BIRZEIT UNIVERSITY

Faculty of Engineering and Technology

Master of Software Engineering (SWEN)

MSCE6342: NETWORK SECURITY PROTOCOLS

Second Semester 2021/2022

Student name: Zaina S. Abuabed

Instructor name: Dr. Ahmad S. Alsadeh

Crypto Lab -- Secret-Key Encryption

Encryption is the process of encoding a message so that only authorized parties can view the original message's content. There are two types of encryption secret-key and public-key encryption. In the secret-key encryption, the same key is used for encryption and decryption. However, in public-key encryption, different keys are used for encryption and decryption. In this lab, we will perform a set of tasks to experiment the secret- key encryption.

Task 1: Frequency Analysis:

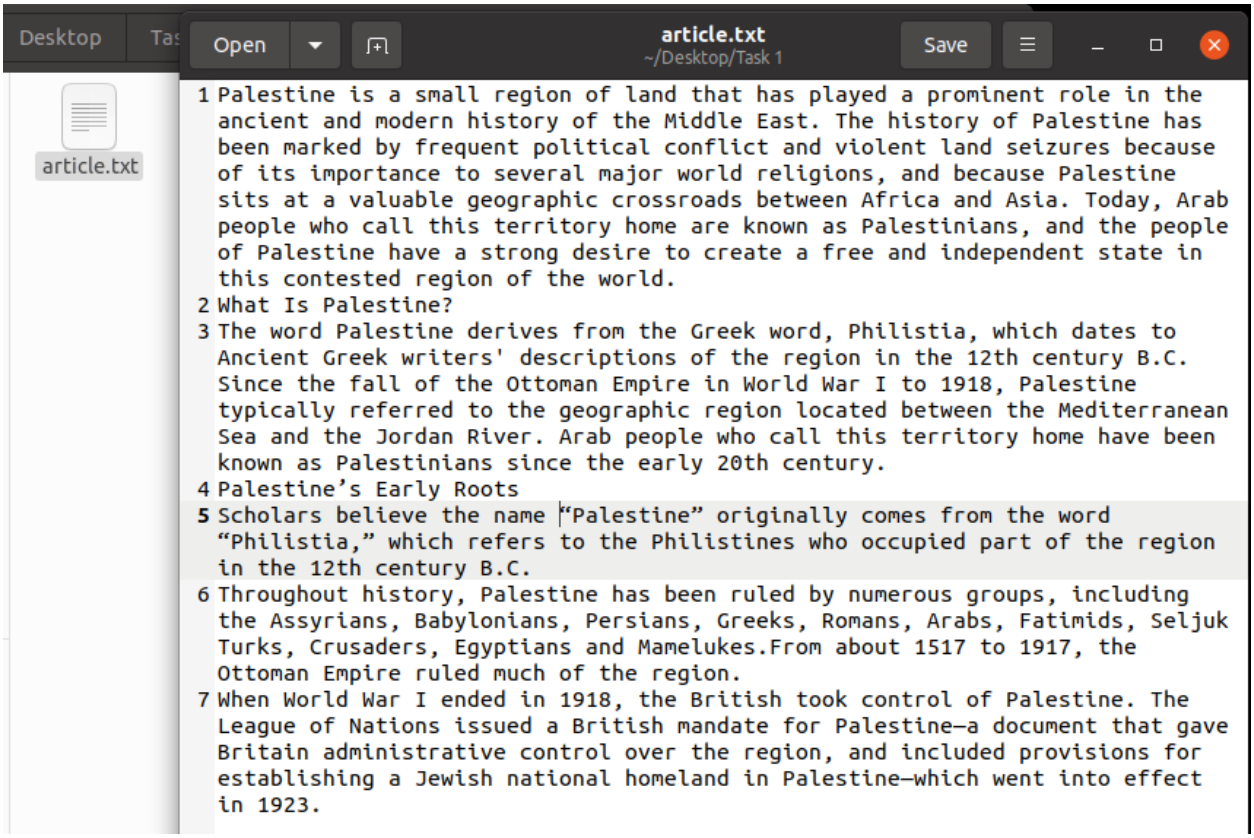
In this task, we will Substitution Cipher where encryption is done by replacing units of plaintext with ciphertext, according to a fixed system. Units may be single letters, pairs of letters, triplets of letters, mixtures of the above, and so forth. Decryption simply performs the inverse substitution. There are two typical substitution ciphers: monoalphabetic and Polyalphabetic.

A fixed mapping table is used in the monoalphabetic substitution cipher. A fixed letter in the plaintext is always mapped to a fixed letter in the cipher. As a result, if a letter appears 100 times in plaintext, the ciphertext letter it maps to will also appear 100 times. The frequency of the symbols in the plaintext will not be modified, despite the fact that they have been changed.

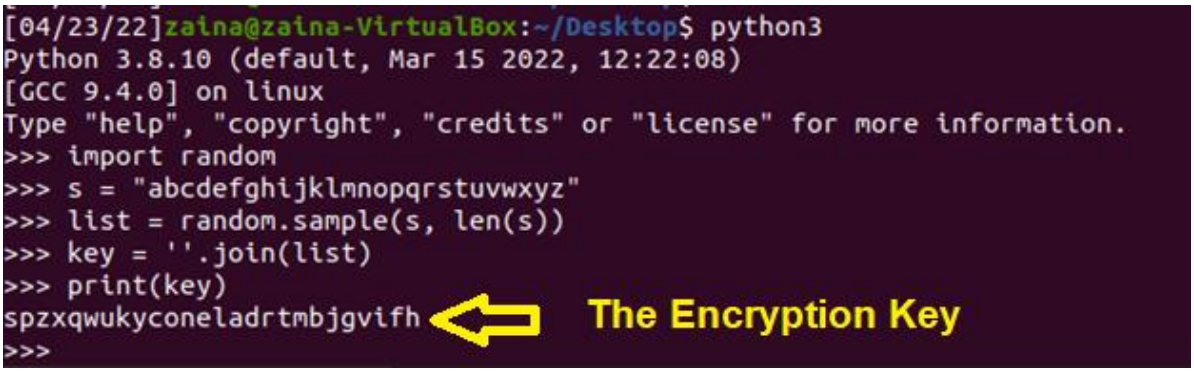
Frequency analysis, which is the study of the frequency of letters or groups of lett ers in a ciphertext, is used to break the monoalphabetic substitution cipher. For example, in English text the common Single letters are T, A, E, I, O, the common 2-letters: TH, HE, IN, ER, and the common 3-letter: THE, AND, and ING.

Part I: Encrypt a text file using monoalphabetic substitution

Step 0: we created a text file with custom text inside named article.txt:

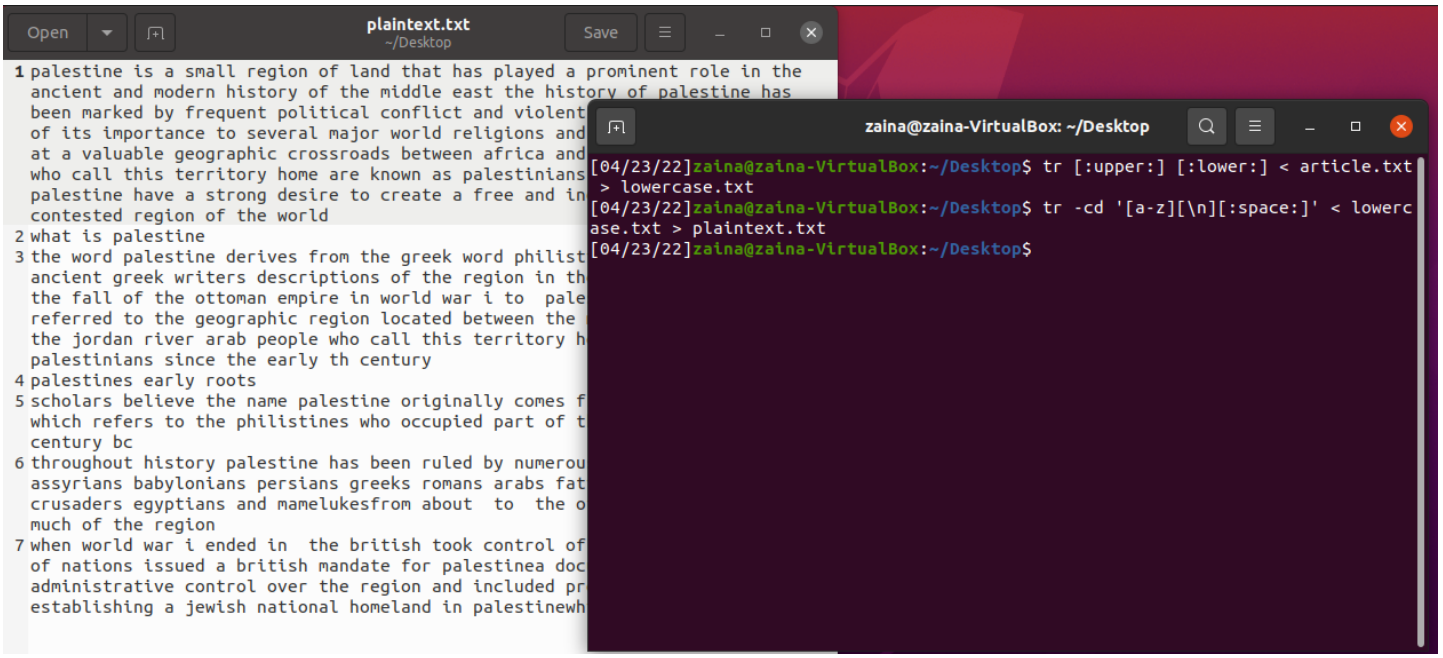


Step1: create the encryption key by applying the given python script:



The Encryption key: spzxqwukyconeladrtmbjgvifh

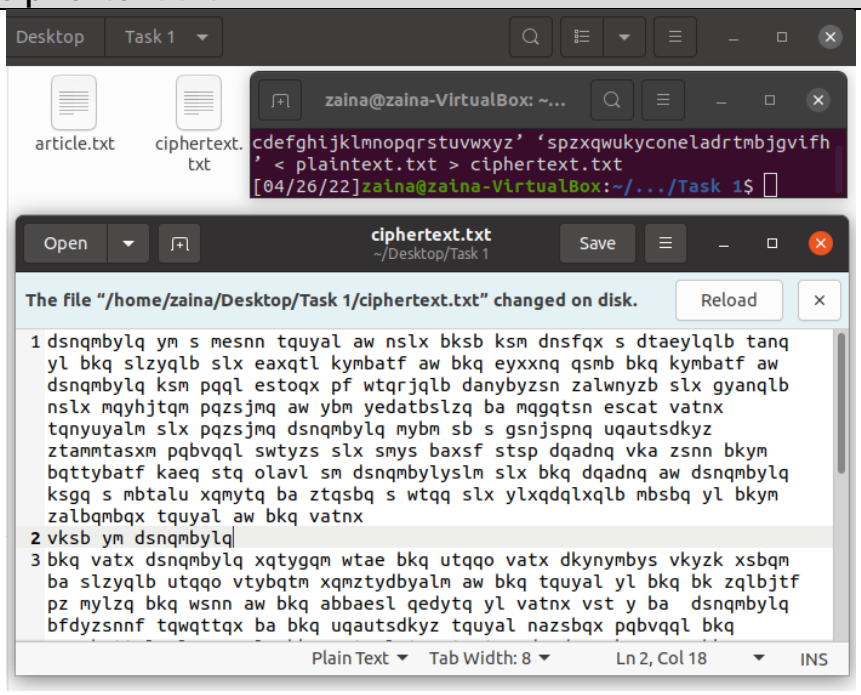
Step 2: for simplicity, we convert the text in the article.txt file to lowercase then eliminate all special characters except letters from a-z, space, and new line.



Step 3: perform monoalphabetic substitution Encryption using tr command

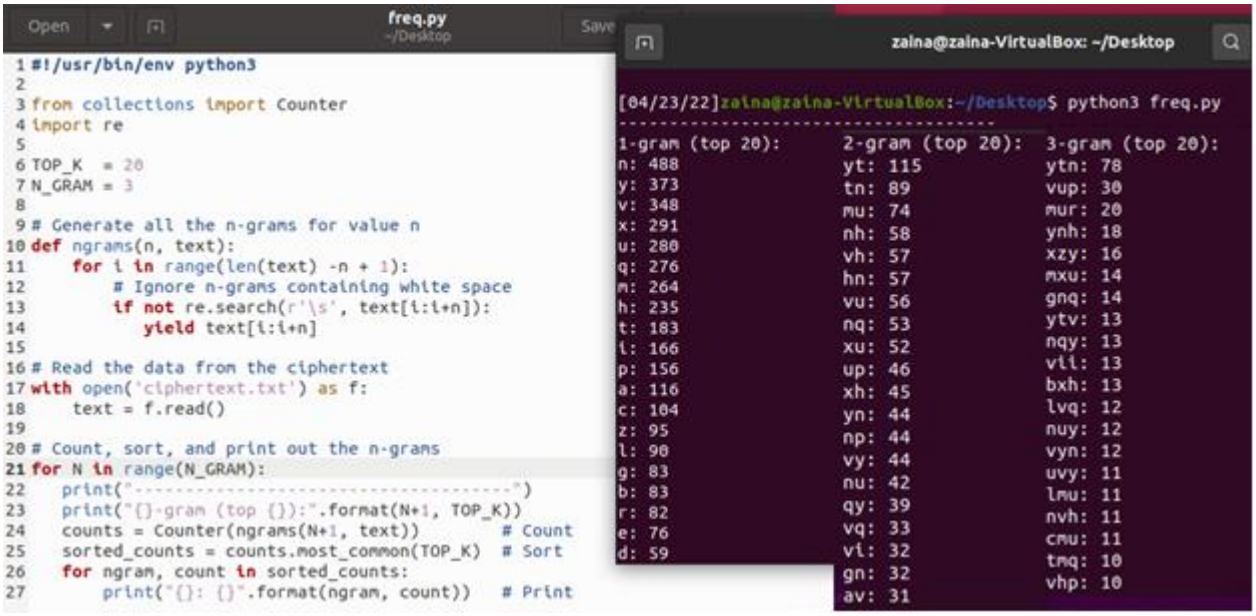
→Encryption:

```
$ tr 'abcdefghijklmnopqrstuvwxyz' 'spzxqwukyconeladrtmbjgvifh' < plaintext.txt > ciphertext.txt
```



Part II: using frequency analysis to figure out the encryption key and the original plaintext of given cipher text.

Step1: run freq.py given script to find the 1-gram , 2-gram , and 3-gram frequencies.



```
#!/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(n, text):
    for i in range(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("{}-gram (top {}):".format(N+1, TOP_K))
    counts = Counter(ngrams(N+1, text))        # Count
    sorted_counts = counts.most_common(TOP_K)  # Sort
    for ngram, count in sorted_counts:
        print("{}: {}".format(ngram, count))   # Print
```

```
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ python3 freq.py
-----------------------------------
1-gram (top 20):      2-gram (top 20):  3-gram (top 20):
n: 488                yt: 115           ytn: 78
y: 373                tn: 89            vup: 30
v: 348                mu: 74            mur: 20
x: 291                nh: 58            ynh: 18
u: 280                vh: 57            xzy: 16
q: 276                hn: 57            mxu: 14
m: 264                vu: 56            gnq: 14
h: 235                nq: 53            ytv: 13
t: 183                xu: 52            nqy: 13
i: 166                up: 46            vii: 13
p: 156                xh: 45            bxh: 13
a: 116                yn: 44            lvq: 12
c: 104                np: 44            nuy: 12
z: 95                 vy: 44            vyn: 12
l: 90                 nu: 42            uvy: 11
g: 83                 qy: 39            lmu: 11
b: 83                 vq: 33            nvh: 11
r: 82                 vi: 32            cmu: 11
e: 76                 gn: 32            tmq: 10
d: 59                 av: 31            vhp: 10
```

Step 2: find the common English text frequencies using Wikipedia

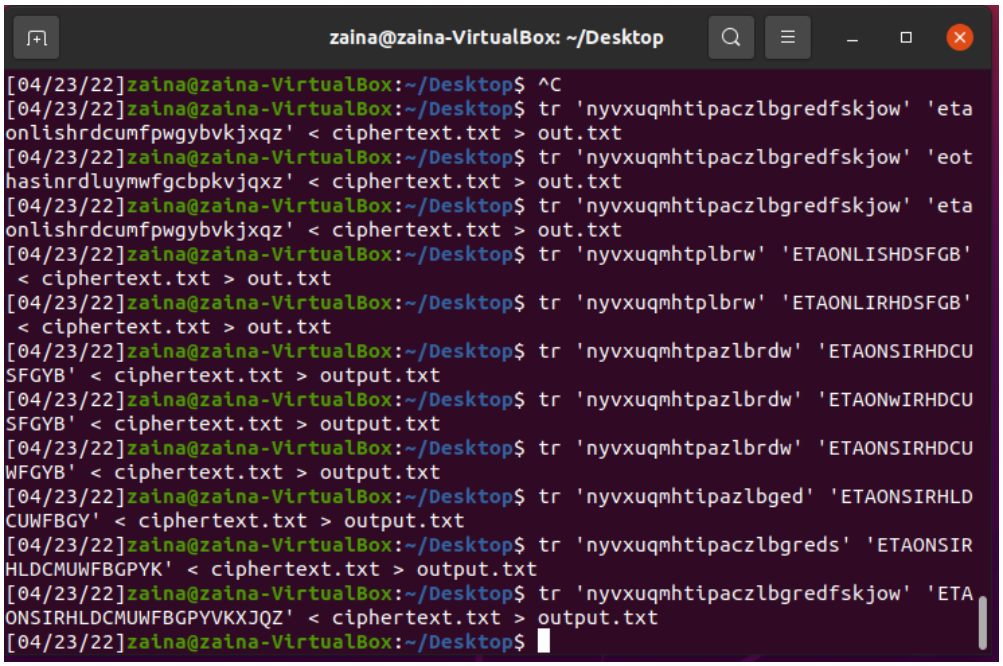
Letters by frequency of appearance in English:

E	12.7 %	**T**	9.1 %	**A**	8.2 %
O	7.5 %	**I**	7.0 %	**N**	6.7 %
S	6.3 %	**H**	6.1 %	**R**	6.0 %
L	4.0 %	**D**	4.3 %	**C**	2.8 %
U	2.8 %	**M**	2.4 %	**W**	2.4 %
F	2.2 %	**G**	2.0 %	**Y**	2.0 %
P	1.9 %	**B**	1.5 %	**V**	1.0 %
K	0.8 %	**J**	0.2 %	**X**	0.2 %
Q	0.1 %	**Z**	0.1 %		

Step 3: using excel sheet to make trials

VLVHP	4	a asd	N	488	12.41%	3931	e	ytn	the	yt	th
UPYTN	4	ndthe	Y	373	9.49%	3931	t	vup	and	mu	in
VPNCD	3	ade	V	348	8.85%	3931	a	mur	ing	nh	
UVYMX	3	natio	X	291	7.40%	3931	o	ynh		vh	ar
VUPYT	3	andth	U	280	7.12%	3931	n	gnq		hn	
NLMYT	3	e ith	Q	276	7.02%	3931	s	xzy		vu	
RIXGN	3	growe	M	264	6.72%	3931	i	nuy		nq	
NAYXH	2		H	235	5.98%	3931	r	bxh			
HNBNH	2		T	183	4.66%	3931	h	vii			
TVYGN	2		I	166	4.22%	3931	L	uvy			
GDYTN	2		P	156	3.97%	3931	d	lmu			
YMXUQ	2	tionl	A	116	2.95%	3931	c	nvh			
GNQYE	2	elty	C	104	2.65%	3931	M	cmu			
			Z	95	2.42%	3931	u				
			L	90	2.29%	3931	w				
			B	83	2.11%	3931	f				
			G	83	2.11%	3931	B				
			R	82	2.09%	3931	G				
			E	76	1.93%	3931	P				
			D	59	1.50%	3931	y				
			F	49	1.25%	3931	V				
			S	19	0.48%	3931	K				
			K	5	0.13%	3931	X				
			J	5	0.13%	3931	Q				
			O	4	0.10%	3931	J				
			W	1	0.03%	3931	Z				
			total	**3931**							

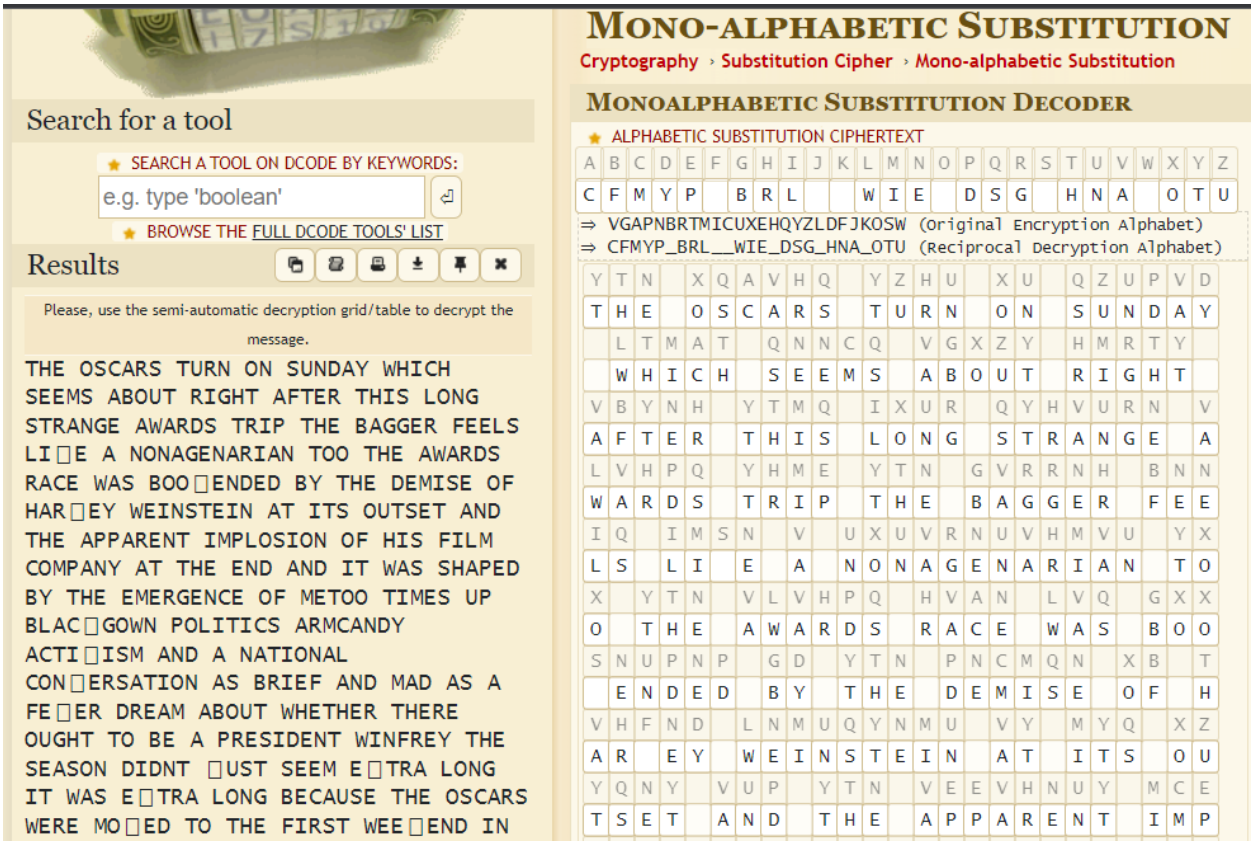
After figuring some letters we substitute the corresponding letters and look at the result to find where we the correct and wrong mapping. Here is some substitution.



Sample from decoded cither of the earliest trials.

THE OLaARL TzRN ON LzNDAd SHIaH LEEcL AgOzT RIGHT AFTER THIL iONG LTRANGE ASARDL TRIe THE gAGGER FEEiL iIsE A NONAGENARIAN TOO
THE ASARDL RAaE SAL gOOsENDED gd THE DEcILE OF HARfEd SEINLTEIN AT ITL OzTLET AND THE AeeARENT IceiOLION OF HIL FIic aOceANd AT THE END AND IT SAL LHAeED gd THE EcERGENaE OF cETOO TIcEL ze giAasGOSN eOiITIaL ARcaANDd AaTIfILc AND A NATIONAi aONfERLATION AL gRIEF AND cAD AL A FEfER DREAc AgOzT SHETHER THERE OzGHT TO gE A eRELIDENT SINFREd THE LEALON DIDNT ozLT LEEc EkTRA iONG IT SAL EkTRA iONG gEaAzLE THE OLaARL SERE cOfED TO THE FIRLT SEEsEND IN cARaH TO AfOID aONFiIaTING SITH THE aiOLING aEREcONd OF THE SINTER OidceIaL THANsL edEONGaHANG
ONE gIG jzELTION LzRROzNDING THIL dEARL AaADEcd ASARDL IL HOS OR IF THE aEREcONd SIii ADDRELL cETOO ELeEaIAiid AFTER THE GOiDEN GiOgEL SHIaH gEaAcE A ozgliANT aOcINGOzT eARTd FOR TIcEL ze THE cOfEcENT LeEARHEADED gd eOSERFzi HOiidSOOD SOcEN SHO HEieED RAILE cIiiIONL OF DOiiARL TO FIGHT LEkzAi HARALLcENT AROzND THE aOzNTRd LIGNAiING THEIR LzeeORT GOiDEN GiOgEL ATTENDEEL LSATHED THEcLEifEL IN giAas
LeORTED iAeEi eINL AND LOzNDED OFF AgOzT LEkILT eOSER IcgAiANaEL FROc THE RED aAReET AND THE LTAGE ON THE AIR E SAL aAiiED OzT AgOzT eAd INEjzITd AFTER ITL FORcER ANaHOR aATT LADiER jzIT ONaE LHE iEARNED THAT LHE SAL cAsING FAR iELL THAN A cAiE aOHOLT AND DzRING THE aEREcONd NATAiIE eORTcAN TOOs A gizNT
AND LATILFdING DIG AT THE AiicAiE ROLTER OF NOcINATED DIREaTORL HOS aOziD THAT gE TOeeED

There is a helpful online tool for manual decryption instead of using excel sheet. It substitutes the guessed character in the whole text instantly and view the result immediately.

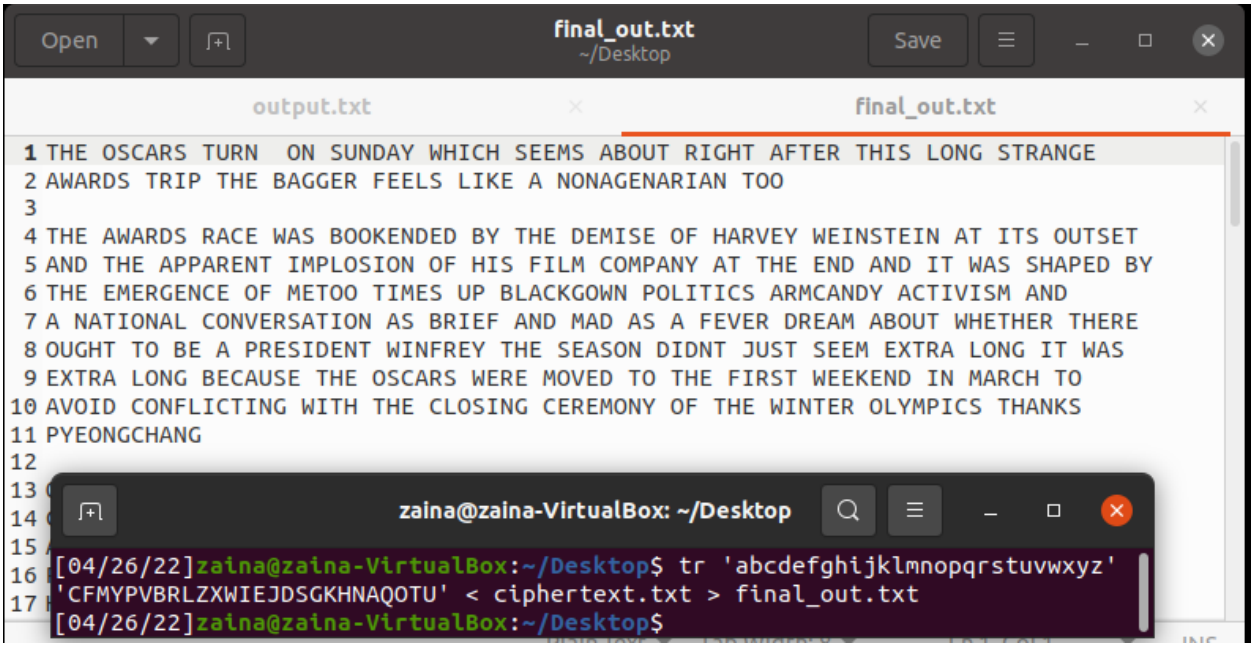


Result: the Key is CFMYPVBRLZXWIEJDSGKHNAQOTU

Step 4: Decrypt the cipher text using the key:

After some trials we found all corresponding letters and finally decrypt the ciohertext.

```
$tr 'abcdefghijklmnopqrstuvwxyz' 'CFMYPVBRLZXWIEJDSGKHNAQOTU' < ciphertext.txt > final_out.txt
```



The final result is described below:

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 QUST 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 JUESTION SURROUNDING THIS YEARS ACADEMY AWARDS IS HOW OR IF THE CEREMONY WILL ADDRESS METOO ESPECIALLY AFTER THE GOLDEN GLOBES WHICH BECAME A QUBILANT 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 INEJUITY AFTER ITS FORMER ANCHOR CATT SADLER JUIT 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 QUST 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 QUDD 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 EJUALLY 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 UNEJUALLY 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

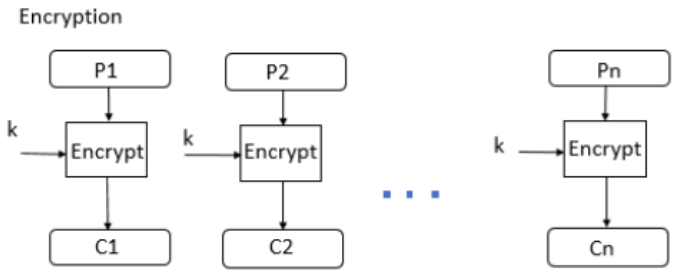
SINCE BRAVEHEART IN THIS YEAR THE BEST ENSEMBLE SAG ENDED UP GOING TO THREE BILLBOARDS WHICH IS SIGNIFICANT BECAUSE ACTORS MAKE UP THE ACADEMYS LARGEST BRANCH THAT FILM WHILE DIVISIVE ALSO WON THE BEST DRAMA GOLDEN GLOBE AND THE BAFTA BUT ITS FILMMAKER MARTIN MCDONAGH WAS NOT NOMINATED FOR BEST DIRECTOR 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

In this task we encrypt/decrypt the article.txt file which was used in task1 using ECB, CBC, CFB, OFB modes using openssl library.

1. Electronic Code Block (ECB) mode

AES128 ECB Encryption:

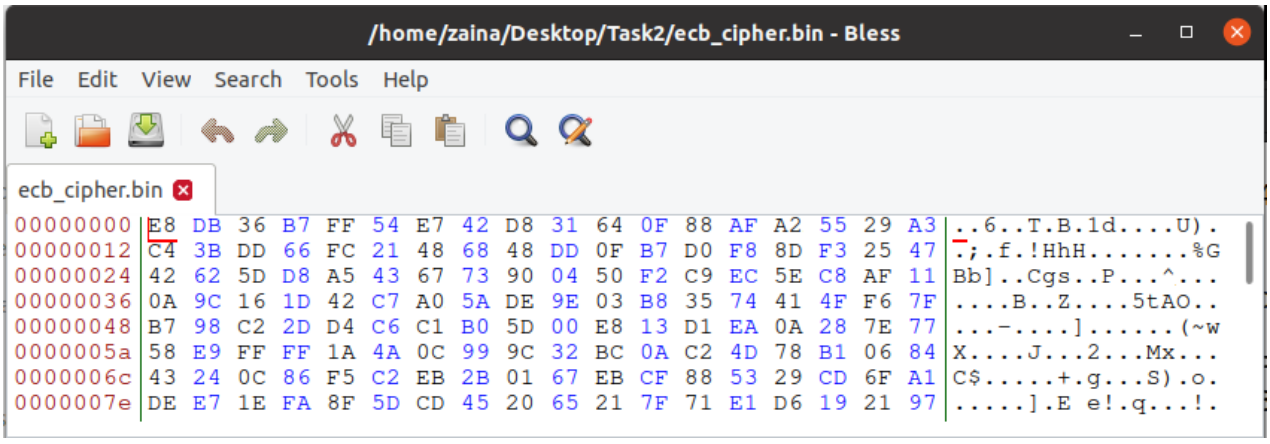


For AES128 ECB encryption we need only the secret key as a 128-bit number (16 byte) represented in hexadecimal. We use openssl library to do the encryption.

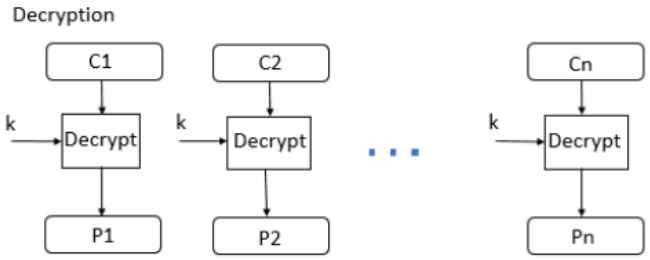
```
$openssl enc  -aes-128-ecb  -e -in article.txt -out ecb_cipher.bin -K 00112233445566778889aabbccddeeff
```

```
[04/23/22]zaina@VM:~/.../Task2$ openssl enc  -aes-128-ecb  -e -in article.txt -o
ut ecb_cipher.bin -K 00112233445566778889aabbccddeeff
```

Then, we use Bless Hex editor to view the encrypted file



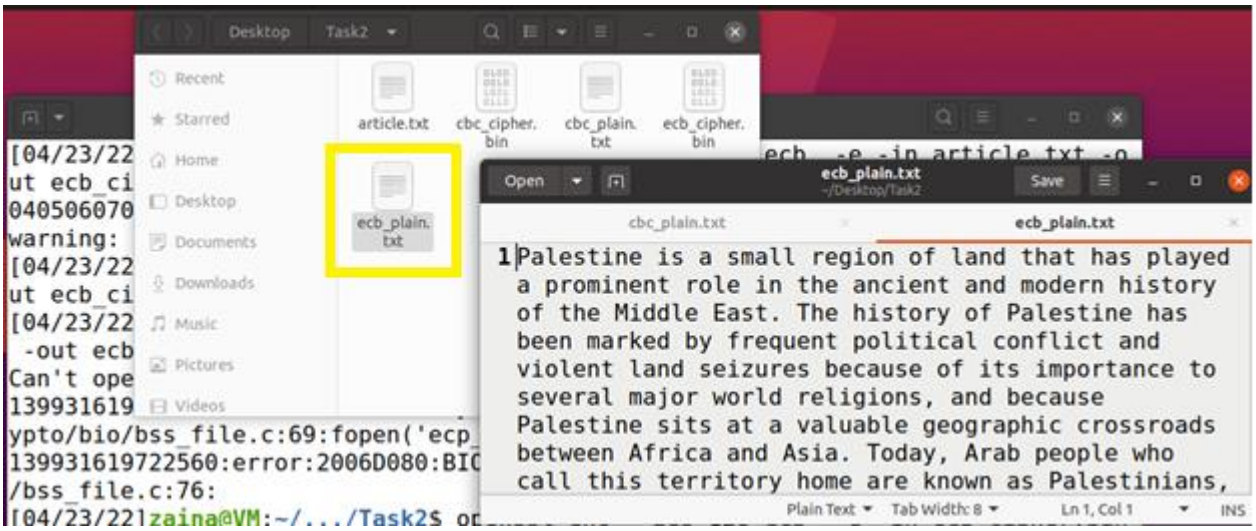
AES128 ECB Decryption:




```
$ openssl enc  -aes-128-ecb  -d -in ecb_cipher.bin -out ecb_plain.txt -K 00112233445566778889aabbccddeeff
```

```
[04/23/22]zaina@VM:~/.../Task2$ openssl enc  -aes-128-ecb  -d -in ecb_cipher.bin
 -out ecb_plain.txt -K 00112233445566778889aabbccddeeff
```

The decrypted file was viewed using text editor. We note that the original plain text retrieved completely.



To emphasize the result, we compare the original file before encryption, and the decrypted one using diff command.

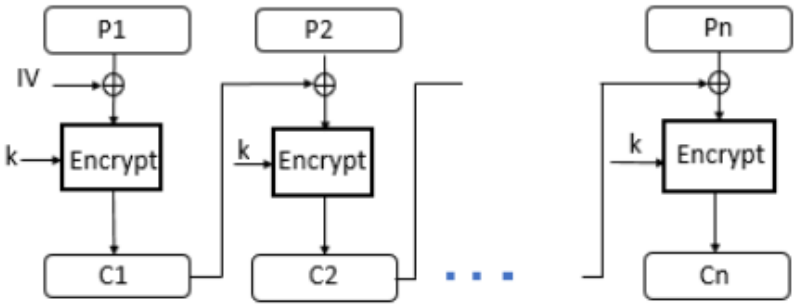
```
$ diff article.txt ecb_plain.txt
```

```
[04/23/22]zaina@VM:~/.../Task2$ diff article.txt ecb_plain.txt
[04/23/22]zaina@VM:~/.../Task2$
```

Since nothing happened, then both files are identical. Moreover, ECB encryption and decryption were performed successfully.

2. Cipher Block Chaining (CBC) Mode

AES128 CBC Encryption:



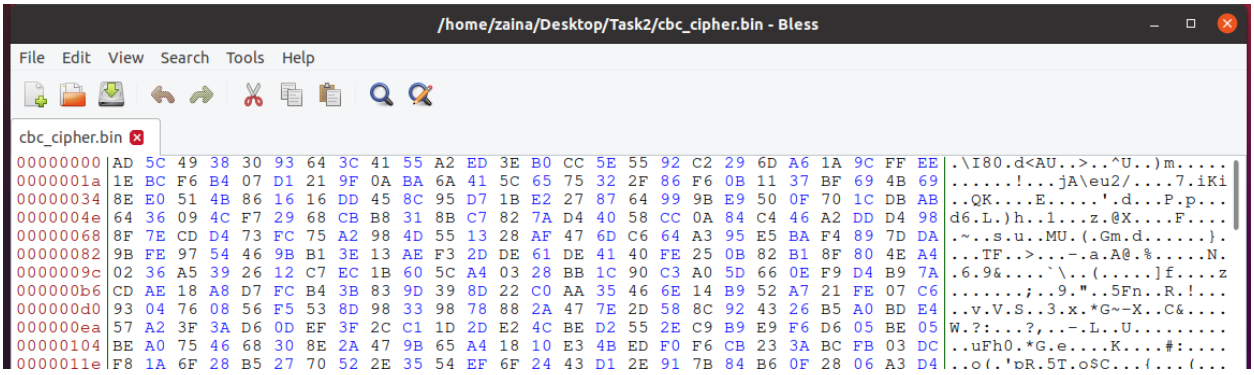
For AES128 CBC encryption we need both the secret key and the initialization vector. We use openssl library to do the encryption. The following command is proposed in the instructions.

```
$ openssl enc  -aes-128-cbc  -e -in plaintext.txt -out cbc_cipher.bin -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
```

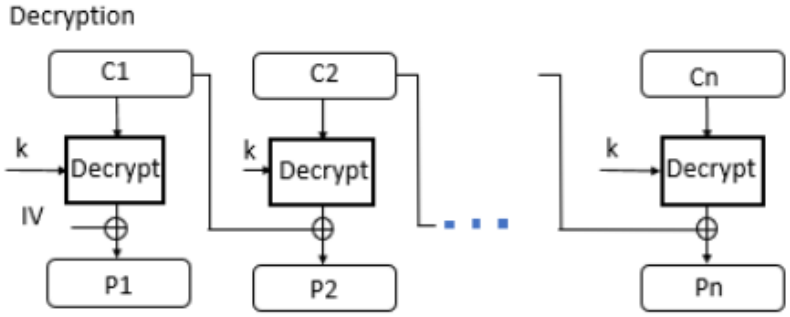
When we applied this command, a warning error appeared means that the initialization vector Is short and it was padded with 0 value bytes. The initialization vector (iv) should be 128 bit (16 bytes) instead of 8 bytes. So we modify the iv → 01020304050607080102030405060708 and redo the encryption.

```
[04/23/22]zaina@VM:~/.../Task2$ openssl enc  -aes-128-cbc  -e -in article.txt -o
ut cbc_cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[04/23/22]zaina@VM:~/.../Task2$ openssl enc  -aes-128-cbc  -e -in article.txt -o
ut cbc_cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708010203
0405060708
```

Here is the output cipher as appeared in Bless.



AES128 CBC Decrypt:




```
$openssl enc  -aes-128-cbc  -d -in cbc_cipher.bin -out cbc_plain.txt -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
```

The decrypted file was viewed using text editor. We note that the original plain text retrieved completely.

To emphasize the result, we compare the original file before encryption, and the decrypted one using diff command.

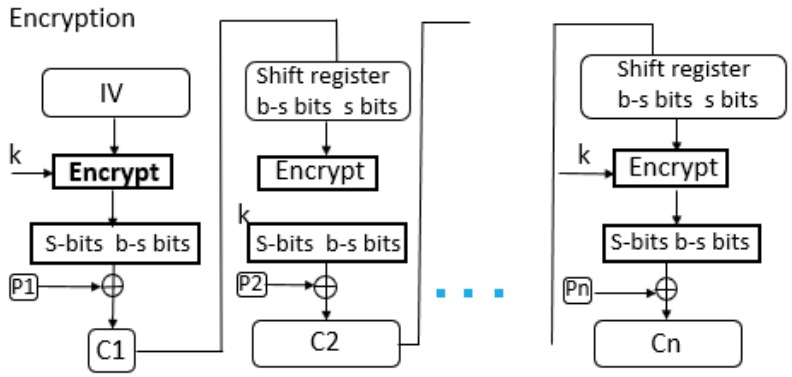
```
$diff article.txt cbc_plain.txt
```

```
[04/23/22]zaina@VM:~/.../Task2$ diff article.txt cbc_plain.txt
[04/23/22]zaina@VM:~/.../Task2$
```

Since nothing happened, then both files are identical. And both CBC encryption and decryption were performed successfully.

3. Cipher Feedback (CFB) Mode

AES128 CFB Encryption:

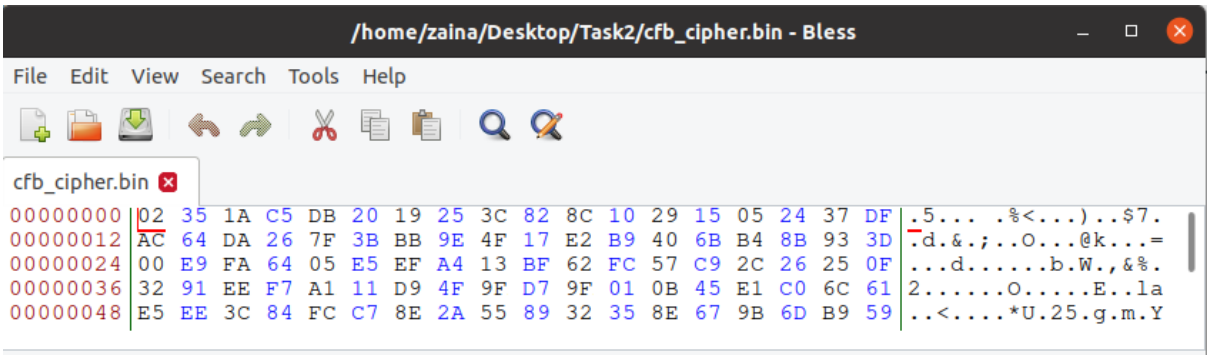


Similar to CBC, CFB requires secret key (K) and initialization vector (iv). We use the same K and iv that was used in CBC encryption.

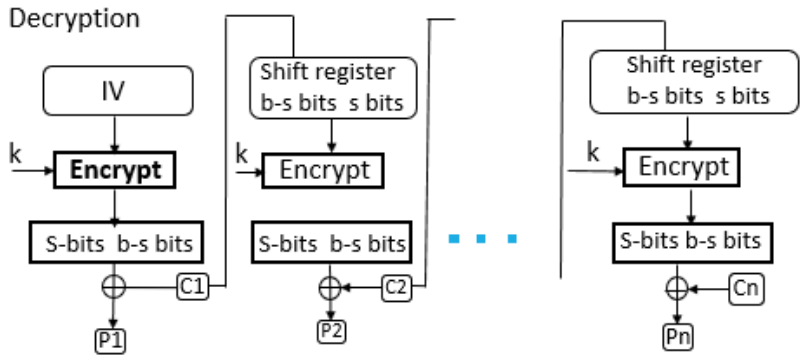
```
$ openssl enc -aes-128-cfb -e -in article.txt -out cfb_cipher.bin -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
```

```
[04/23/22]zaina@VM:~/.../Task2$ openssl enc  -aes-128-cfb  -e -in article.txt -o
ut cfb_cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708010203
0405060708
```

Here is the output cipher as appeared in Bless.



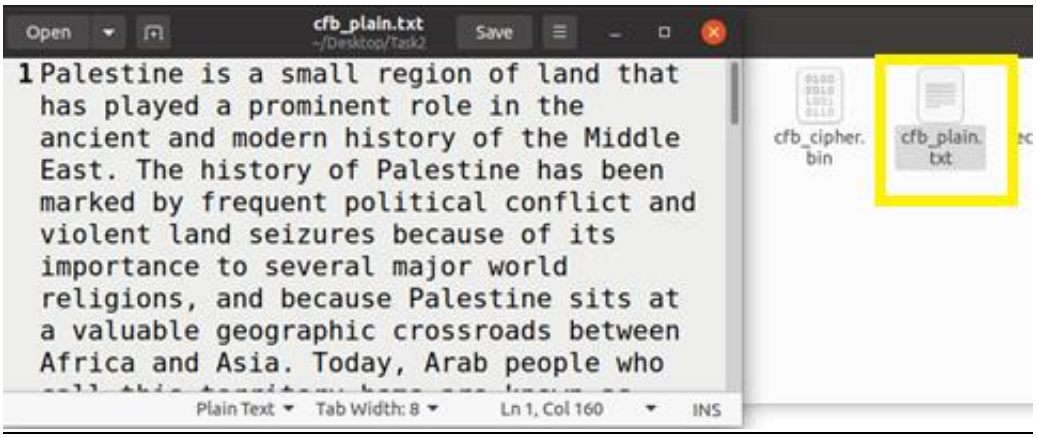
AES128 CFB Decrypt:




```
$ openssl enc  -aes-128-cfb  -d -in cfb_cipher.bin -out cfb_plain.txt -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
```

```
[04/23/22]zaina@VM:~/.../Task2$ openssl enc  -aes-128-cfb  -d -in cfb_cipher.bin
 -out cfb_plain.txt -K 00112233445566778889aabbccddeeff -iv 01020304050607080102
030405060708
```

The decrypted file was viewed using text editor. We note that the original plain text retrieved completely.



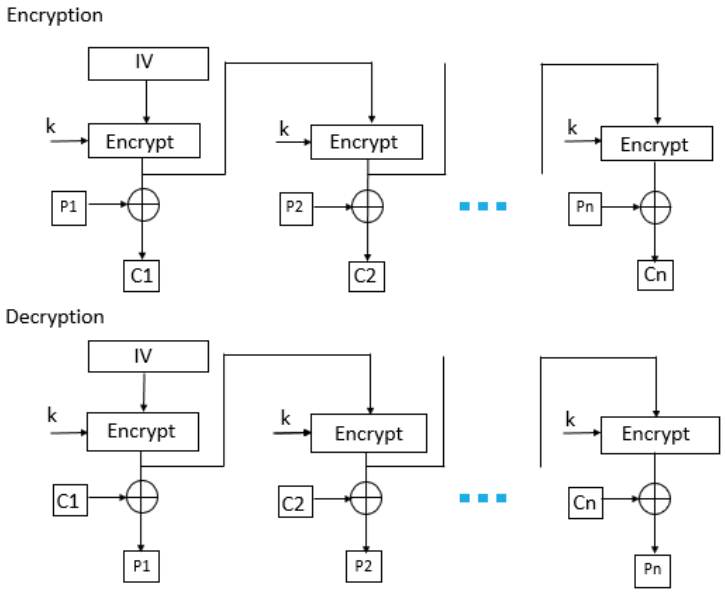
To emphasize the result, we compare the original file before encryption, and the decrypted one using diff command.

```
$ diff cfb_plain.txt article.txt
```

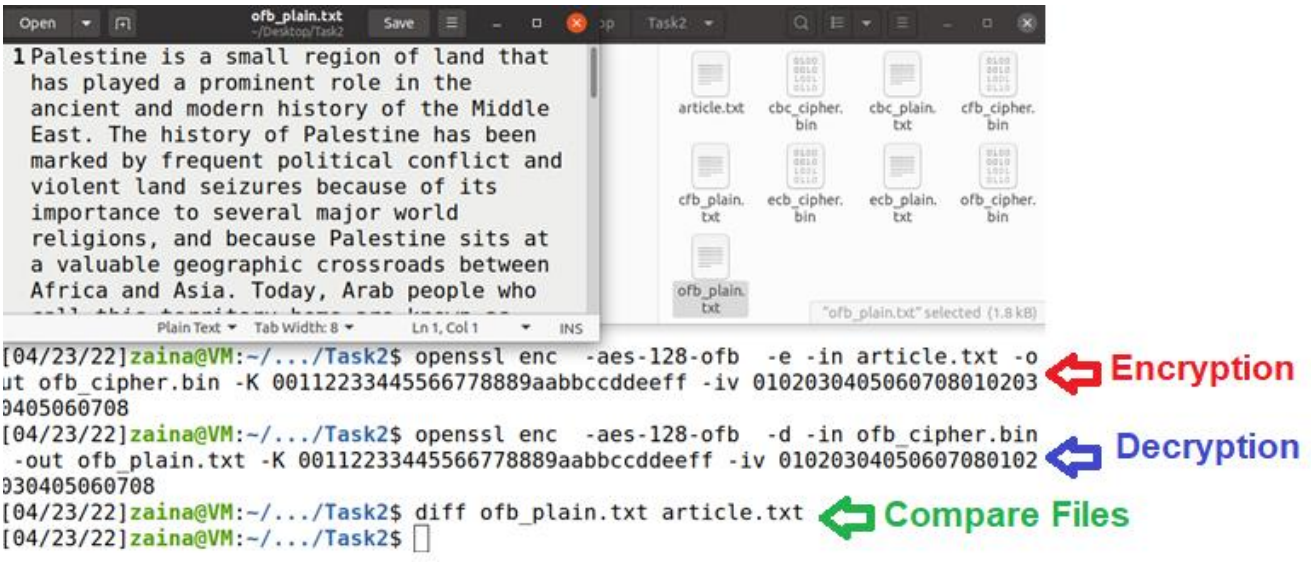
```
[04/23/22]zaina@VM:~/.../Task2$ diff cfb_plain.txt article.txt
[04/23/22]zaina@VM:~/.../Task2$
```

Since nothing happened, then both files are identical. And both CFB encryption and decryption were performed successfully.

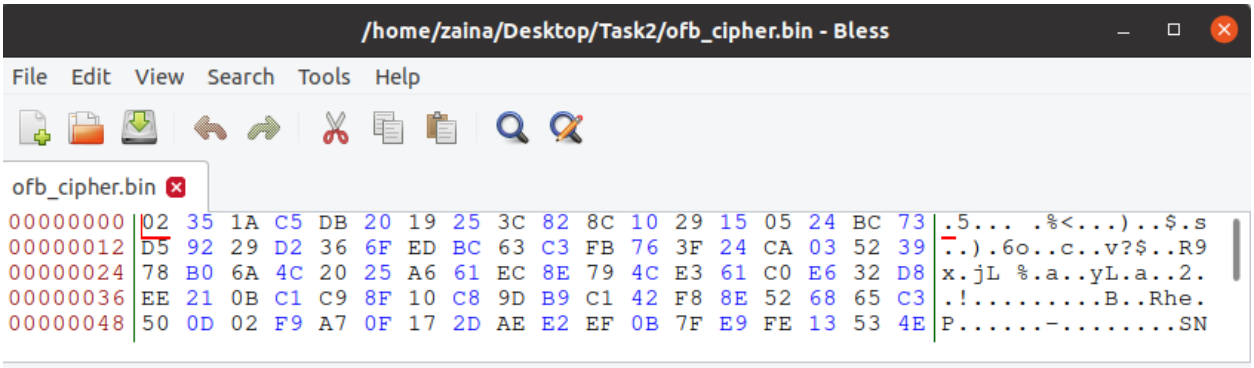
4. Output Feedback (OFB) Mode




```
$ openssl enc  -aes-128-ofb  -e -in article.txt -out ofb_cipher.bin -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$ openssl enc  -aes-128-ofb  -d -in ofb_cipher.bin -out ofb_plain.txt -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$ diff ofb_plain.txt article.txt
```



The encrypted file is shown bellow

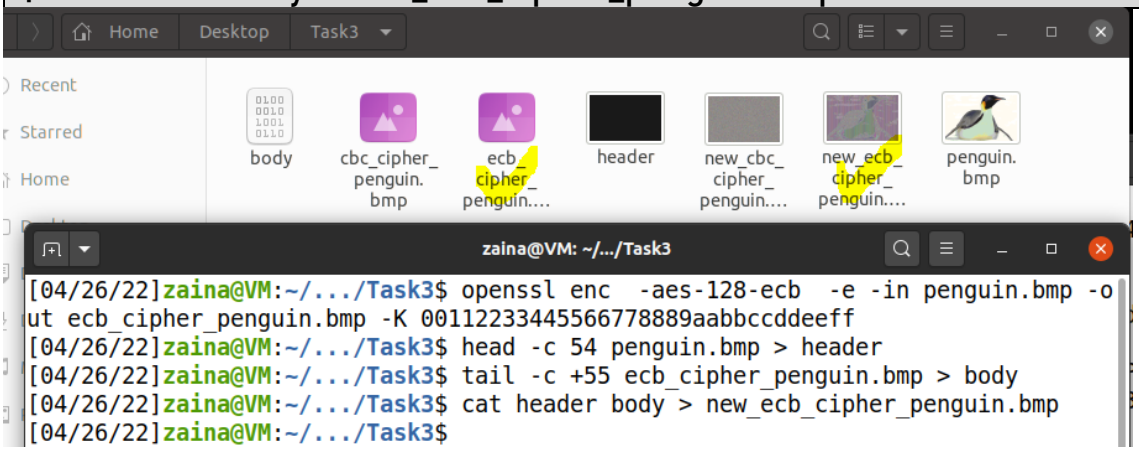


After Using the Openssl to Encrypt/decrypt with ECB, CBC, CFB, OFB, we find that each mode produced different cipher except for the first block in OFB and CFB modes. The remaining blocks are different.

Task 3 Encryption Mode – ECB vs. CBC

Step1: encrypt the penguin.bmp using ECB mode then correct the header to make the encrypted picture readable by photo viewer:

```
$openssl enc  -aes-128-ecb  -e -in penguin.bmp -out
ecb_cipher_penguin.bmp -K 00112233445566778889aabbccddeeff
$head -c 54 penguin.bmp > header
$tail -c +55 ecb_cipher_penguin.bmp > body
$cat header body > new_ecb_cipher_penguin.bmp
```



The encrypted picture in a photo viewer program is shown below:

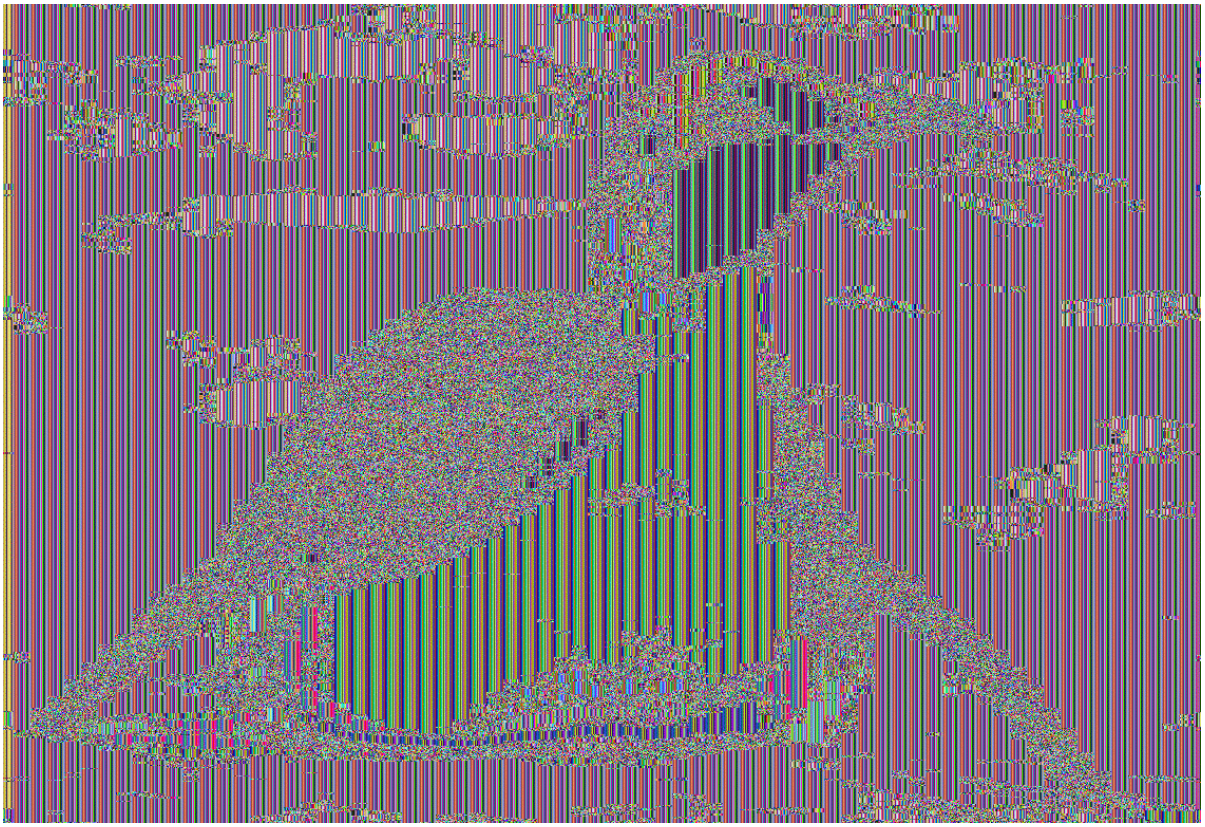
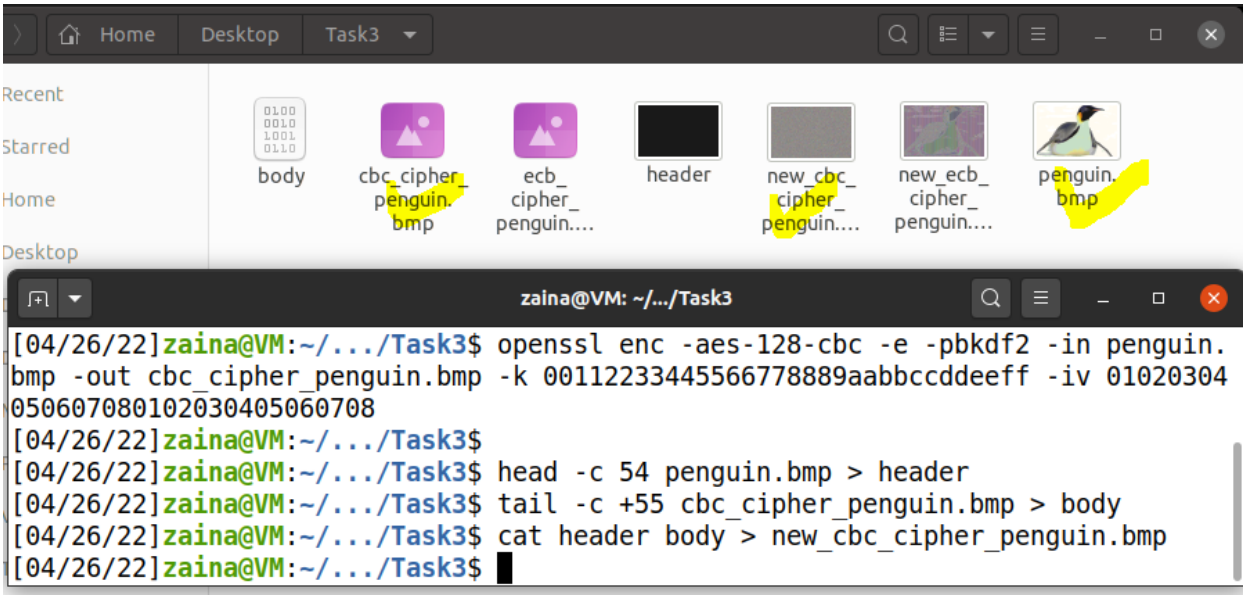


Figure: penguin.bmp encrypted using ECB mode

From the output picture, we note that the encrypted .bmp picture look like the original picture. This is because the ECB mode divide the original file into blocks each with 128 bit. While encrypting the ECB encrypt each block separately. That's means, none of the blocks can affect any other blacks. Therefore, the blocks that have the same color in the original file will be decrypted to the same color in the encrypted file.

Step2: encrypt the penguin.bmp using CBC mode:

```
$openssl enc -aes-128-cbc -e -pbkdf2 -in penguin.bmp -out
cbc_cipher_penguin.bmp -k 00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$head -c 54 penguin.bmp > header
$tail -c +55 cbc_cipher_penguin.bmp > body
$cat header body > new_cbc_cipher_penguin.bmp
```



The encrypted picture in a photo viewer program is shown below. The encrypted picture does not reveal anything about the original one. The reason is that in CBC mode each block affects the encryption in the following block by XORing the previous cipher with the next block then encrypt the result. Therefore, even if the original blocks is the same the encrypted ones will not be the same.

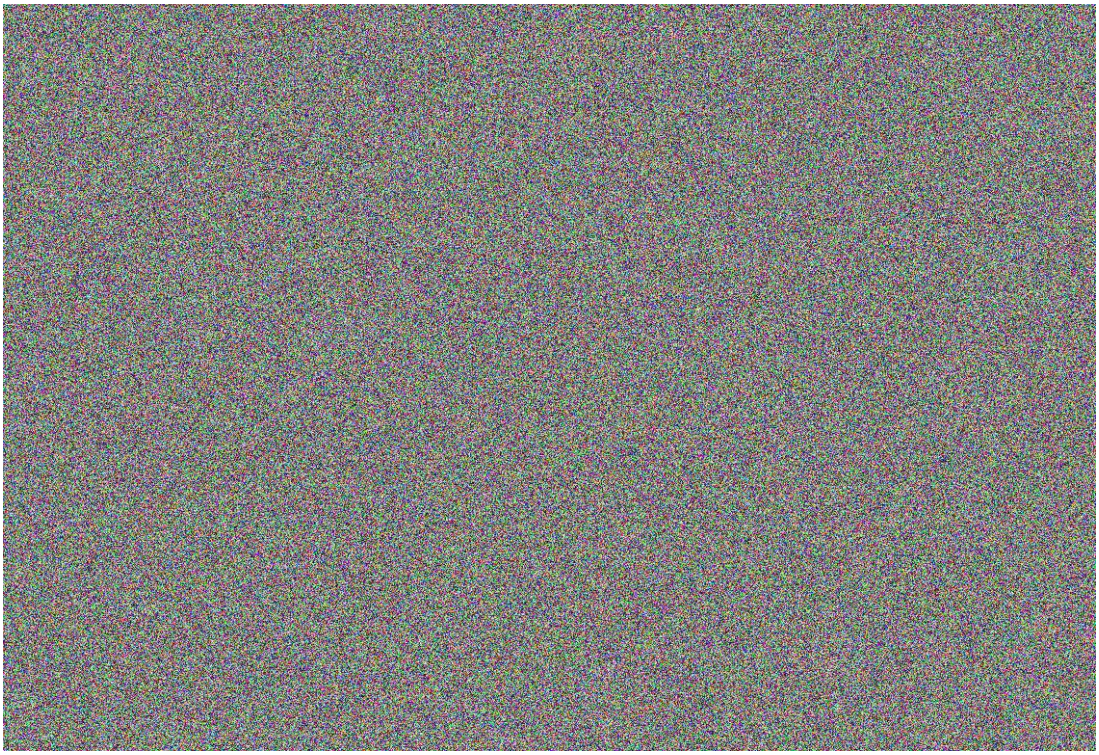


Figure: penguin.bmp encrypted using CBC mode

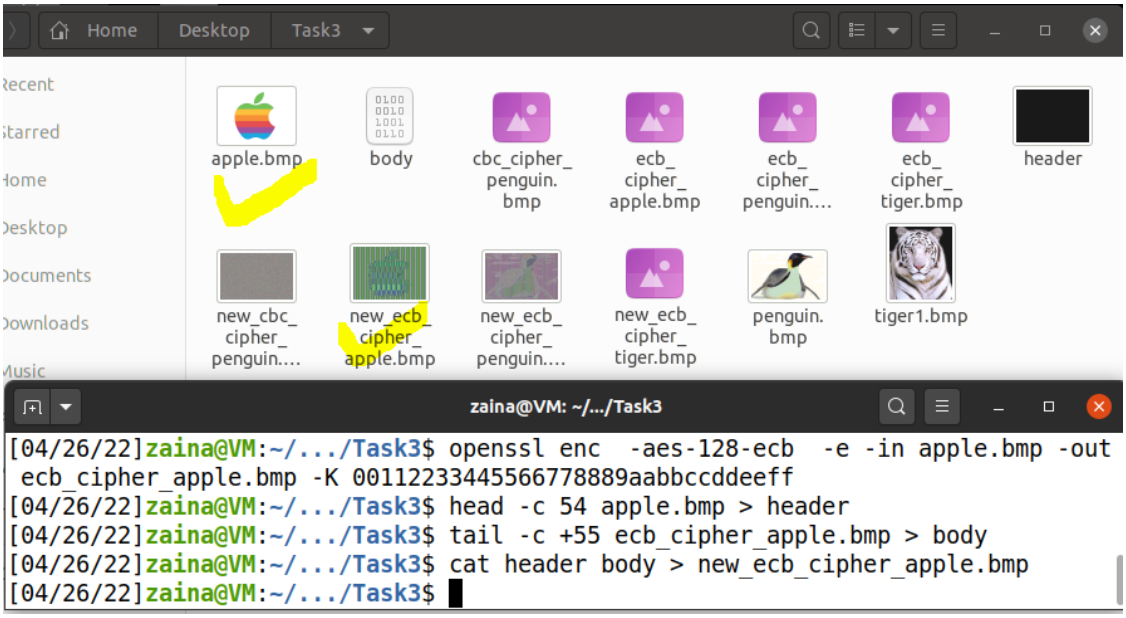
Additional task:

We repeat the same steps for other .bmp picture

Figure: apple.bmp

ECB encryption:

```
$openssl enc  -aes-128-ecb  -e -in apple.bmp -out
ecb_cipher_apple.bmp -K 00112233445566778889aabbccddeeff
$ head -c 54 apple.bmp > header
$ tail -c +55 ecb_cipher_apple.bmp > body
$ cat header body > new_ecb_cipher_apple.bmp
```



The output:

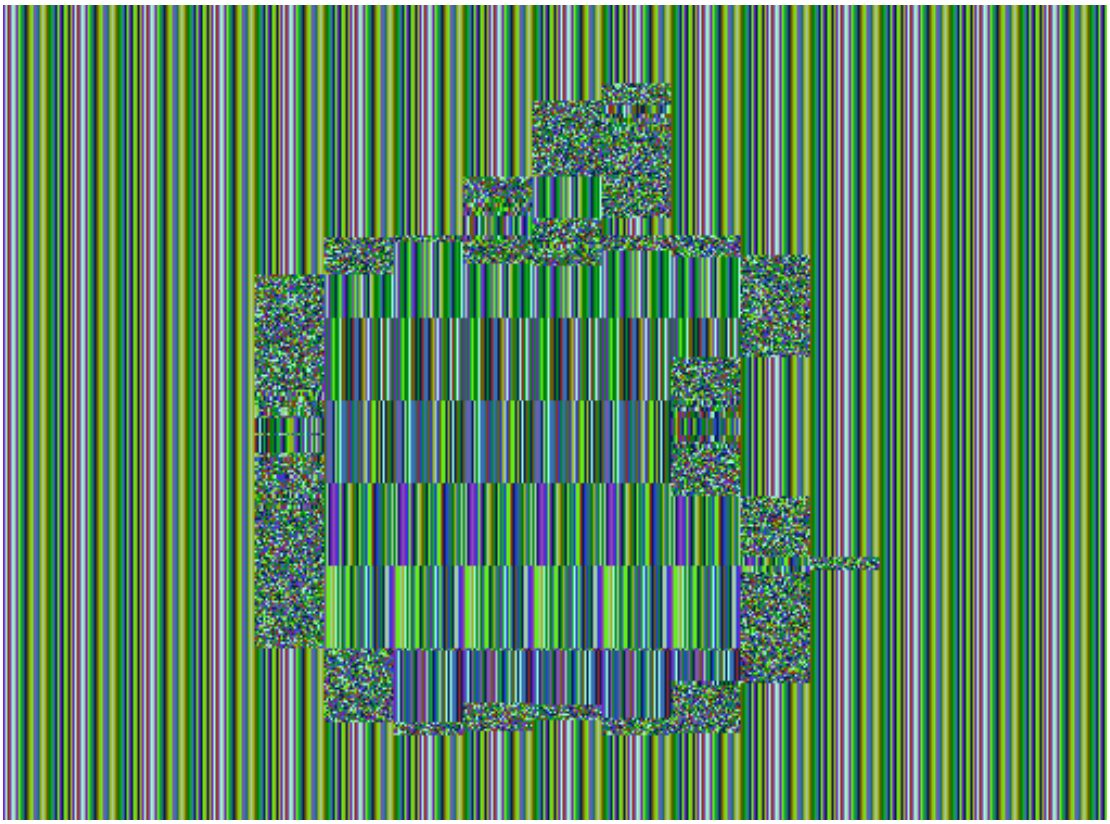
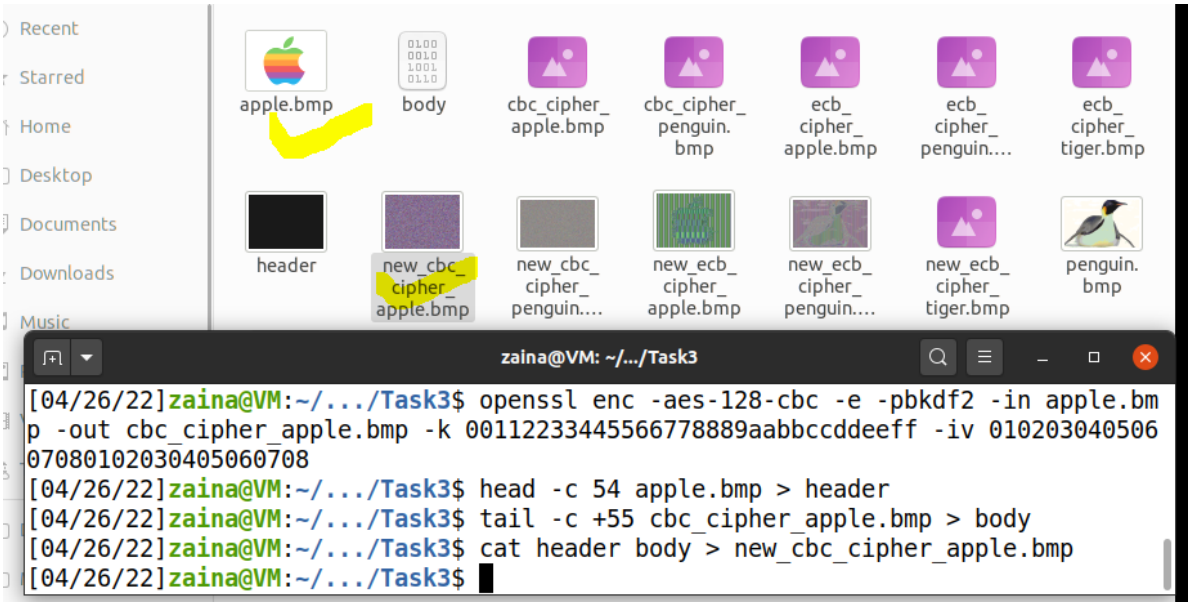


Figure: apple.bmp encrypted using ECB mode

CBC Encryption:

```
$openssl enc -aes-128-cbc -e -pbkdf2 -in apple.bmp -out
cbc_cipher_apple.bmp -k 00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$head -c 54 apple.bmp > header
$tail -c +55 cbc_cipher_apple.bmp > body
$cat header body > new_cbc_cipher_apple.bmp
```



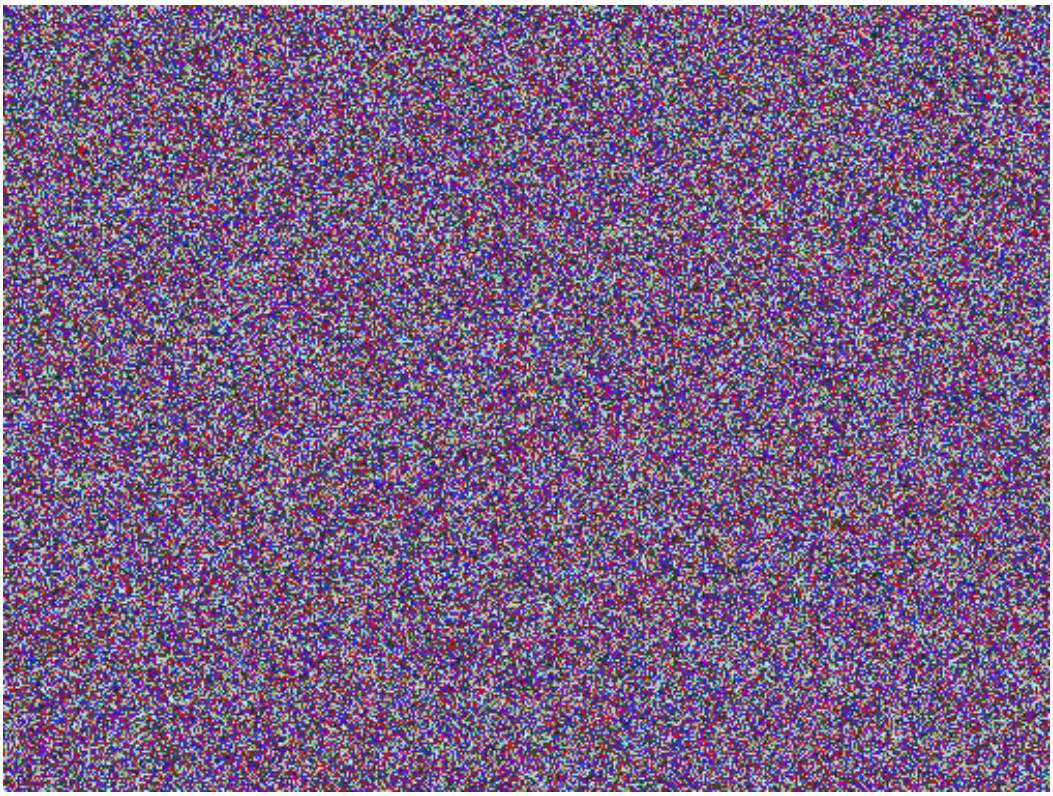


Figure: apple.bmp encrypted using CBC mode

Task 4: Padding

Step1: create a 5 bytes file using echo command, then encrypt it with ECB, CBC, OFB, and OFB

1) Create the file and measure its size using ls command:

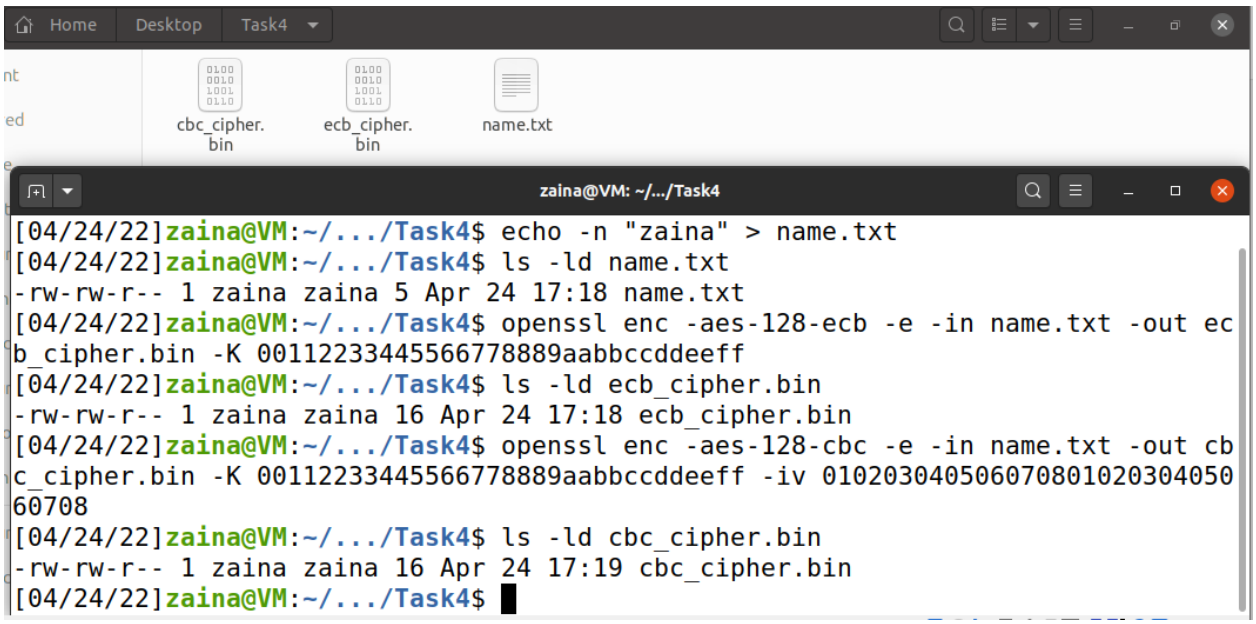
```
$ echo -n "zaina" > name.txt
$ ls -ld name.txt
-rw-rw-r-- 1 zaina zaina 5 Apr 24 17:18 name.txt
```

2) Encrypt the file using ECB and measure the encrypted file size

```
$ openssl enc -aes-128-ecb -e -in name.txt -out ecb_cipher.bin -K
00112233445566778889aabbccddeeff
$ ls -ld ecb_cipher.bin
-rw-rw-r-- 1 zaina zaina 16 Apr 24 17:18 ecb_cipher.bin
```

3) Encrypt the file using CBC and measure the encrypted file size

```
$ openssl enc -aes-128-cbc -e -in name.txt -out cbc_cipher.bin -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$ ls -ld cbc_cipher.bin
-rw-rw-r-- 1 zaina zaina 16 Apr 24 17:19 cbc_cipher.bin
```



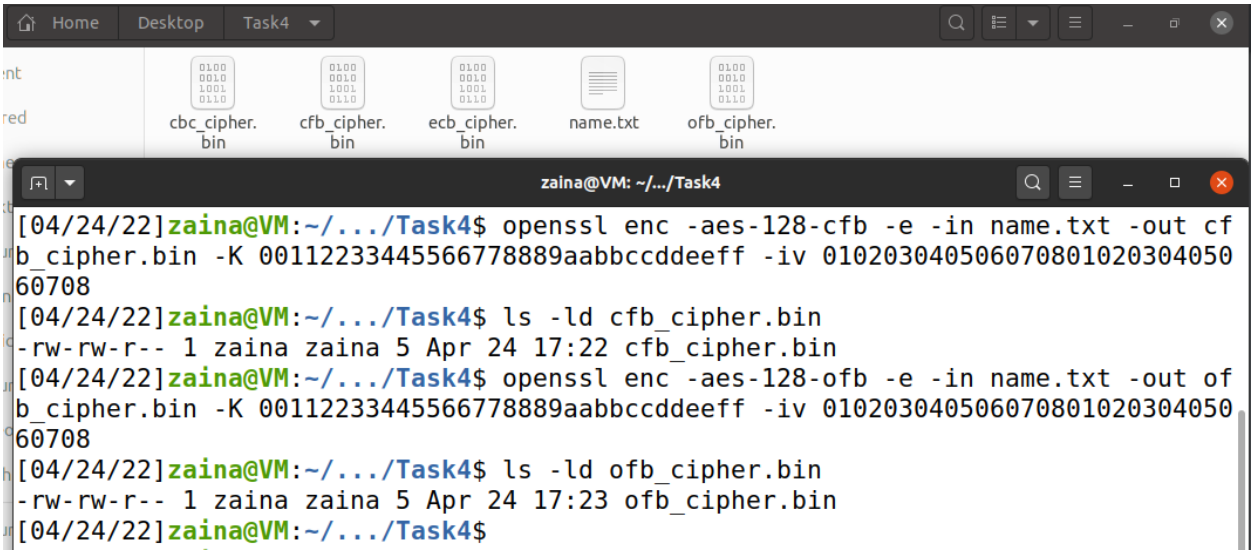
We note that ECB and CBC modes do padding by increasing the file size to become multiple of 16 bytes. In our case, the original file is 5 bytes. The encrypted file bacame one block (16 byte).

4) Encrypt the file using CFB and measure the encrypted file size

```
$ openssl enc -aes-128-cfb -e -in name.txt -out cfb_cipher.bin -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$ ls -ld cfb_cipher.bin
-rw-rw-r-- 1 zaina zaina 5 Apr 24 17:22 cfb_cipher.bin
```

5) Encrypt the file using OFB and measure the encrypted file size

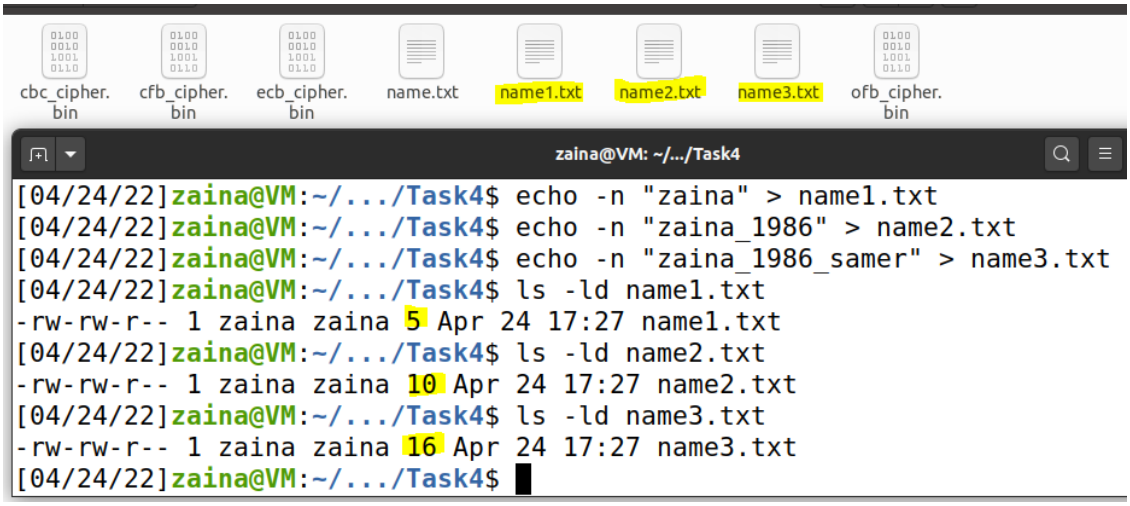
```
$ openssl enc -aes-128-ofb -e -in name.txt -out ofb_cipher.bin -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$ ls -ld ofb_cipher.bin
-rw-rw-r-- 1 zaina zaina 5 Apr 24 17:23 ofb_cipher.bin
```



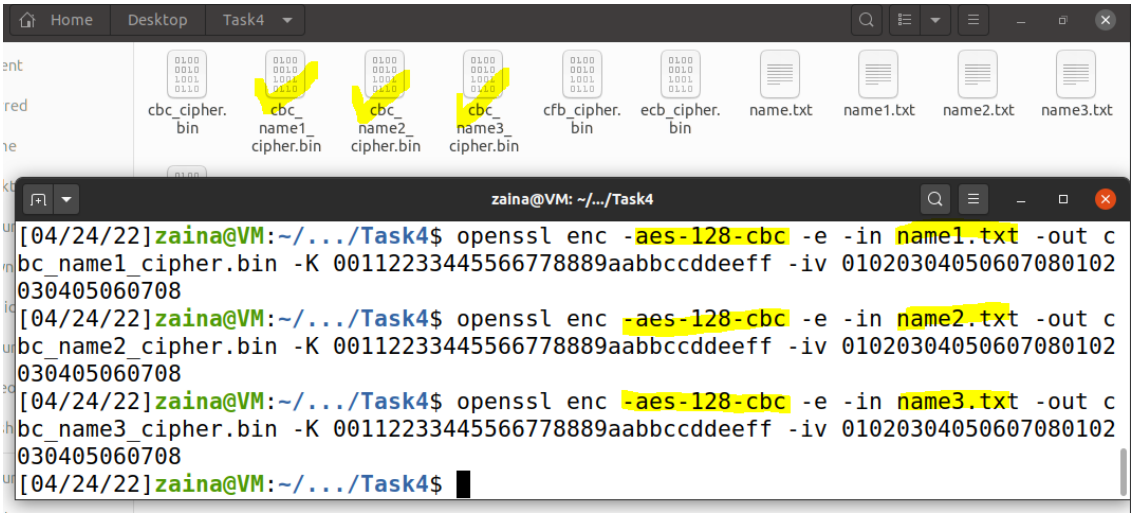
We note that CFB and OFB modes do not pad the original file. In our case, the original file is 5 bytes. The encrypted file remained the same size (5 byte) for both encryption modes.

Step2: create three files 5 bytes 10 bytes, 16 bytes using echo command, then encrypt them with CBC command

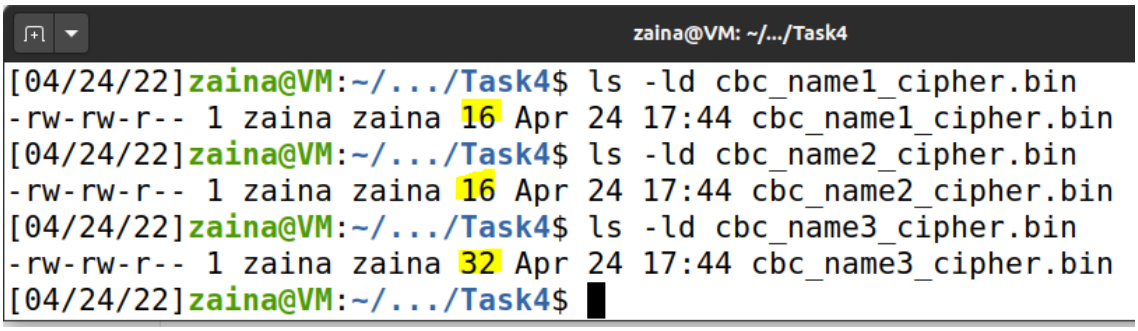
1) Create the required files and measure the size of each file using ls command.



2) Encrypt the three files using CBC mode



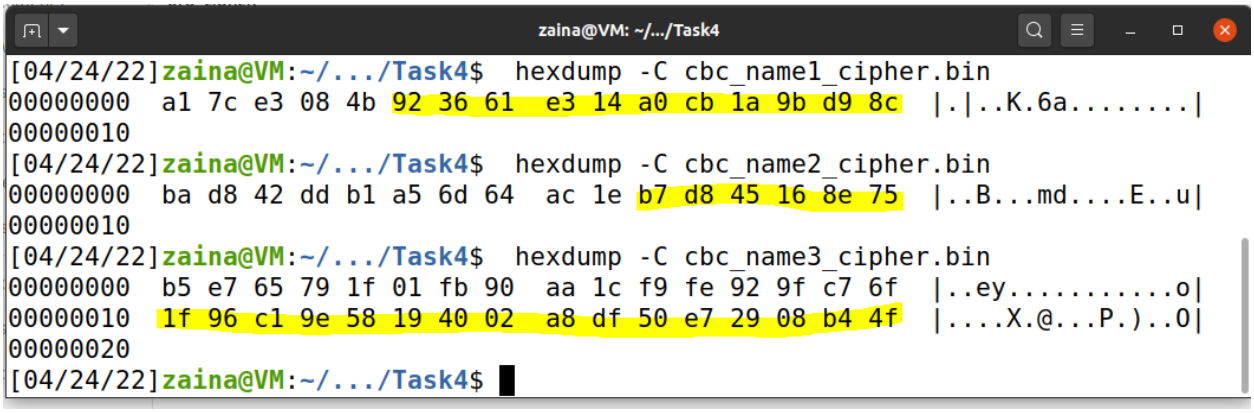
3) Measure the size of the encrypted files



Result: name3.txt was 16 bytes and becomes 32 bytes after padding (2 blocks).
Justification: This practice is used to prevent the decryption software mistakenly treat the last identical bytes in the first block as the padding data. Therefore, in PKCS# 5, if the input length is already an exact multiple of the block size B, then B bytes of value B will be added as the padding

4) View the encrypted files (the marked bytes are the pad data after encryption)

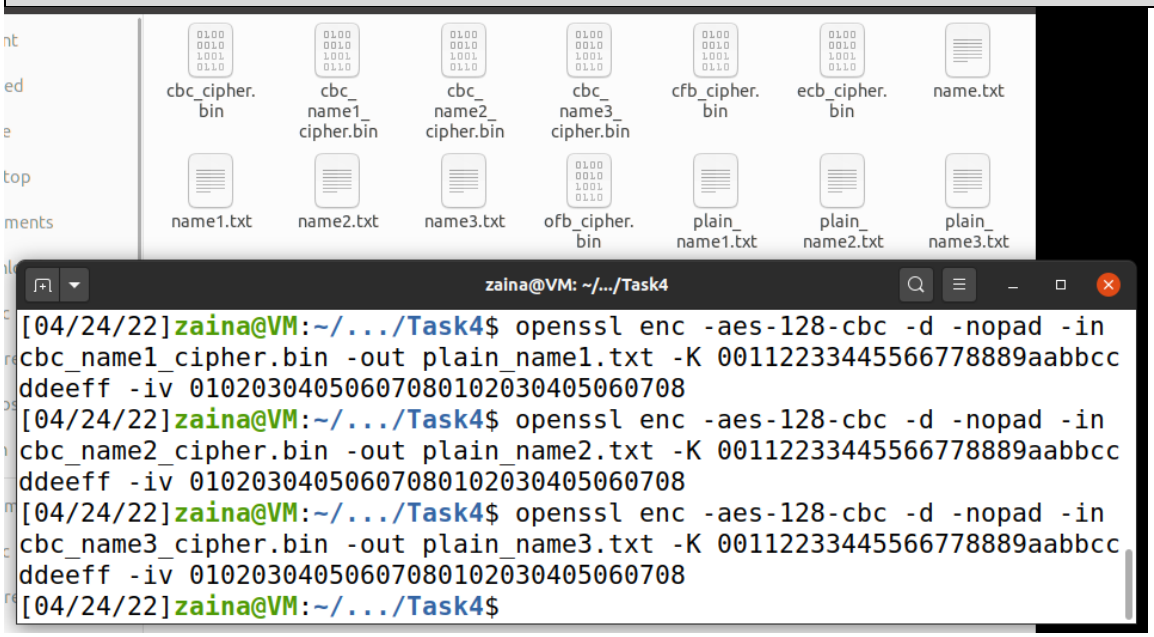
Method1: using ls command



Method 2: using Bless hex editor:

5) Decrypt files with –nopad option the find the value of the added bytes

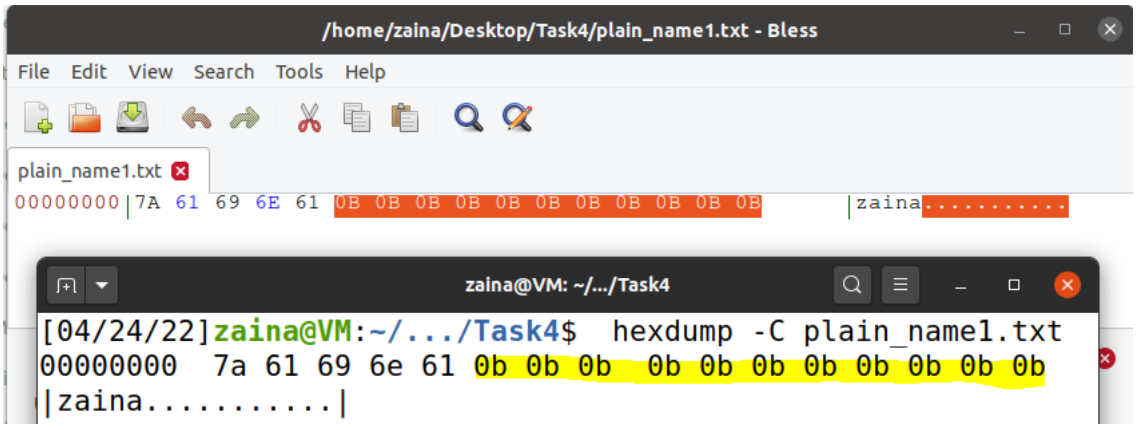
```
$ openssl enc -aes-128-cbc -d -nopad -in cbc_name1_cipher.bin -out plain_name1.txt -K 00112233445566778889aabbccddeeff -iv 01020304050607080102030405060708
$ openssl enc -aes-128-cbc -d -nopad -in cbc_name2_cipher.bin -out plain_name2.txt -K 00112233445566778889aabbccddeeff -iv 01020304050607080102030405060708
$ openssl enc -aes-128-cbc -d -nopad -in cbc_name3_cipher.bin -out plain_name3.txt -K 00112233445566778889aabbccddeeff -iv 01020304050607080102030405060708
```



6) View the values that was added while padding using hexump command or Bless hex editor.

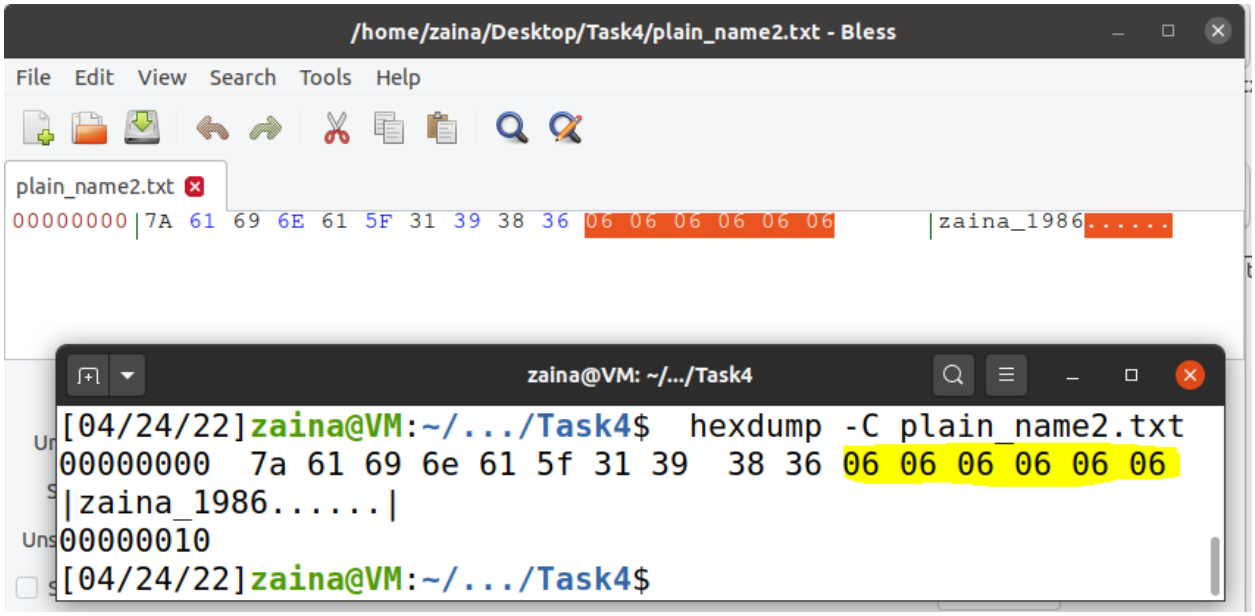
➔ View hex data for plain_name1.txt

```
$ hexdump -C plain_name1.txt
```



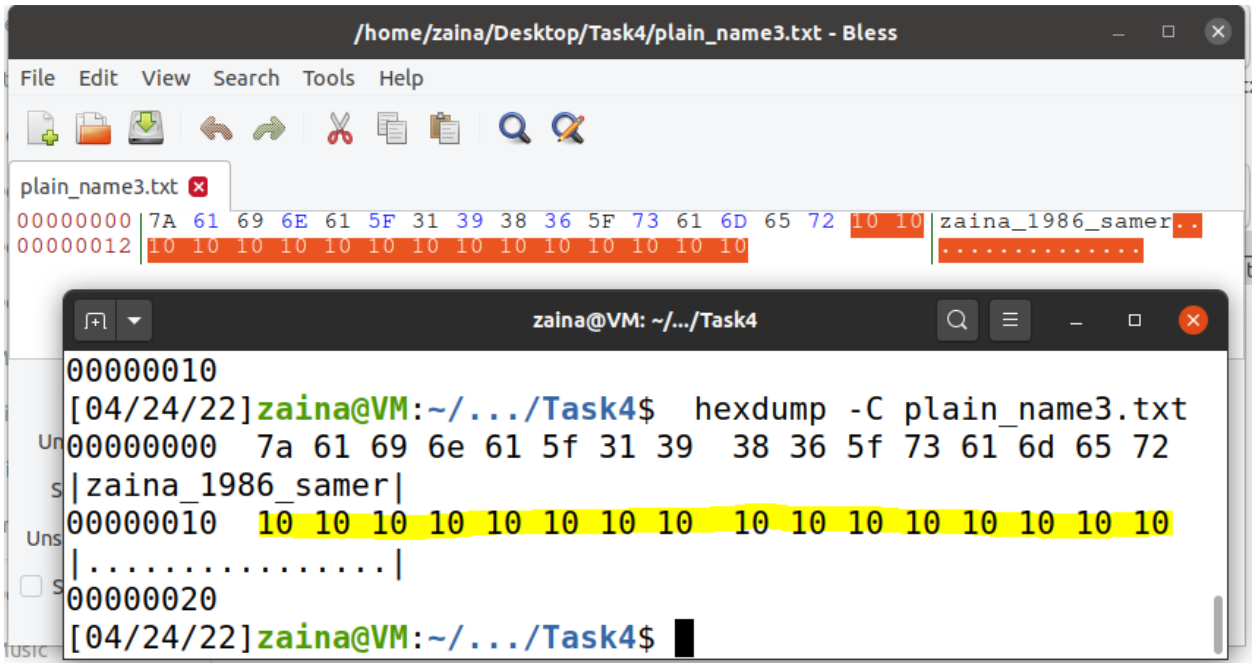
➔ View hex data for plain_name2.txt

```
$ hexdump -C plain_name2.txt
```



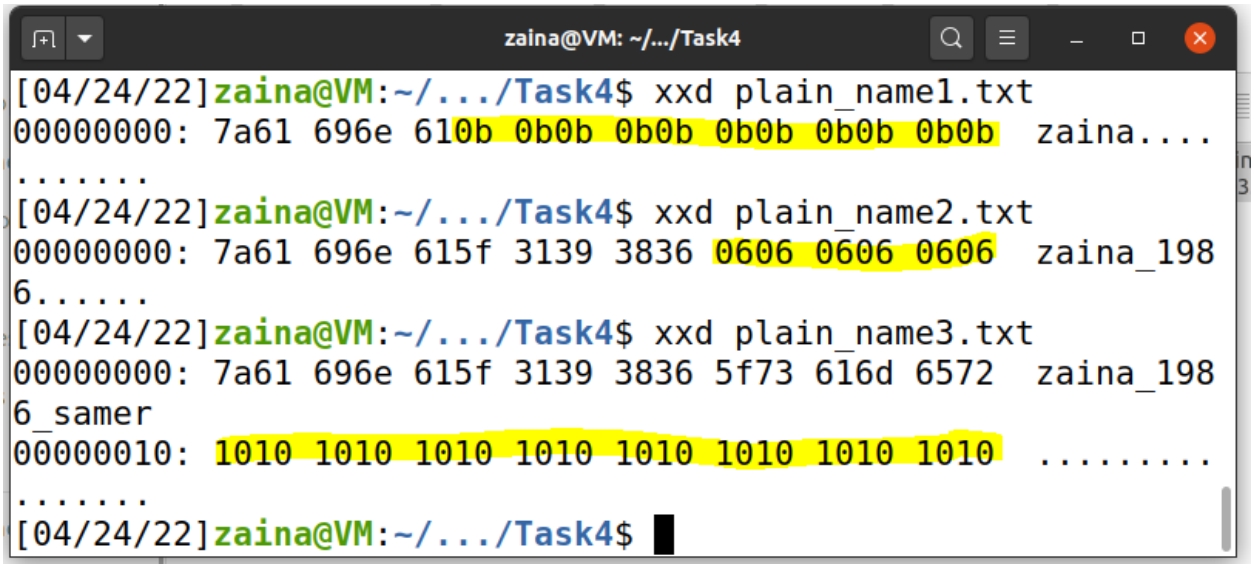
View hex data for plain_name3.txt

```
$ hexdump -C plain_name3.txt
```



Another way to view hex data map using xxd command which view each 2 bytes together

```
$ xxd plain_name1.txt
$ xxd plain_name2.txt
$ xxd plain_name3.txt
```

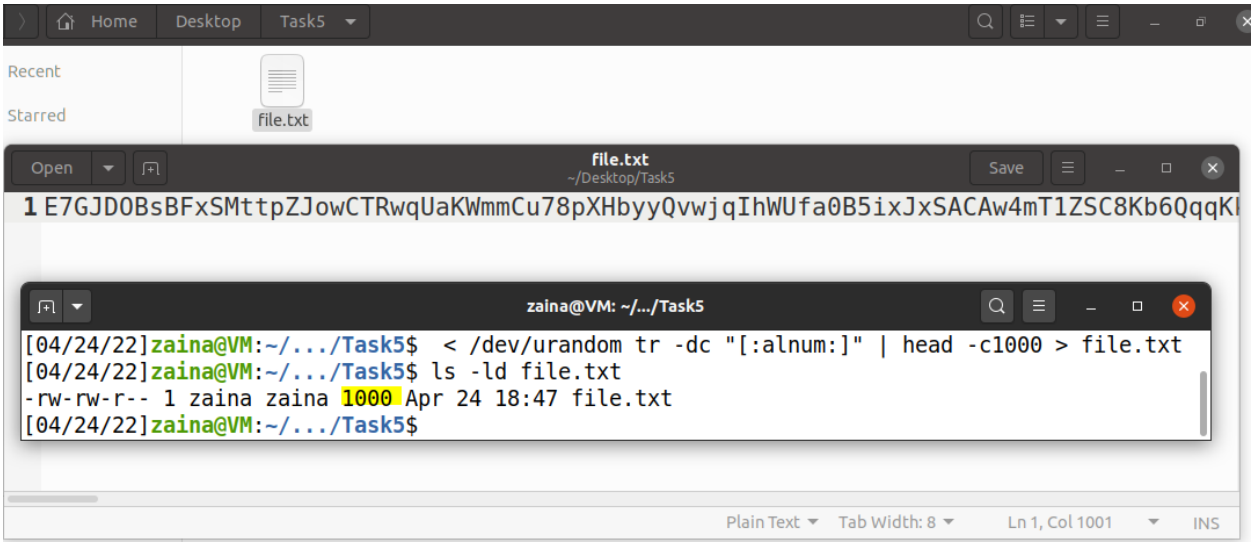


We note that the added data depends on the original file size. The padding schemes that is used with CBC is PKCS#5 and PKCS#7. The difference between them is the allowed block size where PKCS#7 allows 255 block size. Basically, in PKCS# 5, if the block size is B and the last block has K bytes, then B- K bytes of value B-K will be added as the padding.

Task 5: Error Propagation – Corrupted Cipher Text

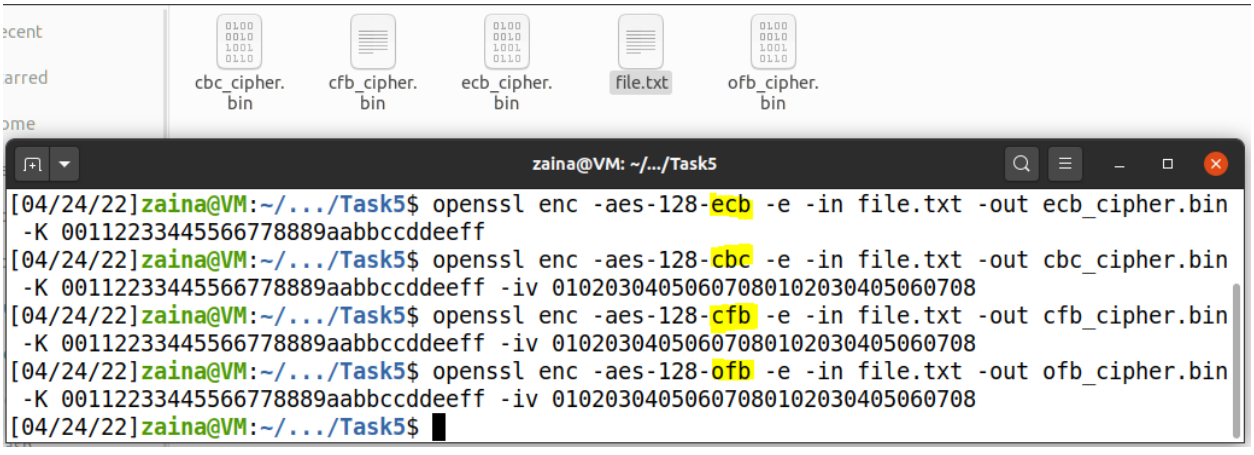
Step1: Create 1000 byte file with random characters then measure its size using ls command.

```
$ < /dev/urandom tr –dc “[ :alnum: ]” | head -c1000 file.txt
$ -ls –ld file.txt
```

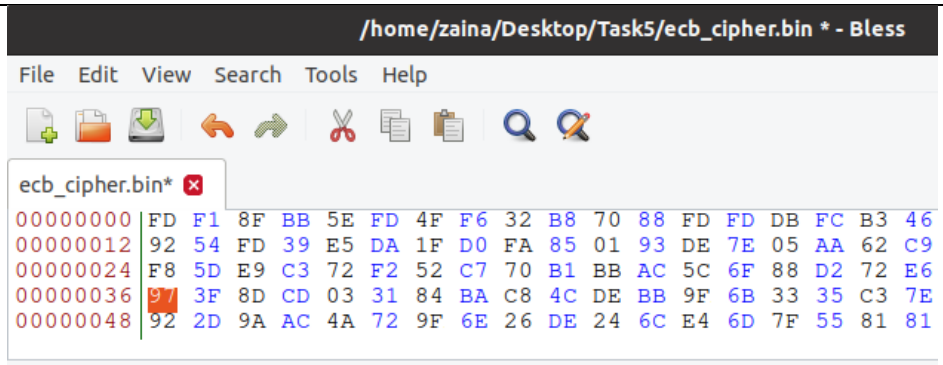
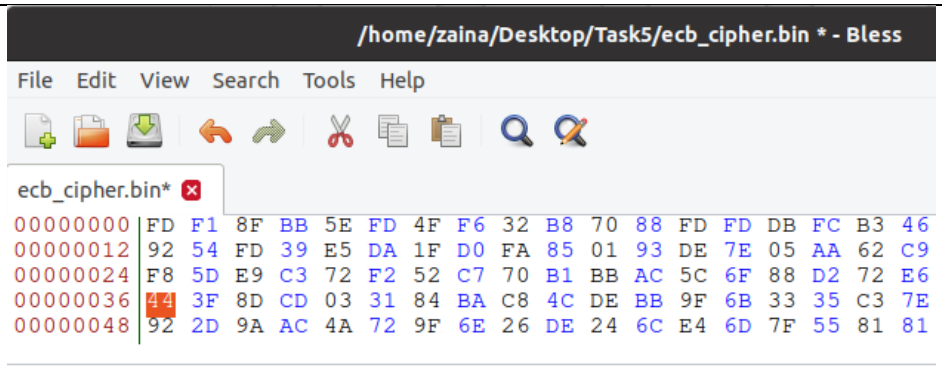
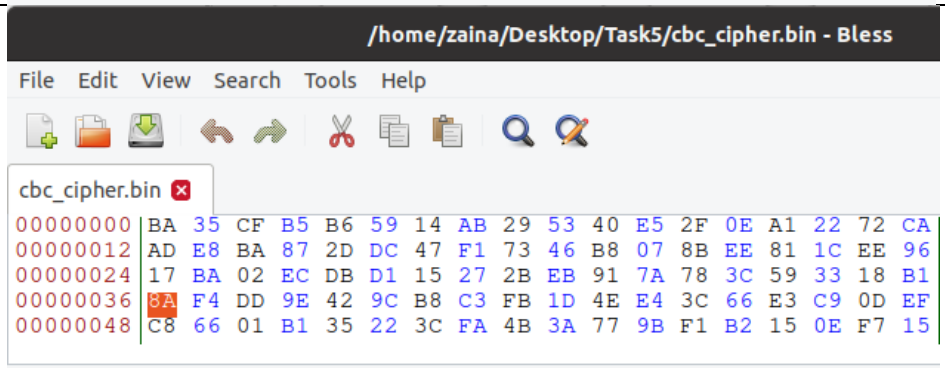
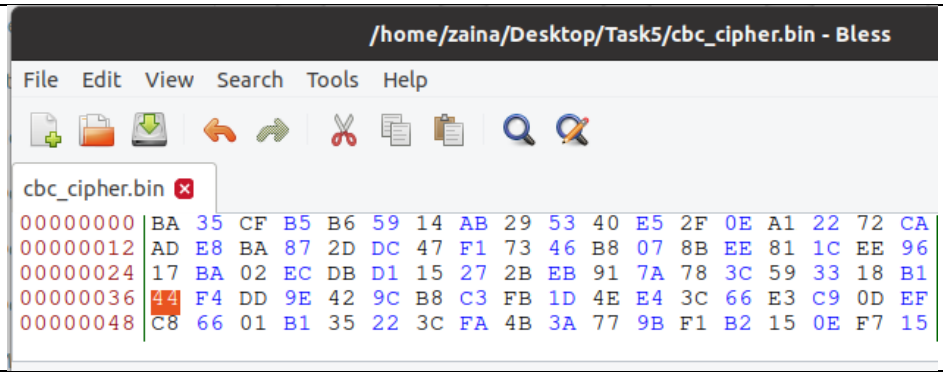


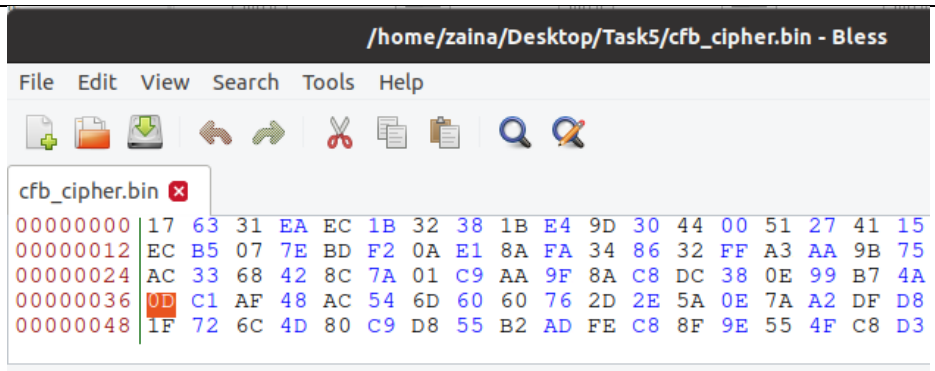
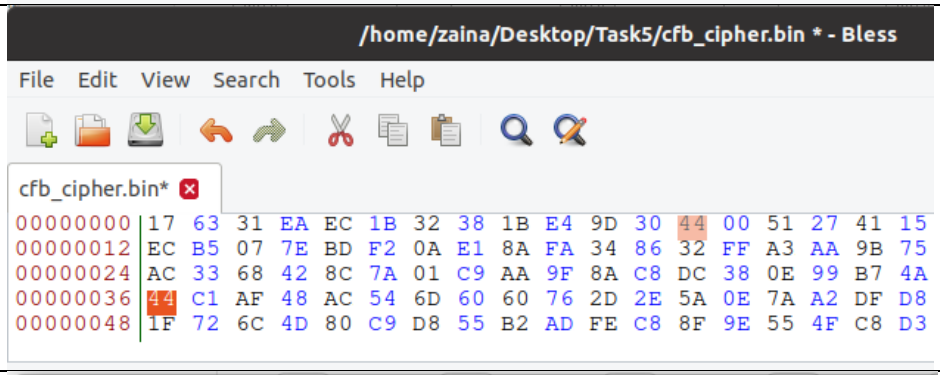
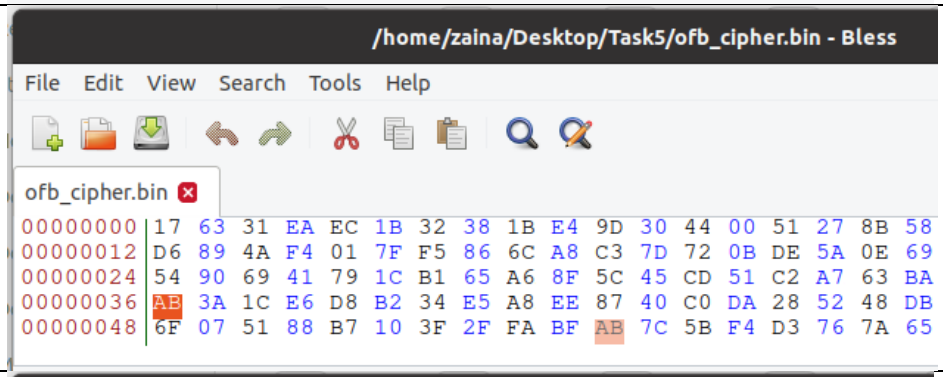
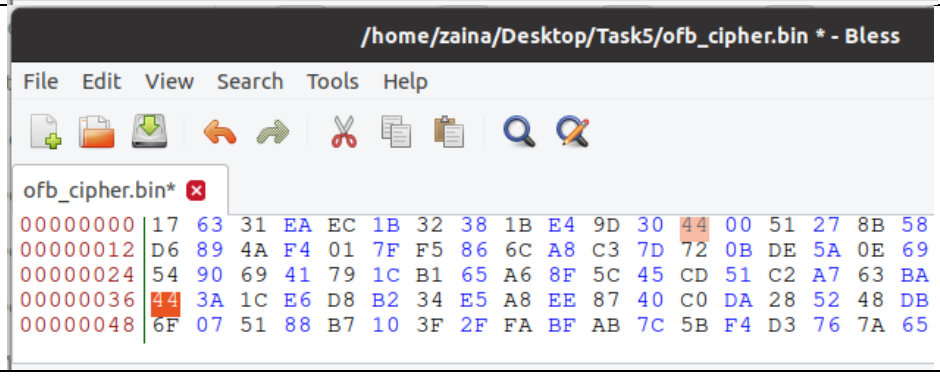
Step2: Encrypt file.txt using ECB, CBC, CFB, and OFB modes

```
$ openssl enc -aes-128-ecb -e -in file.txt -out ecb_cipher.bin -K
00112233445566778889aabbccddeeff
$ openssl enc -aes-128-cbc -e -in file.txt -out cbc_cipher.bin -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$ openssl enc -aes-128-cfb -e -in file.txt -out cfb_cipher.bin -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$ openssl enc -aes-128-ofb -e -in file.txt -out ofb_cipher.bin -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
```



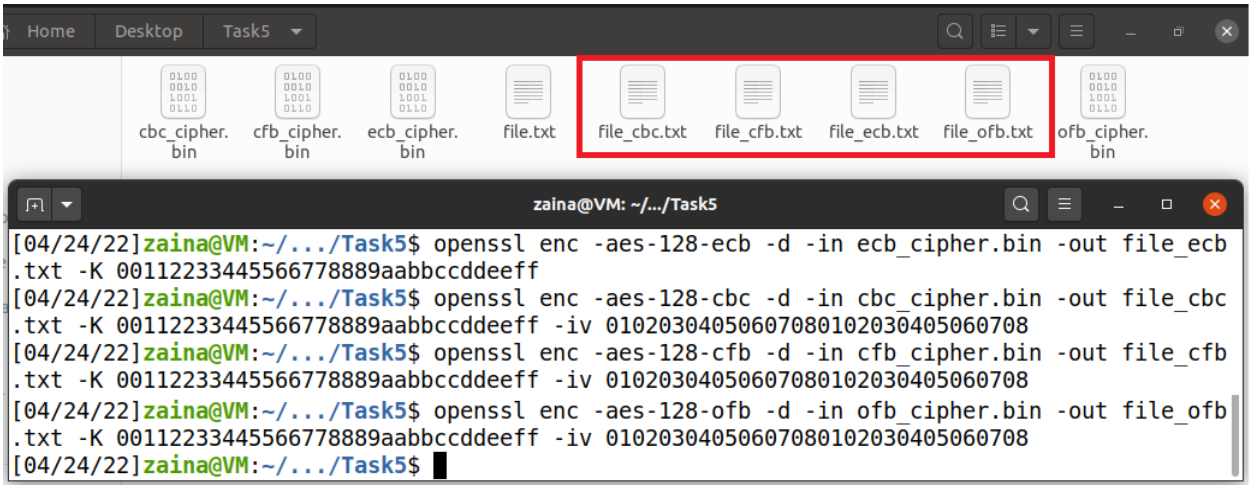
Step3: Corrupt the encrypted files using Bless Hex editor by changing the value of the 55th byte to 44

ECB	Before corruption	
	After corruption	
CBC	Before corruption	
	After corruption	

CFB	Before corruption	
	After corruption	
OFB	Before corruption	
	After corruption	

Step4: Decrypt the corrupted files

```
$ openssl enc -aes-128-ecb -d -in ecb_cipher.bin -out file_ecb.txt -K
00112233445566778889aabbccddeeff
$ openssl enc -aes-128-cbc -d -in cbc_cipher.bin -out file_cbc.txt -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$ openssl enc -aes-128-cfb -d -in cfb_cipher.bin -out file_cfb.txt -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$ openssl enc -aes-128-ofb -d -in ofb_cipher.bin -out file_ofb.txt -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
```

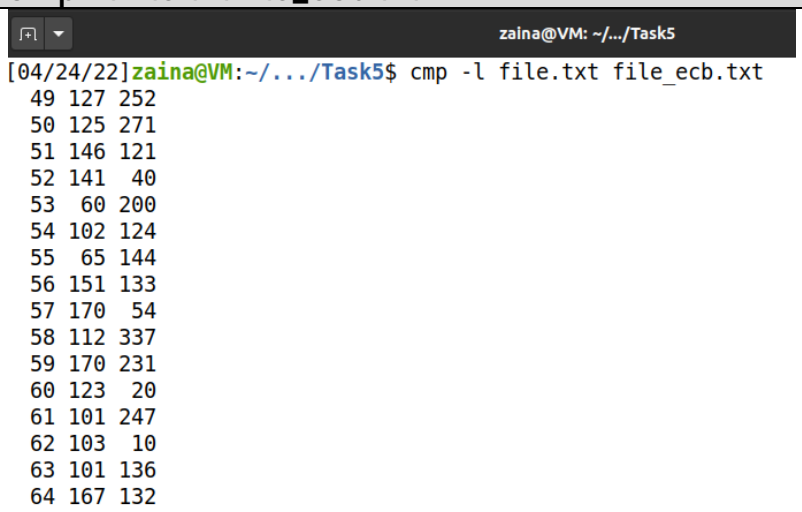


Step 5: compare the decrypted files and the original file using cmp command

The purpose of this step is to find out the effect of corrupting one byte from the encrypted file on the decryption process. Cmp command view the different bytes as 3 columns: the first column is the byte number, the second column is the value from the first file, and the third column is the byte value from the second file. We compared the decrypted file with the original file.

1) ECB Mode

```
$ cmp -l file.txt file_ecb.txt
```



In ECB, the cmp command indicates that there are **16 different** bytes between the original file and decrypted file. The corrupted bytes start from the 49th byte until the 64th byte.

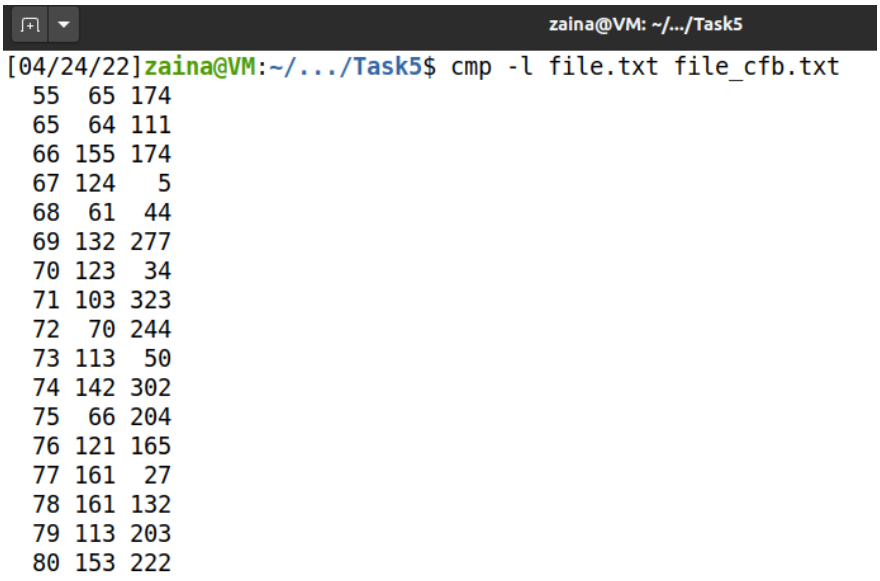
2) CBC mode:

```
$ cmp -l file.txt file_cbc.txt
```

In CBC, the cmp command indicates that there are **17 different** bytes between the original file and decrypted file. The corrupted bytes start from the 49th byte until the 64th byte and the 71st byte.

3) CFB Mode

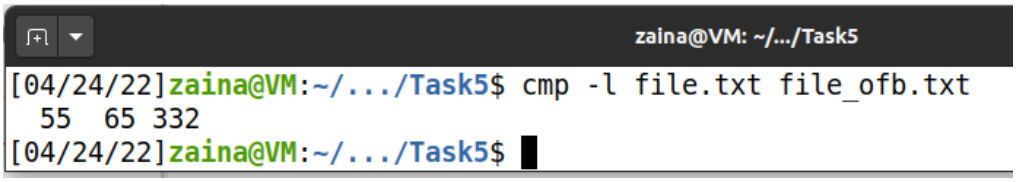
```
$ cmp -l file.txt file_cfb.txt
```



In CFB, the cmp command indicates that there are **17 different** bytes between the original file and decrypted file. The corrupted bytes start from the 55th byte until the 80th byte.

4) OFB Mode

```
$ cmp -l file.txt file_ofb.txt
```



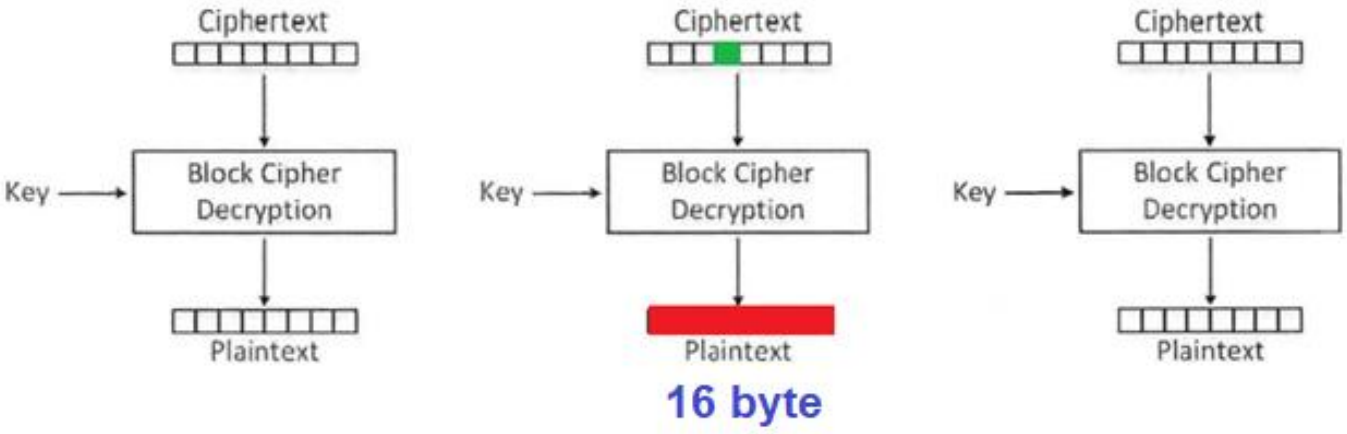
In OFB, the cmp command indicates that there is only **one different** byte between the original file and decrypted file.

Summery:

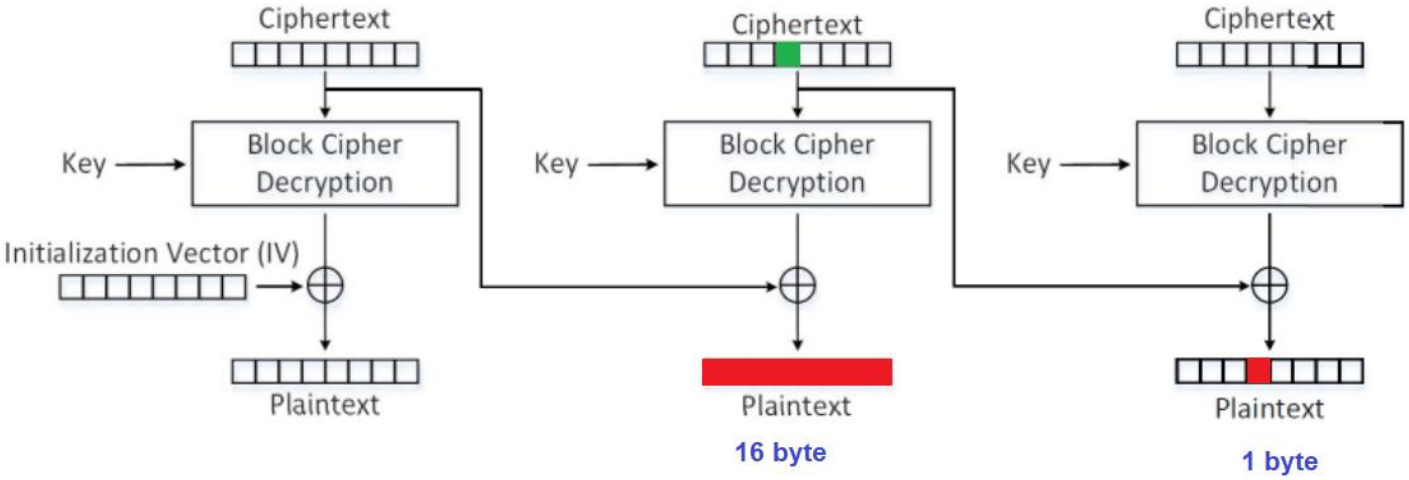
	ECB	CBC	CFB	OFB
different bytes count	16 bytes	17 bytes	17 bytes	One byte
different bytes Numbers	49th →64th	49th →64th + 71st	55th → 80th	55th

Justification: (green → corrupted byte , red → affected bytes after decryption)

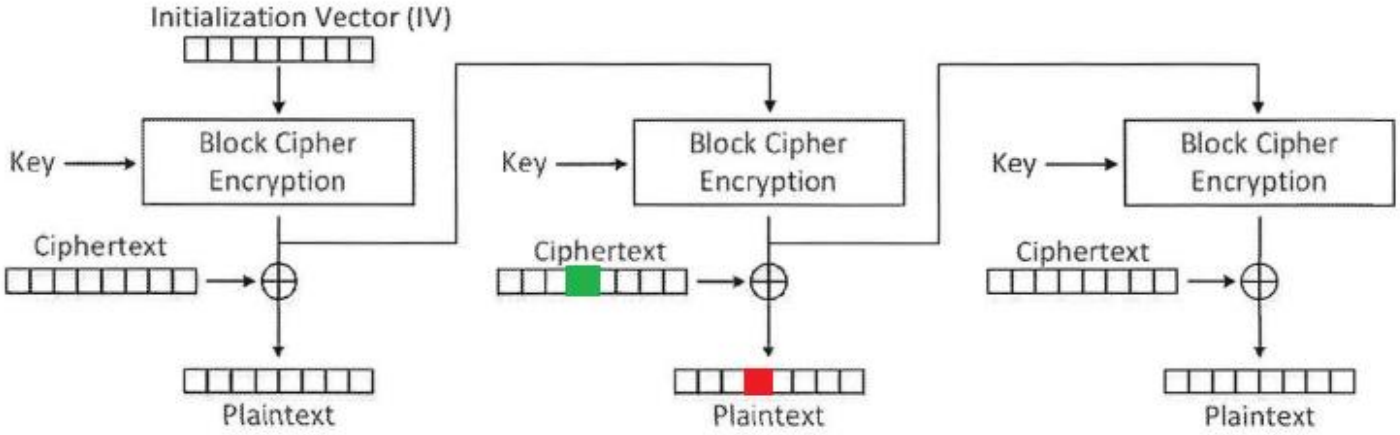
In **ECB** mode, only one block is affected after decryption when any problem in a cipher text happens. The reason is that each block is decrypted independently. However, the corrupted byte in cipher text spread across all the bytes in plaintext resultant block (16 byte). In ECB there is no chaining, thus no error propagation. In our case, the corrupted block starts from the 49th byte and ends with the 64th byte as shown in figure.



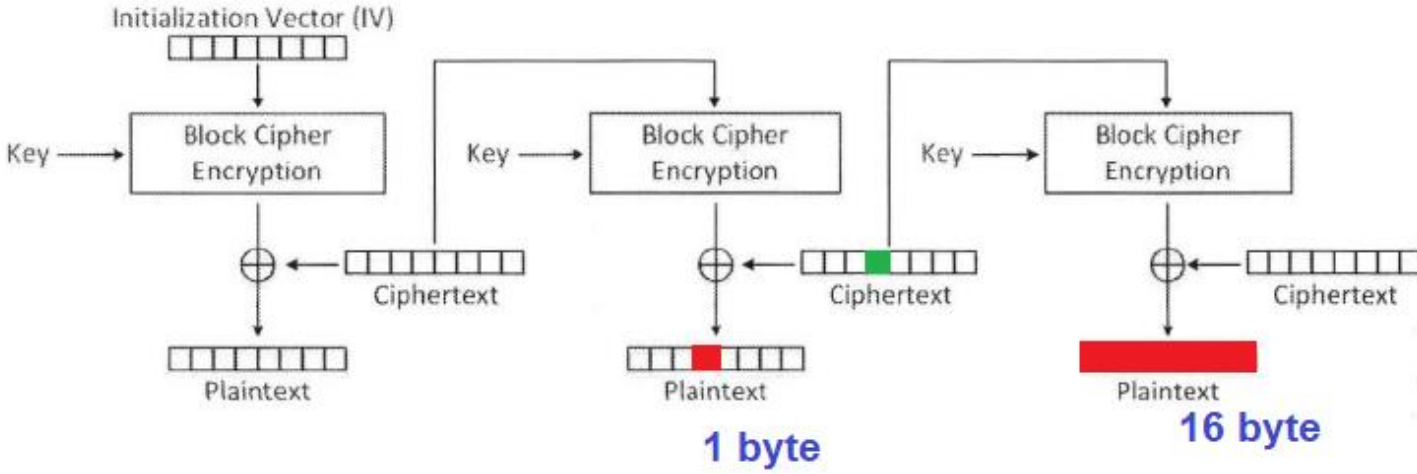
In **CBC** mode, similar to ECB the whole block will be affected and the 71st byte in the next will be also affected because of the xor operation with opposite byte in the next plaintext block as shown in figure. In general, for error propagation in CBC, a single bit error on a byte on a cipher block may change the corresponding byte in the next plaintext block, and changes the current plaintext significantly



OFB mode is the least mode that affected by the error propagation because in the decryption, the cipher block is only xored with the corresponding plaintext and does not interfere in any other previous or next blocks. So only one byte will be affected while decryption by the xor operation as shown in figure.



In **CFB**, a single byte error on the ciphertext may change the corresponding bit on the current plaintext , and changes the next plaintext significantly. See figure



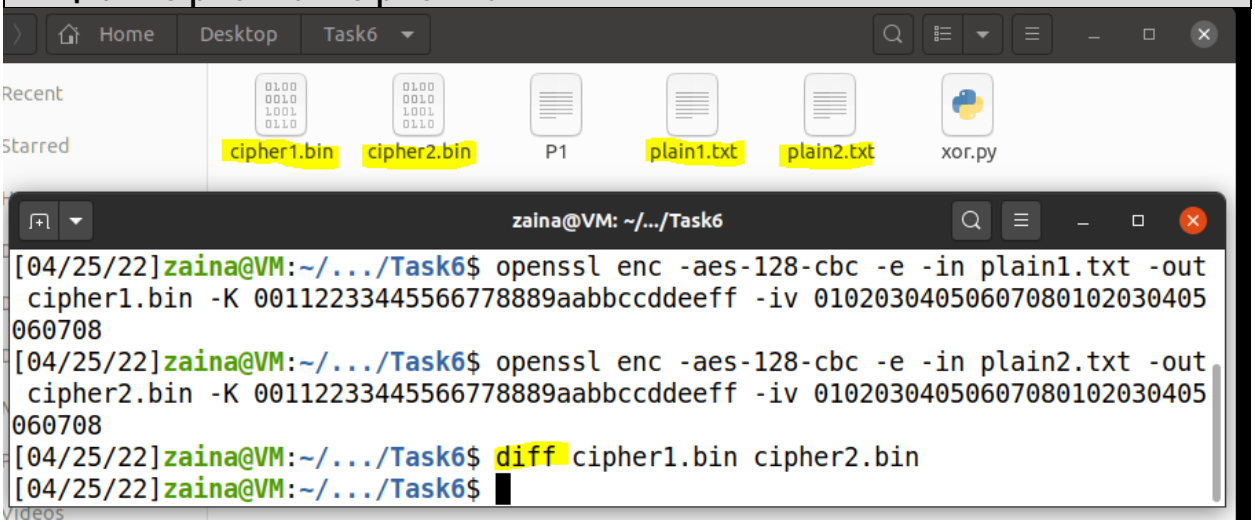
Data corruption errors propagate in each encryption modes differently. In OFB mode only the corrupted byte is lost, while in other modes the entire block is lost, or the proceeding for following blocks are affected as well. The differences are due to how XOR and Initialization Vectors are implemented in each encryption.

Task 6 : Initial Vector (IV) and Common Mistakes

Task 6.1 : IV Experiment

1) Encryption of two identical text files using the same iv then compare the output ciphers

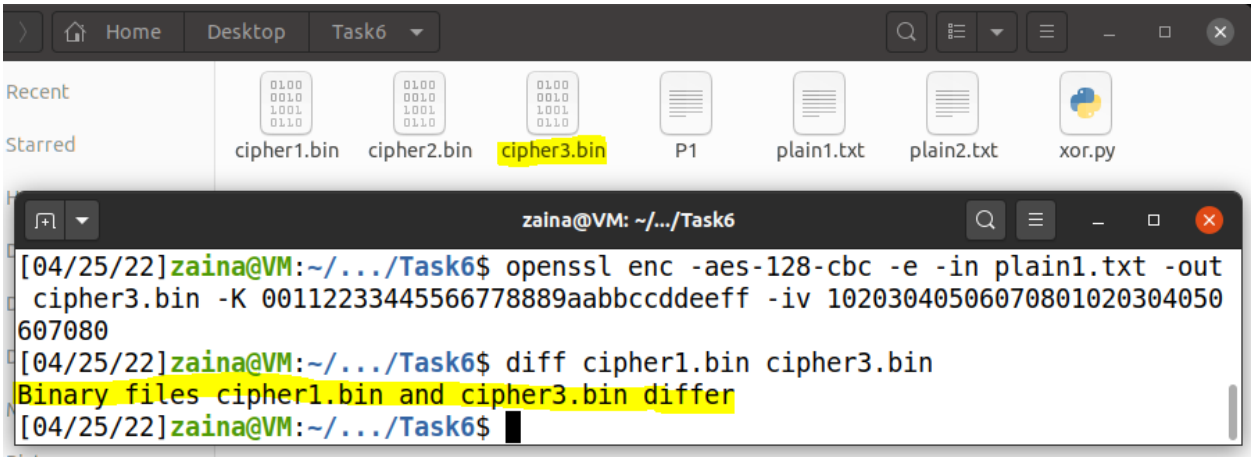
```
$ openssl enc -aes-128-cbc -e -in plain1.txt -out cipher1.bin -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$ openssl enc -aes-128-cbc -e -in plain2.txt -out cipher2.bin -K
00112233445566778889aabbccddeeff -iv
01020304050607080102030405060708
$ diff cipher1.bin cipher2.bin
```



Result: since diff did not show any thing then cpher1 and cipher3 are identical. This may lead to information leak. One may argue that hi s/her plaintext will never repeat, so it is safe to use the same IV. While this may be true for some encryption modes, it is dangerous for other encryption modes such as OFB as we will see in the next tasks.

2) Encryption of two identical text files using different iv values then compare the output ciphers

```
$ openssl enc -aes-128-cbc -e -in plain1.txt -out cipher3.bin -K
00112233445566778889aabbccddeeff -iv
10203040506070801020304050607080
$ diff cipher1.bin cipher3.bin
```



Result: when encrypting two identical files with different iv values, the resultant ciphers are completely different. Thus, each separate file should be encrypted using different iv value.

Task 6.2. Common Mistake: Use the Same IV

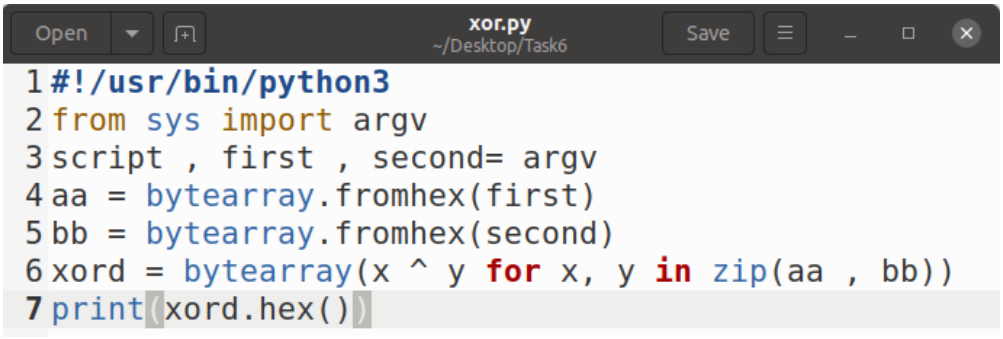
1) Create the known plain text and save it into P1 then Convert the data to hex strings

```
$ echo -n "This is a known message!" >P1
$ xxd -p P1
546869732069732061206b6e6f776e206d65737361676521
```

2) Write a python script that perform xor operation between two arguments The python script is provided in the SEED book chapter 21.5 page 487

```
#!/usr/bin/python3
from sys import argv
script, first, second = argv
aa = bytearray.fromhex(first)
bb = bytearray.fromhex(second)
xord = bytearray(x^y for x,y in zip(aa, bb))
print(xord.hex())
```

We write the previous script and save it as xor.py file

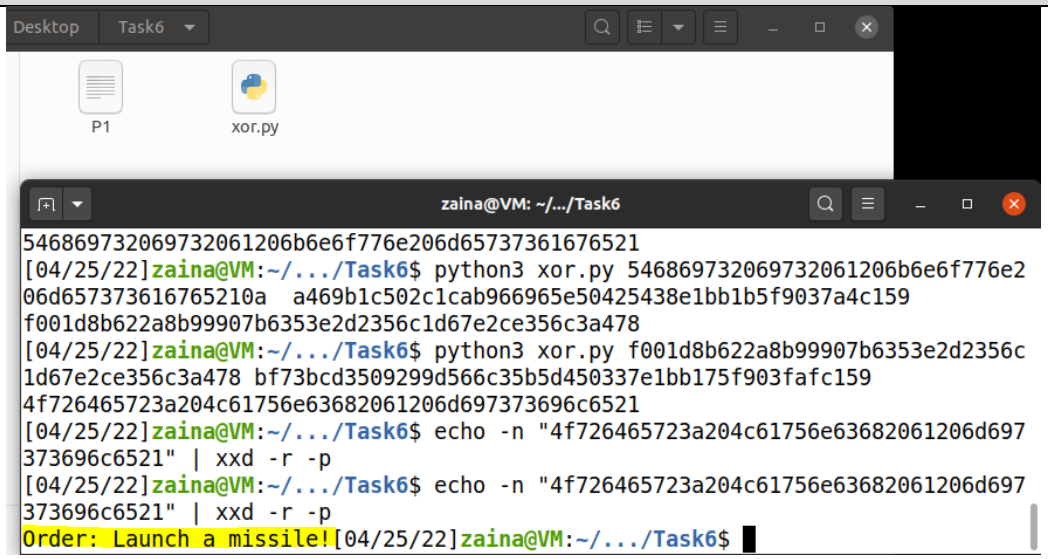


3) According to the known-plaintext attack model, if P1 is disclosed to an attacker (who also knows CI and C2), nothing about P2 will be disclosed. This is not true anymore if P1 and P2 are encrypted with the same IV: by XORing PI and CI , we will get the output stream ; further XORing the output stream with C2 will produce P2.

```
# p1 xor c1
$ python3 xor.py
546869732069732061206b6e6f776e206d657373616765210a
a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478

# the output from previous xor c2
$ python3 xor.py
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478
bf73bcd3509299d566c35b5d450337e1bb175f903fafc159
4f726465723a204c61756e63682061206d697373696c6521
```

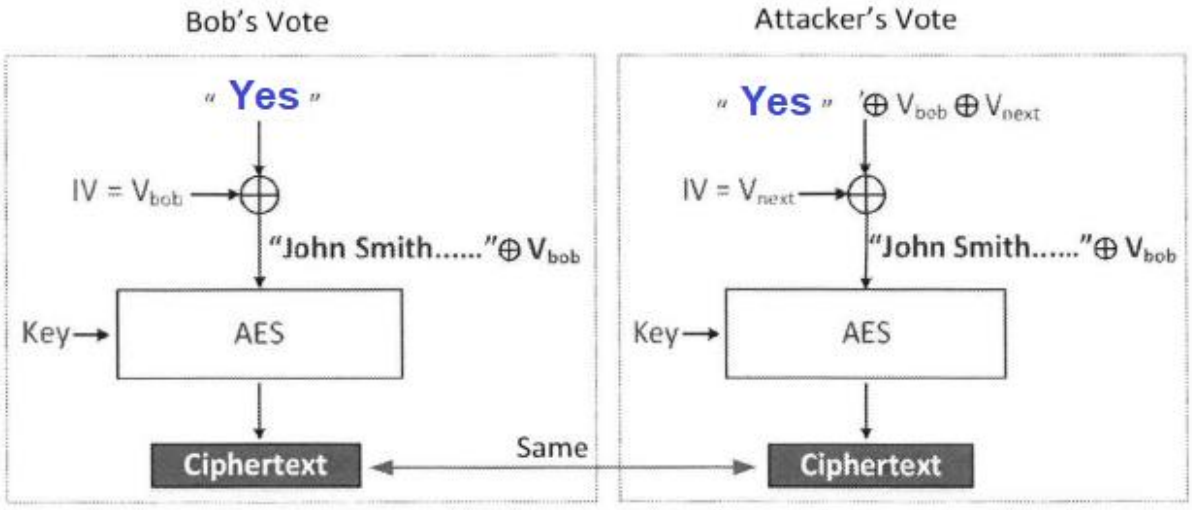
4) Convert the hex string to ASCII string

```
$ echo -n
"4f726465723a204c61756e63682061206d697373696c6521" | xxd -r
-p
Order: Launch a missile!
```



Task 6.3. Common Mistake: Use a Predictable IV

To solve this part, we rely on the principle how CBC mode operate. When IV is predictable a simple xor operation can reveal the secret message (vote) as shown in figure



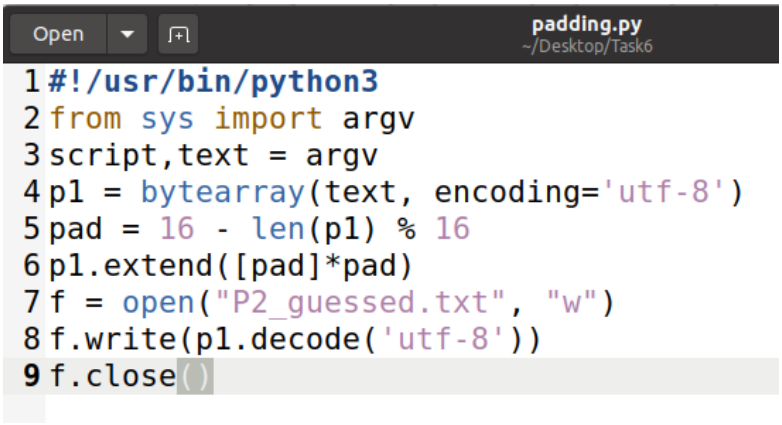
First, we request oracle encryption service using Netcat command

```
$ nc 10.9.0.80 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: fdd735b3c01f6265931b0a558e0ac004
The IV used    : 4f0b16e6815e1908b0a7382d985757f1

Next IV        : 71788dfb815e1908b0a7382d985757f1
Your plaintext :
```

```
[04/27/22]zaina@VM:~/.../Labsetup$ nc 10.9.0.80 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: fdd735b3c01f6265931b0a558e0ac004
The IV used    : 4f0b16e6815e1908b0a7382d985757f1

Next IV        : 71788dfb815e1908b0a7382d985757f1
Your plaintext :
```

Result: oracle encryption provides Bobs cipher, the used IV and the next IV which is a predictable value. Our task is to reveal Bob's plaintext.

1) Guess Bob's answer either "Yes" or "No": we choose "Yes"
2) Pad the guessed plaintext to become 16 byte: we write python script padding.py and run it with the guessed value "Yes". The padded message saved in P2_guessed.txt file.




```
$ python3 padding.py "Yes"
```

3) View the padded message as hex value

```
$ xxd -p P2_guessed.txt
5965730d0d0d0d0d0d0d0d0d0d0d0d0d
```

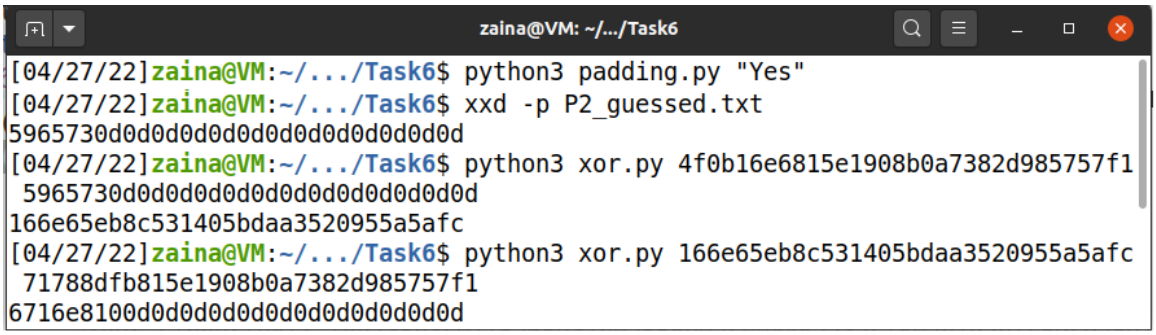
4) XORing the guessed message with Bobs used IV

```
$ python3 xor.py 4f0b16e6815e1908b0a7382d985757f1
5965730d0d0d0d0d0d0d0d0d0d0d0d0d
166e65eb8c531405bdaa3520955a5afc
```

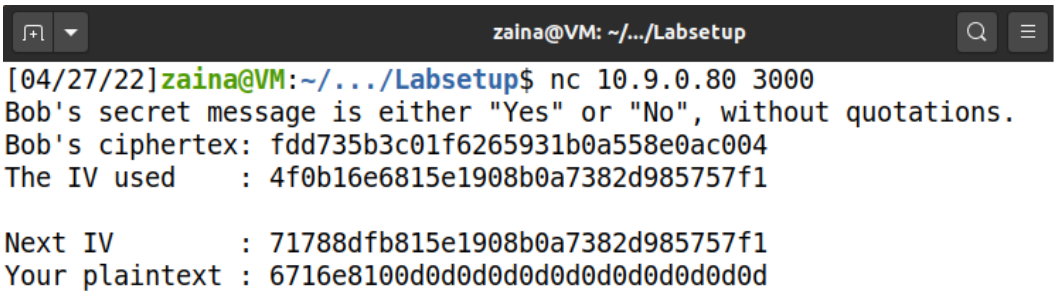
5) XORing the result from the last step with the predicted IV

```
$ python3 xor.py 166e65eb8c531405bdaa3520955a5afc
71788dfb815e1908b0a7382d985757f1
6716e8100d0d0d0d0d0d0d0d0d0d0d0d
```

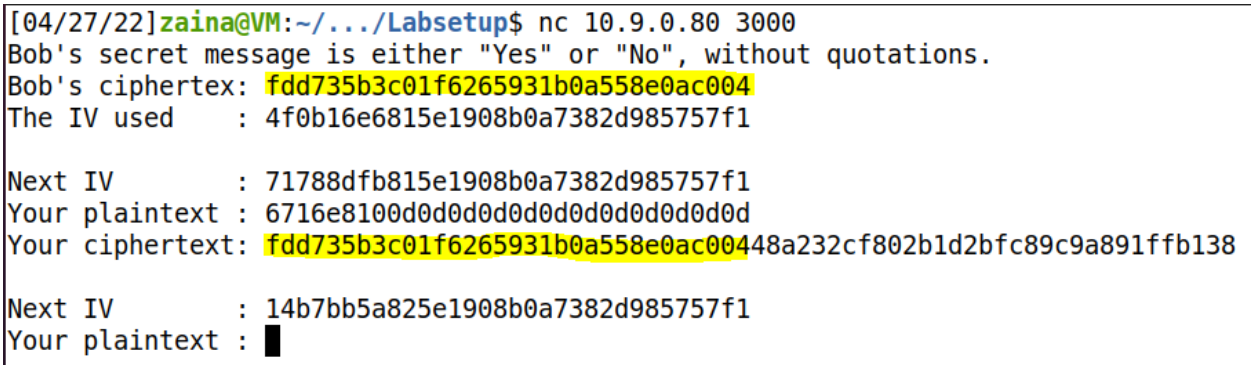
The previous five steps are shown in the following screenshot



6) Insert the output of step 5 to Oracle Encryption as “Your plaintext” and wait the response which will be the encryption of the provided guessed plaintext:



7) Compare the provided cipher with Bobs cipher.



We note that the first block of the provided cipher is the same as Bob’s cipher. This indicates that our guess is correct and Bob’s message is “Yes”. If our guess was “No” then the ciphers will not be the same. To make sure of our guess we will repeat the same steps for “No” guess in the next step.

We also note that the new cipher is 2 blocks because of the padding because we see in task4 that is the message is 16 bytes then the encrypted cipher will be 32 bytes. But in the decryption all the added values will be deleted.

8) Repeat all steps for the “No” guess but we must use the new predictable IV

```
$ python3 padding.py "No"
$ xxd -p P2_guessed.txt
$ python3 xor.py 4e6f0e0e0e0e0e0e0e0e0e0e0e0e0e0e
4f0b16e6815e1908b0a7382d985757f1
016418e88f501706bea93623965959ff
$ python3 xor.py 016418e88f501706bea93623965959ff
14b7bb5a825e1908b0a7382d985757f1
15d3a3b20d0e0e0e0e0e0e0e0e0e0e0e
```

The ciphers are not identical, so Bob’s message is not “No”. As a conclusion, IV should be random and should not be predictable. Otherwise an attacker can exploit this issue and reveal the sniffed message.

End