMERGE IT IS ALL TERRIBLY CONFUSING BUT APPARENTLY THE CONSENSUS FAVORITE COMES OUT AHEAD IN THE END THIS MEANS THAT ENDOSEASON AWARDS CHATTER INVARILY INVOLVES TORTURED SPECULATION ABOUT WHICH FILM WOULD MOST LIKELY VOTE FOR METOO ACTIVISTS AND WHICH FILM WOULD NOT AND WHICH FILM WOULD GET THE BIGGEST METOO VOTE AND WHICH FILM WOULD GET THE BIGGEST VOTE IN THE END THE PREDICTION IS OFTEN CORRECT BUT NOT ALWAYS AS IN THE CASE OF THE BEST SUPPORTING ACTRESS WHERE REVENANT OR THE BIG SHORT THE PRIZE WENT TO SPOTLIGHT LAST YEAR NEARLY ALL THE FORECASTERS DECLARED LA LA LAND THE PRESUMPTIVE WINNER AND FOR TWO AND A HALF MINUTES THEY WERE CORRECT BEFORE AN ENVELOPE S NAUFI WAS REVEALED AND THE RIGHTFUL WINNER MOONLIGHT WAS CROWNED THIS YEAR AWARDS WATCHERS ARE UNEQUALLY DIVIDED BETWEEN THREE BILLBOARDS OUTSIDE EBBING MISSOURI THE FAVORITE AND THE SHAPE OF WATER WHICH IS THE BAGGERS PREDICTION WITH A FEW FORECASTING A HAIL MARY WIN FOR GET OUT BUT ALL OF THOSE FILMS HAVE HISTORICAL OSCARVOTING PATTERNS AGAINST THEM THE SHAPE OF WATER HAS NOMINATIONS MORE THAN ANY OTHER FILM AND WAS ALSO NAMED THE YEARS BEST BY THE PRODUCERS AND DIRECTORS GUILDS YET IT WAS NOT NOMINATED FOR A SCREEN ACTORS GUILD AWARD FOR BEST ENSEMBLE AND NO FILM HAS WON BEST PICTURE WITHOUT PREVIOUSLY LANDING AT LEAST THE ACTORS NOMINATION SINCE BRAVEHEART IN THIS YEAR THE BEST ENSEMBLE SAG ENDED UP GOING TO THREE BILLBOARDS WHICH IS SIGNIFICANT BECAUSE ACTORS MAKE UP THE ACADEMYS LARGEST BRANCH TH
```

Figure 6: The decryption result

## Task 2: Encryption using different ciphers and modes

In this task, various encryption algorithms and modes was used, “openssl enc” command has been used to encrypt/decrypt file, AES 128 bits Cipher Block Chain, Blow Fish Cipher Block Chain, and Cipher Feedback were the three block ciphers we tried. The figure below illustrates how the various ciphers in the cipher texts differ:

```
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher1.bin -K 0011223344556677889aabccddeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cfb -e -in plain.txt -out cipher2.bin -K 0011223344556677889aabccddeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -ofc -e -in plain.txt -out cipher3.bin -K 0011223344556677889aabccddeff -iv 0102030405060708
SaraAyham@saraammar:~/Lab$
```

Figure 7: encryption using different cipher modes

```
SaraAyham@saraammar:~$ cd /home/SaraAyham/Lab
SaraAyham@saraammar:~/Lab$ cat cipher1.bin
+U*****1*****m*F*Co*
+
++++t9;x***x*U5mXvSaraAyham@saraammar:~/Lab$ cat cipher2.bin
***F*R***+iu|.PjCm***G*v*SaraAyham@saraammar:~/Lab$ cat cipher3.bin
u+;g+o+o+o+o+o+Bz=-q+o
++B+[++I+9+hSaraAyham@saraammar:~/Lab$
SaraAyham@saraammar:~/Lab$ SaraAyham@saraammar:~/Lab$
```

Figure 8: Encryption result and the cipher formats using different modes

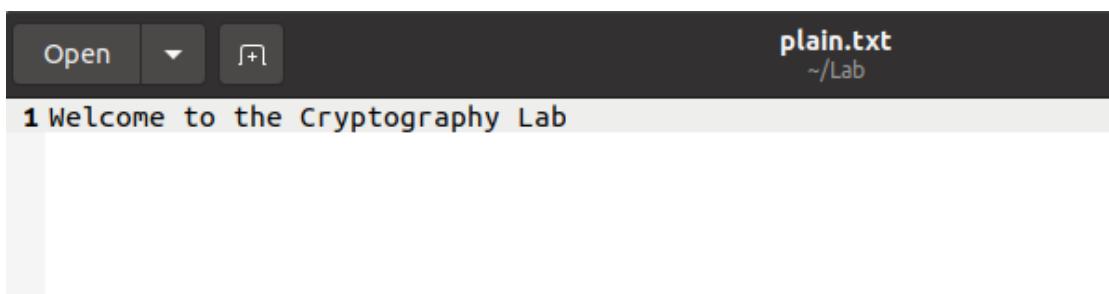
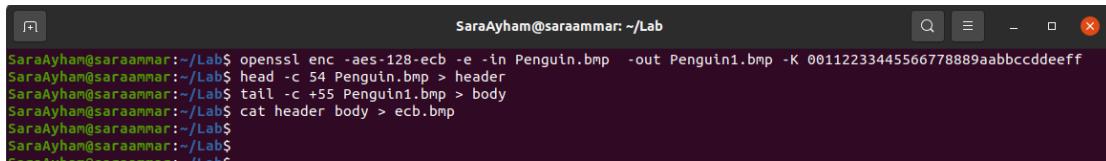


Figure 9: Plaintext used

## Task 3: Encryption mode ECB vs CBC

The penguin image (.bmp) has been encrypted by three cipher modes: Electronic Codebook Mode and Cipher Block Chaining Mode with Initialization Vector (IV) and AES-128 Cipher Block Chaining .Therefore, the body of the image was encrypted and the original header has been put, the following figures show the encryption using three cipher modes:

1. Encryption image using ECB:



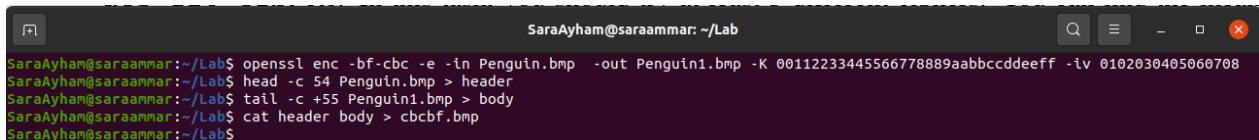
```
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ecb -e -in Penguin.bmp -out Penguin1.bmp -K 00112233445566778889aabbccddeeff
SaraAyham@saraammar:~/Lab$ head -c 54 Penguin.bmp > header
SaraAyham@saraammar:~/Lab$ tail -c +55 Penguin1.bmp > body
SaraAyham@saraammar:~/Lab$ cat header body > ecb.bmp
SaraAyham@saraammar:~/Lab$
SaraAyham@saraammar:~/Lab$
```

Figure 10: Encryption use ECB



Figure 11: Encryption result using ECB

2. Encryption result using CBC mode with IV:



```
SaraAyham@saraammar:~/Lab$ openssl enc -bf-cbc -e -in Penguin.bmp -out Penguin1.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708
SaraAyham@saraammar:~/Lab$ head -c 54 Penguin.bmp > header
SaraAyham@saraammar:~/Lab$ tail -c +55 Penguin1.bmp > body
SaraAyham@saraammar:~/Lab$ cat header body > cbcfb.bmp
SaraAyham@saraammar:~/Lab$
```

Figure 12: Encryption using CBC with IV

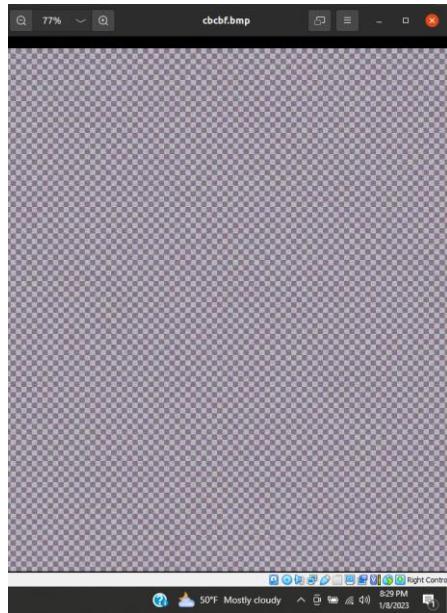


Figure 13: Encryption Result using CBC with IV

### 3. Encryption image using AES-128 CBC:

```
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -e -in Penguin.bmp -out Penguin1.bmp -K 00112233445566778889aabcccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ head -c 54 Penguin.bmp > header
SaraAyham@saraammar:~/Lab$ tail -c +55 Penguin1.bmp > body
SaraAyham@saraammar:~/Lab$ cat header body > cbc.bmp
SaraAyham@saraammar:~/Lab$
```

Figure 14: Encryption using AES-128 CBC

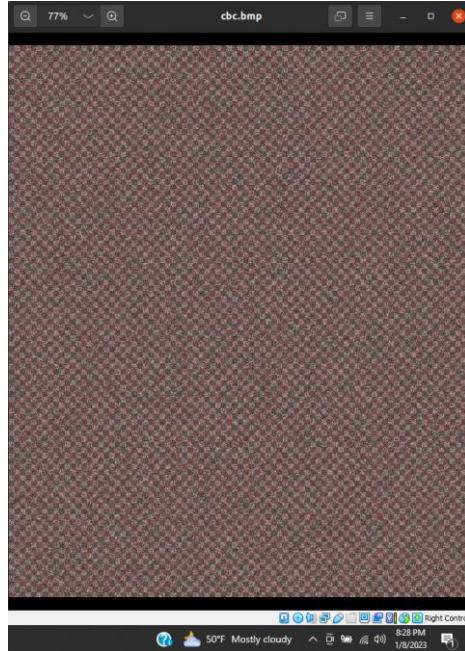
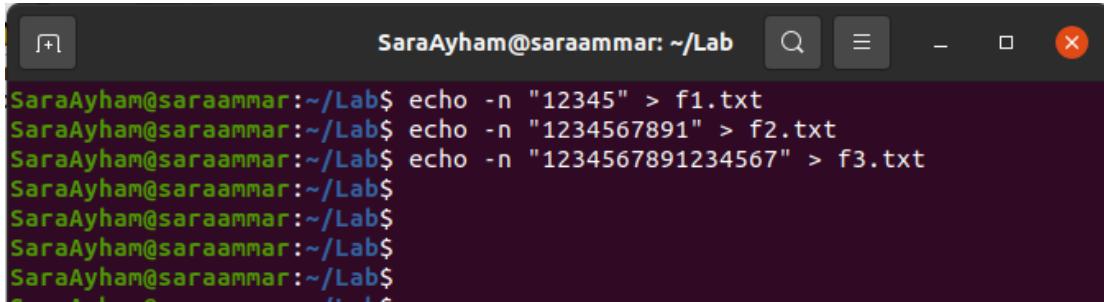


Figure 15: Encryption result of AES-128 CBC

## Task 4: Padding

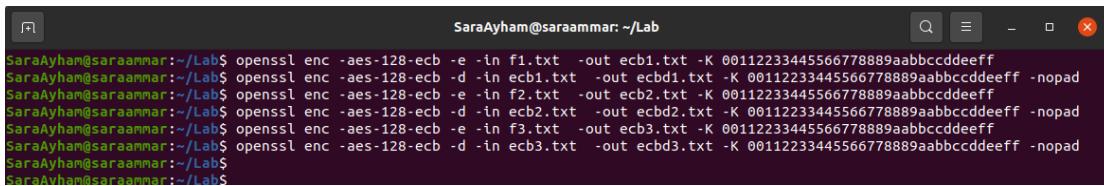
In this task, the three files with different lengths was encrypted using three ciphers: ECB, CBC and CFB. Firstly, the three files has been created, which contains f1 = 5 bytes, f2 =10 bytes, f3=16 bytes, files has been created using the command “echo -n” as shown in figure below:



```
SaraAyham@saraammar:~/Lab$ echo -n "12345" > f1.txt
SaraAyham@saraammar:~/Lab$ echo -n "1234567891" > f2.txt
SaraAyham@saraammar:~/Lab$ echo -n "1234567891234567" > f3.txt
SaraAyham@saraammar:~/Lab$
```

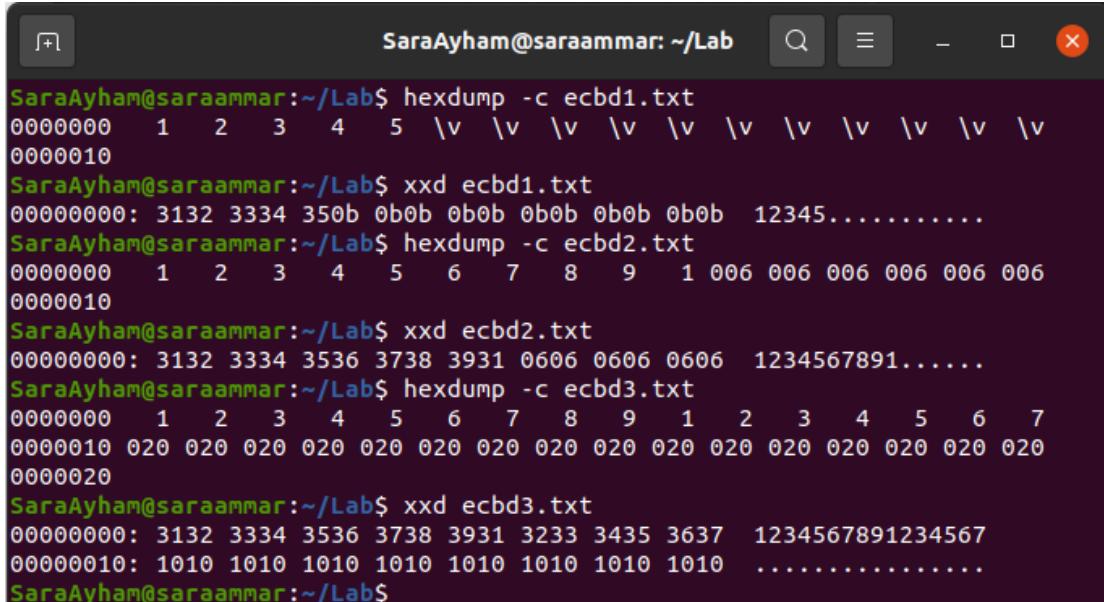
Figure 16: Creating three files

### 1. Encryption three files using ECB



```
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ecb -e -in f1.txt -out ecb1.txt -K 001122334556677889aabcccddeeff
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ecb -d -in ecb1.txt -out ecb1.txt -K 001122334556677889aabcccddeeff -nopad
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ecb -e -in f2.txt -out ecb2.txt -K 001122334556677889aabcccddeeff
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ecb -d -in ecb2.txt -out ecb2.txt -K 001122334556677889aabcccddeeff -nopad
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ecb -e -in f3.txt -out ecb3.txt -K 001122334556677889aabcccddeeff
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ecb -d -in ecb3.txt -out ecb3.txt -K 001122334556677889aabcccddeeff -nopad
SaraAyham@saraammar:~/Lab$
```

Figure 17: Encryption files using ECB

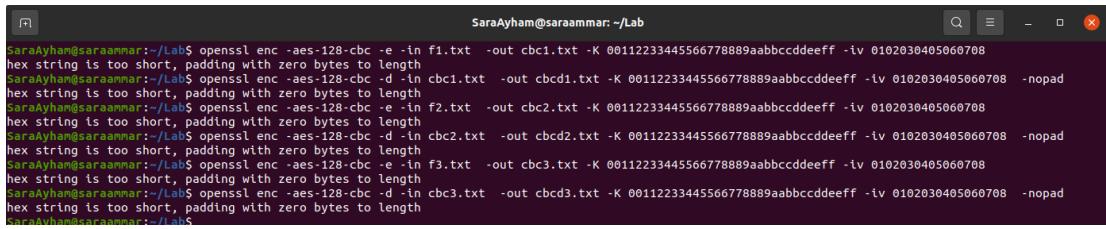


```
SaraAyham@saraammar:~/Lab$ hexdump -c ecb1.txt
00000000  1 2 3 4 5 \v \v
00000010
SaraAyham@saraammar:~/Lab$ xxd ecb1.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 12345.....
SaraAyham@saraammar:~/Lab$ hexdump -c ecb2.txt
00000000  1 2 3 4 5 6 7 8 9 1 006 006 006 006 006
00000010
SaraAyham@saraammar:~/Lab$ xxd ecb2.txt
00000000: 3132 3334 3536 3738 3931 0606 0606 0606 1234567891.....
SaraAyham@saraammar:~/Lab$ hexdump -c ecb3.txt
00000000  1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7
00000010 020 020 020 020 020 020 020 020 020 020 020 020 020 020 020
00000020
SaraAyham@saraammar:~/Lab$ xxd ecb3.txt
00000000: 3132 3334 3536 3738 3931 3233 3435 3637 1234567891234567
00000010: 1010 1010 1010 1010 1010 1010 1010 1010 .....
```

Figure 18: encrypted files result using ECB

## 2. Encryption files using CBC:

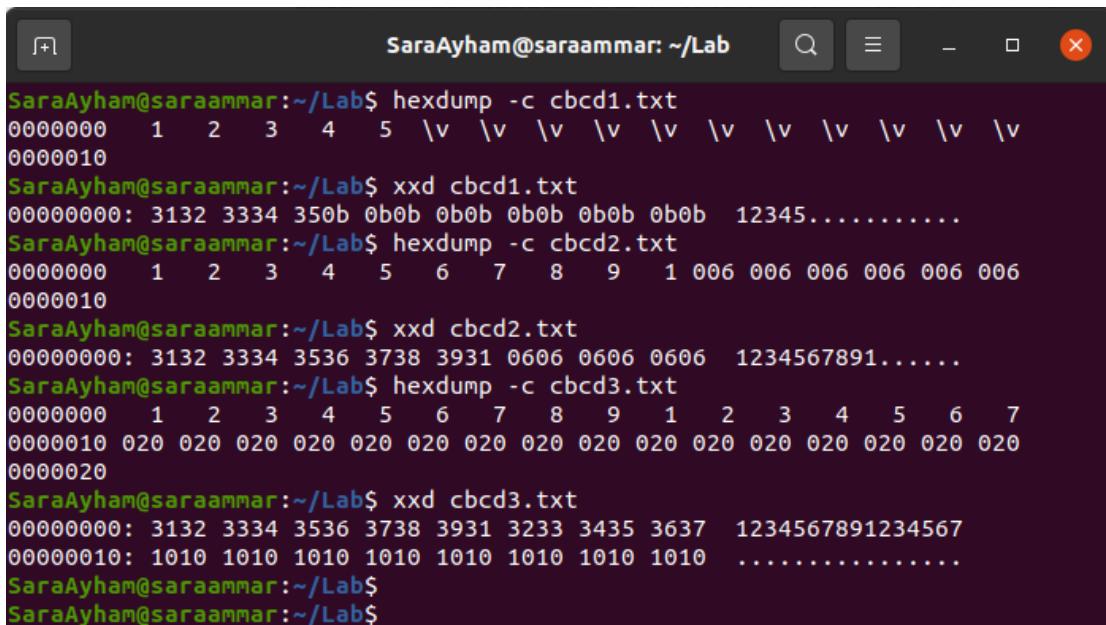
The first encrypted file in CBC included 11 bytes of 0X0B as padding data because the total block size in CBC is 16 bytes, and B in HEX equals 11 in decimal. As padding data, the second file adds 6 bytes of 0X06. The padding bytes for the third file are 16 bytes of 0X10 (10 in HEX=16 in decimal), as indicated in the figure below, because its length is greater than the block size of 16.



```
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -e -in f1.txt -out cbc1.txt -K 00112233445566778889aabccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -d -in cbc1.txt -out cbc1d.txt -K 00112233445566778889aabccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -e -in f2.txt -out cbc2.txt -K 00112233445566778889aabccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -d -in cbc2.txt -out cbc2d.txt -K 00112233445566778889aabccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -e -in f3.txt -out cbc3.txt -K 00112233445566778889aabccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -d -in cbc3.txt -out cbc3d.txt -K 00112233445566778889aabccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$
```

Figure 19: Encryption files using CBC

As shown in the figure below, the CBC is the same concept as padding the ECB.



```
SaraAyham@saraammar:~/Lab$ hexdump -c cbc1.txt
00000000  1 2 3 4 5 \v \v
00000010
SaraAyham@saraammar:~/Lab$ xxd cbc1.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 12345.....
SaraAyham@saraammar:~/Lab$ hexdump -c cbc2.txt
00000000  1 2 3 4 5 6 7 8 9 1 006 006 006 006 006 006
00000010
SaraAyham@saraammar:~/Lab$ xxd cbc2.txt
00000000: 3132 3334 3536 3738 3931 0606 0606 0606 1234567891.....
SaraAyham@saraammar:~/Lab$ hexdump -c cbc3.txt
00000000  1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7
00000010 020 020 020 020 020 020 020 020 020 020 020 020 020 020 020
00000020
SaraAyham@saraammar:~/Lab$ xxd cbc3.txt
00000000: 3132 3334 3536 3738 3931 3233 3435 3637 1234567891234567
00000010: 1010 1010 1010 1010 1010 1010 1010 1010 1010 ..... .
SaraAyham@saraammar:~/Lab$ 
SaraAyham@saraammar:~/Lab$
```

Figure 20: Encryption result of files using CBC and padding files

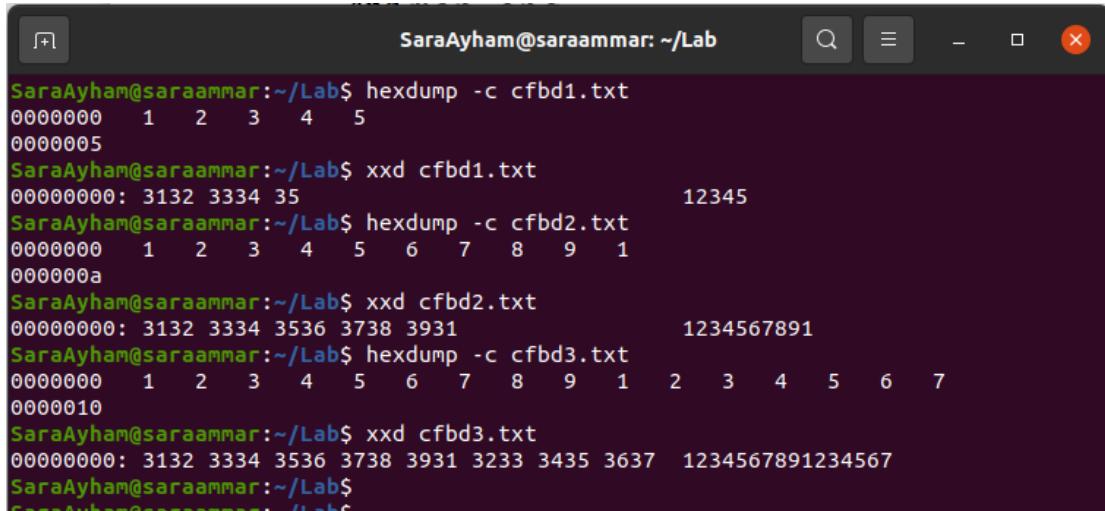
### 3. Encryption files using CFB:

Contrary to CBC and ECB, which require padding, CFB's cipher text size is the same as plain text size, as illustrated in the figure below:



```
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cfb -e -in f1.txt -out cfbd1.txt -K 00112233445566778889aabcccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cfb -d -in cfbd1.txt -out cfbd1.txt -K 00112233445566778889aabcccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cfb -e -in f2.txt -out cfbd2.txt -K 00112233445566778889aabcccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cfb -d -in cfbd2.txt -out cfbd2.txt -K 00112233445566778889aabcccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cfb -e -in f3.txt -out cfbd3.txt -K 00112233445566778889aabcccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cfb -d -in cfbd3.txt -out cfbd3.txt -K 00112233445566778889aabcccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$
```

Figure 21: Encryption files using CFB

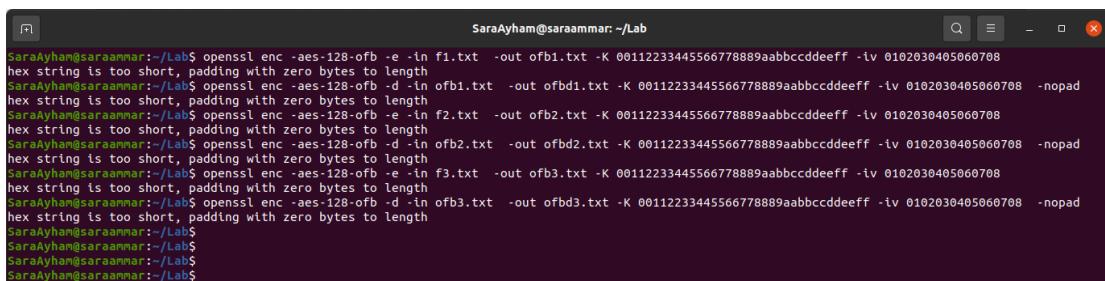


```
SaraAyham@saraammar:~/Lab$ hexdump -c cfbd1.txt
00000000  1 2 3 4 5
00000005
SaraAyham@saraammar:~/Lab$ xxd cfbd1.txt
00000000: 3132 3334 35          12345
SaraAyham@saraammar:~/Lab$ hexdump -c cfbd2.txt
00000000  1 2 3 4 5 6 7 8 9 1
0000000a
SaraAyham@saraammar:~/Lab$ xxd cfbd2.txt
00000000: 3132 3334 3536 3738 3931          1234567891
SaraAyham@saraammar:~/Lab$ hexdump -c cfbd3.txt
00000000  1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7
00000010
SaraAyham@saraammar:~/Lab$ xxd cfbd3.txt
00000000: 3132 3334 3536 3738 3931 3233 3435 3637 1234567891234567
SaraAyham@saraammar:~/Lab$
```

Figure 22: Encryption result using CFB

### 4. Encryption files using OFB:

Since OFB does not have padding, as illustrated in the figure below, it is the same concept as CFB in terms of padding.



```
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ofb -e -in f1.txt -out ofb1.txt -K 00112233445566778889aabcccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ofb -d -in ofb1.txt -out ofb1.txt -K 00112233445566778889aabcccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ofb -e -in f2.txt -out ofb2.txt -K 00112233445566778889aabcccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ofb -d -in ofb2.txt -out ofb2.txt -K 00112233445566778889aabcccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ofb -e -in f3.txt -out ofb3.txt -K 00112233445566778889aabcccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ofb -d -in ofb3.txt -out ofb3.txt -K 00112233445566778889aabcccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$
```

Figure 23: Encryption files using OFB

```
SaraAyham@saraammar:~/Lab$ hexdump -c ofbd1.txt
00000000  1 2 3 4 5
00000005
SaraAyham@saraammar:~/Lab$ xxd ofbd1.txt
00000000: 3132 3334 35                                12345
SaraAyham@saraammar:~/Lab$ hexdump -c ofbd2.txt
00000000  1 2 3 4 5 6 7 8 9 1
0000000a
SaraAyham@saraammar:~/Lab$ xxd ofbd2.txt
00000000: 3132 3334 3536 3738 3931                  1234567891
SaraAyham@saraammar:~/Lab$ hexdump -c ofbd3.txt
00000000  1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7
00000010
SaraAyham@saraammar:~/Lab$ xxd ofbd3.txt
00000000: 3132 3334 3536 3738 3931 3233 3435 3637  1234567891234567
SaraAyham@saraammar:~/Lab$
```

*Figure 24: Encryption result using OFB*

## Task 5: Error Propagation – Corrupted Cipher Text

In this task, the error proration property of encryption modes have been understand using the following steps:

1. The text file was created with size 1.1 Kb, as shown below:

*Figure 25: Creating file with 1.1 Kb size*

- The file has been encrypted using AES-128 cipher with four different modes, as shown below:
- Using ECB:

```

SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-ecb -e -in file.txt -out ecbE.txt -K 0011223344556677889aabccddeff
SaraAyham@saraammar:~/Lab$ bless hexdump < ecbE.txt
Gtk-MESSAGE: 20:44:08.000: Failed to load module "canberra-gtk-module"
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/export_patterns'.
Could not find file "/home/SaraAyham/.config/bless/export_patterns"
SaraAyham@saraammar:~/Lab$ hexdump < ecbD.txt
SaraAyham@saraammar:~/Lab$ bless hexdump < ecbD.txt

```

The terminal shows the command to encrypt 'file.txt' using AES-128 ECB mode with key '0011223344556677889aabccddeff'. It then uses the 'bless hexdump' command to view the resulting ciphertext 'ecbE.txt'. A message from the 'canberra-gtk-module' is displayed, indicating it failed to load. Finally, it attempts to view the decrypted file 'ecbD.txt'.

Figure 26: encryption and decryption using AES-128 ECB

After this step, the 55<sup>th</sup> byte in the encrypted file was edited, and the corrupted byte was achieved using bless hex editor, then the corrupted cipher text was decrypted using the correct key and IV, and the result was as shown in figure below: (note that the result of decryption is in the figure above). **In ECB, the block size completely has been corrupted.**

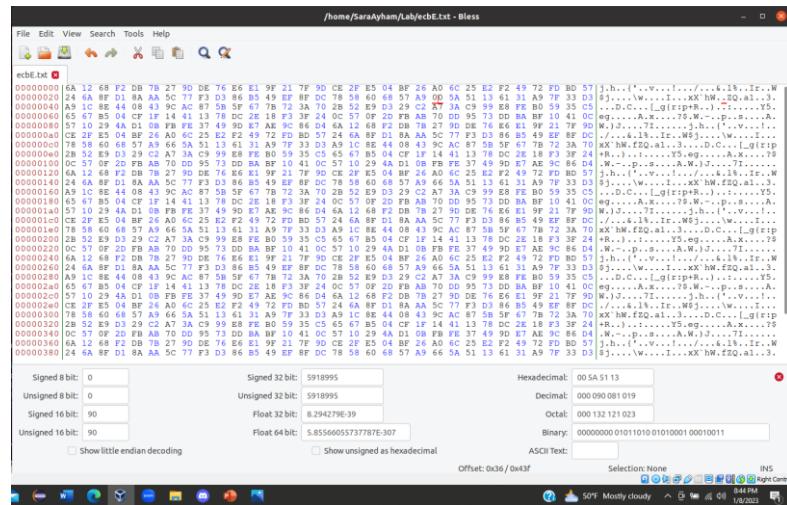


Figure 27: Edit 55th byte of the cipher text with ECB

## b. Using CBC:

```

SaraAyham@saraammar:~/Lab$ SaraAyham@saraammar:~/Lab$ SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -e -in file.txt -out cbcE.txt -K 00112233445566778889aabccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ bless cbcE.txt
Gtk-Message: 20:53:56.589: Failed to load module "canberra-gtk-module"
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/export_patterns'.
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -d -in cbcE.txt -out cbcD.txt -K 00112233445566778889aabccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ hexdump -c cbcD.txt
00000000 S a r a A y h a m S a r a A y h a m
00000010 a m S a r a A y h a m S a r a A y h a m
00000020 y h a m S a r a A y h a m S a r a A y h a m
00000030 223 | i { 005 200 216 + + u + 211 016 C + +
00000040 a r a A y h g m S a r a A y h a m S a r a A y h a m
00000050 m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000060 n a M S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000070 r a y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000080 r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000090 S a r a A y h a m S a r a A y h a m S a r a A y h a m
000000a0 a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
000000b0 j h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
000000c0 a y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
000000d0 a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
000000e0 m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
000000f0 h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000100 A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000110 r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000120 S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000130 a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000140 y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000150 a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000160 a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000170 m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000180 h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000190 A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
000001a0 r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
000001b0 S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
000001c0 a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
000001d0 y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
000001e0 a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
000001f0 a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000200 m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000210 h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000220 A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m
00000230 r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m S a r a A y h a m

```

Figure 28: Encryption and decryption using AES-128 CBC

Signed 8 bit:	0	Signed 32 bit:	6476582	Hexadecimal:	00 62 D3 26
Unsigned 8 bit:	0	Unsigned 32 bit:	6476582	Decimal:	000 098 211 038
Signed 16 bit:	98	Float 32 bit:	9.075624E-39	Octal:	000 142 323 046
Unsigned 16 bit:	98	Float 64 bit:	8.3773147847146E-307	Binary:	00000000 01100011 11010011 00100110

Show little endian decoding      Show unsigned as hexadecimal      ASCII Text:

Offset: 0x36 / 0x43F      Selection: None      INS

Figure 29: Edit 55th byte of the cipher text with CBC

### c. Using CFB:

```

SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cfb -e -in file.txt -out cfbE.txt -K 0011223344556677889aabccddeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ bless cfbE.txt
Gtk-Message: 21:00:11.838: Failed to load module "canberra-gtk-module"
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find file "/home/SaraAyham/.config/bless/export_patterns"
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cfb -d -in cfbE.txt -out cfbD.txt -K 0011223344556677889aabccddeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ hexdump -c cfbD.txt
000000 S a r a A y h a m S a r a r a A y h
000001 a m S a r a r a A y h a m S a r a r a A
000002 y h a m S a r a r a A y h a m S a r a r
000003 a A y h a m z a r a A y h a m S a r a r
000004 033 u q r 223 235 x 206 005 o o v ♦ \
000005 m S a r a r a A y h a m S a r a r a A y
000006 h a m S a r a r a A y h a m S a r a r a
000007 A y h a m S a r a r a A y h a m S a r a r
000008 a A y h a m S a r a r a A y h a m S a r a r
000009 S a r a r a A y h a m S a r a r a A y h a m
00000a a m S a r a r a A y h a m S a r a r a A y h a m
00000b y h a m S a r a r a A y h a m S a r a r a A y h a m
00000c a A y h a m S a r a r a A y h a m S a r a r a A y h a m
00000d a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
00000e m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
00000f h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
000010 A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
000011 r a A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
000012 S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
000013 a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
000014 y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
000015 a A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
000016 a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
000017 m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
000018 h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
000019 e A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
00001a0 r a A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
00001b0 S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
00001c0 a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
00001d0 d y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
00001e0 a A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
00001f0 a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
0000200 m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
0000210 h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
0000220 A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
0000230 r a A y h a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
0000240 S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m
0000250 a m S a r a r a A y h a m S a r a r a A y h a m S a r a r a A y h a m

```

Figure 30: Encryption and decryption using AES-128 CFB

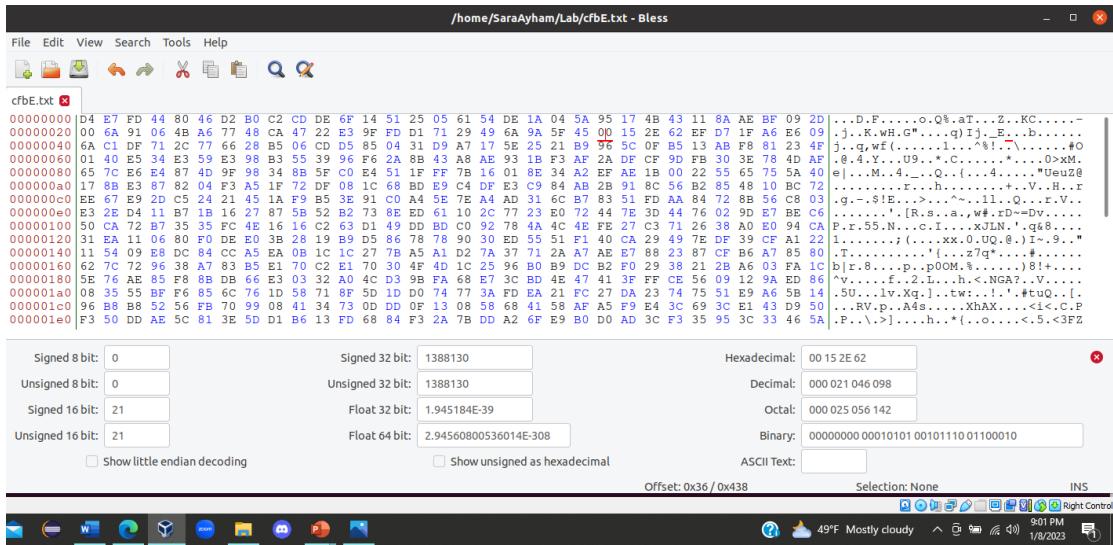


Figure 31: Edit 55th byte of the cipher text with CFB

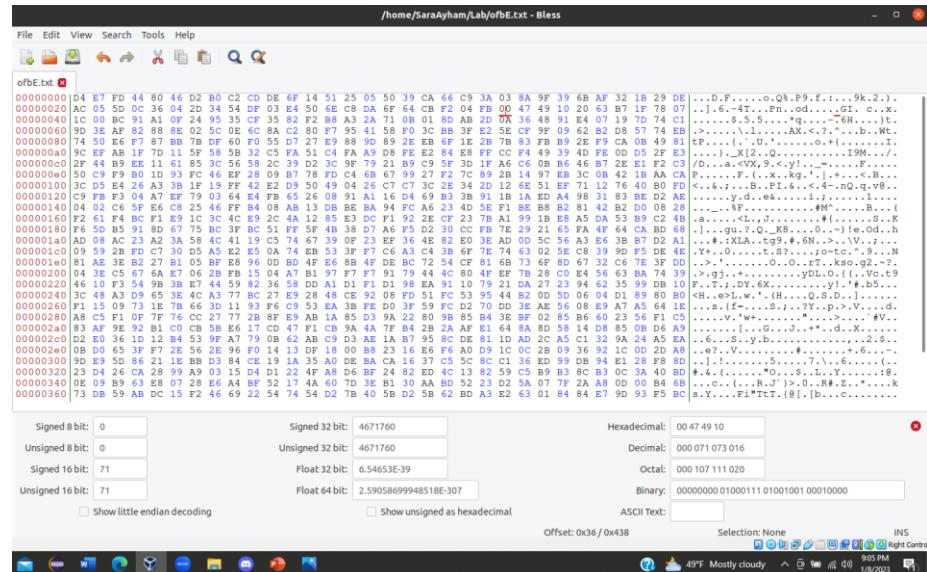
Also in CBC and CFB, the block size (16 Bytes) has been corrupted.

d. Using OFB:

```
SaraAyham@sarammar:~/Lab$ SaraAyham@sarammar:~/Lab$ openssl enc -aes-128-ofb -e -in file.txt -out ofbE.txt -K 0011223344556677889aabccddeff -iv 0102030405060708 hex string is too short, padding with zero bytes to length
SaraAyham@sarammar:~/Lab$ bless ofbE.txt
Gtk-Message: 21:05:28.785: Failed to load module "canberra-gtk-module"
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find file "/home/SaraAyham/.config/bless/export_patterns"

SaraAyham@sarammar:~/Lab$ SaraAyham@sarammar:~/Lab$ openssl enc -aes-128-ofb -d -in ofbE.txt -out ofbD.txt -K 0011223344556677889aabccddeff -iv 0102030405060708 hex string is too short, padding with zero bytes to length
SaraAyham@sarammar:~/Lab$ hexdump -c ofbD.txt
00000000  S a r a   A y h a m s a r a   A y h
00000010  a m S a r a   A y h a m s a r a   A y h
00000020  y h a m s a r a   A y h a m s a r a   A
00000030  a A y h a m   ◆ a r a   A y h a m s
00000040  a r a   A y h a m s a r a   A y h a m s
00000050  m s a r a   A y h a m s a r a   A y h a m s
00000060  h a m s a r a   A y h a m s a r a   A y h a m s
00000070  T a y h a m s a r a   A y h a m s a r a   A y h a m s
00000080  r r a   A y h a m s a r a   A y h a m s a r a   A y h a m s
00000090  S a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s
000000a0  a m S a r a   A y h a m s a r a   A y h a m s a r a   A
000000b0  y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
000000c0  a A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
000000d0  a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
000000e0  m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
000000f0  h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000100  A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000110  r r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000120  S a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000130  a m S a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000140  y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000150  a A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000160  a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000170  m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000180  h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000190  A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
000001a0  r r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
000001b0  S a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
000001c0  a m S a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
000001d0  y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
000001e0  a A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
000001f0  a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000200  m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000210  h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000220  A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
00000230  s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A y h a m s a r a   A
```

*Figure 32: Encryption and decryption using AES-128 OFB*



*Figure 33: Edit 55th byte of the cipher text with OFB*

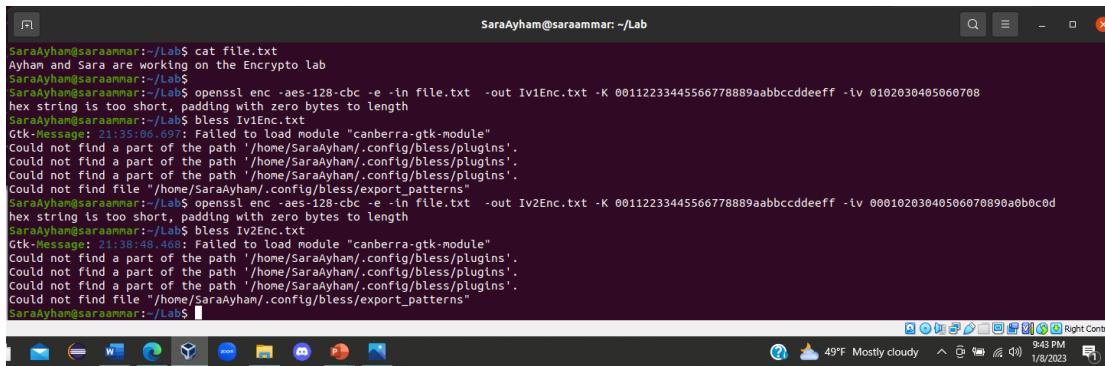
In OFB, the corrupted cipher text is only affected by the corrupted byte and not by any other changes.

## Task 6: Initial Vector (IV) and Common Mistakes

The objective in this task is to understand the problems if an IV is not selected properly.

### 6.1. IV Experiment

Encrypt the same plaintext using two different IVs:



```
SaraAyham@saraammar:~/Lab$ cat file.txt
Ayhan and Sara are working on the Encrypto lab
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -e -in file.txt -out Iv1Enc.txt -K 00112233445566778899aabccddeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ bless Iv1Enc.txt
Gtk-WARNING **: 21:38:48.077: Failed to load module "canberra-gtk-module"
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find file "/home/SaraAyham/.config/bless/export_patterns"
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -e -in file.txt -out Iv2Enc.txt -K 00112233445566778899aabccddeff -iv 00010203040506070890a0b0c0d
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ bless Iv2Enc.txt
Gtk-WARNING **: 21:38:48.078: Failed to load module "canberra-gtk-module"
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find file "/home/SaraAyham/.config/bless/export_patterns"
SaraAyham@saraammar:~/Lab$
```

Figure 34: Encrypting the same plaintext using two different IVs

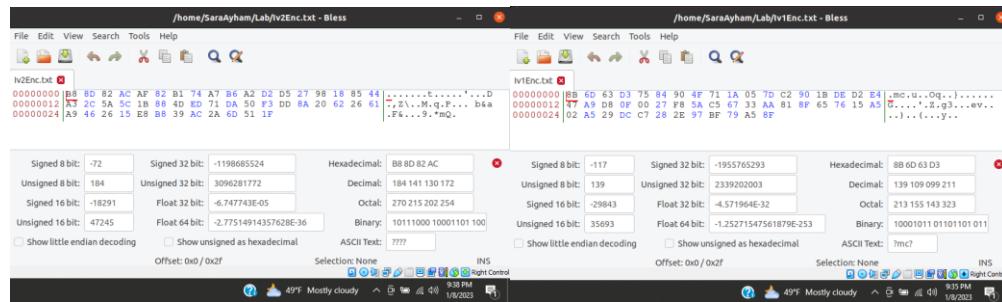
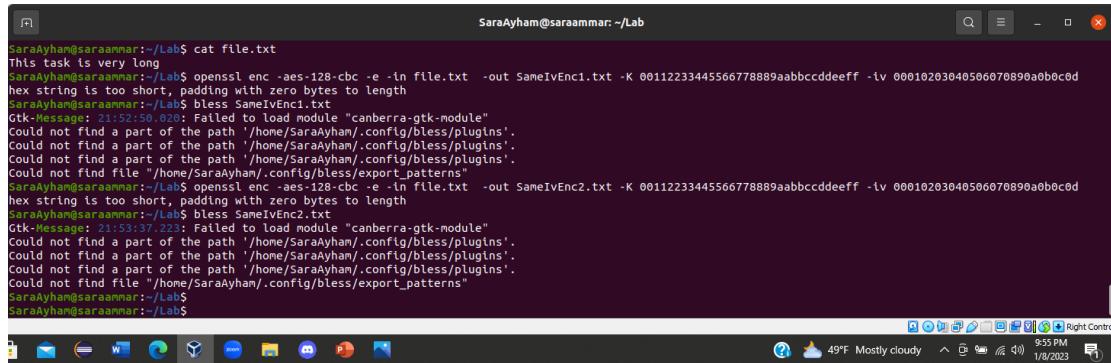


Figure 35: The resulting ciphertexts from using two different IVs

We note from the results mentioned above that the resulting encrypted texts differ from each other, although the encrypted texts are identical, and the reason for this difference is the difference in the IVs.

Encrypt the same plaintext using same IV:



```
SaraAyham@saraammar:~/Lab$ cat file.txt
This task is very long
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -e -in file.txt -out SameIVEnc1.txt -K 00112233445566778899aabbccddeeff -iv 00010203040506070890a0b0c0d
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ bless SameIVEnc1.txt
Gtk-Message: 21:52:58.020: Failed to load module "canberra-gtk-module"
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find file "/home/SaraAyham/.config/bless/export_patterns"
SaraAyham@saraammar:~/Lab$ openssl enc -aes-128-cbc -e -in file.txt -out SameIVEnc2.txt -K 00112233445566778899aabbccddeeff -iv 00010203040506070890a0b0c0d
hex string is too short, padding with zero bytes to length
SaraAyham@saraammar:~/Lab$ bless SameIVEnc2.txt
Gtk-Message: 21:53:37.223: Failed to load module "canberra-gtk-module"
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find a part of the path '/home/SaraAyham/.config/bless/plugins'.
Could not find file "/home/SaraAyham/.config/bless/export_patterns"
SaraAyham@saraammar:~/Lab$
```

Figure 36: Encrypting the same plaintext using the same IV

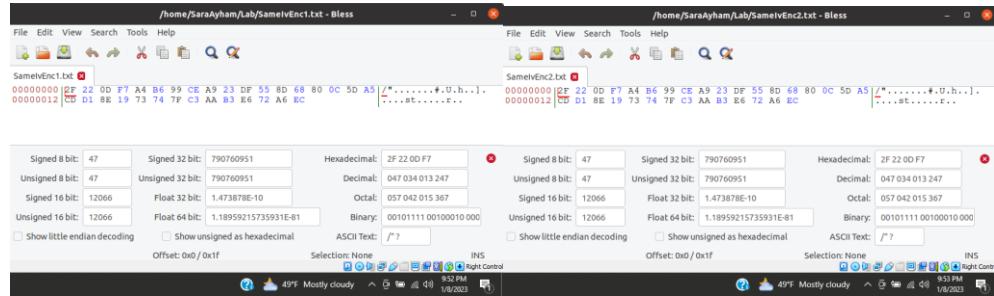


Figure 37: The resulting ciphertexts from using the same IV

We note from the above results that the resulting ciphertexts are similar, and the reason for this is that the same IV is used in the cipher process for the same text.

When the same IV is used with the same plaintext, they yield the same cipher text. Once a part is cracked, it makes it easier to follow the other plain text. This is a major flaw to security. So, same IV's must be avoided.

## 6.2. Common Mistake: Use the Same IV

Here we will discuss the issue that if the plaintext is not repeated, is it safe to use the same IV?

We assumed that the attacker got plaintext (P1), ciphertext (C1) and ciphertext (C2), and the encryption process was done using the same IV:

Plaintext (P1): This is a known message!

Ciphertext (C1): a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159

Plaintext (P2): (unknown to you)

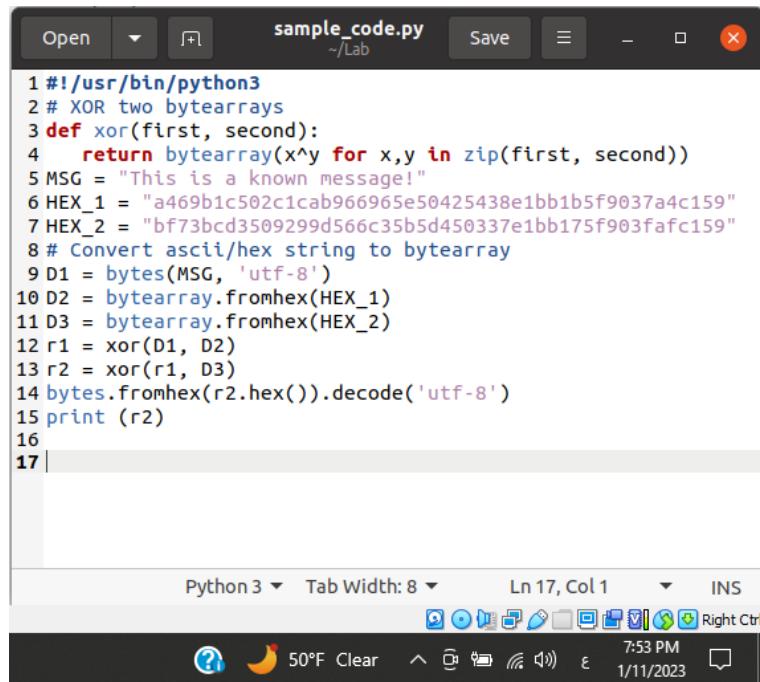
Ciphertext (C2): bf73bcd3509299d566c35b5d450337e1bb175f903fafc159

By formula,

$$P1 \text{ XOR } P2 = C1 \text{ XOR } C2$$

$$\text{So, } P2 = C1 \text{ XOR } C2 \text{ XOR } P1$$

We have been able to find out the actual content of P2 based on C2, P1 and C1.



The screenshot shows a Jupyter Notebook cell with the following Python code:

```
1 #!/usr/bin/python3
2 # XOR two bytearrays
3 def xor(first, second):
4     return bytearray(x^y for x,y in zip(first, second))
5 MSG = "This is a known message!"
6 HEX_1 = "a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159"
7 HEX_2 = "bf73bcd3509299d566c35b5d450337e1bb175f903fafc159"
8 # Convert ascii/hex string to bytearray
9 D1 = bytes(MSG, 'utf-8')
10 D2 = bytearray.fromhex(HEX_1)
11 D3 = bytearray.fromhex(HEX_2)
12 r1 = xor(D1, D2)
13 r2 = xor(r1, D3)
14 bytes.fromhex(r2.hex()).decode('utf-8')
15 print (r2)
16
17 |
```

The notebook interface includes tabs for 'Python 3' and 'Tab Width: 8'. The status bar at the bottom shows the current line (Ln 17, Col 1), mode (INS), and system information like battery level (50F), time (7:53 PM), and date (1/11/2023).

Figure 38: Sample\_code to find P2 content

*Figure 39: P2 Content*

P2 String =Order: Launch a missile!

So, if the plaintext is not repeated, it is not safe to use the same IV.

### 6.3. Common Mistake: Use a Predictable IV

This task is more inclined towards chosen-plaintext attack than known-plaintext attack. The IV that are used are different but predictable.

By formula,

$$P2Bob = (IV2 \text{ xor } IV1 \text{ xor } P1Bob)$$

Since, we know the IV1, IV2 and the P1 of Bob, the formula can be used to find the P2 that would be sent by Bob since the IV is predictable.

*Figure 40: P2 Bob message*

From the above results, it turns out that Bob's first message is **yes**, because the ciphertext output for  $(IV_1 \text{ xor } IV_{\text{next}} \text{ xor } P(\text{yes}))$  is the same ciphertext output for the first message.

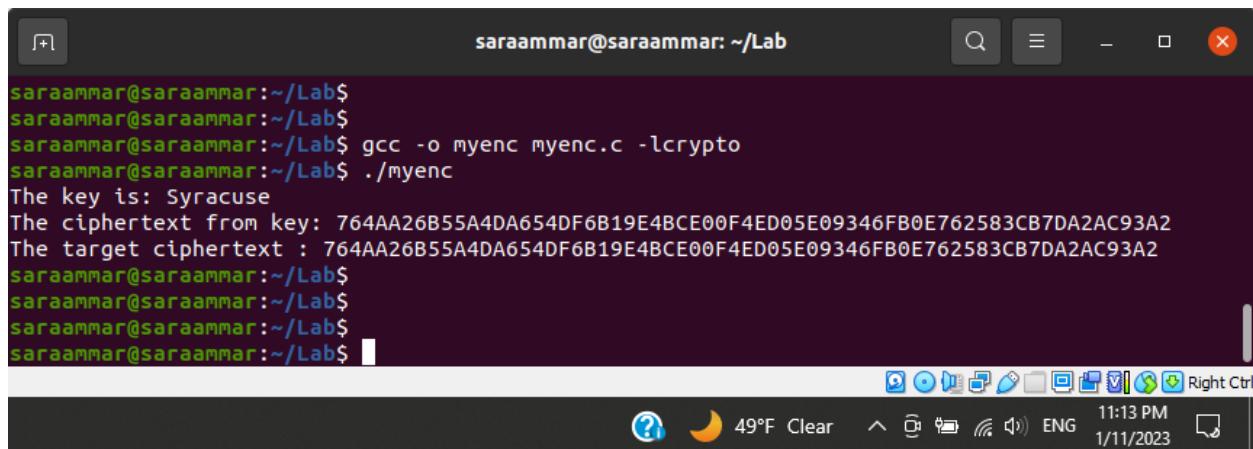
So, using predictable IV is not secure.

## Task 7: Programming using the Crypto Library

In this task, we have a given plaintext and a ciphertext, our job is to find the key that is used for the encryption. We know that the aes-128-cbc cipher is used for the encryption, and the key used to encrypt this plaintext is an English word shorter than 16 characters from a typical English dictionary.

- Plaintext (total 21 characters): This is a top secret.
- Ciphertext (in hex):  
764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2
- IV (in hex): aabbccddeeff00998877665544332211

We have written a C program to find the encryption key. [See Appendix2](#)



The screenshot shows a terminal window with a dark background. The terminal session starts with the user's name and location: `saraammar@saraammar:~/Lab$`. The user then runs a command to compile a C program named `myenc.c` using `gcc -o myenc myenc.c -lcrypto`. After compilation, the user runs the executable `./myenc`. The output of the program is displayed in the terminal, showing the key as "Syracuse" and the ciphertext as "764AA26B55A4DA654DF6B19E4BCE00F4ED05E09346FB0E762583CB7DA2AC93A2". The terminal window also includes a system tray at the bottom with various icons and status information like the date and time.

Figure 41: The result of the key

The key is: Syracuse (has number 22287 in the dictionary).

## Conclusion

in conclusion, we learned how apply the theoretical information about encryption and decryption in secure-key encryption to get familiar with these concepts, there various ways used to learn the difference and security between each other , we faced an small problems in the last task ; because the task wasn't easy to be understood, after the task was understood and analyzed the task the solution was easy, this lab was very important for our, cause we gained an small expertise of how encrypt and decrypt the messages with unique keys.

## Appendix

### 1. The plaintext before decryption:

```
#!/bin/env python3
```

```
Cipher = []
```

```
repetition = {}
```

```
fileName = open('/home/saraammar/Lab/ciphertext.txt', '+r')
```

```
for line in fileName:
```

```
    for element in line.split(' '):
```

```
        Cipher.append(element.strip())
```

```
Cipher = list(filter(None, Cipher))
```

```
buffer = {}
```

```
for word in Cipher:
```

```
    for character in word:
```

```
        if character not in buffer.keys():
```

```
            buffer[character] = 1
```

```
        else:
```

```
            buffer[character] += 1
```

```
repetition = dict(sorted(buffer.items(), key=lambda x:x[1], reverse=True))
```

```
k = 0
```

```
for word in Cipher:
```

```
    freq = list(word)
```

```
j = 0
```

```
for character in word:
```

```
    if character == 'n':
```

```
        freq[j] = 'E'
```

```
    elif character == 't':
```

```
        freq[j] = 'H'
```

```
    elif character == 'y':
```

```
        freq[j] = 'T'
```

```
    elif character == 'v':
```

```
        freq[j] = 'A'
```

```
    elif character == 'x':
```

```
        freq[j] = 'O'
```

```
    elif character == 'u':
```

```
        freq[j] = 'N'
```

```
    elif character == 'b':
```

```
        freq[j] = 'F'
```

```
    elif character == 'p':
```

```
        freq[j] = 'D'
```

```
    elif character == 'g':
```

```

        freq[j] = 'B'
    elif character == 'h':
        freq[j] = 'R'
    elif character == 'm':
        freq[j] = 'T'
    elif character == 'i':
        freq[j] = 'L'
    elif character == 'q':
        freq[j] = 'S'
    elif character == 's':
        freq[j] = 'K'
    elif character == 'f':
        freq[j] = 'V'
    elif character == 'z':
        freq[j] = 'U'
    elif character == 'd':
        freq[j] = 'Y'
    elif character == 'c':
        freq[j] = 'M'
    elif character == 'r':
        freq[j] = 'G'
    elif character == 'e':
        freq[j] = 'P'
    elif character == 'a':
        freq[j] = 'C'
    elif character == 't':
        freq[j] = 'W'
    elif character == 'k':
        freq[j] = 'X'
    elif character == 'o':
        freq[j] = 'J'
    elif character == 'j':
        freq[j] = 'Q'
    elif character == 'w':
        freq[j] = 'Z'
    else:
        freq[j] = character
    j += 1
tempString = "".join(freq)
Cipher[k] = tempString
k += 1

for i in Cipher:
    print(i, end=" ")

```

2. C code:

```
// main.cpp
#include <openssl/aes.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <stdlib.h>

#define charMaxLeng 20

unsigned char AscToHex(unsigned char Char){
    int aChar = (int)Char;

    if((aChar>=0x30)&&(aChar<=0x39))
        aChar -= 0x30;
    else if((aChar>=0x41)&&(aChar<=0x46))
        aChar -= 0x37;
    else if((aChar>=0x61)&&(aChar<=0x66))
        aChar -= 0x57;
    else
        aChar = 0xff;

    return aChar;
}

unsigned char HexToAsc(unsigned char aHex){
    if((aHex>=0)&&(aHex<=9))
        aHex += 0x30;
    else if((aHex>=10)&&(aHex<=15))//A-F
        aHex += 0x37;
    else
        aHex = 0xff;

    return aHex;
}

unsigned char* str2hex(char *str) {
    unsigned char *ret = NULL;
    int str_len = strlen(str);
    int i = 0;
    // printf("%d \n", str_len);
    // printf("%s\n", str);
    assert((str_len%2) == 0);
    ret = (char *)malloc(str_len/2);
    for (i = 0;i < str_len; i = i+2 ) {
        sscanf(str+i,"%2hhx",&ret[i/2]);
    }
}
```

```

    }
    return ret;
}

char *padding_buf(char *buf,int size, int *final_size) {
    char *ret = NULL;
    int ppadding_size = AES_BLOCK_SIZE - (size % AES_BLOCK_SIZE);
    int i;

    *final_size = size + ppadding_size;
    ret = (char *)malloc(size+ppadding_size);
    memcpy( ret, buf, size);
    if (ppadding_size!=0) {
        for (i =size;i < (size+ppadding_size); i++ ) {
            ret[i] = ppadding_size;
        }
    }

    return ret;
}

void printf_buff(char *buff,int size) {
    int i = 0;
    for (i=0;i<size;i ++ ) {
        // printf( "%02X ", (unsigned char)buff[i] );
        printf( "%02X", (unsigned char)buff[i] );
        if ((i+1) % 8 == 0) {
            // printf("\n");
        }
    }

    printf("\n");
}

void encrpyt_buf(char *raw_buf, char **encrpyt_buf, int len, char* kkey) {
    AES_KEY aes;
    unsigned char *key = str2hex(kkey);
    unsigned char *iv = str2hex("aabbccddeeff00998877665544332211");
    AES_set_encrypt_key(key,128,&aes);
    AES_cbc_encrypt(raw_buf,*encrpyt_buf,len,&aes,iv,AES_ENCRYPT);

    free(key);
    free(iv);
}

void decrpyt_buf(char *raw_buf, char **encrpyt_buf, int len, char* kkey) {
    AES_KEY aes;
    unsigned char *key = str2hex(kkey);

```

```

unsigned char *iv = str2hex("aabbccddeeff00998877665544332211");
AES_set_decrypt_key(key,128,&aes);
AES_cbc_encrypt(raw_buf,*encrpy_buf,len,&aes,iv,AES_DECRYPT);

free(key);
free(iv);
}

int main(int argc, char* argv[]) {
    char*
target="764AA26B55A4DA654DF6B19E4BCE00F4ED05E09346FB0E762583CB7DA2AC93A2";
    FILE* p=NULL;
    if((p=fopen("words.txt","r"))==NULL)
    {
        printf("ERROR");
    }

    char buffer[charMaxLeng];
    char buf2[charMaxLeng];
    int flag=0;

    while (!feof(p)){
        int i=0;
        memset(buffer,'0', charMaxLeng * sizeof(char));
        memset(buf2,'0', charMaxLeng * sizeof(char));
        fgets(buffer, charMaxLeng, p);
        while(i<charMaxLeng){
            buf2[i]=buffer[i];
            i+=1;
        }
        size_t len = strlen(buffer);
        if (len == 1)      continue;
        // printf("%s", buffer);

        char *raw_buf = NULL;
        char *after_padding_buf = NULL;
        int padding_size = 0;
        char *encrypt_buf = NULL;
        char *decrypt_buf = NULL;

        i=0;
        unsigned char* key=NULL;
        key=(unsigned char*)malloc(33);

        while(i < strlen(buffer)){
            unsigned char letter = buffer[i];

            key[2*i] = HexToAsc(letter/0x10);

```

```

key[2*i+1] = HexToAsc(letter%0x10);
// printf("%d\n", i);

++i;
if(i==0x0f || buffer[i] < 0x20)
    break;
}
while(i < 0x10){
    key[2*i] = '2';
    key[2*i+1] = '3';
    ++i;
}
key[0x20]='\0';
// printf("%s\n", key);
raw_buf = (char *)malloc(21);
memcpy(raw_buf,"This is a top secret.",21);
after_padding_buf = padding_buf(raw_buf,21,&padding_size);

encrypt_buf = (char *)malloc(padding_size);
encrpyt_buf(after_padding_buf,&encrypt_buf, padding_size, key);

// printf("%d\n", strlen(target));
i=0;
char temp='\0';
flag=1;
while(i<padding_size){
    temp = HexToAsc((unsigned char)encrypt_buf[i]/0x10);
    if(temp!=target[2*i]){
        flag=0;
        break;
    }
    temp = HexToAsc((unsigned char)encrypt_buf[i]%0x10);
    if(temp!=target[2*i+1]){
        flag=0;
        break;
    }
    i+=1;
}
if(flag==0){
    continue;
}
printf("The key is: %s", buf2);
printf("The ciphertext from key: ");
printf_buff(encrypt_buf,padding_size);
printf("The target ciphertext : %s\n", target);

free(raw_buf);
free(after_padding_buf);
free(encrypt_buf);

```

```
    free(decrypt_buf);
    // printf("%02X\n", (unsigned char)(T));
    break;
}
fclose(p);
return 0;
}
```