

SEED LABS – ENCRYPTION AND DECRYPTION

Frequency Analysis

The image shows a Linux desktop environment with a dock containing icons for a file manager, terminal, and other applications. Four text editor windows are open:

- Text Editor (Sep 16 12:19)**: Displays the original article: "A computer is a machine that can be programmed to carry out sequences of arithmetic or logical operations (computation) automatically. Modern digital electronic computers can perform generic sets of operations known as programs. These programs enable computers to perform a wide range of tasks. A computer system is a nominally complete computer that includes the hardware, operating system (main software), and peripheral equipment needed and used for full operation. This term may also refer to a group of computers that are linked and function together, such as a computer network or computer cluster."
- Text Editor (Sep 16 12:20)**: Displays the lowercase version of the article.
- Text Editor (Sep 16 12:22)**: Displays the ciphertext, which is a highly encrypted version of the article.
- Text Editor (Sep 16 12:25)**: Displays the decrypted plaintext, which is identical to the original article.

The terminal window at the bottom shows the command used to decrypt the ciphertext:

```
[09/13/23] seed@VM:~$ tr 'abcdefghijklmnopqrstuvwxyz' 'CFMYPVBRLQXWIEJDSGKHNAZOTU' < ciphertext.txt > decrypted_plain.txt
```

ENCRYPTION KEY :

CFMYPVBRXLQXWIEJDSGKHNAZOTU

The screenshot shows a terminal window titled "Text Editor" with the file "decrypted_plain.txt" open. The content of the file is a poem about the Oscars and the #MeToo movement. The poem discusses the awards race, Harvey Weinstein's demise, and the emergence of the #MeToo movement. It also mentions the Golden Globes and the Black Lives Matter movement. The poem concludes with a call to action against sexism and harassment.

```
1 THE OSCARS TURN ON SUNDAY WHICH SEEMS ABOUT RIGHT AFTER THIS LONG STRANGE
2 AWARDS TRIP THE BAGGER FEELS LIKE A NONAGENARIAN TOO
3
4 THE AWARDS RACE WAS BOOKENDED BY THE DEMISE OF HARVEY WEINSTEIN AT ITS OUTSET
5 AND THE APPARENT IMPLOSION OF HIS FILM COMPANY AT THE END AND IT WAS SHAPED BY
6 THE EMERGENCE OF METOO TIMES UP BLACKGOWN POLITICS ARMCANDY ACTIVISM AND
7 A NATIONAL CONVERSATION AS BRIEF AND MAD AS A FEVER DREAM ABOUT WHETHER THERE
8 OUGHT TO BE PRESIDENT WINFREY THE SEASON DIDNT JUST SEEM EXTRA LONG IT WAS
9 EXTRA LONG BECAUSE THE OSCARS WERE MOVED TO THE FIRST WEEKEND IN MARCH TO
10 AVOID CONFLICTING WITH THE CLOSING CEREMONY OF THE WINTER OLYMPICS THANKS
11 PYEONGCHANG
12
13 ONE BIG QUESTION SURROUNDING THIS YEARS ACADEMY AWARDS IS HOW OR IF THE
14 CEREMONY WILL ADDRESS METOO ESPECIALLY AFTER THE GOLDEN GLOBES WHICH BECAME
15 A JUBILANT COMINGOUT PARTY FOR TIMES UP THE MOVEMENT SPEARHEADED BY
16 POWERFUL HOLLYWOOD WOMEN WHO HELPED RAISE MILLIONS OF DOLLARS TO FIGHT SEXUAL
17 HARASSMENT AROUND THE COUNTRY
18
19 SIGNALING THEIR SUPPORT GOLDEN GLOBES ATTENDEES SWATHED THEMSELVES IN BLACK
20 SPORTED LAPEL PINS AND SOUNDED OFF ABOUT SEXIST POWER IMBALANCES FROM THE RED
21 CARPET AND THE STAGE ON THE AIR E WAS CALLED OUT ABOUT PAY INEQUALITY AFTER
22 ITS FORMER ANCHOR CATT SADLER QUIT ONCE SHE LEARNED THAT SHE WAS MAKING FAR
23 LESS THAN A MALE COHOST AND DURING THE CEREMONY NATALIE PORTMAN TOOK A BLUNT
24 AND SATISFYING DIG AT THE ALLMALE ROSTER OF NOMINATED DIRECTORS HOW COULD
25 THAT BE TOPPED
26
27 AS IT TURNS OUT AT LEAST IN TERMS OF THE OSCARS IT PROBABLY WONT BE
28
29 WOMEN INVOLVED IN TIMES UP SAID THAT ALTHOUGH THE GLOBES SIGNIFIED THE
30 INITIATIVES LAUNCH THEY NEVER INTENDED IT TO BE JUST AN AWARDS SEASON
31 CAMPAIGN OR ONE THAT BECAME ASSOCIATED ONLY WITH REDCARPET ACTIONS INSTEAD
32 A SPOKESWOMAN SAID THE GROUP IS WORKING BEHIND CLOSED DOORS AND HAS SINCE
33 AMASSED MILLION FOR ITS LEGAL DEFENSE FUND WHICH AFTER THE GLOBES WAS
34 FLOODED WITH THOUSANDS OF DONATIONS OF OR LESS FROM PEOPLE IN SOME
35 COUNTRIES
36
37
38 NO CALL TO WEAR BLACK CLOTHING WENT OUT IN ADVANCE OF THE OSCARS TONIGHT THE
```

The screenshot shows a Jupyter Notebook interface with a single cell containing Python code. The code defines a function to generate n-grams from a given text and then reads a file named "ciphertext.txt" to find the most common n-grams. The output shows the top 20 1-grams and their counts.

```
[1]: #!/usr/bin/env python3
from collections import Counter
import re

TOP_K = 20
N_GRAM = 3

# Generate all the n-grams for value n
def ngrams(text):
    for i in range(1, len(text) - n + 1):
        # Ignore n-grams containing white space
        if not re.search(r'\s', text[i:i+n]):
            yield text[i:i+n]

# Read the data from the ciphertext
with open('ciphertext.txt') as f:
    text = f.read()

# Count, sort, and print out the n-grams
for N in range(N_GRAM):
    print("-----")
    print("1-gram (top %d):" % (TOP_K))
    counts = Counter(ngrams(N+1, text)) # Count
    sorted_counts = counts.most_common(TOP_K) # Sort
    for ngram, count in sorted_counts:
        print("{:<10}: {}".format(ngram, count)) # Print
```

```
1-gram (top 20):
n: 486
y: 373
v: 348
x: 291
u: 280
q: 276
w: 264
h: 235
t: 183
l: 166
p: 156
e: 116
c: 104
z: 95
l: 90
g: 83
b: 83
r: 83
```

TASK DONE AND OBSERVATION:

- By running the given code snippet I was able to get the words with highest frequencies.
- Using the frequency analysis method, I have referred the words with higher frequency (3 gram, 2 gram, 1 gram frequency) in the wikipedia and replaced the encrypted data it with the corresponding plain text.
- Did this multiple times using hit and trial method manually for all the alphabets.
- Finally found the correct encryption key and decrypted and displayed the corresponding plaintext.
- CFMYPVBRLQXWIEJDSGKHNAZOTU is the Encryption key
- Decrypted data is printed below for reference.

DECRYPTED PLAIN TEXT :

THE OSCARS TURN ON SUNDAY WHICH SEEMS ABOUT RIGHT AFTER THIS LONG
STRANGE
AWARDS TRIP THE BAGGER FEELS LIKE A NONAGENARIAN TOO

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

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

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

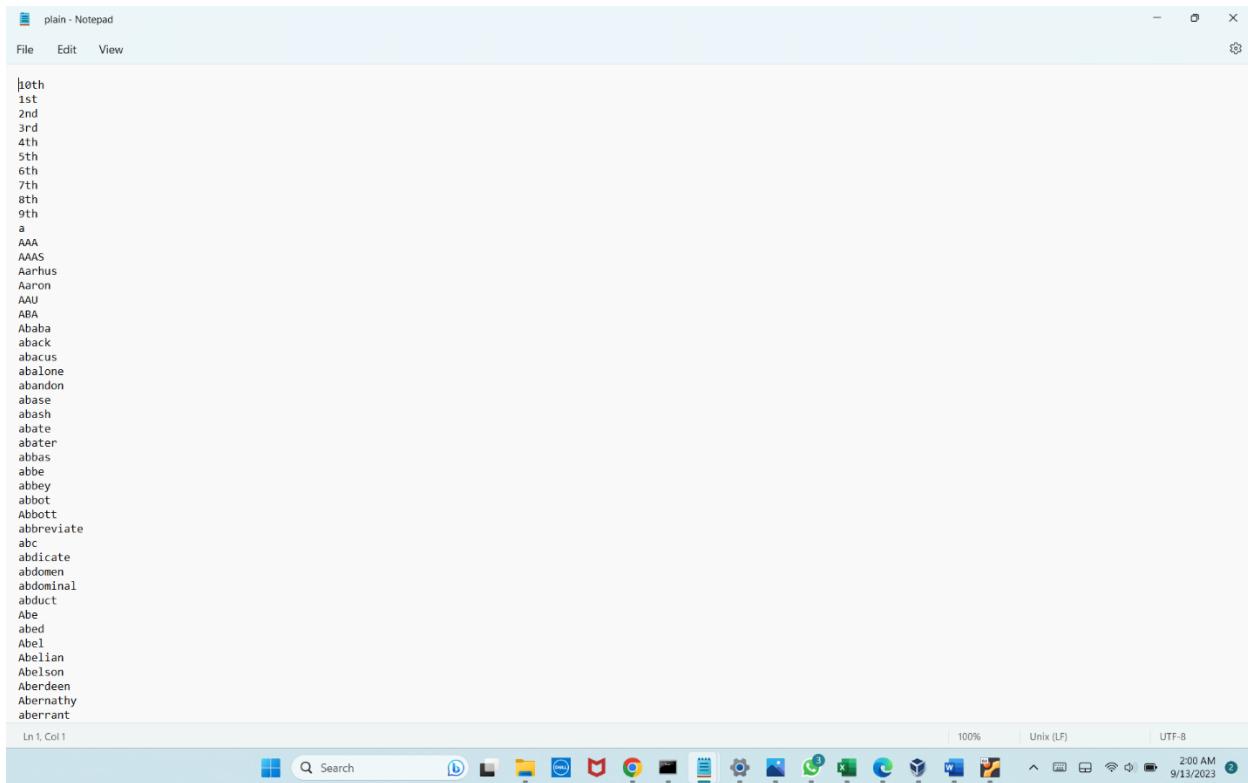
LARGEST BRANCH THAT FILM WHILE DIVISIVE ALSO WON THE BEST DRAMA GOLDEN GLOBE

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

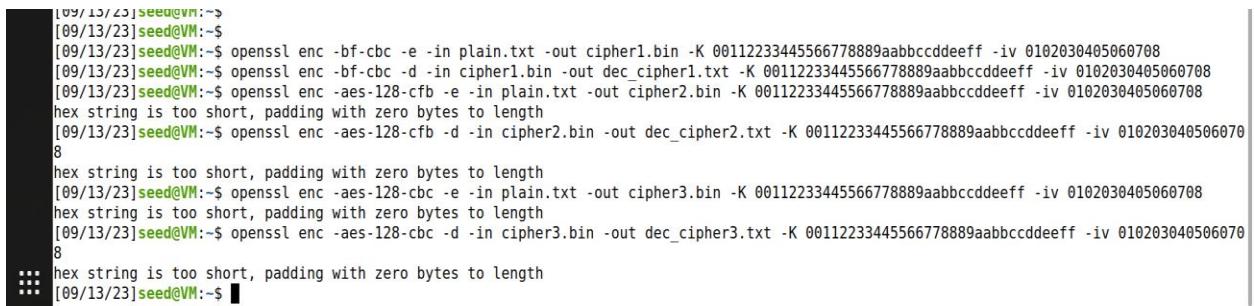
Encryption using Different Ciphers and Modes

PLAIN TEXT :



A screenshot of a Windows Notepad window titled "plain - Notepad". The window contains a large amount of text, mostly names and terms, listed one per line. Some of the entries include: 10th, 1st, 2nd, 3rd, 4th, 5th, 6th, 7th, 8th, 9th, a, AAA, AAAS, Aarhus, Aaron, AAU, ABA, Ababa, aback, abacus, abalone, abandon, abase, abash, abate, abater, abbas, abbe, abbey, abbott, Abbott, abbreviate, abc, abdicate, abdomen, abdominal, abduct, Abe, abed, Abel, Abelian, Abelson, Aberdeen, Abernathy, aberrant. The Notepad window has a standard Windows title bar with "File", "Edit", and "View" options. At the bottom, it shows "Ln 1, Col 1", "100%", "Unix (LF)", "UTF-8", and a timestamp "2:00 AM 9/13/2023". Below the Notepad window is the Windows taskbar with various pinned icons.

ENCRYPTION AND DECRYPTION CODE



A screenshot of a terminal window on a Linux system. The prompt is "seed@VM:~>". The user runs several OpenSSL commands to encrypt and decrypt files using different cipher modes. The commands and their outputs are:

- [09/13/23] seed@VM:~\$ openssl enc -bf-cbc -e -in plain.txt -out cipher1.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
- [09/13/23] seed@VM:~\$ openssl enc -bf-cbc -d -in cipher1.bin -out dec_cipher1.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
- [09/13/23] seed@VM:~\$ openssl enc -aes-128-cfb -e -in plain.txt -out cipher2.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
- [09/13/23] seed@VM:~\$ openssl enc -aes-128-cfb -d -in cipher2.bin -out dec_cipher2.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
- [09/13/23] seed@VM:~\$ hex string is too short, padding with zero bytes to length
- [09/13/23] seed@VM:~\$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher3.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
- [09/13/23] seed@VM:~\$ hex string is too short, padding with zero bytes to length
- [09/13/23] seed@VM:~\$ openssl enc -aes-128-cbc -d -in cipher3.bin -out dec_cipher3.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
- [09/13/23] seed@VM:~\$ hex string is too short, padding with zero bytes to length

BLOW FISH IN CBC MODE ENCRYPTION – CIPHER TEXT

BLOW FISH IN CBC MODE DECRYPTION

DECRYPTING IN CBC MODE DECRYPTION

File Edit View

```
10th  
1st  
2nd  
3rd  
4th  
5th  
6th  
7th  
8th  
9th  
a  
AAA  
AAAS  
Aarhus  
Aaron  
AAU  
ABA  
Ababa  
aback  
abacus  
abalone  
abandon  
abase  
abash  
abate  
abater  
abbas  
abbe  
abbey  
abbot  
Abbott  
abbreviate  
abc  
abdicate  
abdomen  
abdominal  
abduct  
Abe  
abed  
Abel  
Abelian  
Abelson  
Aberdeen  
Abernathy  
aberrant
```

Ln 1, Col 1

100% Unix (LF) UTF-8

4:19 PM 9/13/2023

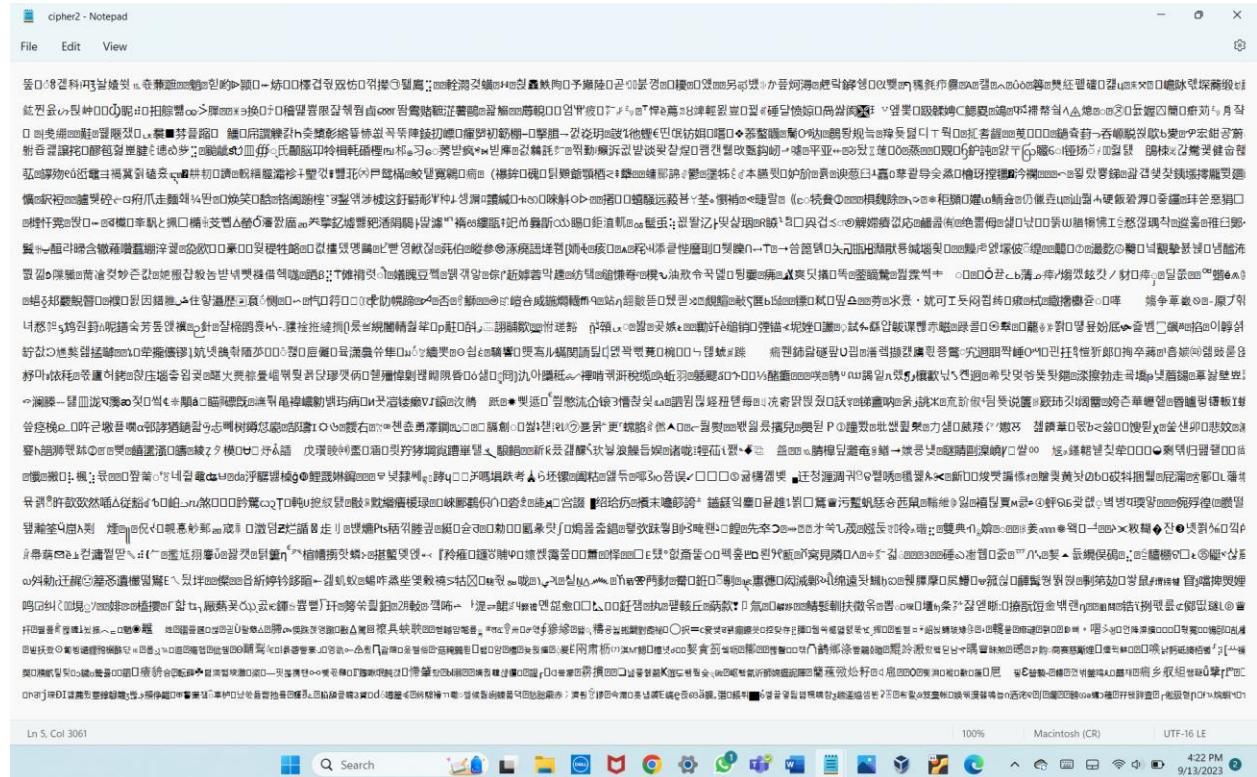
AES 128 CBC MODE ENCRYPTION – CIPHER TEXT

```
cipher3 - Notepad
File Edit View
很長的二進位數字串，代表AES 128 CBC模式加密後的密文。
```

AES 128 CBC MODE DECRYPTION

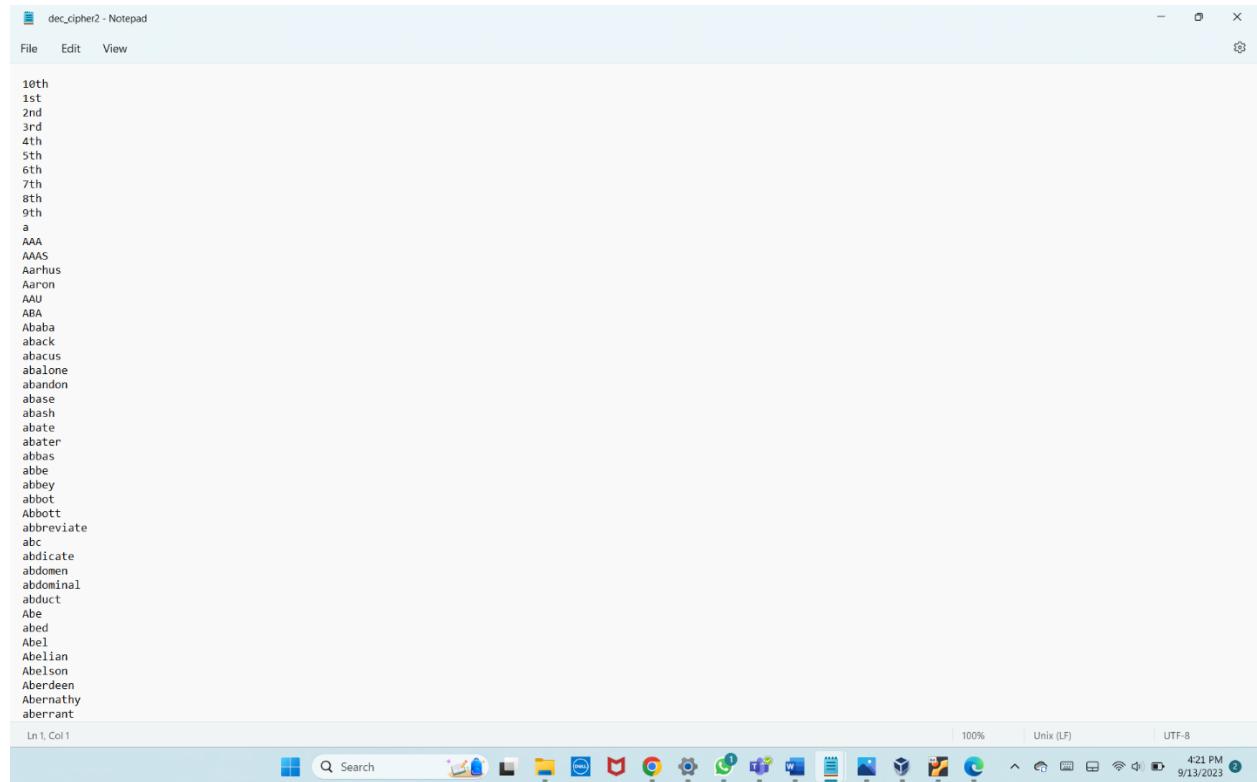
```
dec_cipher3 - Notepad
File Edit View
解密後的明文，顯示為亂碼。
```

AES – 128 CFB MODE ENCRYPTION – CIPHER TEXT



The screenshot shows a Notepad window titled "cipher2 - Notepad". The content of the text area is a long string of binary data represented as ASCII characters, which is the encrypted AES-128 CFB mode ciphertext. The text area starts with "00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00" and continues with a repeating pattern of binary digits.

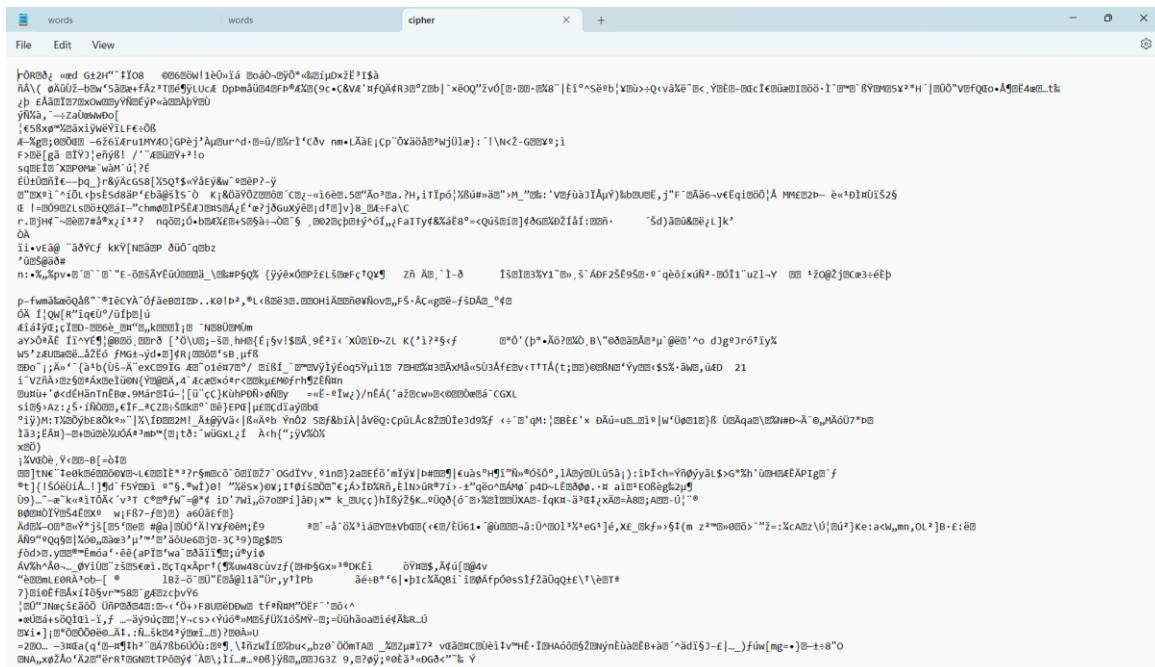
AES – 128 CFB MODE DECRYPTION



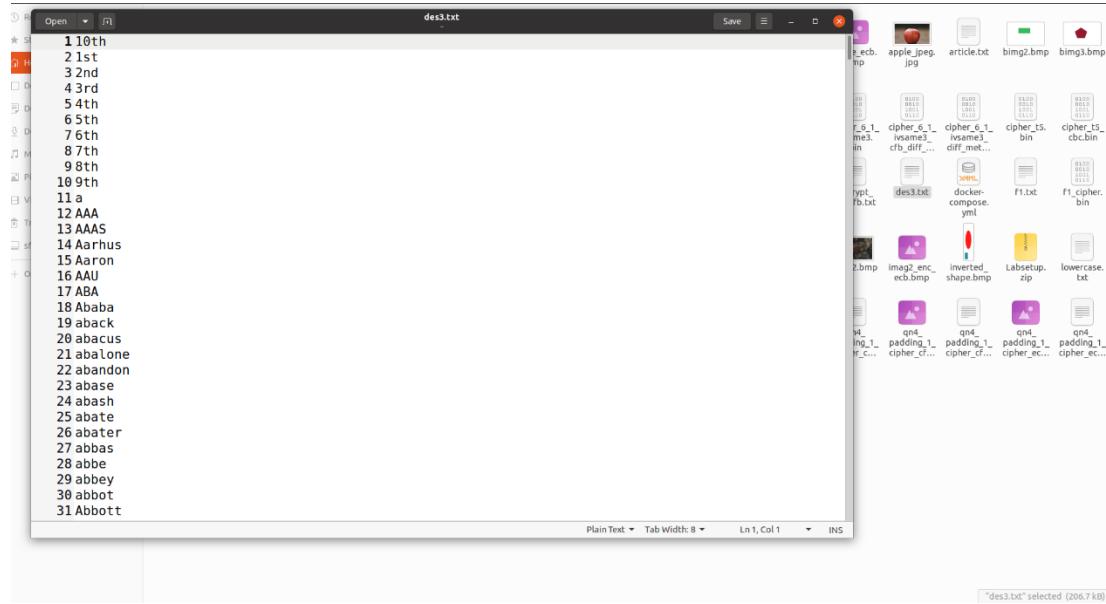
The screenshot shows a Notepad window titled "dec_cipher2 - Notepad". The content of the text area is a long string of binary data represented as ASCII characters, which is the decrypted AES-128 CFB mode plaintext. The text area starts with "00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00" and continues with a repeating pattern of binary digits.

DES CODE SNIPPET:

DES ENCRYPTED TEXT :



DES DECRYPTED TEXT :



TASK DONE AND OBSERVATION :

Tried encryption and decryption for a plain text using

- AES-128 in CBC mode
- Blow Fish in CBC mode
- AES-128 IN CFB mode
- DES3

I was able to encrypt and decrypt by using the openssl command for the above ciphers and modes and understood the modes and flagoptions for encryption and decryption

Encryption Mode – ECB vs. CBC

1. Image



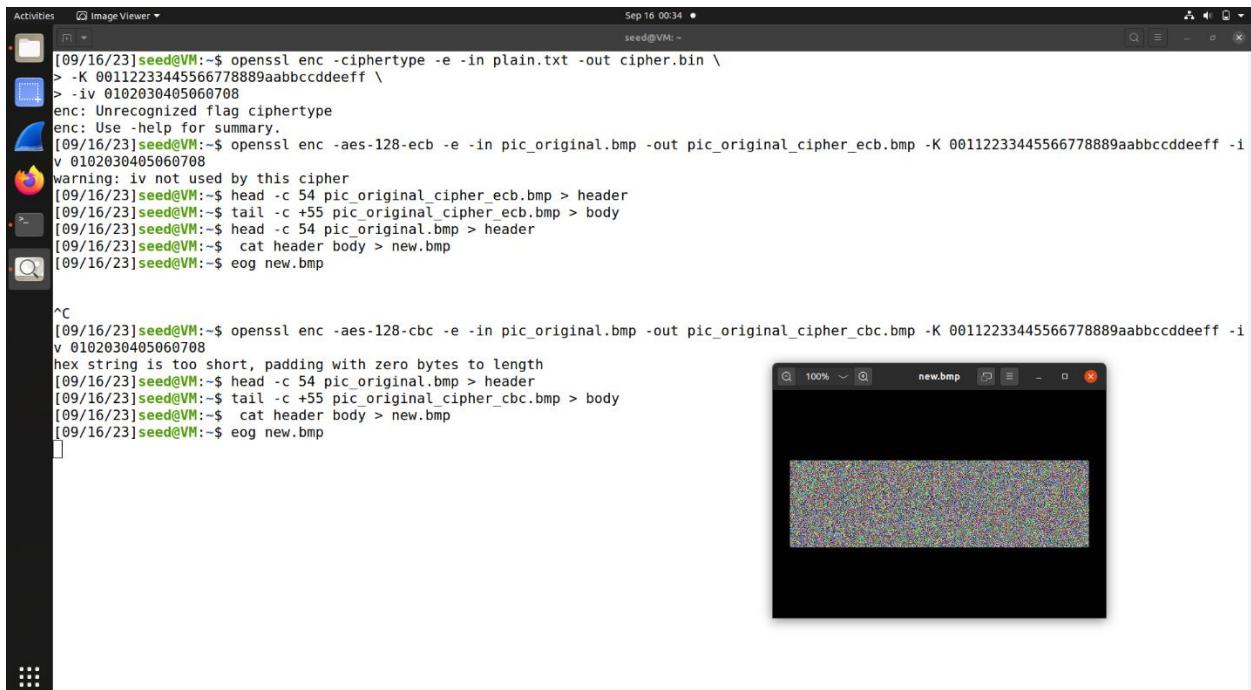
ENCRYPTED IMAGE IN ECB :

```
Activities Image Viewer Sep 16 00:31 seed@VM:~$ openssl enc -ciphertype -e -in plain.txt -out cipher.bin \
> -K 00112233445566778889aabcccddeeff \
> -iv 0102030405060708
enc: Unrecognized flag ciphertype
enc: Use -help for summary.
[09/16/23]seed@VM:~$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out pic_original_cipher_ecb.bmp -K 00112233445566778889aabcccddeeff -i
v 0102030405060708
warning: iv not used by this cipher
[09/16/23]seed@VM:~$ head -c 54 pic_original_cipher_ecb.bmp > header
[09/16/23]seed@VM:~$ tail -c +55 pic_original_cipher_ecb.bmp > body
[09/16/23]seed@VM:~$ head -c 54 pic_original.bmp > header
[09/16/23]seed@VM:~$ cat header body > new.bmp
[09/16/23]seed@VM:~$ eog new.bmp
```

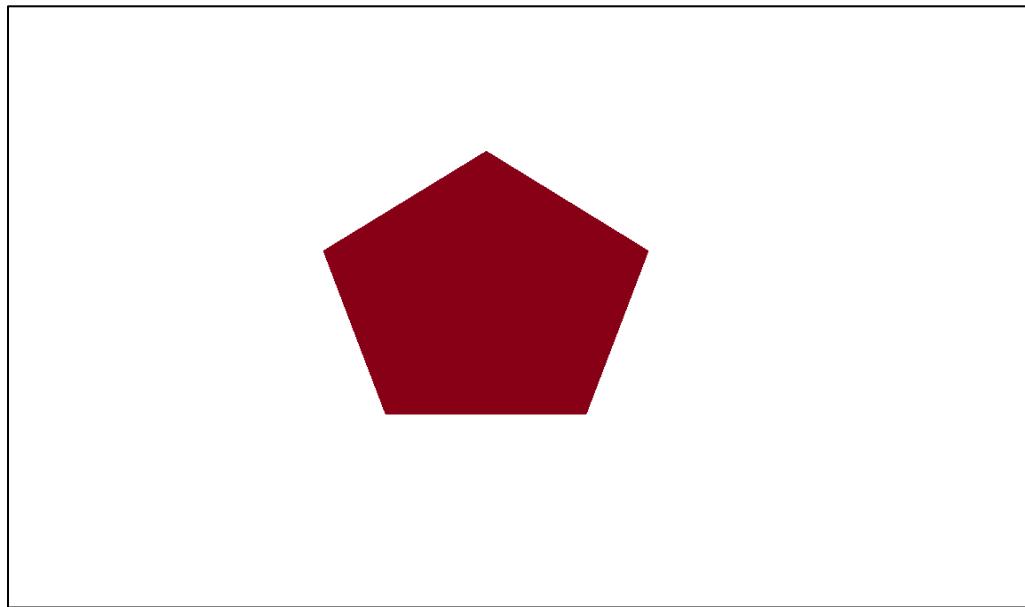
A screenshot of a terminal window showing the command line steps to encrypt an image using ECB mode with OpenSSL. The terminal shows the user running commands to generate a cipher file, then splitting it into header and body, and finally concatenating them back together. To the right of the terminal is a screenshot of an image viewer showing the original image (a red oval and a blue rectangle) and the encrypted image (a distorted version where the red oval has a halftone pattern and the blue rectangle is partially visible).

ENCRYPTED IMAGE IN CBC :

```
Activities Image Viewer Sep 16 00:34 • seed@VM:~  
[09/16/23]seed@VM:~$ openssl enc -ciphertype -e -in plain.txt -out cipher.bin \  
> -K 00112233445566778889aabccddeeff \  
> -iv 0102030405060708  
enc: Unrecognized flag ciphertype  
enc: Use -help for summary.  
[09/16/23]seed@VM:~$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out pic_original_cipher_ecb.bmp -K 00112233445566778889aabccddeeff -i  
v 0102030405060708  
warning: iv not used by this cipher  
[09/16/23]seed@VM:~$ head -c 54 pic_original_cipher_ecb.bmp > header  
[09/16/23]seed@VM:~$ tail -c +55 pic_original_cipher_ecb.bmp > body  
[09/16/23]seed@VM:~$ head -c 54 pic_original.bmp > header  
[09/16/23]seed@VM:~$ cat header body > new.bmp  
[09/16/23]seed@VM:~$ eog new.bmp  
  
^C  
[09/16/23]seed@VM:~$ openssl enc -aes-128-cbc -e -in pic_original.bmp -out pic_original_cipher_cbc.bmp -K 00112233445566778889aabccddeeff -i  
v 0102030405060708  
hex string is too short, padding with zero bytes to length  
[09/16/23]seed@VM:~$ head -c 54 pic_original.bmp > header  
[09/16/23]seed@VM:~$ tail -c +55 pic_original_cipher_cbc.bmp > body  
[09/16/23]seed@VM:~$ cat header body > new.bmp  
[09/16/23]seed@VM:~$ eog new.bmp
```

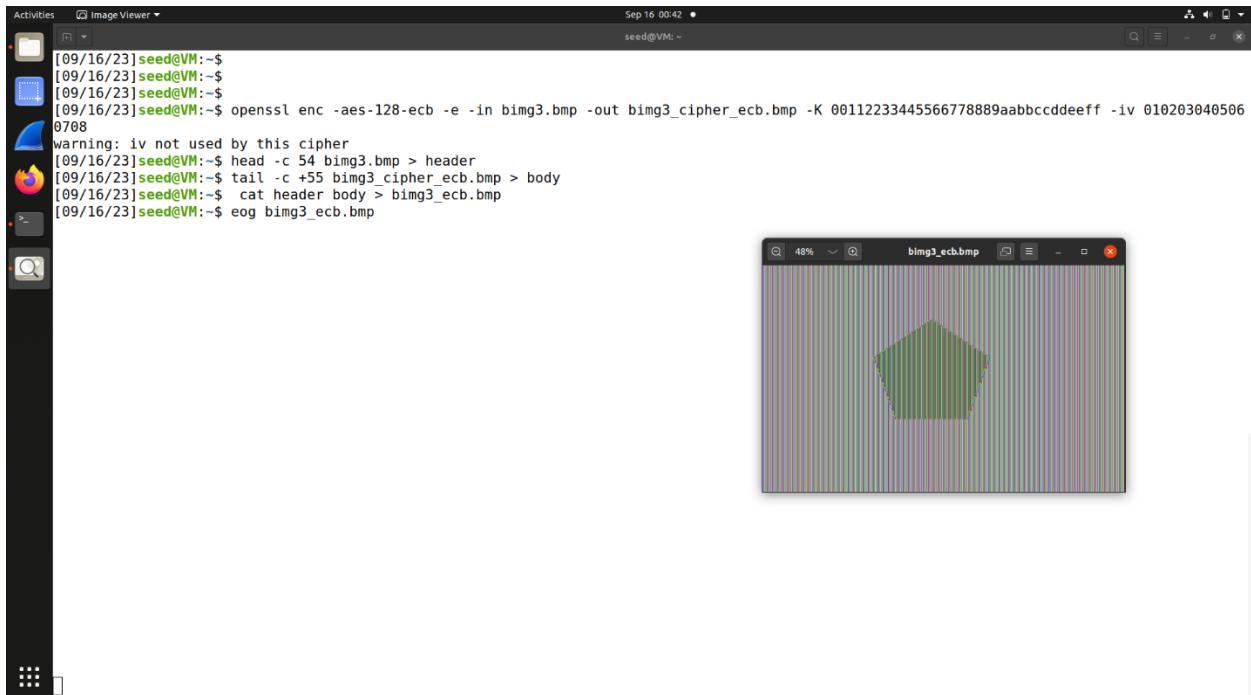


2. IMAGE



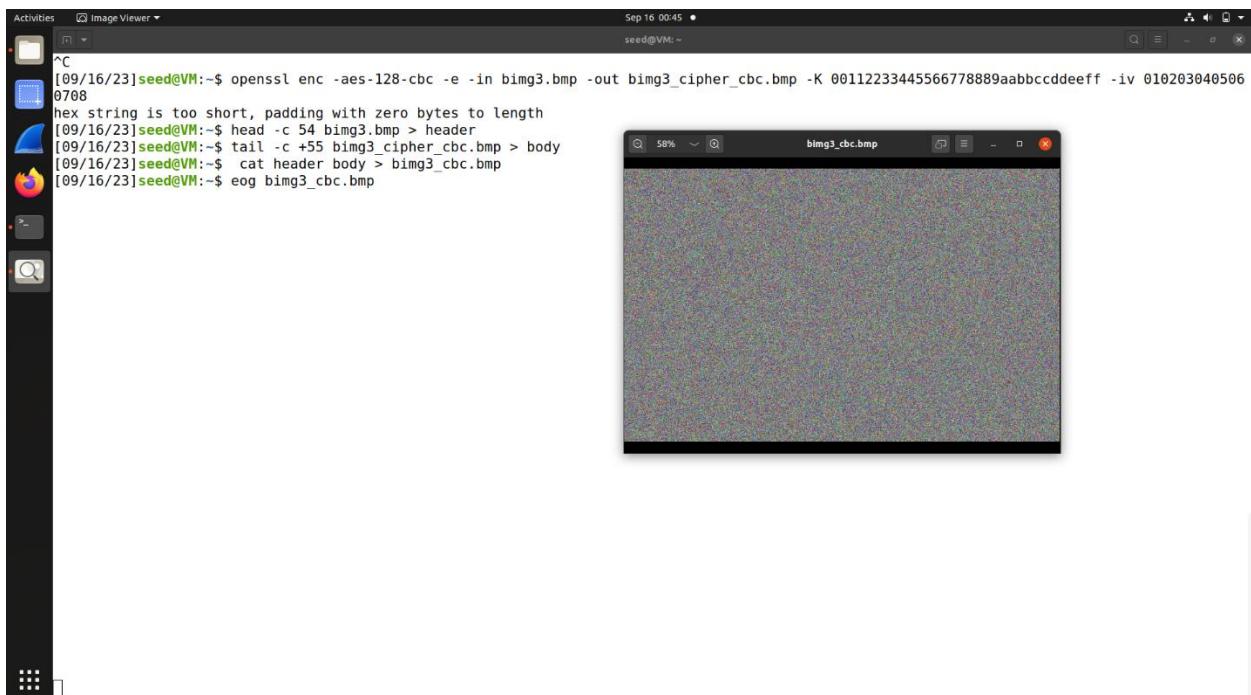
ENCRYPTED IMAGE IN ECB :

```
Activities  Image Viewer  Sep 16 00:42  seed@VM:~  
[09/16/23]seed@VM:~$ [09/16/23]seed@VM:~$ [09/16/23]seed@VM:~$ [09/16/23]seed@VM:~$ openssl enc -aes-128-ecb -e -in bimg3.bmp -out bimg3_cipher_ecb.bmp -K 00112233445566778889aabccddeff -iv 010203040506  
0708  
warning: iv not used by this cipher  
[09/16/23]seed@VM:~$ head -c 54 bimg3.bmp > header  
[09/16/23]seed@VM:~$ tail -c +55 bimg3_cipher_ecb.bmp > body  
[09/16/23]seed@VM:~$ cat header body > bimg3_ecb.bmp  
[09/16/23]seed@VM:~$ eog bimg3_ecb.bmp
```



ENCRYPTED IMAGE IN CBC :

```
Activities  Image Viewer  Sep 16 00:45  seed@VM:~  
^C  
[09/16/23]seed@VM:~$ openssl enc -aes-128-cbc -e -in bimg3.bmp -out bimg3_cipher_cbc.bmp -K 00112233445566778889aabccddeff -iv 010203040506  
0708  
hex string is too short, padding with zero bytes to length  
[09/16/23]seed@VM:~$ head -c 54 bimg3.bmp > header  
[09/16/23]seed@VM:~$ tail -c +55 bimg3_cipher_cbc.bmp > body  
[09/16/23]seed@VM:~$ cat header body > bimg3_cbc.bmp  
[09/16/23]seed@VM:~$ eog bimg3_cbc.bmp
```



OBSERVATION :

- It is observed the at the CBC mode of encryption is the best as compared to ECB mode of encryption.
- Because, In ECB mode we are able to predict the picture even after the encryption
- But we cant predict the picture in CBC mode
- After doing the encryption of given image, I tried with the new image and I could see the same behavior of CBC mode performing better than ECB mode

Padding

1.

```
^C
[09/16/23]seed@VM:~$ echo -n "abcdefghijkl" > qn4_padding_1.txt
[09/16/23]seed@VM:~$ openssl enc -aes-128-cbc -e -in qn4_padding_1.txt -out qn4_padding_1_cipher.bmp -K 00112233445566778889aabccddeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/16/23]seed@VM:~$ openssl enc -aes-128-ecb -e -in qn4_padding_1.txt -out qn4_padding_1_cipher_ecb.bmp -K 00112233445566778889aabccddeff -iv 0102030405060708
warning: iv not used by this cipher
[09/16/23]seed@VM:~$ openssl enc -aes-128-cfb -e -in qn4_padding_1.txt -out qn4_padding_1_cipher_cfb.bmp -K 00112233445566778889aabccddeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/16/23]seed@VM:~$ openssl enc -aes-128-ofb -e -in qn4_padding_1.txt -out qn4_padding_1_cipher_ofb.bmp -K 00112233445566778889aabccddeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ 
```

```
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ openssl enc -aes-128-cbc -d -nopad -in qn4_padding_1_cipher.bmp -out qn4_padding_1_cipher_cbc_d.txt -K 00112233445566778889aabccddeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/16/23]seed@VM:~$ openssl enc -aes-128-cfb -d -nopad -in qn4_padding_1_cipher_cfb.bmp -out qn4_padding_1_cipher_cfb_d.txt -K 00112233445566778889aabccddeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/16/23]seed@VM:~$ openssl enc -aes-128-ecb -d -nopad -in qn4_padding_1_cipher_ecb.bmp -out qn4_padding_1_cipher_ecb_d.txt -K 00112233445566778889aabccddeff -iv 0102030405060708
warning: iv not used by this cipher
[09/16/23]seed@VM:~$ openssl enc -aes-128-ofb -d -nopad -in qn4_padding_1_cipher_ofb.bmp -out qn4_padding_1_cipher_ofb_d.txt -K 00112233445566778889aabccddeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ 
```

```
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ hexdump -C p1.txt
hexdump: p1.txt: No such file or directory
[09/16/23]seed@VM:~$ hexdump -C qn4_padding_1_cipher_cbc_d.txt
00000000 61 62 63 64 65 66 67 68 69 6a 06 06 06 06 06 |abcdefghijkl....|
00000010
[09/16/23]seed@VM:~$ xxd qn4_padding_1_cipher_cbc_d.txt
00000000: 6162 6364 6566 6768 696a 0606 0606 0606 abcde
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ hexdump -C qn4_padding_1_cipher_cfb_d.txt
00000000 61 62 63 64 65 66 67 68 69 6a |abcdefghijkl|
0000000a
[09/16/23]seed@VM:~$ xxd qn4_padding_1_cipher_cfb_d.txt
00000000: 6162 6364 6566 6768 696a abcde
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ hexdump -C qn4_padding_1_cipher_ecb_d.txt
00000000 61 62 63 64 65 66 67 68 69 6a 06 06 06 06 06 |abcdefghijkl....|
00000010
[09/16/23]seed@VM:~$ xxd qn4_padding_1_cipher_ecb_d.txt
00000000: 6162 6364 6566 6768 696a 0606 0606 0606 abcde
[09/16/23]seed@VM:~$ hexdump -C qn4_padding_1_cipher_ofb_d.txt
00000000 61 62 63 64 65 66 67 68 69 6a |abcdefghijkl|
0000000a
[09/16/23]seed@VM:~$ xxd qn4_padding_1_cipher_ofb_d.txt
00000000: 6162 6364 6566 6768 696a abcde
[09/16/23]seed@VM:~$ 
[09/16/23]seed@VM:~$ 
```

OBSERVATION:

- ECB and CBC mode requires padding, because it requires fixed block size to encrypt the text. (It may be like 128 or 265).
- Here the padding bits 06 is appended the end.
- CFB and OFB does not require padding, because in these modes cipher text is always the same length as of the plain text as they do the operation byte by byte.

2.

FILE 1 (5 BYTES):

```
Activities Terminal Sep 13 20:49
[09/13/23]seed@VM:~$ echo -n "1234567890123456" > f3.txt
[09/13/23]seed@VM:-
[09/13/23]seed@VM:-
[09/13/23]seed@VM:~$ echo -n "12345" > f1.txt
[09/13/23]seed@VM:~$ openssl enc -aes-128-cbc -e -in f1.txt -out f1_cipher.bin
enter aes-128-cbc encryption password:
Verifying : enter aes-128-cbc encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[09/13/23]seed@VM:~$ ls
apple.bmp body ciphertext.txt Downloads f3.txt header_imag1 Music new.bmp shapes.bmp
apple.jpeg.jpg body1 decrypted_plain.txt f1_cipher.bin header1 imag1.bmp new1.bmp new_imag.bmp Templates
bimg2.bmp body2 Desktop f1.txt header1 imag2.bmp new2.bmp Pictures Videos
bimg3.bmp body_imag2 Documents f2.txt header2 inverted_shape.bmp new_bimg.bmp Public
[09/13/23]seed@VM:-
[09/13/23]seed@VM:-
[09/13/23]seed@VM:~$ openssl enc -ciphertextype -e -in plain.txt -out cipher.bin \
> -K 00112233445566778889aabccddeeff \
> -iv 0102030405060708
enc: Unrecognized flag ciphertextype
enc: Use -help for summary.
[09/13/23]seed@VM:~$ openssl enc -ciphertextype -e -in f1.txt -out f1_cipher.bin -K 00112233445566778889aabccddeeff -iv 0102030405060708
enc: Unrecognized flag ciphertextype
enc: Use -help for summary.
[09/13/23]seed@VM:~$ openssl enc -aes-128-cbc -e -in f1.txt -out f1_cipher.bin -K 00112233445566778889aabccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/13/23]seed@VM:~$ openssl enc -aes-128-cbc -d -nopad -in f1_cipher.bin -out f1_dec.txt -K 00112233445566778889aabccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/13/23]seed@VM:~$ hexdump -C f1_dec.txt
00000000 31 32 33 34 35 0b 0b 0b 0b 0b 0b 0b 0b 0b |12345.....|
00000010
[09/13/23]seed@VM:~$ xxd f1_dec.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 12345..... .
[09/13/23]seed@VM:~$ ls -lrt
total 20976
drwxr-xr-x 2 seed seed 4096 Nov 24 2020 Videos
drwxr-xr-x 2 seed seed 4096 Nov 24 2020 Templates
```

FILE2 (10 BYTES) :

```
Activities Terminal Sep 13 20:55 seed@VM:~  
Try: sudo apt install <deb name>  
[09/13/23]seed@VM:~$ openssl enc -aes-128-cbc -d -nopad -in f2_cipher.bin -out f2_dec.txt -K 00112233445566778889aabccddeff -iv 0102030405060708  
hex string is too short, padding with zero bytes to length  
[09/13/23]seed@VM:~$ hexdump -C f1_dec.txt  
00000000 31 32 33 34 35 0b |12345.....|  
00000010  
[09/13/23]seed@VM:~$ hexdump -C f2_dec.txt  
00000000 31 32 33 34 35 36 37 38 39 30 06 06 06 06 06 06 |1234567890.....|  
00000010  
[09/13/23]seed@VM:~$ xxd f1_dec.txt  
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b 12345.....  
[09/13/23]seed@VM:~$ xxd f2_dec.txt  
00000000: 3132 3334 3536 3738 3930 0606 0606 0606 1234567890.....  
[09/13/23]seed@VM:~$  
[09/13/23]seed@VM:~$ echo -n "1234567890" > f2.txt  
[09/13/23]seed@VM:~$ openssl enc -aes-128-cbc -e -in f2.txt -out f2_cipher.bin -K 00112233445566778889aabccddeff -iv 0102030405060708  
hex string is too short, padding with zero bytes to length  
[09/13/23]seed@VM:~$ openssl enc -aes-128-cbc -d -nopad -in f2_cipher.bin -out f2_dec.txt -K 00112233445566778889aabccddeff -iv 0102030405060708  
hex string is too short, padding with zero bytes to length  
[09/13/23]seed@VM:~$ hexdump -C f2_dec.txt  
00000000 31 32 33 34 35 36 37 38 39 30 06 06 06 06 06 06 |1234567890.....|  
00000010  
[09/13/23]seed@VM:~$ xxd f2_dec.txt  
00000000: 3132 3334 3536 3738 3930 0606 0606 0606 1234567890.....  
[09/13/23]seed@VM:~$
```

FILE 3 (16 BYTES):

The terminal window shows the following sequence of commands:

```
[09/13/23] seed@VM:~$ echo -n "1234567890" > f2.txt
[09/13/23] seed@VM:~$ openssl enc -aes-128-cbc -e -in f2.txt -out f2_cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/13/23] seed@VM:~$ openssl enc -aes-128-cbc -d -nopad -in f2_cipher.bin -out f2_dec.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/13/23] seed@VM:~$ hexdump -C f2_dec.txt
00000000 31 32 33 34 35 36 37 38 39 30 06 06 06 06 06 |1234567890....|
00000010
[09/13/23] seed@VM:~$ xxd f2_dec.txt
00000000: 3132 3334 3536 3738 3930 0606 0606 0606 1234567890.....
[09/13/23] seed@VM:~$ hexdump -C f3_dec.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
[09/13/23] seed@VM:~$ xxd f3_dec.txt
00000000: 3132 3334 3536 3738 3930 3132 3334 3536 1234567890123456
00000010: 1010 1010 1010 1010 1010 1010 1010 1010 .....
[09/13/23] seed@VM:~$
```

PLAIN TEXT , ENCRYPTED , DEENCRYPTED FILE SIZES :

The terminal window shows the following file sizes:

File	Size (bytes)
f1.txt	5
f1_cipher.bin	16
f1_dec.txt	16
f2.txt	10
f2_cipher.bin	16
f2_dec.txt	16
f3.txt	16
f3_cipher.bin	32
f3_dec.txt	32

Sizes of encrypted files are (in bytes)

- 16 bytes for the file that contains 5 (bytes) characters
- 16 bytes for the file that contains 10 (bytes) characters
- 32 bytes for the file that contains 16 (bytes) characters

Padding bits

- In File 1 that contains 5 (bytes) characters, the padding bits 0b is appended to make it 128 bits
- In File 2 that contains 10 (bytes) characters, the padding bits 06 is appended to make it 128 bits
- In File 3 that contains 16 (bytes) characters, no padding bits are added as it already 128 bits long

Error Propagation – Corrupted Cipher Text

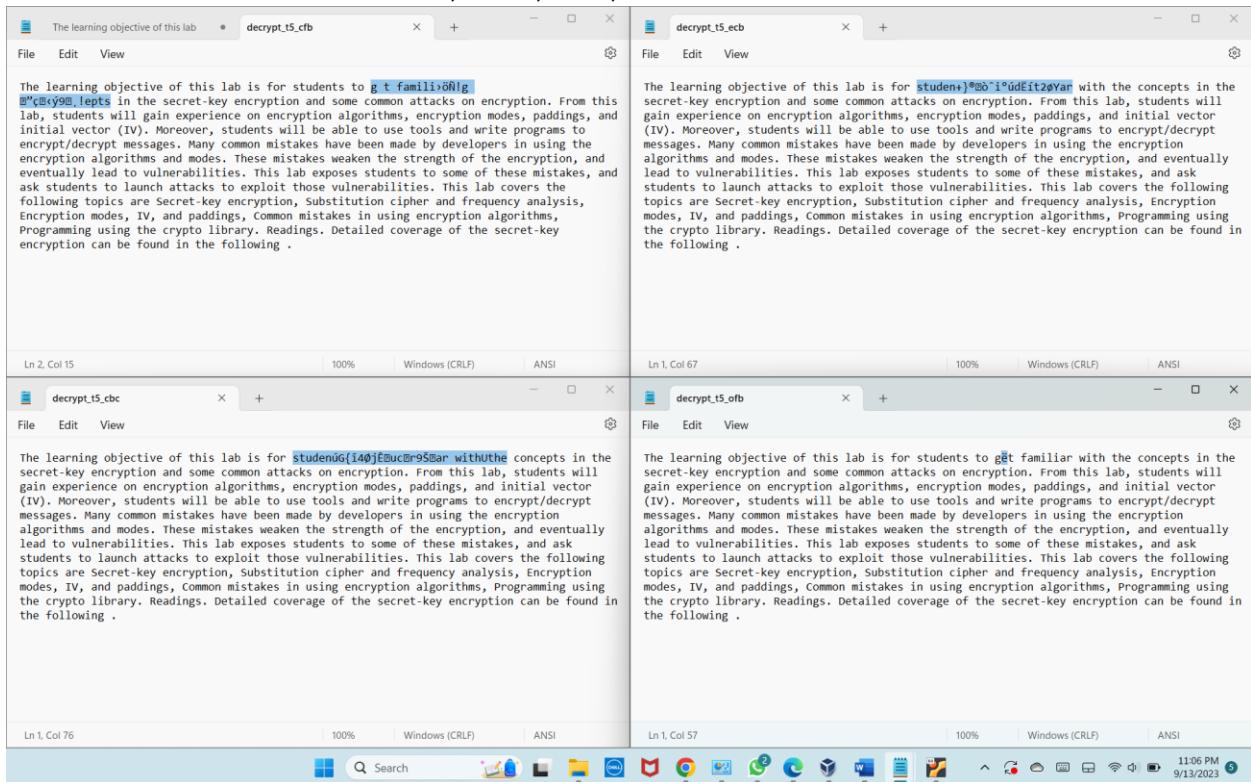
CORRUPTING THE TEXT AT 55TH BYTE

The image consists of four vertically stacked screenshots of the HexEdit browser-based online hex editor, demonstrating the propagation of errors in cipher text.

- Screenshot 1:** Shows the original cipher text (cipher_t5-Copy.bin) with the 55th byte highlighted in yellow. The byte value is 55 (hex). The text pane shows readable ASCII characters.
- Screenshot 2:** Shows the corrupted cipher text (corrupted_cipher.bin) after changing the 55th byte to 00 (hex). The text pane shows partially corrupted ASCII characters.
- Screenshot 3:** Shows the corrupted cipher text (corrupted_cbc.bin) after changing the 55th byte to 00 (hex). The text pane shows more corrupted ASCII characters.
- Screenshot 4:** Shows the corrupted cipher text (corrupted_ecb.bin) after changing the 55th byte to 00 (hex). The text pane shows severely corrupted and illegible ASCII characters.

In all screenshots, the file size is 1,000 bytes. The "Data Inspector (Little-endian)" section shows the byte values for each row. The "File Information" section shows the file name and type. The "Go To" panel on the right shows the current address (0x00000038), last address (0x000003E7), and search bar.

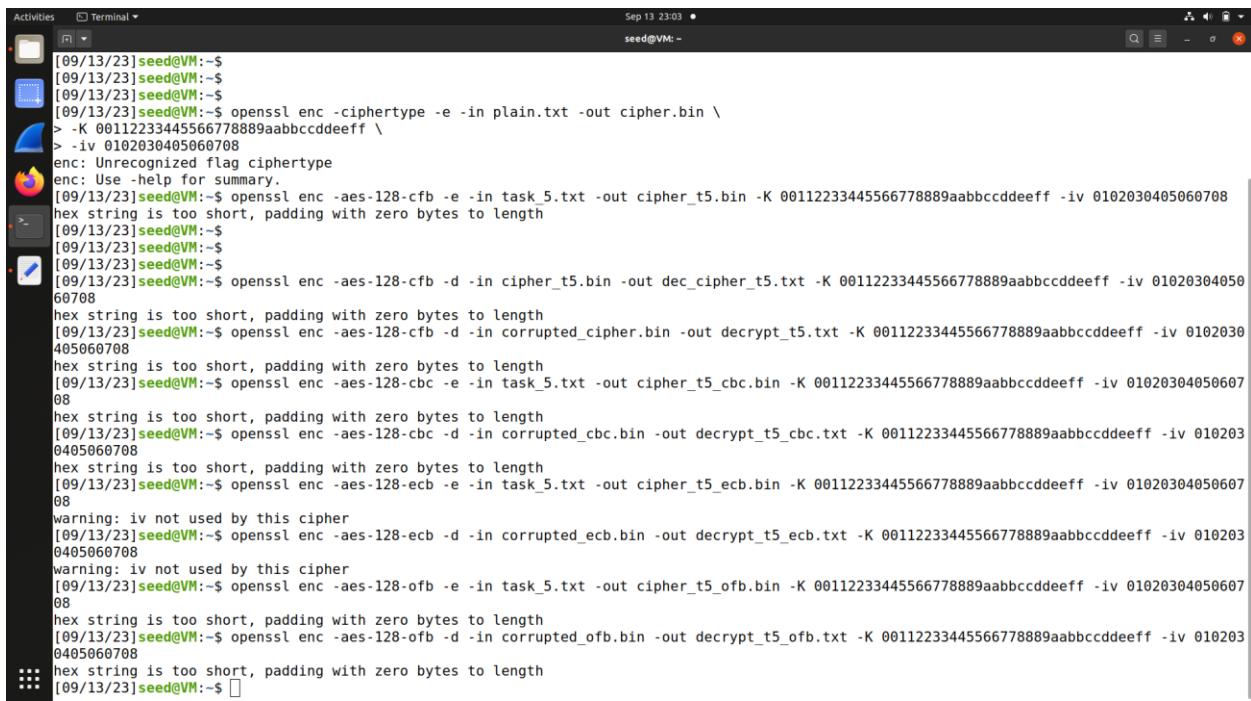
CORRUPTED TEXT FOR CFB, CBC, ECB, OFB RESPECTIVELY



LINUX SCRIPTS :

The image shows a Linux desktop environment with several windows open. In the foreground, a terminal window is active, showing a series of commands and their outputs related to encryption. The terminal session starts with creating a file 'f1.txt' containing the intended text, then echoing it to standard output. It then attempts to use 'openssl enc' with various flags, which results in errors due to unrecognized flags or incorrect usage. Finally, it uses the correct command to encrypt 'plain.txt' to 'cipher.bin' using AES-128-CFB mode with a specific key and IV, and then decrypts it back to 'dec_cipher_t5.txt'. The terminal window also shows the date and time (Sep 13 22:45) and the user's name (seed@VM:~).

```
[09/13/23]seed@VM:~$ echo -n "12345" > f1.txt
[09/13/23]seed@VM:~$ echo -n "The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption and some common attacks on encryption. From this lab, students will gain experience on encryption algorithms, encryption modes, paddings, and initial vector (IV). Moreover, students will be able to use tools and write programs to encrypt/decrypt messages. Many common mistakes have been made by developers in using the encryption algorithms and modes. These mistakes weaken the strength of the encryption, and eventually lead to vulnerabilities. This lab exposes students to some of these mistakes, and ask students to launch attacks to exploit those vulnerabilities. This lab covers the following topics are Secret-key encryption, Substitution cipher and frequency analysis, Encryption modes, IV, and paddings, Common mistakes in using encryption algorithms, Programming using the crypto library. Readings. Detailed coverage of the secret-key encryption can be found in the following ." > task_5.txt
[09/13/23]seed@VM:~$ 
[09/13/23]seed@VM:~$ 
[09/13/23]seed@VM:~$ 
[09/13/23]seed@VM:~$ openssl enc -ciphertextype -e -in plain.txt -out cipher.bin \
> -K 00112233445566778899aabcccddeeff \
> -iv 0102030405060708
enc: Unrecognized flag ciphertextype
enc: Use -help for summary.
[09/13/23]seed@VM:~$ openssl enc -aes-128-cfb -e -in task_5.txt -out cipher_t5.bin -K 00112233445566778899aabcccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/13/23]seed@VM:~$ 
[09/13/23]seed@VM:~$ 
[09/13/23]seed@VM:~$ 
[09/13/23]seed@VM:~$ openssl enc -aes-128-cfb -d -in cipher_t5.bin -out dec_cipher_t5.txt -K 00112233445566778899aabcccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/13/23]seed@VM:~$ openssl enc -aes-128-cfb -d -in corrupted_cipher.bin -out decrypt_t5.txt -K 00112233445566778899aabcccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[09/13/23]seed@VM:~$ 
```



The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "Terminal" and the date and time are "Sep 13 23:03". The user is running a series of OpenSSL commands to encrypt and decrypt files, specifically task_5.txt, using various cipher modes (cfb, t5, cbc, ECB, OFB) and different padding options (PKCS7, OAEP). The terminal output includes error messages such as "hex string is too short, padding with zero bytes to length" and "warning: iv not used by this cipher". The user's session ID is "seed@VM:~\$".

```
[09/13/23]seed@VM:~$  
[09/13/23]seed@VM:~$  
[09/13/23]seed@VM:~$  
[09/13/23]seed@VM:~$ openssl enc -ciphertextype -e -in plain.txt -out cipher.bin \  
> -K 00112233445566778889aabccddeff \  
> -iv 0102030405060708  
enc: Unrecognized flag ciphertextype  
enc: Use -help for summary.  
[09/13/23]seed@VM:~$ openssl enc -aes-128-cfb -e -in task_5.txt -out cipher_t5.bin -K 00112233445566778889aabccddeff -iv 0102030405060708  
hex string is too short, padding with zero bytes to length  
[09/13/23]seed@VM:~$  
[09/13/23]seed@VM:~$  
[09/13/23]seed@VM:~$  
[09/13/23]seed@VM:~$ openssl enc -aes-128-cfb -d -in cipher_t5.bin -out dec_cipher_t5.txt -K 00112233445566778889aabccddeff -iv 0102030405060708  
hex string is too short, padding with zero bytes to length  
[09/13/23]seed@VM:~$ openssl enc -aes-128-cfb -d -in corrupted_cipher.bin -out decrypt_t5.txt -K 00112233445566778889aabccddeff -iv 0102030405060708  
hex string is too short, padding with zero bytes to length  
[09/13/23]seed@VM:~$ openssl enc -aes-128-cbc -e -in task_5.txt -out cipher_t5_cbc.bin -K 00112233445566778889aabccddeff -iv 0102030405060708  
hex string is too short, padding with zero bytes to length  
[09/13/23]seed@VM:~$ openssl enc -aes-128-cbc -d -in corrupted_cbc.bin -out decrypt_t5_cbc.txt -K 00112233445566778889aabccddeff -iv 0102030405060708  
hex string is too short, padding with zero bytes to length  
[09/13/23]seed@VM:~$ openssl enc -aes-128-ecb -e -in task_5.txt -out cipher_t5_ecb.bin -K 00112233445566778889aabccddeff -iv 0102030405060708  
warning: iv not used by this cipher  
[09/13/23]seed@VM:~$ openssl enc -aes-128-ecb -d -in corrupted_ecb.bin -out decrypt_t5_ecb.txt -K 00112233445566778889aabccddeff -iv 0102030405060708  
warning: iv not used by this cipher  
[09/13/23]seed@VM:~$ openssl enc -aes-128-ofb -e -in task_5.txt -out cipher_t5_ofb.bin -K 00112233445566778889aabccddeff -iv 0102030405060708  
hex string is too short, padding with zero bytes to length  
[09/13/23]seed@VM:~$ openssl enc -aes-128-ofb -d -in corrupted_ofb.bin -out decrypt_t5_ofb.txt -K 00112233445566778889aabccddeff -iv 0102030405060708  
hex string is too short, padding with zero bytes to length  
[09/13/23]seed@VM:~$
```

TASK DONE AND OBSERVATION

- Created a file of 1000 bytes long and encrypted it using AES 128 Cipher
- Corrupted the 55th bit using hexEdit online tool
- Then decrypted the corrupted file
- It is observed that some characters were in non readable form and in case of OFB mode only the corresponding byte was unreadable and others were decrypted properly
- Hence Im able to infer that the OFB mode is the least affected mode to corruption out of the above modes

Initial Vector (IV) and Common Mistakes

Case1 : SAME IV

```

decrypt_t5_cbc cipher_6_1_ivsame3
File Edit View
Ln 1, Col 1 100% Unix (LF) ANSI
cipher_6_1_ivsame3_ctb_diff_plaint
File Edit View
Ln 1, Col 1 100% Unix (LF) ANSI
11:50 PM 9/13/2023

```

The screenshots show two windows side-by-side. Both windows have a title bar with the file name and are displaying the same block of ciphertext. The ciphertext appears to be a long string of characters, likely binary or hex, with some recognizable patterns like '0x' and '01'. The windows are running on a Windows operating system, as indicated by the taskbar at the bottom.

Case 2 : DIFFERENT IV

```

cipher_6_1_iv2
File Edit View
Ln 1, Col 1 100% Unix (LF) ANSI
The learning objec * decrypt_t5_cfb cipher_6_1_iv1
File Edit View
Ln 1, Col 1 100% Unix (LF) ANSI
11:50 PM 9/13/2023

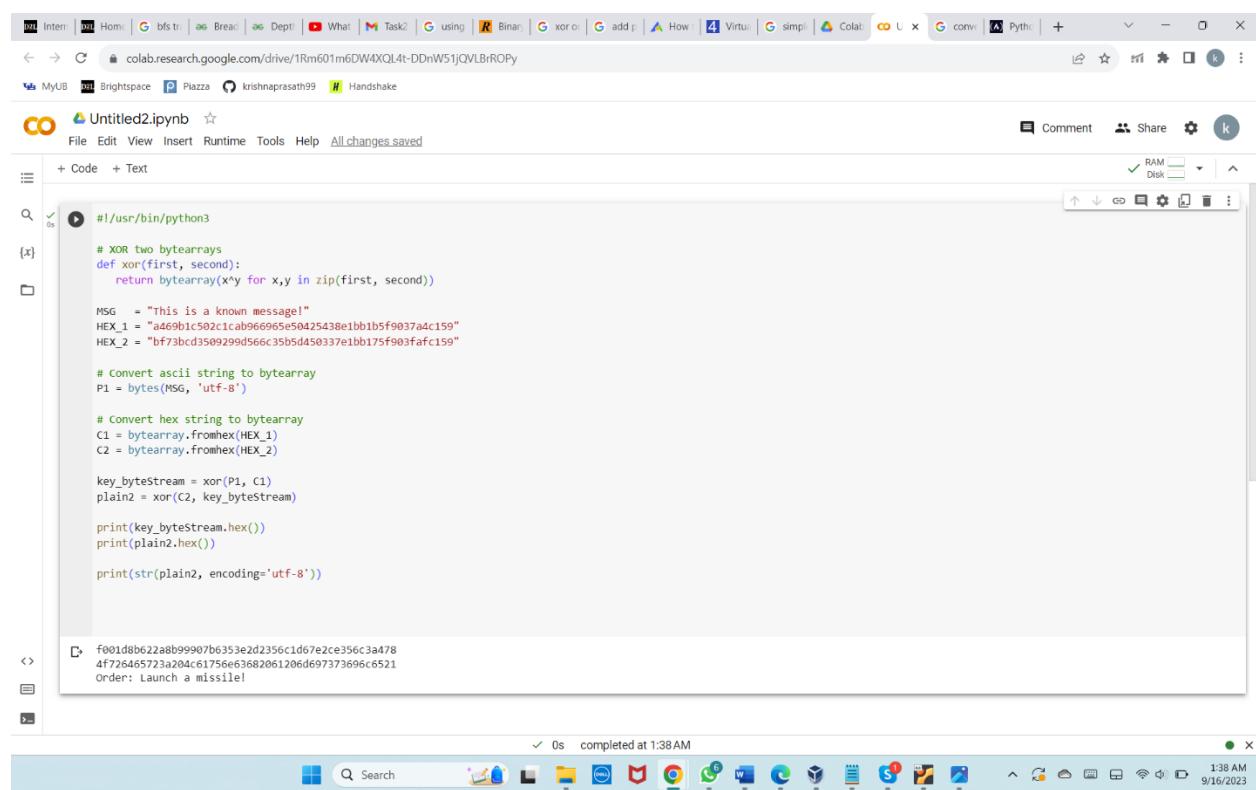
```

The screenshots show two windows side-by-side. The left window is titled 'cipher_6_1_iv2' and the right window is titled 'cipher_6_1_iv1'. Both windows are displaying ciphertext, but the content is visibly different. The right window's ciphertext includes several characters that are not present in the left window's, such as 'A', 'E', 'I', 'O', 'U', 'Y', and 'Z'. The windows are running on a Windows operating system, as indicated by the taskbar at the bottom.

OBSERVATION :

- In case1 while using the same IV, some of the encrypted text's pattern are repeated.
- So it is easier for the attacker to predict the pattern of the cipher and find the text easily.
- But when we used different IV (In case 2), even for the same text, the cipher text differs.
- Hence, we can understand that the IVs should be different

Common Mistake: Use of Same IV



```

# !/usr/bin/python3

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

MSG = "This is a known message!"
HEX_1 = "a4e9b1c5021ca0966965e50425438e1bb1b5f9037a4c159"
HEX_2 = "bf73bcd3509299d566c35bd450337e1bb175f903fafc159"

# Convert ascii string to bytearray
P1 = bytes(MSG, 'utf-8')

# Convert hex string to bytearray
C1 = bytearray.fromhex(HEX_1)
C2 = bytearray.fromhex(HEX_2)

key_bytostream = xor(P1, C1)
plain2 = xor(C2, key_bytostream)

print(key_bytostream.hex())
print(plain2.hex())

print(str(plain2, encoding='utf-8'))

```

Output:

```

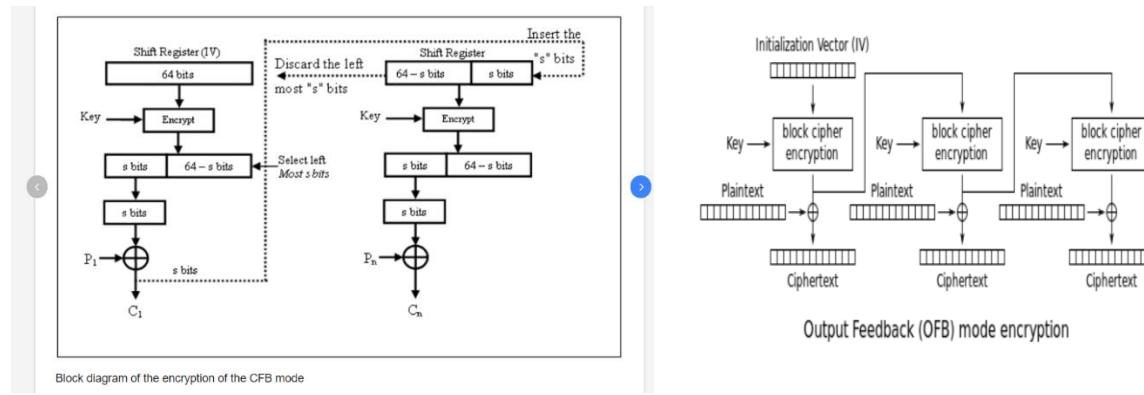
f001d8b622a8b9907b6353e2d2356c1d67e2ce356c3a478
4f7264f572a204c61756e63682061206d697373696c6521
Order: Launch a missile!

```

Plain text P2 is Order: Launch a missile!

TASK DONE AND OBSERVATION :

- As the xor operation is commutative, if we do the xor operation on plaintext (in byte array) with its corresponding cipher text we could get the key, because cipher text is obtained by doing the xor operation on plain text and key.
- Hence we can find the key.
- After finding the key, we can do the xor operation with key and other cipher text to find the plain text



- From the block diagrams of CFB and OFB mode we can understand that the, for the first 64 bits of data the IV is same,
- Then the encrypted key is given as IV for next 64 bits in OFB
- In CFB its xor with the Plain text and its given as input of IV for the next 64 bits.
- Hence if we replace OFB with CFB in this experiment, **the first 64 bits of data can be revealed** as there is no change of logic in the IV

Common Mistake: Use a Predictable IV

```
[09/15/23]seed@VM:~$ cd Labsetup/  
[09/15/23]seed@VM:~/Labsetup$ docker ps  
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
3688741ee8a8 seed-image-encryption "/bin/sh -c ./server" 58 seconds ago Up 56 seconds oracle-10.9.0  
.80  
[09/15/23]seed@VM:~/Labsetup$
```

TASK DONE AND OBSERVATION :

- As the cipher is AES-128-CBC mode, and the plain text is either ‘Yes’ or ‘No’, I tried to convert the string ‘Yes’ to hexadecimal form and added the padding bits at the end.
 - Then I did the xor operation on current IV with the padded data as xor operation is commutative
 - Then did the xor operation of the result with the next IV, so that if it is passed as the plain text it give the same cipher text.
 - Thus from this it can be concluded that the plain text is ‘Yes’
 - I was able to understand that if the attacker have the current IV, next IV and the

Programming using the Crypto Library AES-128-CBC

Source code :

```
from Crypto.Cipher import *
import binascii

plaintext = "This is a top secret."
given_ciphertext="764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da
2ac93a2"
init_vector="aabbccddeeff00998877665544332211"

pad_size = 32-len(plaintext)
padding = chr(pad_size) * (pad_size)
plaintext = plaintext + padding
print("Plain text  -----> ",plaintext)

iv_byte_arr=bytearray.fromhex(init_vector)
print("Hex iv ----->",iv_byte_arr);
f=open("words.txt","r")
allKeys=f.read().splitlines()

for key in allKeys:

    if len(key)==0:
        continue

    if len(key)>16:
        continue

    elif len(key)%16 !=0:
        key_length=len (key)
        while key_length %16!=0:
            key =key+"#"
            key_length = key_length+1

    new_ciphertext = AES.new(bytes (key, "utf-8"), AES.MODE_CBC, iv_byte_arr)
    byte_cipher = new_ciphertext.encrypt (bytes (plaintext, "utf-8"))
    hex_cipher_text=str(binascii.hexlify(byte_cipher).decode())

    if (hex_cipher_text==given_ciphertext):
```

```

finalKey = key
break

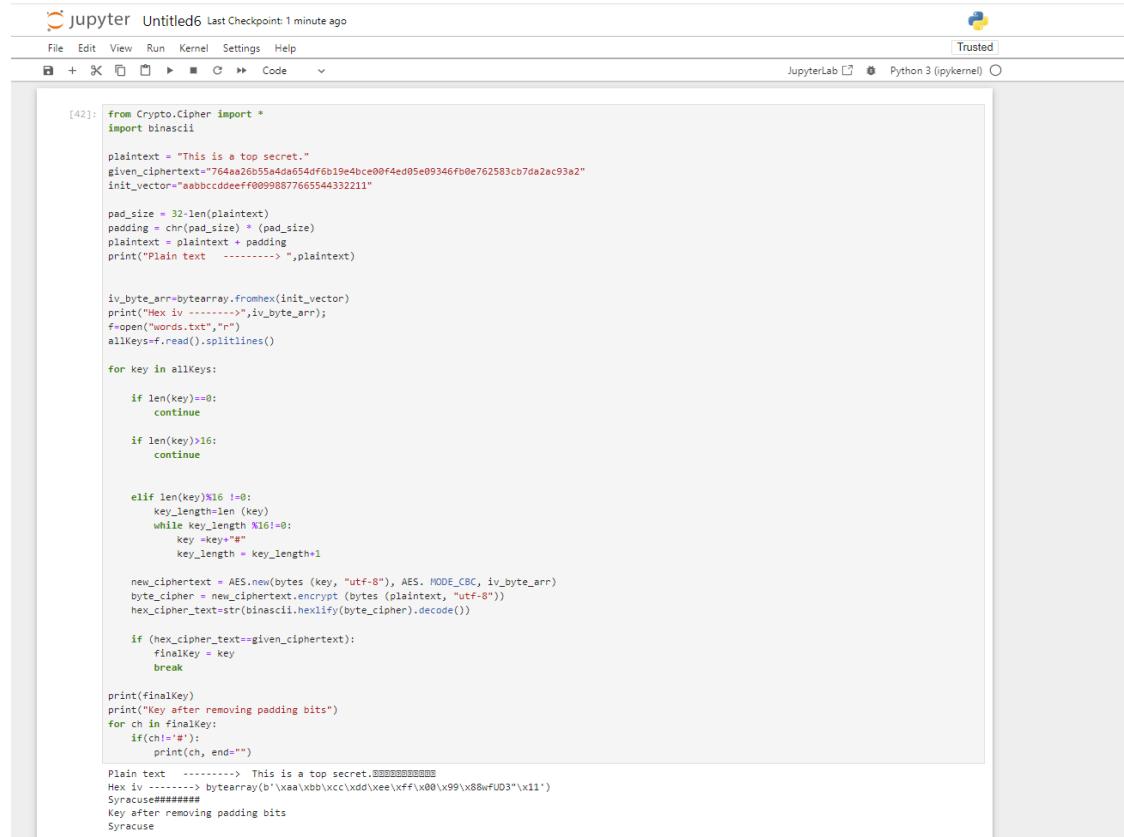
print(finalKey)
print("Key after removing padding bits")
for ch in finalKey:
    if(ch!='#'):
        print(ch, end="")

```

Output :

Plain text -----> This is a top secret.
Hex iv -----> bytearray(b'\xaa\xbb\xcc\xdd\xee\xff\x00\x99\x88wfUD3"\x11')
Syracuse#####
Key after removing padding bits
Syracuse

Hence the key is **Syracuse**



The screenshot shows a Jupyter Notebook interface with the following details:

- Kernel:** Python 3 (ipykernel)
- Cell 42:** Contains the provided Python code for cracking the AES-CBC cipher.
- Output:**
 - Plain text -----> This is a top secret.
 - Hex iv -----> bytearray(b'\xaa\xbb\xcc\xdd\xee\xff\x00\x99\x88wfUD3"\x11')
 - Syracuse#####
 - Key after removing padding bits
 - Syracuse

Code Logic :

- Importing the necessary cipher library
- Add 11 byte padding (32 - 21) to the plain text, 21 is length of plain text
- Convert the given hexadecimal initial vector to byte array
- Read all the keys from the given file
- For each key, validate the size, add '#' to make it 16 bytes length (if length < 16)
- Encrypt the plain text using each key
- If the encrypted cipher text matches with the given cipher text, break the loop and exit and print the key

Output :

Hence we are able to get the Key as Syracuse with the additional padded bits '#' at the end. As the length of key is 8 (8 padding bits are added). Then padding bits are removed and the key is displayed.