

CPS 633 Section 09

Secret-Key Encryption Lab

Group 18

Roxie Reginold (501087897)

Hetu Virajkumar Patel (501215707)

Sayyada Aisha Mehvish (501106795)

TASK 1: Frequency Analysis

This task involved decrypting an encrypted file using frequency analysis.

We ran the `freq.py` script on the encrypted file, which generated statistics for n-grams, including single letter, bigram, and trigram frequencies.

Running freq.py script

```
[09/26/24] seed@VM:~/.../Files$ ./freq.py
```

```
-----
```

```
1-gram (top 20):
```

```
n: 488  
y: 373  
v: 348  
x: 291  
u: 280  
q: 276  
m: 264  
h: 235  
t: 183  
i: 166  
p: 156  
a: 116  
c: 104  
z: 95  
l: 90  
g: 83  
b: 83  
r: 82  
e: 76  
d: 59
```

```
-----
```

```
2-gram (top 20):
```

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

```

3-gram (top 20):
ytn: 78
vup: 30
mur: 20
ynh: 18
xzy: 16
mxu: 14
gnq: 14
ytv: 13
nqy: 13
vii: 13
bxh: 13
lvq: 12
nuy: 12
vyn: 12
uvy: 11
lmu: 11
nvh: 11
cmu: 11
tmq: 10
vhp: 10

```

We compared the script's output with standard frequency statistics for English language n-grams, referencing:

<https://en.wikipedia.org/wiki/Bigram>

<https://en.wikipedia.org/wiki/Trigram>

By comparing our frequency analysis results with standard English language patterns, we made initial inferences:

- n likely corresponds to E (the most common letter in English)
- yt probably represents TH (a common bigram)
- ytn likely stands for THE (the most frequent trigram)

We progressively substituted encrypted characters with their probable plaintext counterparts, refining our substitutions as more text became readable.

This table shows the mapping between the encrypted letters to their respective plaintext letters:

Letter to encrypt	Plain text Letter	Letter to encrypt	Plain text Letter
y	T	t	H
n	E	v	A
d	Y	u	N
p	D	q	S
z	U	a	C

h	R	l	W
m	I	x	O
f	V	g	B
e	P	i	L
r	G	b	F
s	K	j	Q
o	J	k	X
w	Z	c	M

Throughout this process, we utilized the tr command to perform character substitutions. Our final decryption command was:

```
tr 'nyvtimuqxpclahzdfjgrebsokw' 'ET AHLINSODMWCRUYVQBGPFKJXZ' < ciphertext.txt >
out1.txt
```



```
[09/28/24] seed@VM:~/.../Files$ tr 'nyvtimuqxpclahzdfjgrebsokw' 'ET AHLINSODMWCRUYVQBGPFKJXZ' < ciphertext.txt > out1.txt
[09/28/24] seed@VM:~/.../Files$
```

Output produced after running the final decryption command:

```
[09/28/24]seed@VM:~/.../Files$ cat out1.txt
THE OSCARS TURN ON SUNDAY WHICH SEEMS ABOUT RIGHT AFTER THIS LONG STRANGE
AWARDS TRIP THE BAGGER FEELS LIKE A NONAGENARIAN TOO
```

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

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

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

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

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

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

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

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 ITS VOTES ARE REDISTRIBUTED TO THE MOVIES THAT GARNERED THE ELIMINATED BALLOTS SECONDPLACE VOTES AND THIS CONTINUES UNTIL A WINNER EMERGES

IT IS ALL TERRIBLY CONFUSING BUT APPARENTLY THE CONSENSUS FAVORITE COMES OUT AHEAD IN THE END THIS MEANS THAT ENDOFSEASON AWARDS CHATTER INVARIBLY INVOLVES TORTURED SPECULATION ABOUT WHICH FILM WOULD MOST LIKELY BE VOTERS

IT IS ALL TERRIBLY CONFUSING BUT APPARENTLY THE CONSENSUS FAVORITE COMES OUT AHEAD IN THE END THIS MEANS THAT END OF SEASON AWARDS CHATTER INVARIABLY INVOLVES TORTURED SPECULATION ABOUT WHICH FILM WOULD MOST LIKELY BE VOTERS SECOND OR THIRD FAVORITE AND THEN EQUALLY TORTURED CONCLUSIONS ABOUT WHICH FILM MIGHT PREVAIL

IN IT WAS A TOSSUP BETWEEN BOYHOOD AND THE EVENTUAL WINNER BIRDMAN IN WITH LOTS OF EXPERTS BETTING ON THE 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 SNAFU 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 THAT FILM WHILE DIVISIVE ALSO WON THE BEST DRAMA GOLDEN GLOBE AND THE BAFTA BUT ITS FILMMAKER MARTIN MCDONAGH WAS NOT NOMINATED FOR BEST DIRECTOR AND APART FROM ARGO MOVIES THAT LAND BEST PICTURE WITHOUT ALSO EARNING BEST DIRECTOR NOMINATIONS ARE FEW AND FAR BETWEEN

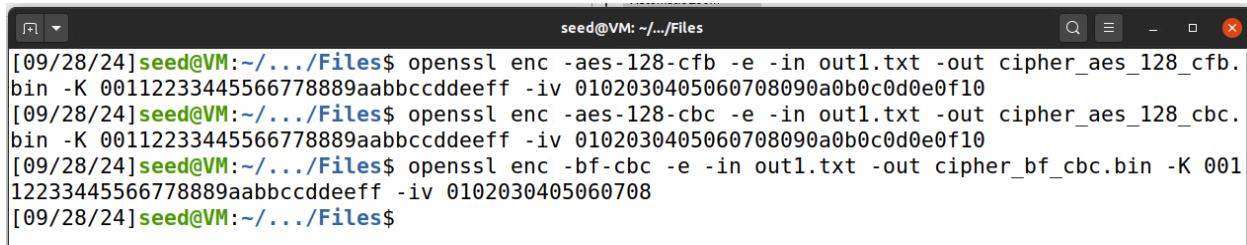
TASK 2: Encryption using Different Ciphers and Modes

OpenSSL is a toolkit for SSL and TLS protocols in which the library contains cryptography functions like operations, creating and managing private keys, public keys and parameters.

We ran the output from task1 and encrypted it using 3 different methods:

- -aes-128-cbc
- -aes-128-cfb
- -bf-cbc

Screenshot of encryption via terminal using different methods:



```
seed@VM: ~/.../Files
[09/28/24] seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in out1.txt -out cipher_aes_128_cfb.bin -K 00112233445566778889aabccddeeff -iv 0102030405060708090a0b0c0d0e0f10
[09/28/24] seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in out1.txt -out cipher_aes_128_cbc.bin -K 00112233445566778889aabccddeeff -iv 0102030405060708090a0b0c0d0e0f10
[09/28/24] seed@VM:~/.../Files$ openssl enc -bf-cbc -e -in out1.txt -out cipher_bf_cbc.bin -K 00112233445566778889aabccddeeff -iv 0102030405060708
[09/28/24] seed@VM:~/.../Files$
```

AES-128-CBC

AES-128-CBC (Advanced Encryption Standard with a 128-bit key in Cipher Block Chaining mode) is a symmetric encryption method that processes data in blocks of 128 bits. It uses a 128-bit key for both encryption and decryption. In CBC mode, each block of plaintext is combined with the previous ciphertext block using an XOR operation before encryption. The first block uses an Initialization Vector (IV) to start the process. This method helps hide patterns in the plaintext, making it more secure than simpler modes like ECB.

BF-CBC

BF-CBC (Blowfish in Cipher Block Chaining mode) is a symmetric encryption method that works with 64-bit blocks of data. It uses a variable-length key ranging from 32 to 448 bits. Like AES-CBC, Blowfish in CBC mode combines each plaintext block with the previous ciphertext block using XOR before encrypting it.

AES-128-CFB

AES-128-CFB (AES with a 128-bit key in Cipher Feedback mode) also uses the AES algorithm but operates differently. In CFB mode, the previous ciphertext is encrypted and then combined with the current plaintext using XOR to produce the next ciphertext block. This turns the block cipher into a stream cipher, allowing for the encryption of data in smaller units than the block size. CFB mode does not require padding.

Task 3: Encryption Mode – ECB vs. CBC

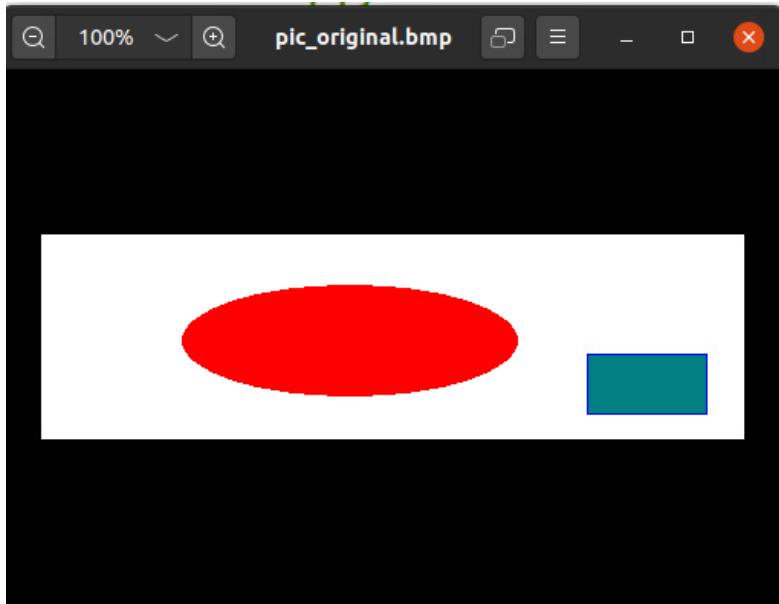
In this task, we were given a .bmp-type image. We had to encrypt it using the ECB encryption mode and CBC encryption mode.

The following commands were run to encrypt this picture using the ECB and CBC mode.

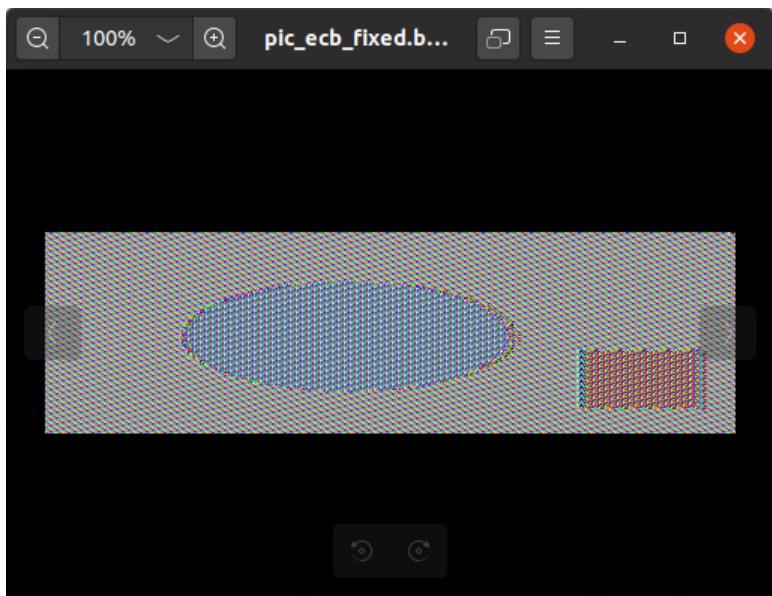
```
[09/28/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out pic_ecb.bmp  
-K 00112233445566778889aabcccddeeff  
[09/28/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in pic_original.bmp -out pic_cbc.bmp  
-K 00112233445566778889aabcccddeeff -iv 0102030405060708090a0b0c0d0e0f10  
[09/28/24]seed@VM:~/.../Files$ head -c 54 pic_original.bmp > header  
[09/28/24]seed@VM:~/.../Files$ tail -c +55 pic_ecb.bmp > body_ecb  
[09/28/24]seed@VM:~/.../Files$ tail -c +55 pic_cbc.bmp > body_cbc  
[09/28/24]seed@VM:~/.../Files$ tail -c +55 pic_cbc.bmp > body_cbc_fixed.bmp  
[09/28/24]seed@VM:~/.../Files$ cat header body_ecb > pic_ecb_fixed.bmp  
[09/28/24]seed@VM:~/.../Files$ cat header body_cbc > pic_cbc_fixed.bmp  
[09/28/24]seed@VM:~/.../Files$ eog pic_ecb_fixed.bmp  
[09/28/24]seed@VM:~/.../Files$ eog pic_cbc_fixed.bmp  
[09/28/24]seed@VM:~/.../Files$ eog pic_original.bmp  
[09/28/24]seed@VM:~/.../Files$ eog pic_ecb_fixed.bmp  
[09/28/24]seed@VM:~/.../Files$ eog pic_cbc_fixed.bmp
```

In this process, we see that all the image bytes are encrypted but we just want to cipher them from the 55th byte onwards. Therefore we grab the first 54 bytes of the original image (since that is the header for a .bmp file) and append the bytes from 55 offset to the rest of the file. This results in no errors when displaying the encrypted image.

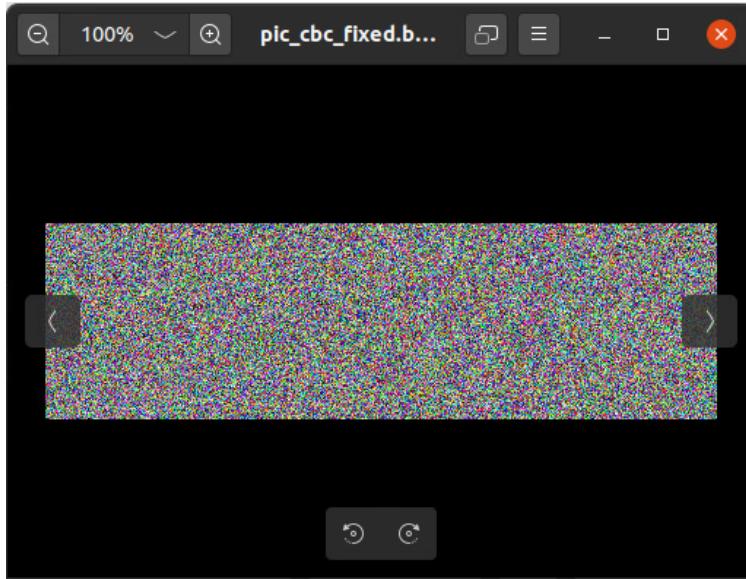
Original image:



Observations for ECB mode:



Observations for CBC mode:



Results:

Based on our observations, we can conclude that in CBC (Cipher Block Chaining) mode, the encrypted image is difficult to distinguish from the original image. However, in ECB (Electronic Code Book) mode, the picture can still be recognized. This difference occurs because:

- In ECB mode, each block of data is encrypted separately.
- In CBC mode, the encryption of each block depends on the previous block.

For CBC, the process works like this:

- The system takes the encrypted data from the previous block.
- It combines this with the current block's data using an XOR operation.
- The result is then encrypted.

This chaining process makes CBC more secure, as it hides patterns in the original data better than ECB mode.

We then repeated the same procedure with another image. We got the image from

<https://filesampleshub.com/format/image/bmp>

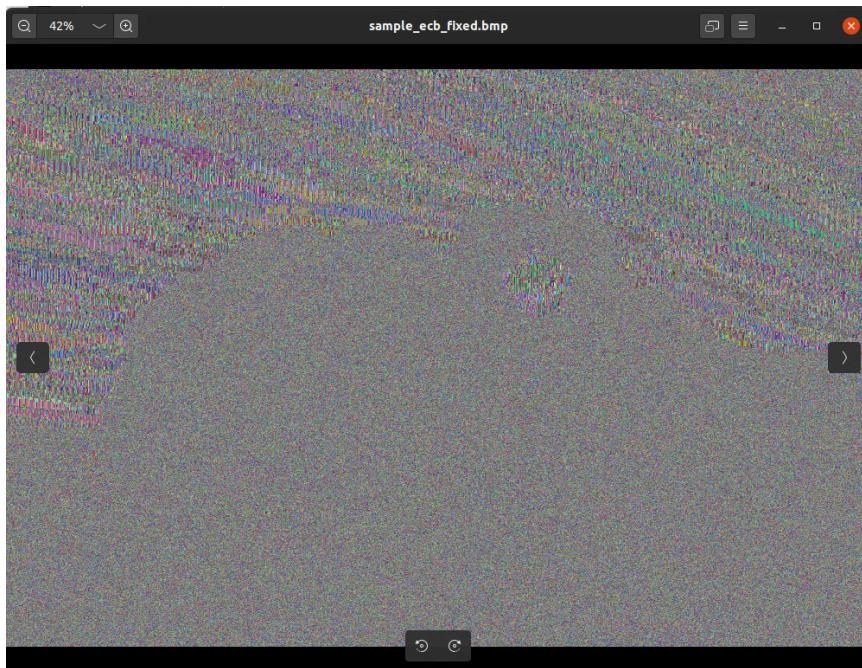
The following commands were run to encrypt this sample picture using the ECB and CBC mode.

```
[09/28/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in sample.bmp -out sample_ecb.bmp -K 00112233445566778889aabcccddeeff  
[09/28/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in sample.bmp -out sample_cbc.bmp -K 00112233445566778889aabcccddeeff -iv 0102030405060708090a0b0c0d0e0f10  
[09/28/24]seed@VM:~/.../Files$ head -c 54 sample.bmp > header  
[09/28/24]seed@VM:~/.../Files$ tail -c +55 sample_ecb.bmp > sample_body_ecb  
[09/28/24]seed@VM:~/.../Files$ tail -c +55 sample_cbc.bmp > sample_body_cbc  
[09/28/24]seed@VM:~/.../Files$ cat header sample_body_ecb > sample_ecb_fixed.bmp  
[09/28/24]seed@VM:~/.../Files$ cat header sample_body_cbc > sample_cbc_fixed.bmp  
[09/28/24]seed@VM:~/.../Files$
```

Original sample image



Observations for ECB mode:



Observations for CBC mode:

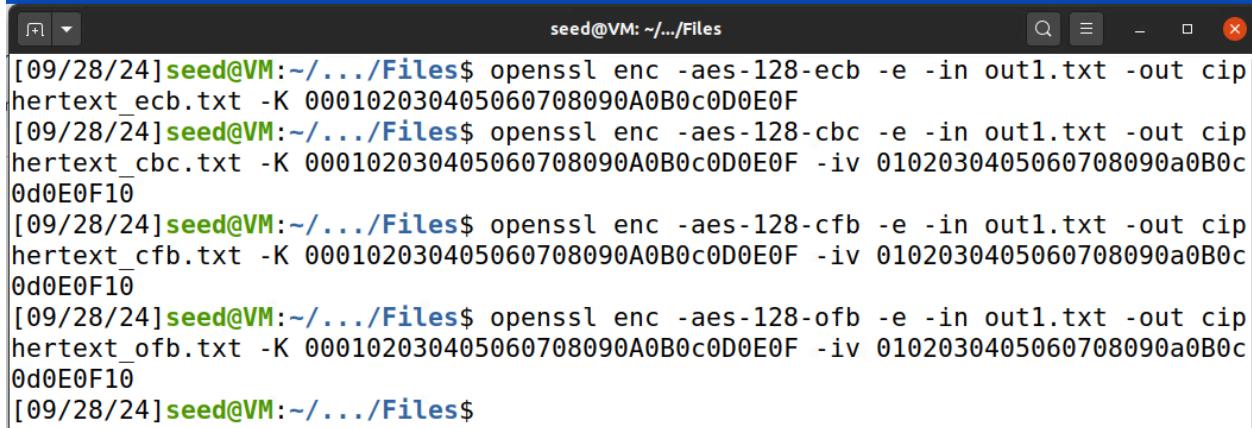


Results:

We observe the same for the reasons mentioned above. CBC mode was able to encrypt the picture, such that the elements of the original image were fully hidden or hard to detect.

Task 4: Padding

1. In this task, we ran the file with different modes such as ECB, CBC, CFB and OFB.



```
seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in out1.txt -out ciphertext_ecb.txt -K 000102030405060708090A0B0c0D0E0F
[09/28/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in out1.txt -out ciphertext_cbc.txt -K 000102030405060708090A0B0c0D0E0F -iv 0102030405060708090a0B0c0d0E0F10
[09/28/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in out1.txt -out ciphertext_cfb.txt -K 000102030405060708090A0B0c0D0E0F -iv 0102030405060708090a0B0c0d0E0F10
[09/28/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in out1.txt -out ciphertext_ofb.txt -K 000102030405060708090A0B0c0D0E0F -iv 0102030405060708090a0B0c0d0E0F10
[09/28/24]seed@VM:~/.../Files$
```

We know that block ciphers operate on fixed-size blocks of data. When the data size is not a multiple of the block size, padding is required to fill the last block.

ECB (Electronic Code Book):

Padding Required: Yes

Reason: ECB operates on fixed-size blocks, so padding is needed when data is not a perfect multiple of the block size extra bits are needed to be added to make it fit.

CBC (Cipher Block Chaining):

Padding Required: Yes

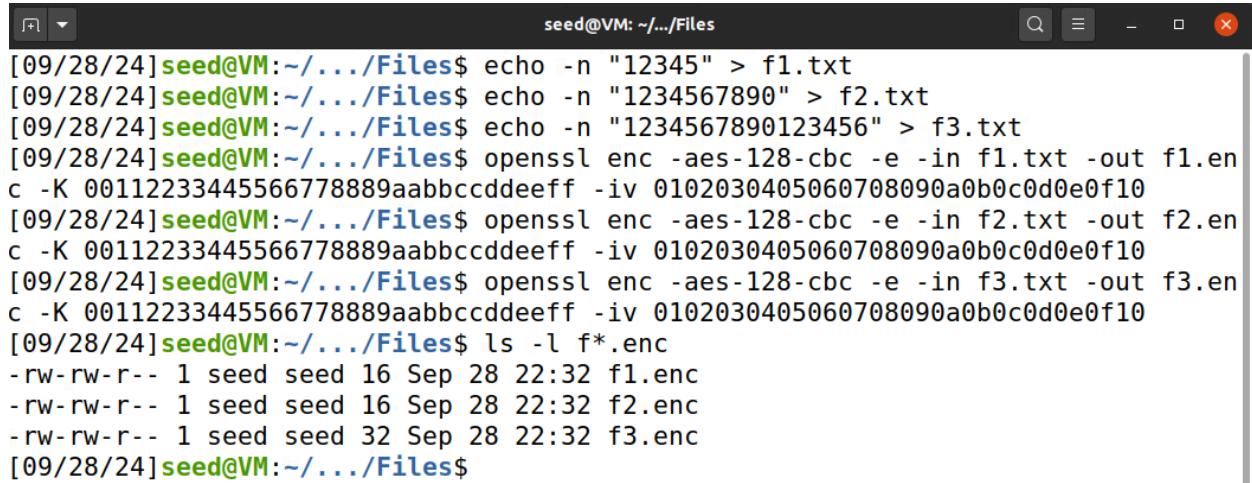
Reason: Like ECB, CBC works on blocks and requires padding if the plaintext size is not a multiple of the block size.

CFB (Cipher Feedback)/ OFB (Output Feedback):

Padding Required: No

Reason: These modes function like stream ciphers, processing data bit by bit or in smaller increments, which allows them to work without needing the data to fit into complete blocks. Since they both don't need fixed-sized blocks, any size of data can be used for an adaptable-sized block.

2. First we create 3 files which contain 5 bytes, 10 bytes, and 16 bytes, respectively. We then use "openssl enc -aes-128-cbc -e" to encrypt these three files using 128-bit AES with CBC mode:



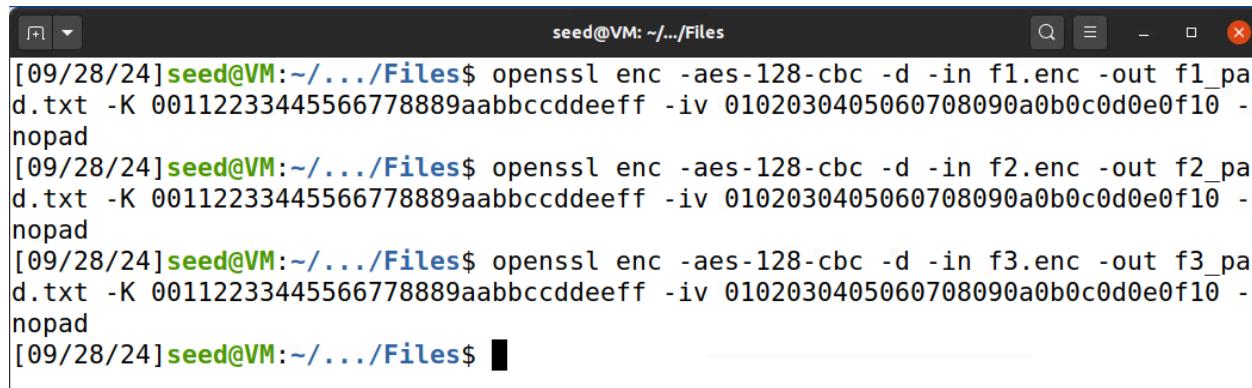
```
[09/28/24] seed@VM:~/.../Files$ echo -n "12345" > f1.txt
[09/28/24] seed@VM:~/.../Files$ echo -n "1234567890" > f2.txt
[09/28/24] seed@VM:~/.../Files$ echo -n "1234567890123456" > f3.txt
[09/28/24] seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f1.txt -out f1.enc -K 00112233445566778889aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
[09/28/24] seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f2.txt -out f2.enc -K 00112233445566778889aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
[09/28/24] seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f3.txt -out f3.enc -K 00112233445566778889aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
[09/28/24] seed@VM:~/.../Files$ ls -l f*.enc
-rw-rw-r-- 1 seed seed 16 Sep 28 22:32 f1.enc
-rw-rw-r-- 1 seed seed 16 Sep 28 22:32 f2.enc
-rw-rw-r-- 1 seed seed 32 Sep 28 22:32 f3.enc
[09/28/24] seed@VM:~/.../Files$
```

We find the size of the encrypted files to be as follows:

- f1.enc: 16 bytes
- f2.enc: 16 bytes
- f3.enc: 32 bytes

The reason for this is that AES has a block size of 16 bytes. Therefore the sizes are always padded to the nearest multiples of 16.

Decryption:



```
[09/28/24] seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in f1.enc -out f1_padded.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10 -nopad
[09/28/24] seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in f2.enc -out f2_padded.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10 -nopad
[09/28/24] seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in f3.enc -out f3_padded.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10 -nopad
[09/28/24] seed@VM:~/.../Files$
```

Viewing the padded content of file1

```
seed@VM: ~/.../Files$ hexdump -C f1_pad.txt
00000000  31 32 33 34 35 0b |12345....|
00000010
[09/28/24]seed@VM:~/.../Files$ xxd f1_pad.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b 12345.....
[09/28/24]seed@VM:~/.../Files$
```

Viewing the padded content of file2

```
seed@VM: ~/.../Files$ hexdump -C f2_pad.txt
00000000  31 32 33 34 35 36 37 38 39 30 06 06 06 06 06 |1234567890....|
00000010
[09/28/24]seed@VM:~/.../Files$ xxd f2_pad.txt
00000000: 3132 3334 3536 3738 3930 0606 0606 0606 1234567890.....
[09/28/24]seed@VM:~/.../Files$
```

Viewing the padded content of file3

```
seed@VM: ~/.../Files$ hexdump -C f3_pad.txt
00000000  31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 |1234567890123456|
00000010  10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 |.....|
00000020
[09/28/24]seed@VM:~/.../Files$ xxd f3_pad.txt
00000000: 3132 3334 3536 3738 3930 3132 3334 3536 1234567890123456
00000010: 1010 1010 1010 1010 1010 1010 1010 1010 .....
[09/28/24]seed@VM:~/.../Files$
```

Results::

f1.txt (5 bytes):

Padding added: 11 bytes (0x0b) repeated.

f2.txt (10 bytes):

Padding added: 6 bytes (0x06) repeated.

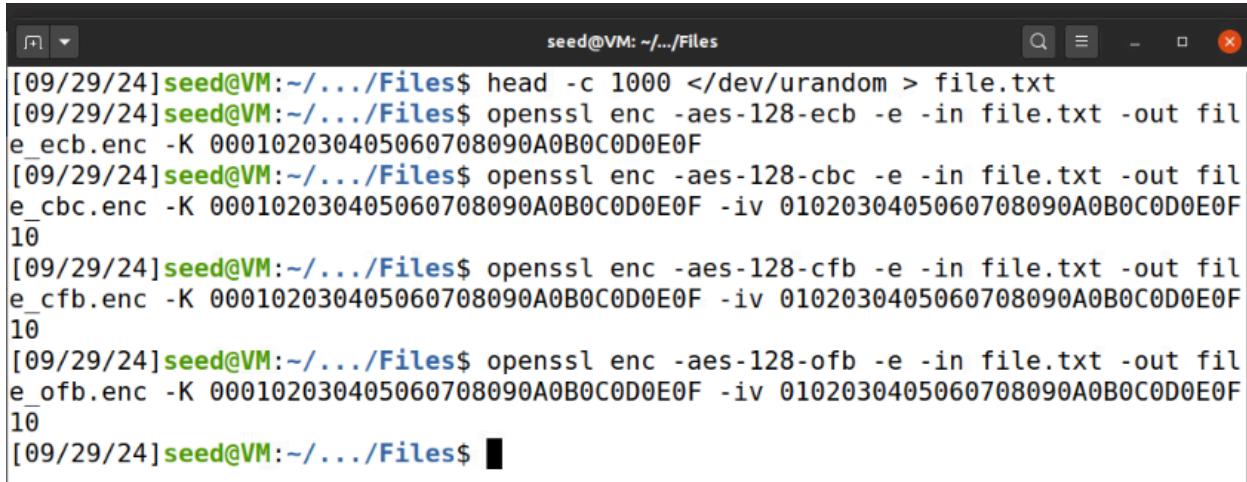
f3.txt (16 bytes):

Full padding block added: 16 bytes (0x10) repeated.

Task 5: Error Propagation – Corrupted Cipher Text

In this task we first created a text file of 1000 bytes by running this command: **head -c 1000 </dev/urandom > file.txt**.

We then encrypted the file using the AES-128 cipher.



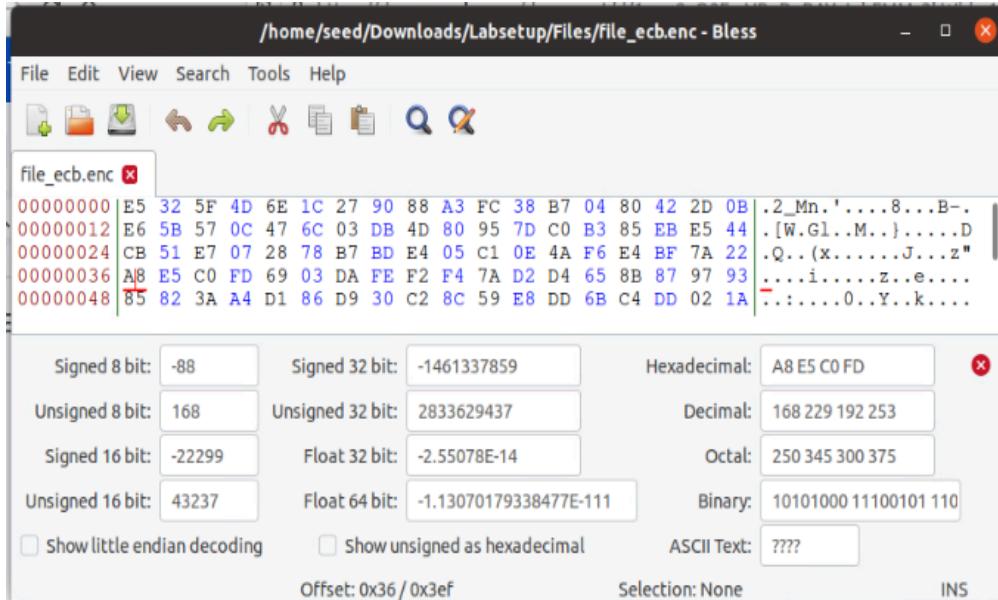
```
seed@VM:~/.../Files$ head -c 1000 </dev/urandom > file.txt
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in file.txt -out file_ecb.enc -K 000102030405060708090A0B0C0D0E0F
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in file.txt -out file_cbc.enc -K 000102030405060708090A0B0C0D0E0F -iv 0102030405060708090A0B0C0D0E0F10
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in file.txt -out file_cfb.enc -K 000102030405060708090A0B0C0D0E0F -iv 0102030405060708090A0B0C0D0E0F10
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in file.txt -out file_ofb.enc -K 000102030405060708090A0B0C0D0E0F -iv 0102030405060708090A0B0C0D0E0F10
[09/29/24]seed@VM:~/.../Files$
```

Prediction before conducting task:

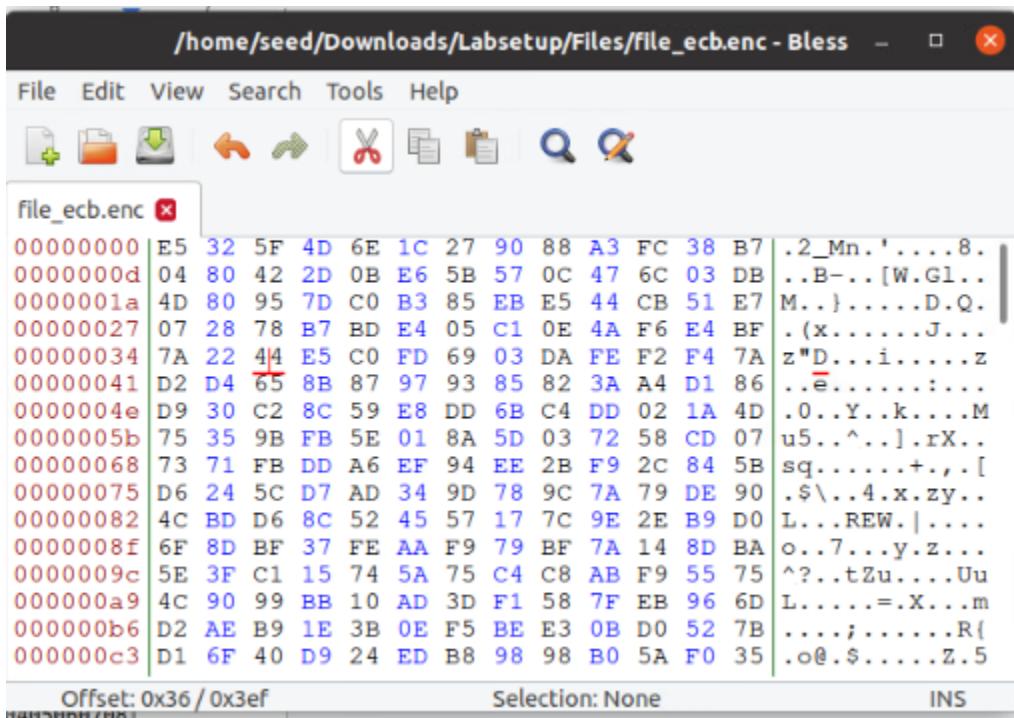
When decrypting the corrupted file, the outcomes will differ based on the encryption mode employed. In modes such as ECB and OFB, we should not anticipate significant errors, as these modes do not depend on the ciphertext from prior steps. Conversely, in modes like CBC and CFB, which rely on the previous ciphertext, we can expect a higher number of errors and more pronounced changes.

ECB Mode:

Before the corruption of the 55th byte:



After the corruption of the 55th byte:



CBC Mode:

Before the corruption of the 55th byte:

/home/seed/Downloads/Labsetup/Files/file_cbc.enc - Bless

File Edit View Search Tools Help

file_cbc.enc

00000000	84 81 3C 99 B1 1F 16 F5 22 8A F4 E2 4D 56 DA D2 15 33	..<....."....MV...3
00000012	3F A4 DA DE 13 83 0B AF 7F 73 14 7B 98 54 C9 34 52 2C	?.....s.{.T.4R,
00000024	BF 55 25 E1 D2 AB 8B 2C 0F C6 EC C1 B8 42 84 51 51 3F	.U%.....,.....B.QQ?
00000036	CF 5C 39 C3 C0 BB 36 B8 BE FC 8F 15 FC 0D AD 7E 40 61	\9...6.....~@a
00000048	AF 93 55 3A E5 5B FD 79 2A 6C 8F 0A 5D D2 04 FB A9 96	..U:[.y*1..]....
0000005a	53 4C B7 9E B8 26 E5 24 52 8C 5B F6 2B 16 D2 6C 92 F1	SL...&.\$R.[.+.1..
0000006c	15 2D A8 50 E2 E3 1A 77 07 5E 72 38 74 76 E7 81 AA 2E	.-.P...w.^r8tv....
0000007e	22 0D 41 18 8F 33 CE 70 43 F9 22 16 69 6C 8C 6D 2F 90	".A..3.pC.".il.m..
00000090	88 05 DA 45 A7 C1 64 EF 3A F2 DF 70 B3 E8 2B 19 10 BC	...E..d.:..p..+....
000000a2	A8 56 E2 CA 46 44 9E 84 C0 02 01 E3 71 1A C7 7B 09 A2	.V..FD.....q.{..
000000b4	F1 6F 70 2C 6C B0 86 D0 68 F5 B7 4A 36 58 41 C8 DC D7	.op,1...h..J6XA...
000000c6	2F 85 14 CC 8F 2D 92 C8 A4 A5 3D 93 26 1A A8 E9 8F 9F	/....-....=.&....
000000d8	56 2F C4 21 FC 57 5B EO 51 83 D8 D2 2B 15 60 53 44 33	V/.!W[.Q...+.`SD3
000000ea	61 5D E9 56 63 F7 D2 7A 4E 5E 7D 37 2F 78 9F 3B EF E9 a].Vc..zN^}7/x.;...	a].Vc..zN^}7/x.;...
000000cf	E6 AF F7 5E 62 6C 87 F3 91 54 2A D1 EC 58 4F D1 BC 85	...~bl...T*.XO...
0000010e	14 81 45 63 21 49 F9 4F 70 FA 96 5B CA 1A AC 82 F1 07	..Ec!I.Op..[.....

Offset: 0x36 / 0x3ef Selection: None INS

After the corruption of the 55th byte:

/home/seed/Downloads/Labsetup/Files/file_cbc.enc - Bless

File Edit View Search Tools Help

file_cbc.enc

00000000	84 81 3C 99 B1 1F 16 F5 22 8A F4 E2 4D	..<....."....M
0000000d	56 DA D2 15 33 3F A4 DA DE 13 83 0B AF	V...3?.....
0000001a	7F 73 14 7B 98 54 C9 34 52 2C BF 55 25	.s.{.T.4R,.U%
00000027	E1 D2 AB 8B 2C 0F C6 EC C1 B8 42 84 51,.....B.Q
00000034	51 3F 44 5C 39 C3 C0 BB 36 B8 BE FC 8F	Q?D\9...6....
00000041	15 FC 0D AD 7E 40 61 AF 93 55 3A E5 5B~@a..U:[
0000004e	FD 79 2A 6C 8F 0A 5D D2 04 FB A9 96 53	.y*1..]....S
0000005b	4C B7 9E B8 26 E5 24 52 8C 5B F6 2B 16	L...&.\$R.[.+.
00000068	D2 6C 92 F1 15 2D A8 50 E2 E3 1A 77 07	.l...-..P...w.
00000075	5E 72 38 74 76 E7 81 AA 2E 22 0D 41 18	^r8tv....".A.
00000082	8F 33 CE 70 43 F9 22 16 69 6C 8C 6D 2F	.3.pC.".il.m/
0000008f	90 88 05 DA 45 A7 C1 64 EF 3A F2 DF 70E..d.:..p
0000009c	B3 E8 2B 19 10 BC A8 56 E2 CA 46 44 9E	.+....V..FD.
000000a9	84 C0 02 01 E3 71 1A C7 7B 09 A2 F1 6Fq...{...o
000000b6	70 2C 6C B0 86 D0 68 F5 B7 4A 36 58 41	p,1...h..J6XA
000000c3	C8 DC D7 2F 85 14 CC 8F 2D 92 C8 A4 A5	.../.....-

T... Offset: 0x37 / 0x3ef Selection: None INS

CFB Mode:

Before the corruption of the 55th byte:

/home/seed/Downloads/Labsetup/Files/file_cfb.enc - Bless

File Edit View Search Tools Help

file_cfb.enc

00000000	FC F9 74 52 C0 27 71 34 06 9F 61 70 21	..tR.'q4..ap!
0000000d	C8 60 02 D1 37 9A E7 50 84 86 71 4A 0B	.'..7..P..qJ.
0000001a	83 34 EE BD 64 FB DB 28 B3 E5 86 AF 72	.4..d..(....r
00000027	3D C9 FE 32 1C A7 B4 23 FA B0 84 BD 70	=..2...#....p
00000034	3A F0 30 35 B1 D1 C5 43 66 BC 0C 70 E5	:.05...cf..p.
00000041	DC CB 6D B8 5A 63 31 94 3C 19 EF C3 8F	..m.Zcl.<....
0000004e	0F EB F1 BF CE 26 C2 6A 31 A9 1C 7C A7&.j1.. .
0000005b	C6 0B AA DA C2 CF E7 01 1B 8D 22 C2 0B".
00000068	FA 93 BF 3A 12 03 28 E4 8A 87 E3 B4 76:(....v
00000075	65 16 59 B1 81 67 6F D0 71 4E 8F A8 3C	e.Y..go.qN..<
00000082	3D 12 07 CE 11 27 E4 40 4C 75 47 00 99	=....'.@LuG..
0000008f	5B 67 1C F6 E3 64 28 49 92 42 A0 D3 39	[g...d(I.B..9
0000009c	95 49 AE CD 10 44 01 A0 E4 F4 E6 DA C2	.I...D.....
000000a9	D3 3D 91 E6 5E E4 5D CA E6 55 CF DC D6	.=..^.]..U...
000000b6	25 B8 19 35 29 DA EC 19 35 91 04 76 1D	%..5)...5..v.
000000c3	FF 97 CD F7 4E FA 12 4B A5 18 2C CO B2N..K...,..

Offset: 0x36 / 0x3e7 Selection: None INS

After the corruption of the 55th byte:

/home/seed/Downloads/Labsetup/Files/file_cfb.enc - Bless

File Edit View Search Tools Help

file_cfb.enc

00000000	FC F9 74 52 C0 27 71 34 06 9F 61 70 21	..tR.'q4..ap!
0000000d	C8 60 02 D1 37 9A E7 50 84 86 71 4A 0B	.'..7..P..qJ.
0000001a	83 34 EE BD 64 FB DB 28 B3 E5 86 AF 72	.4..d..(....r
00000027	3D C9 FE 32 1C A7 B4 23 FA B0 84 BD 70	=..2...#....p
00000034	3A F0 44 B5 B1 D1 C5 43 66 BC 0C 70 E5	:.D5...cf..p.
00000041	DC CB 6D B8 5A 63 31 94 3C 19 EF C3 8F	..m.Zcl.<....
0000004e	0F EB F1 BF CE 26 C2 6A 31 A9 1C 7C A7&.j1.. .
0000005b	C6 0B AA DA C2 CF E7 01 1B 8D 22 C2 0B".
00000068	FA 93 BF 3A 12 03 28 E4 8A 87 E3 B4 76:(....v
00000075	65 16 59 B1 81 67 6F D0 71 4E 8F A8 3C	e.Y..go.qN..<
00000082	3D 12 07 CE 11 27 E4 40 4C 75 47 00 99	=....'.@LuG..
0000008f	5B 67 1C F6 E3 64 28 49 92 42 A0 D3 39	[g...d(I.B..9
0000009c	95 49 AE CD 10 44 01 A0 E4 F4 E6 DA C2	.I...D.....
000000a9	D3 3D 91 E6 5E E4 5D CA E6 55 CF DC D6	.=..^.]..U...
000000b6	25 B8 19 35 29 DA EC 19 35 91 04 76 1D	%..5)...5..v.
000000c3	FF 97 CD F7 4E FA 12 4B A5 18 2C CO B2N..K...,..

T... Offset: 0x37 / 0x3e7 Selection: None INS

OFB Mode:

Before the corruption of the 55th byte:

/home/seed/Downloads/Labsetup/Files/file_ofb.enc - Bless

File Edit View Search Tools Help

file_ofb.enc	00000000	FC F9 74 52 C0 27 71 34 06 9F 61 70 21	..tR.'q4..ap!
0000000d	C8 60 02 4B 9C 8C 2E 06 1D A6 C9 66 7C	.`K.....f	
0000001a	B5 33 6D 7B 0C 89 D5 CB 95 C7 2F 6F FA	.3m{.....o.	
00000027	4D 72 A6 E0 EE 0C FB 04 0C 73 08 A0 B7	Mr.....s...	
00000034	93 10 89 8D 75 4E 69 99 FD 49 E0 B2 64	...uNi..I..d	
00000041	37 FF 1C CE 08 27 E4 1E 48 7D 77 2B 0A	7....'..H}w+.	
0000004e	95 7A D5 68 E9 89 7C 5D 17 C7 76 B5 E5	.z.h..]..v..	
0000005b	90 5B 5C D3 F4 EB EF AF 9A B3 79 F0 FE	.[\.....y..	
00000068	F0 D4 4F 24 20 5C 20 72 FE 69 12 C8 17	..O\$ \ r.i...	
00000075	86 A4 56 4E 44 16 92 98 D6 79 64 93 77	..VND....yd.w	
00000082	F9 CC AE 60 39 43 DE 16 41 FA E7 BF 71	...`9C..A...q	
0000008f	93 A1 0C AE CC FF C9 0F FE 1F 55 0D 7FU..	
0000009c	7C E4 B3 10 A2 7B 7B A1 4F 18 AB 8F E2{.O....	
000000a9	BA CE F6 EA BF FF 0B 51 B6 3D 65 AE 4FQ.=e.O	
000000b6	CC A7 82 84 60 A4 6D CF 66 E8 9E 98 C6`m.f....	
000000c3	E2 8B A9 DC 3E 15 C6 DA DB 11 F4 AD 1A>.....	

Offset: 0x36 / 0x3e7 Selection: None INS

After the corruption of the 55th byte:

/home/seed/Downloads/Labsetup/Files/file_ofb.enc - Bless

File Edit View Search Tools Help

file_ofb.enc	00000000	FC F9 74 52 C0 27 71 34 06 9F 61 70 21	..tR.'q4..ap!
0000000d	C8 60 02 4B 9C 8C 2E 06 1D A6 C9 66 7C	.`K.....f	
0000001a	B5 33 6D 7B 0C 89 D5 CB 95 C7 2F 6F FA	.3m{.....o.	
00000027	4D 72 A6 E0 EE 0C FB 04 0C 73 08 A0 B7	Mr.....s...	
00000034	93 10 44 8D 75 4E 69 99 FD 49 E0 B2 64	..D.uNi..I..d	
00000041	37 FF 1C CE 08 27 E4 1E 48 7D 77 2B 0A	7....'..H}w+.	
0000004e	95 7A D5 68 E9 89 7C 5D 17 C7 76 B5 E5	.z.h..]..v..	
0000005b	90 5B 5C D3 F4 EB EF AF 9A B3 79 F0 FE	.[\.....y..	
00000068	F0 D4 4F 24 20 5C 20 72 FE 69 12 C8 17	..O\$ \ r.i...	
00000075	86 A4 56 4E 44 16 92 98 D6 79 64 93 77	..VND....yd.w	
00000082	F9 CC AE 60 39 43 DE 16 41 FA E7 BF 71	...`9C..A...q	
0000008f	93 A1 0C AE CC FF C9 0F FE 1F 55 0D 7FU..	
0000009c	7C E4 B3 10 A2 7B 7B A1 4F 18 AB 8F E2{.O....	
000000a9	BA CE F6 EA BF FF 0B 51 B6 3D 65 AE 4FQ.=e.O	
000000b6	CC A7 82 84 60 A4 6D CF 66 E8 9E 98 C6`m.f....	
000000c3	E2 8B A9 DC 3E 15 C6 DA DB 11 F4 AD 1A>.....	

T... Offset: 0x37 / 0x3e7 Selection: None INS

We then decrypt the files accordingly

```
seed@VM: ~/.../Files$ openssl enc -aes-128-ecb -d -in file_ecb.enc -out file_ecb_decrypted.txt -K 000102030405060708090A0B0C0D0E0F
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in file_cbc.enc -out file_cbc_decrypted.txt -K 000102030405060708090A0B0C0D0E0F -iv 0102030405060708090A0B0C0D0E0F10
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in file_cfb.enc -out file_cfb_decrypted.txt -K 000102030405060708090A0B0C0D0E0F -iv 0102030405060708090A0B0C0D0E0F10
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in file_ofb.enc -out file_ofb_decrypted.txt -K 000102030405060708090A0B0C0D0E0F -iv 0102030405060708090A0B0C0D0E0F10
[09/29/24]seed@VM:~/.../Files$
```

Using the xxd command to view the hexdump of each file:

```
[09/29/24]seed@VM:~/.../Files$ xxd file.txt > file_hex.txt
[09/29/24]seed@VM:~/.../Files$ xxd file_ecb_decrypted.txt > file_ecb_decrypted_hex.txt
[09/29/24]seed@VM:~/.../Files$ xxd file_cbc_decrypted.txt > file_cbc_decrypted_hex.txt
[09/29/24]seed@VM:~/.../Files$ xxd file_cfb_decrypted.txt > file_cfb_decrypted_hex.txt
[09/29/24]seed@VM:~/.../Files$ xxd file_ofb_decrypted.txt > file_ofb_decrypted_hex.txt
[09/29/24]seed@VM:~/.../Files$
```

ECB mode observations:

```
seed@VM: ~/.../Files$ diff file_hex.txt file_ecb_decrypted_hex.txt
4c4
< 00000030: 2d05 36d3 d307 713e 2038 f15b 1725 4a6a  -.6...q> 8.[.%Jj
---
> 00000030: d491 f7b6 9837 4bbb f55b f273 89c4 3485  .....7K..[.s..4.
[09/29/24]seed@VM:~/.../Files$
```

Only the block containing the corrupted byte is affected. This is because the ECB mode is not a chain, and no block depends on the previous one. Therefore, 1 block (16 bytes) is corrupted (the block containing the 55th byte).

CBC mode observations:

The block containing the corrupted byte and the following block are affected. Since the current block is always dependent on the previous block, the affected area is much larger. Here, 2 blocks (32 bytes) are corrupted (the block containing the 55th byte and the following block).

CFB mode observations:

The block containing the corrupted byte and the following block are affected.

CFB operates as a stream cipher. The corrupted bit will cause all following bits to be XORed with incorrect values, propagating the error throughout.

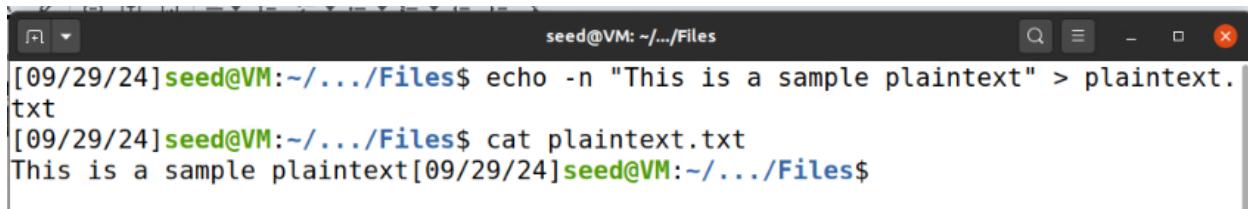
OFB mode observations:

Only the single corrupted byte is affected. OFB generates a keystream independent of the ciphertext. A corrupted bit in the ciphertext only affects the corresponding plaintext bit only.

Task 6: Initial Vector (IV) and Common Mistakes

Task 6.1. IV Experiment

In this task, we first created a sample plaintext.



```
seed@VM: ~/.../Files$ echo -n "This is a sample plaintext" > plaintext.txt
[09/29/24]seed@VM:~/.../Files$ cat plaintext.txt
This is a sample plaintext[09/29/24]seed@VM:~/.../Files$
```

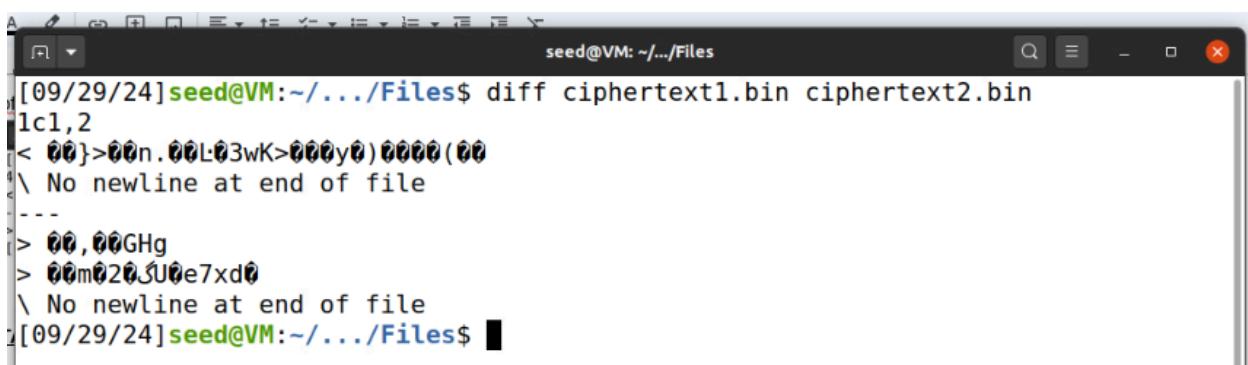
We then encrypted the file with different Initialization vectors and the same key:

- IV1 = 9b6b75321d3e4f896574234895135d73
- IV2 = 5b3a12393d7b0f128538254972538e96



```
seed@VM: ~/.../Files$ openssl enc -aes-128-cbc -e -in plaintext.txt -out ciphertext1.bin -K 000102030405060708090A0B0C0D0E0F -iv 9b6b75321d3e4f896574234895135d73
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plaintext.txt -out ciphertext2.bin -K 000102030405060708090A0B0C0D0E0F -iv 5b3a12393d7b0f128538254972538e96
[09/29/24]seed@VM:~/.../Files$
```

We then check if there is any difference between the two encrypted files:



```
seed@VM: ~/.../Files$ diff ciphertext1.bin ciphertext2.bin
1c1,2
< 00}>00n.00L03wK>000y0)0000(00
\ No newline at end of file
---
> 00,00GHg
> 00m0203U0e7xd0
\ No newline at end of file
[09/29/24]seed@VM:~/.../Files$
```

```
seed@VM: ~/.../Files$ xxd ciphertext1.bin > ciphertext1_hex.bin
[09/29/24]seed@VM:~/.../Files$ xxd ciphertext2.bin > ciphertext2_hex.bin
[09/29/24]seed@VM:~/.../Files$ diff ciphertext1_hex.bin ciphertext2_hex.bin
1,2c1,2
< 00000000: c9fd 7d3e deee 6e2e afca c4bf 9533 774b ..}...n.....3wK
< 00000010: 3ed6 c0aa 79ef 291e 97aa a28b 28eb c41c >...y.)....(...
---
> 00000000: 161b 2bfc ccca 2c13 96fa 0547 4867 0ad2 ..+....,...GHg..
I> 00000010: d76d e8ac daaf a032 5504 f365 3778 6497 .m....2U..e7xd.
[09/29/24]seed@VM:~/.../Files$
```

Observation:

Different IVs produce different ciphertexts for the same plaintext.

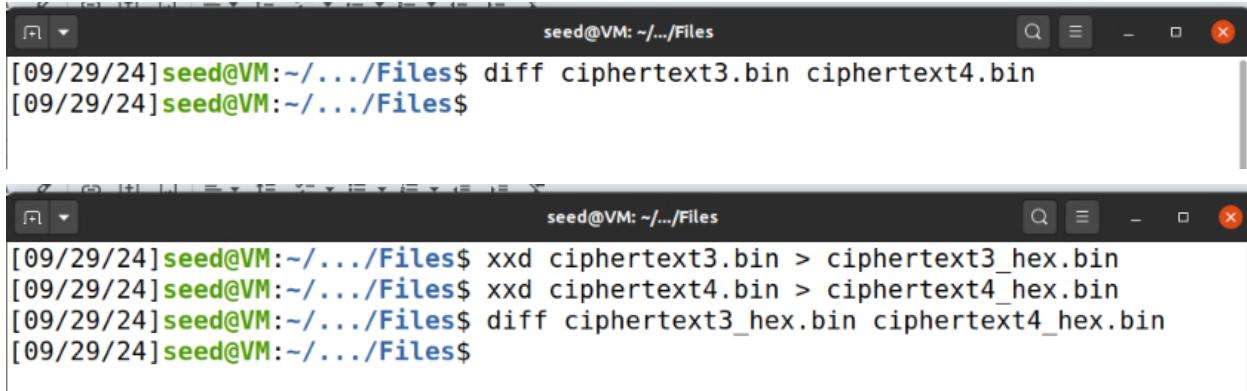
Now we perform the same procedure using the same IVs.

IV = 4a6b75321d3c4f896574234895135a73

Encrypting the file using the same IV:

```
seed@VM: ~/.../Files$ openssl enc -aes-128-cbc -e -in plaintext.txt -o
ut ciphertext3.bin -K 000102030405060708090A0B0C0D0E0F -iv 4a6b75321d3c4f8965742
34895135a73
[09/29/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plaintext.txt -ou
t ciphertext4.bin -K 000102030405060708090A0B0C0D0E0F -iv 4a6b75321d3c4f89657423
4895135a73
[09/29/24]seed@VM:~/.../Files$
```

We then check if there is any difference between the two encrypted files:



The image shows two terminal windows side-by-side. Both windows have a title bar 'seed@VM: ~/.../Files' and a red close button.

The top terminal window contains the command: [09/29/24] seed@VM:~/.../Files\$ diff ciphertext3.bin ciphertext4.bin

The bottom terminal window contains the following commands:
[09/29/24] seed@VM:~/.../Files\$ xxd ciphertext3.bin > ciphertext3_hex.bin
[09/29/24] seed@VM:~/.../Files\$ xxd ciphertext4.bin > ciphertext4_hex.bin
[09/29/24] seed@VM:~/.../Files\$ diff ciphertext3_hex.bin ciphertext4_hex.bin
[09/29/24] seed@VM:~/.../Files\$

Observation:

When using the same IV, we notice that it produces identical ciphertexts. This demonstrates why IVs need to be unique - reusing an IV can reveal patterns in the encrypted data. Therefore having different IVs is important to ensure that the encrypted files differ. IVs need to be unique to prevent identical ciphertexts for confidentiality and security purposes.

Task 6.2. Common Mistake: Use the Same IV

In this task, we have to find the value of “P2”, based on the values of C2, P1, and C1.

We changed the sample code provided accordingly so that it gave out the output for P2.

```
seed@VM: ~/.../Files
#!/usr/bin/python3

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

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

D1 = MSG

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

r1 = xor(D1, D2)
r2 = xor(D2, D3)
r3 = xor(D2, D2)

r4 = xor(r2, D1)
print("Decrypted P2: ", r4.decode())
#print(r1.hex())
#print(r2.hex())
#print(r3.hex())
```

Running the script we get:

```
seed@VM: ~/.../Files
[09/29/24]seed@VM:~/.../Files$ vim sample_code.py
[09/29/24]seed@VM:~/.../Files$ ./sample_code.py
Decrypted P2: Order: Launch a missile!
[09/29/24]seed@VM:~/.../Files$
```

As observed, the plain text message was `Order: Launch a missile!`.

Question - 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.

Answer:

OFB mode requires a new Initialization Vector IV for each message. In OFB, the IV is used to create a special sequence of random-looking data called a keystream. This keystream is then combined with the message to encrypt it. If you use the same IV more than once, you'll get the same keystream. This is dangerous because if an attacker gets two messages encrypted with the same keystream, they might be able to figure out parts of both messages. By using a different IV

each time, you ensure that each message has its unique keystream, making it much harder for attackers to break the encryption.

CFB mode works a bit differently. Here, even if an attacker knows the IV, they can only potentially figure out the first part of the message. This is because CFB uses each piece of encrypted data to help encrypt the next piece. So, after the first part, the IV isn't used anymore. Instead, the previous encrypted piece acts like a new IV for the next piece. This means that even if an attacker knows the IV, they can't easily decrypt the whole message. They might get a small part at the beginning, but the rest stays secret unless they know the encryption key. This makes CFB more resistant to attacks, especially if the IV is somehow compromised or reused.

Task 6.3. Common Mistake: Use a Predictable IV

In this task, we have to find out Bob's message. As the question suggests we know the value of his message can be "Yes" or "No".

Since we are using CBC mode, it is known that using predictable IV can make the algorithm vulnerable, making it possible to discover Bob's message.

We modified the script provided to grab the value of plaintext.

```
#!/usr/bin/python3

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

IV_used = "baf4246d46dcef31428b3ceelccfa86d"
Next_IV = "48b9237d46dcef31428b3ceelccfa86d"
MSG = "Yes"

pad_len = 16 - len(MSG) % 16
D1 = bytearray(MSG, encoding='utf-8')
D1.extend([pad_len] * pad_len)

# Convert hex string to bytearray
D2 = bytearray.fromhex(IV_used)
D3 = bytearray.fromhex(Next_IV)

r1 = xor(D1, D2)

plaintext = xor(r1, D3)

print("Plaintext in hex: ", plaintext.hex())
```

We then run this script to get the value of plaintext in hexadecimal

After receiving this value, we feed it to the oracle and get the ciphertext. We then compare if both of them are equal.

The answer 'Yes' is right. We can see that only the first 16 bytes (128 bits) of the ciphertext are the same as Bob's ciphertext. This happens because the CBC mode needs the input to be exactly 16 bytes long. The word 'Yes' is shorter than 16 bytes, so extra characters (padding) were added to make it 16 bytes. The oracle only accepts input in hex format, so this padded version of 'Yes' was used. This created two blocks of ciphertext: the first block has the encrypted 'Yes' with some padding, and the second block is just the encrypted padding. That's why only the first block matches Bob's ciphertext.

Task 7: Programming using the Crypto Library

```
gcc -o decrypt decrypt.c -lcrypto
```

```
#include <openssl/evp.h>
#include <stdio.h>
#include <string.h>

int main() {
    // Array to store the plaintext "This is a top secret."
    unsigned char plaintext[21];

    // Given ciphertext in hexadecimal format (adding 0x)
764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2
    unsigned char ciphertext[] = {
        0x76, 0x4a, 0xa2, 0x6b, 0x55, 0xa4, 0xda, 0x65,
        0x4d, 0xf6, 0xb1, 0x9e, 0x4b, 0xce, 0x00, 0xf4,
        0xed, 0x05, 0xe0, 0x93, 0x46, 0xfb, 0x0e, 0x76,
        0x25, 0x83, 0xcb, 0x7d, 0xa2, 0xac, 0x93, 0xa2
    };

    // Initialization vector in hexadecimal format
aabbccddeeff00998877665544332211
    unsigned char iv[] = {
        0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff, 0x00, 0x99,
        0x88, 0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11
    };

    // Array to store the encryption key
    unsigned char key[16];
    // Array to store each word from the word list
    char word[16];
    // Array to store the encrypted output
    unsigned char encryptedop[32];

    // Open the plaintext file for reading
    FILE *plaintext_file = fopen("plaintext.txt", "rb");
    if (!plaintext_file) {
        fprintf(stderr, "Error opening plaintext file\n");
        return 1;
    }

    // Read the plaintext from the file
    fread(plaintext, 1, 21, plaintext_file);
```

```
fclose(plaintext_file);

// Open the word list file for reading
FILE *words = fopen("words.txt", "r");
if (!words) {
    fprintf(stderr, "Error opening word list file\n");
    return 1;
}

// Read each word from the word list
while (fgets(word, sizeof(word), words)) {
    // Remove newline character from the word
    word[strcspn(word, "\n")] = 0;

    // Copy the word into the key array and pad with '#'
    memcpy(key, word, strlen(word));
    memset(key + strlen(word), '#', 16 - strlen(word));

    int enc_l, final_l;
    // Create encryption context
    EVP_CIPHER_CTX *ctx;
    ctx = EVP_CIPHER_CTX_new();

    EVP_EncryptInit_ex(ctx, EVP_aes_128_cbc(), NULL, key, iv); // Initialize encryption

    // Encrypt the plaintext
    EVP_EncryptUpdate(ctx, encryptedop, &enc_l, plaintext, strlen((char *)plaintext));
    EVP_EncryptFinal_ex(ctx, encryptedop + enc_l, &final_l); // Finalize the encryption
    enc_l += final_l;

    EVP_CIPHER_CTX_free(ctx); // Free the encryption context

    // Compare the encrypted output with the given ciphertext
    if (memcmp(encryptedop, ciphertext, 32) == 0) {
        printf("Found key: %s\n", word); // Print the found key
        fclose(words); // Close the word list file
        return 0; // Exit the program
    }
}
```

```
    printf("Key not found\n");
    fclose(words); // Close the word list file
    return 1;
}
```

```
[10/01/24]seed@VM:~/.../Files$ gcc -o decrypt decrypt.c -lcrypto
[10/01/24]seed@VM:~/.../Files$ ./decrypt
Found key: Syracuse
```

We stored the plaintext in a file using `echo -n "This is a top secret." > plaintext.txt`. Then, we retrieved it, ensured it was 21 bytes long, and stored it in a char array. The given ciphertext and IV are stored in char arrays, where each element in C represents 1 byte. Both are in hexadecimal format.

Next, we allocated memory for the key, encrypted output, and word. The word has a size of 16 bytes since that's the size of one AES-128 block.

We open words.txt and read each word, padding it with # if it's less than 16 bytes to fit the AES-128 block size. The `memcpy()` function copies the word into the key, and `memset()` fills the remaining space with padding.

The key is set using the padded word, and we create an encryption context with `EVP_CIPHER_CTX_new()`, which initializes the context needed for encryption. The `EVP_EncryptInit_ex()` function sets up the encryption using AES-128 in CBC mode, with the provided key and IV. This prepares the encryption operation.

Next, `EVP_EncryptUpdate()` encrypts the plaintext in blocks, writing the result to the encrypted array and storing the length of the encrypted data. `EVP_EncryptFinal_ex()` completes the encryption process by handling any remaining data and final padding.

The `memcmp()` function compares the encrypted output (`encryptedop`) with the given ciphertext. If they match, the correct key is found, and the program prints it.

If no match is found after going through all the words, the program prints "Key not found." The encryption context is cleaned up with EVP_CIPHER_CTX_free() to prevent memory leaks, and fclose() is used to properly close the file.

In this case, the key was found, meaning the correct word “Syracuse” from words.txt successfully decrypted the ciphertext, verifying it as the encryption key.