



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:

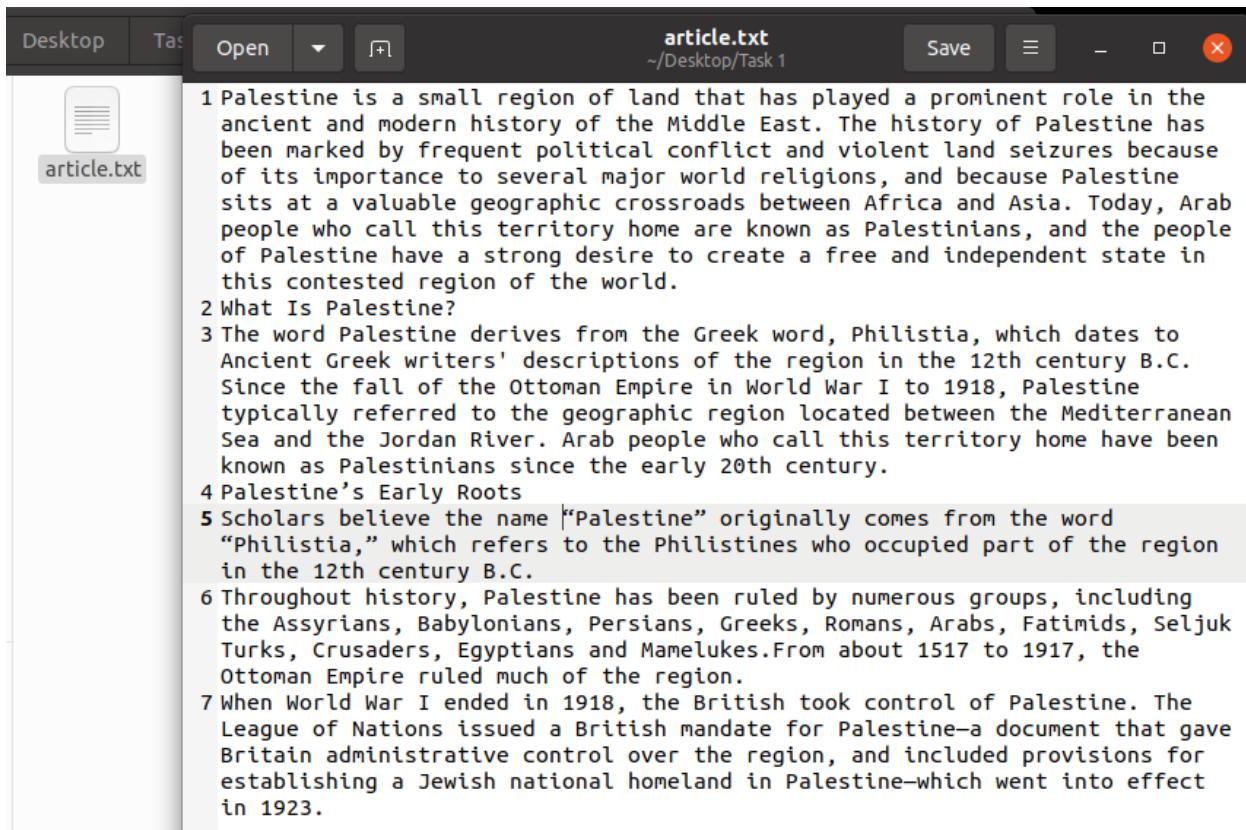
In this task, we will Substitution Cipher where encryption is done by replacing units of plaintext with ciphertext, according to a fixed system. Units may be single letters, pairs of letters, triplets of letters, mixtures of the above, and so forth. Decryption simply performs the inverse substitution.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



```
Desktop Task Open article.txt Save article.txt ~/Desktop/Task 1
1 Palestine is a small region of land that has played a prominent role in the ancient and modern history of the Middle East. The history of Palestine has been marked by frequent political conflict and violent land seizures because of its importance to several major world religions, and because Palestine sits at a valuable geographic crossroads between Africa and Asia. Today, Arab people who call this territory home are known as Palestinians, and the people of Palestine have a strong desire to create a free and independent state in this contested region of the world.
2 What Is Palestine?
3 The word Palestine derives from the Greek word, Philistia, which dates to Ancient Greek writers' descriptions of the region in the 12th century B.C. Since the fall of the Ottoman Empire in World War I to 1918, Palestine typically referred to the geographic region located between the Mediterranean Sea and the Jordan River. Arab people who call this territory home have been known as Palestinians since the early 20th century.
4 Palestine's Early Roots
5 Scholars believe the name "Palestine" originally comes from the word "Philistia," which refers to the Philistines who occupied part of the region in the 12th century B.C.
6 Throughout history, Palestine has been ruled by numerous groups, including the Assyrians, Babylonians, Persians, Greeks, Romans, Arabs, Fatimids, Seljuk Turks, Crusaders, Egyptians and Mamelukes. From about 1517 to 1917, the Ottoman Empire ruled much of the region.
7 When World War I ended in 1918, the British took control of Palestine. The League of Nations issued a British mandate for Palestine—a document that gave Britain administrative control over the region, and included provisions for establishing a Jewish national homeland in Palestine—which went into effect in 1923.
```

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

```
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ python3
Python 3.8.10 (default, Mar 15 2022, 12:22:08)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> s = "abcdefghijklmnopqrstuvwxyz"
>>> list = random.sample(s, len(s))
>>> key = ''.join(list)
>>> print(key)
spzxqwukyconeladrtmbjgvifh ← The Encryption Key
>>>
```

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.

```
plaintext.txt ~/Desktop
Open Save - x
1 palestine is a small region of land that has played a prominent role in the ancient and modern history of the middle east the history of palestine has been marked by frequent political conflict and violent of its importance to several major world religions and at a valuable geographic crossroads between africa and who call this territory home are known as palestinians palestine have a strong desire to create a free and in contested region of the world
2 what is palestine
3 the word palestine derives from the greek word philist ancient greek writers descriptions of the region in the fall of the ottoman empire in world war i to pale referred to the geographic region located between the the jordan river arab people who call this territory h palestinians since the early th century
4 palestines early roots
5 scholars believe the name palestine originally comes f which refers to the philistines who occupied part of t century bc
6 throughout history palestine has been ruled by numerous assyrians babylonians persians greeks romans arabs fat crusaders egyptians and mamelukesfrom about to the o much of the region
7 when world war i ended in the british took control of nations issued a british mandate for palestine a doc administrative control over the region and included pr establishing a jewish national homeland in palestinewh
```

```
[04/23/22] zaina@zaina-VirtualBox:~/Desktop$ tr [:upper:] [:lower:] < article.txt > lowercase.txt
[04/23/22] zaina@zaina-VirtualBox:~/Desktop$ tr -cd '[a-z][\n][:space:]' < lowercase.txt > plaintext.txt
[04/23/22] zaina@zaina-VirtualBox:~/Desktop$
```

Step 3: perform monoalphabetic substitution Encryption using tr command

→Encryption:

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

```
Desktop Task 1
article.txt ciphertext.txt
zaina@zaina-VirtualBox: ~... < plaintext.txt > ciphertext.txt
[04/26/22] zaina@zaina-VirtualBox:~/.../Task 1$
```

```
ciphertext.txt ~/Desktop/Task 1
Open Save - x
The file "/home/zaina/Desktop/Task 1/ciphertext.txt" changed on disk. Reload x
1 dsnqmbylq ym s mesnn tquyal aw nslx bksb ksm dnsfqx s dtaeylqlb tanq
    yl bkq slzyqlb slx eaxtql kymbatf aw bkq eyxxnq qsmq bkq kymbatf aw
    dsnqmbylq ksm pqql estoqx pf wtqrjqlb danybyzn zalwnyzb slx gyanqlb
    nslx mqyhjtqm pqzsjmq aw ybm yedatbslzq ba mqgqtsn escat vatnx
    tqnyualm slx pqzsjmq dsnqmbylq mybm sb s gsnjspnq uqautsdkz
    ztammtasxm pqbvqql swtyzs slx smys baxsf stsp dqadnq vka zsnn bkym
    bqttbyatf kaeq stq olavl sm dsnqmblyslm slx bkq dqadnq aw dsnqmbylq
    ksgq s mbtau xqmytq ba ztqsq s wtqq slx ylxqdqlxqlb mbsbq yl bkym
    zalbqmbqx tquyal aw bkq vatnx
2 vksb ym dsnqmbylq
3 bkq vatx dsnqmbylq xqtyggm wtae bkq utqqo vatx dkynymbys vkyzk xsbqm
    ba slzyqlb utqqo vtybqtm xqmzytdbyalm aw bkq tquyal yl bkq bk zqlbjtf
    pz mylzq bkq wsnn aw bkq abbaesl qedytq yl vatnx vst y ba dsnqmbylq
    bfdyzsnnf tqwqttqx ba bkq uqautsdkz tquyal nazsbqx pqbvqql bkq
```

Plain Text Tab Width: 8 Ln 2, Col 18 INS

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.

The screenshot shows a terminal window titled 'freq.py' running on a Linux system. The command 'python3 freq.py' is entered, and the output displays the top 20 frequencies for 1-grams, 2-grams, and 3-grams. The 1-gram output includes characters like 'y' (373), 't' (348), 'n' (488), etc. The 2-gram output includes 'yt' (115), 'tn' (89), 'yu' (78), etc. The 3-gram output includes 'ytn' (78), 'vup' (30), 'mnr' (20), etc.

```
freq.py
zaina@zaina-VirtualBox:~/Desktop$ python3 freq.py
[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          mnr: 20
x: 291          nh: 58          ynh: 18
u: 280          qz: 76          xzy: 16
q: 276          hn: 57          mxu: 14
m: 264          vu: 56          gnq: 14
h: 235          t: 53          ytv: 13
t: 183          l: 52          nqy: 13
l: 166          p: 46          vtl: 13
p: 156          a: 45          bxh: 13
c: 104          c: 45          lvq: 12
z: 95           z: 44          nuy: 12
l: 90            np: 44          vyn: 12
g: 83            vy: 44          uvy: 11
b: 83            nu: 42          lmu: 11
r: 82             qy: 39          nvh: 11
e: 76             vq: 33          cmu: 11
d: 59             vi: 32          tmq: 10
gn: 32            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	
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	bkh			
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.

```
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ ^C
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ tr 'nyvxuqmhtipaczbredfskjow' 'eta
onlisdrcumfpwybkjxqz' < ciphertext.txt > out.txt
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ tr 'nyvxuqmhtipaczbredfskjow' 'eot
hasinrdluymwfgcbpkvjxqz' < ciphertext.txt > out.txt
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ tr 'nyvxuqmhtipaczbredfskjow' 'eta
onlisdrcumfpwybkjxqz' < ciphertext.txt > out.txt
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ tr 'nyvxuqmhtplbrw' 'ETAONLISHDSFGB'
< ciphertext.txt > out.txt
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ tr 'nyvxuqmhtplbrw' 'ETAONLIRHDSFGB'
< ciphertext.txt > out.txt
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ tr 'nyvxuqmhtpazlbrdw' 'ETAONSIRHDCU
SFGYB' < ciphertext.txt > output.txt
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ tr 'nyvxuqmhtpazlbrdw' 'ETAONWIRHDCU
SFGYB' < ciphertext.txt > output.txt
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ tr 'nyvxuqmhtpazlbrdw' 'ETAONSIRHDCU
WFGYB' < ciphertext.txt > output.txt
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ tr 'nyvxuqmhtipazlbgd' 'ETAONSIRHLD
CUWFBGY' < ciphertext.txt > output.txt
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ tr 'nyvxuqmhtipaczbred' 'ETAONSIR
HLDCMUWFGBPYK' < ciphertext.txt > output.txt
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$ tr 'nyvxuqmhtipaczbredfskjow' 'ETA
ONSIRHLCMUWFGBPYVKXJQZ' < ciphertext.txt > output.txt
[04/23/22]zaina@zaina-VirtualBox:~/Desktop$
```

Sample from decoded cipher of the earliest trials.

THE OLaARL TzRN ON LzNDAd SHIaH LEEcL AgOzT RIGHT AFTER THIL iONG LTRANGE ASARDL TRle THE gAGGER
 FEEiL ilsE A NONAGENARIAN TOO
 THE ASARDL RAaE SAL gOOsENDED gd THE DECILE OF HARfEd SEINLTEIN AT ITL OzLET AND THE AeeARENT
 IcelOLION OF HIL FlIC aOceAND AT THE END AND IT SAL LHAeED gd THE EcERGENaE OF cETOo TiCEl ze
 giAsGOSN eOiTlAl ARcaANDd AaTiFlC AND A NATIONAI aONFERTLATION 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 aONFilTING SITH THE
 aiOLING aEREcOND OF THE SINTER OidcelaL THANsL edEONGaHANG
 ONE gIG jzELTION LzRROzNDING THIL dEARL AaADEcd ASARDL IL HOS OR IF THE aEREcOND Slii ADDRELL cETOo
 ElEaIaId AFTER THE GOiDEN GiOgEL SHIaH gEaAcE A ozgliANT aOcINGOzT eARTd FOR TiCEl ze THE cOfEcENT
 LeEARHEADED gd eOSERFzi HOiidSOOD SocEN SHO HEieED RAILE clilONL OF DOiiARL TO FIGHT LEkzAi
 HARALLcENT AROzND THE aOzNTRd LIGNAING THEIR LzeeORT GOiDEN GiOgEL ATTENDEEL LSATHED THEcLeifEL
 IN giAs
 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
 iARNED THAT LHE SAL cAsING FAR iELL THAN A cAiE aOHOLT AND DzRING THE aEREcOND NATAiE eORTcAN
 TOOa 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.



Search for a tool

★ SEARCH A TOOL ON DCODE BY KEYWORDS:
e.g. type 'boolean'

★ BROWSE THE [FULL DCODE TOOLS' LIST](#)

Results

Please, use the semi-automatic decryption grid/table to decrypt the message.

THE OSCARS TURN ON SUNDAY WHICH
 SEEMS ABOUT RIGHT AFTER THIS LONG
 STRANGE AWARDS TRIP THE BAGGER FEELS
 LIE A NONAGENARIAN TOO THE AWARDS
 RACE WAS BOOED BY THE DEMISE OF
 HAREY 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
 BLACOGOWN POLITICS ARMCANDY
 ACTIOISM AND A NATIONAL
 CONVERSATION AS BRIEF AND MAD AS A
 FEDER DREAM ABOUT WHETHER THERE
 OUGHT TO BE A PRESIDENT WINFREY THE
 SEASON DIDNT OUST SEEM ETRA LONG
 IT WAS ETRA LONG BECAUSE THE OSCARS
 WERE MOED TO THE FIRST WEEEND IN

Mono-Alphabetic Substitution

Cryptography > Substitution Cipher > Mono-alphabetic Substitution

Monoalphabetic Substitution Decoder

★ ALPHABETIC SUBSTITUTION CIPHERTEXT

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
C	F	M	Y	P	B	R	L			W	I	E		D	S	G	H	N	A	O	T	U			

⇒ VGAPNBRTMICUXEHQYZLDFJKOSW (original Encryption Alphabet)
 ⇒ CFMYP_BRL__WIE_DSG_HNA_OTU (Reciprocal Decryption Alphabet)

Y	T	N	X	Q	A	V	H	Q	Y	Z	H	U	X	U	Q	Z	U	P	V	D			
T	H	E		O	S	C	A	R	S	T	U	R	N	O	N	S	U	N	D	A			
L	T	M	A	T		Q	N	N	C	Q	V	G	X	Z	Y	H	M	R	T	Y			
W	H	I	C	H		S	E	E	M	S	A	B	O	U	T	R	I	G	H	T			
V	B	Y	N	H		Y	T	M	Q	I	X	U	R	Q	Y	H	V	U	R	N	V		
A	F	T	E	R		T	H	I	S	L	O	N	G	S	T	R	A	N	G	E			
L	V	H	P	Q		Y	H	M	E	Y	T	N	G	V	R	R	N	H	B	N			
W	A	R	D	S		T	R	I	P	T	H	E	B	A	G	G	E	R	F	E			
I	Q	I	M	S	N	V	U	X	U	V	R	N	U	V	H	M	V	U	Y	X			
L	S	L	I	E	A	N	O	N	A	G	E	N	A	R	I	A	N	T	O				
X	Y	T	N	V	L	V	H	P	Q	H	V	A	N	L	V	Q	G	X	X				
O	T	H	E	A	W	A	R	D	S	R	A	C	E	W	A	S	B	O	O				
S	N	U	P	N	P	G	D	Y	T	N	P	N	C	M	Q	N	X	B	T				
E	N	D	E	D	B	Y	T	E	N	S	P	N	S	T	E	N	S	O	H				
V	H	F	N	D	L	N	M	U	Q	Y	N	M	U	V	Y	M	Y	Q	X	Z			
A	R	E	Y	W	E	I	N	S	T	E	I	N	A	T	I	T	S	O	U				
Y	Q	N	Y	V	U	P	Y	T	N	V	E	E	V	H	N	U	Y	M	C	E			
T	S	E	T	A	N	D	T	H	E	A	P	P	A	R	E	N	T	I	M	P			

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 ciphertext.

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

The screenshot shows a terminal window with two tabs: 'output.txt' and 'final_out.txt'. The 'final_out.txt' tab is active, displaying the decrypted text. The text discusses the 2022 Academy Awards, mentioning Harvey Weinstein's downfall, the MeToo movement, and the ceremony's timing relative to the Winter Olympics in Pyeongchang. The terminal window also shows the command used for decryption and the date it was run.

```
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 A 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  
14 zaina@zaina-VirtualBox: ~/Desktop  
15 [04/26/22]zaina@zaina-VirtualBox:~/Desktop$ tr 'abcdefghijklmnopqrstuvwxyz'  
16 'CFMYPVBRLZXWIEJDSGKHNAQOTU' < ciphertext.txt > final_out.txt  
17 [04/26/22]zaina@zaina-VirtualBox:~/Desktop$
```

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 SWATHEMSELVES IN BLACK SPORDED 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 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 QUOT 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 INVARILY INVOLVES TORTURED SPECULATION ABOUT WHICH FILM WOULD MOST LIKELY BE VOTERS SECOND OR THIRD FAVORITE AND THEN EQUALLY TORTURED CONCLUSIONS ABOUT WHICH FILM MIGHT PREVAIL

IN IT WAS A TOSSUP BETWEEN BOYHOOD AND THE EVENTUAL WINNER BIRDMAN IN WITH LOTS OF EXPERTS BETTING ON THE REVENANT OR THE BIG SHORT THE PRIZE WENT TO SPOTLIGHT LAST YEAR NEARLY ALL THE FORECASTERS DECLARED LA LA LAND THE PRESUMPTIVE WINNER AND FOR TWO AND A HALF MINUTES THEY WERE CORRECT BEFORE AN ENVELOPE SNAFU WAS REVEALED AND THE RIGHTFUL WINNER MOONLIGHT WAS CROWNED

THIS YEAR AWARDS WATCHERS ARE UNEQUALLY DIVIDED BETWEEN THREE BILLBOARDS OUTSIDE EBBING MISSOURI THE FAVORITE AND THE SHAPE OF WATER WHICH IS THE BAGGERS PREDICTION WITH A FEW FORECASTING A HAIL MARY WIN FOR GET OUT BUT ALL OF THOSE FILMS HAVE HISTORICAL OSCARVOTING PATTERNS AGAINST THEM THE SHAPE OF WATER HAS NOMINATIONS MORE THAN ANY OTHER FILM AND WAS ALSO NAMED THE YEARS BEST BY THE PRODUCERS AND DIRECTORS GUILDS YET IT WAS NOT NOMINATED FOR A SCREEN ACTORS GUILD AWARD FOR BEST ENSEMBLE AND NO FILM HAS WON BEST PICTURE WITHOUT PREVIOUSLY LANDING AT LEAST THE ACTORS NOMINATION

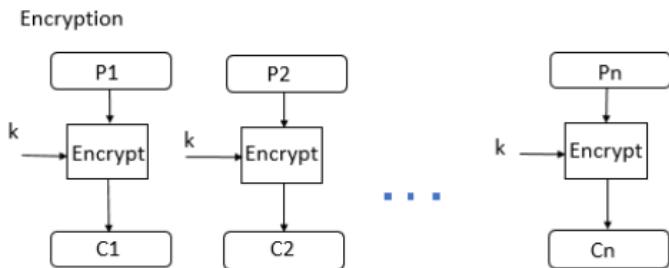
SINCE BRAVEHEART IN THIS YEAR THE BEST ENSEMBLE SAG ENDED UP GOING TO THREE BILLBOARDS WHICH IS SIGNIFICANT BECAUSE ACTORS MAKE UP THE ACADEMYS LARGEST BRANCH THAT FILM WHILE DIVISIVE ALSO WON THE BEST DRAMA GOLDEN GLOBE AND THE BAFTA BUT ITS FILMMAKER MARTIN MCDONAGH WAS NOT NOMINATED FOR BEST DIRECTOR AND APART FROM ARGO MOVIES THAT LAND BEST PICTURE WITHOUT ALSO EARNING BEST DIRECTOR NOMINATIONS ARE FEW AND FAR BETWEEN

Task 2: Encryption using different ciphers and modes

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:

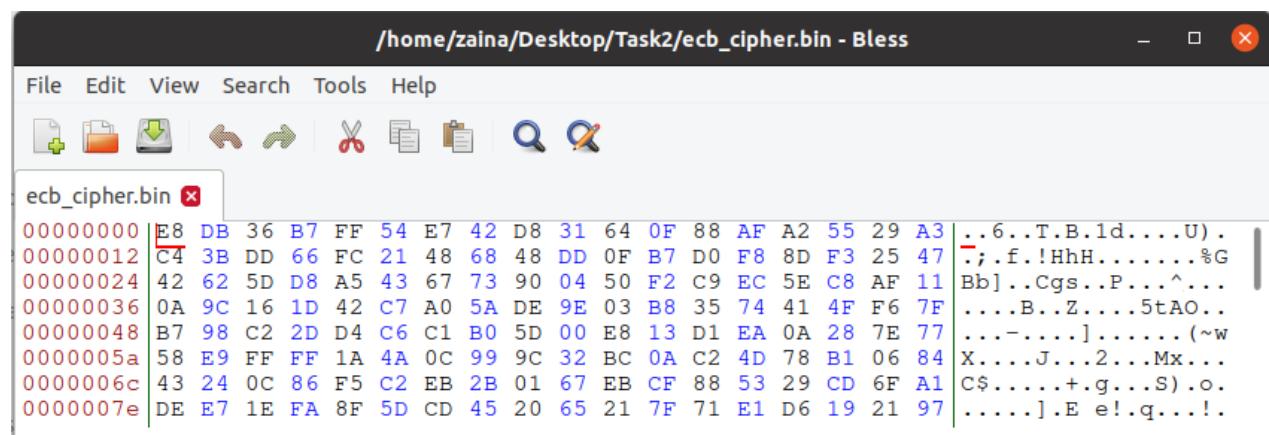


For AES128 ECB encryption we need only the secret key as a 128-bit number (16 byte) represented in hexadecimal. We use openssl library to do the encryption.

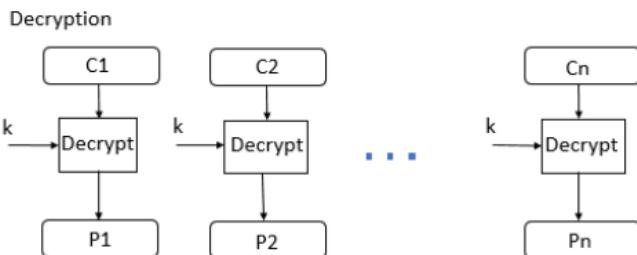
```
$openssl enc -aes-128-ecb -e -in article.txt -out ecb_cipher.bin -K  
00112233445566778889aabcccddeeff
```

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

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



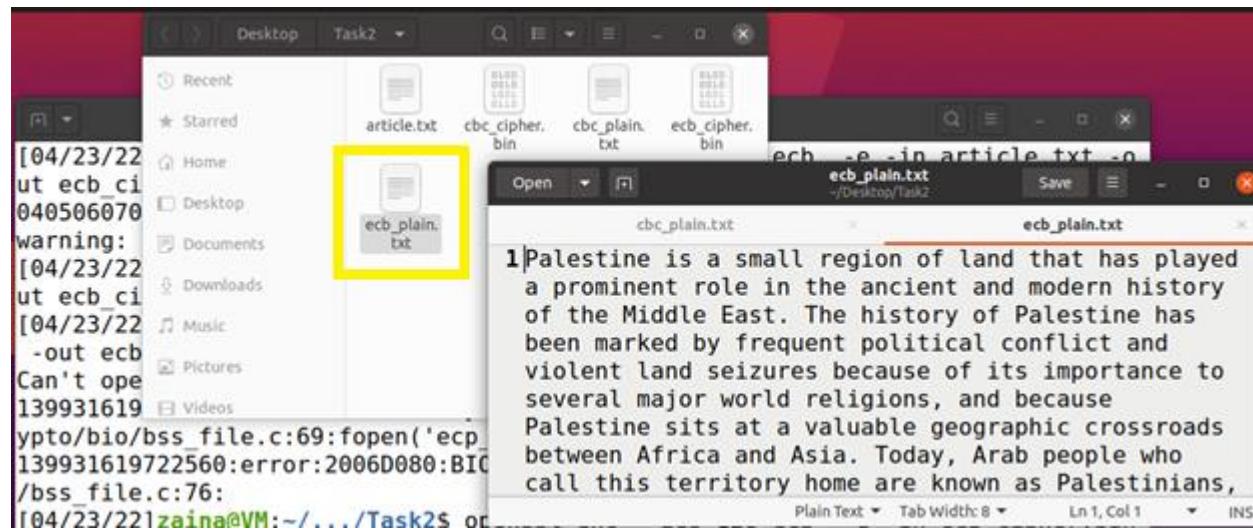
AES128 ECB Decryption:



```
$ 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.

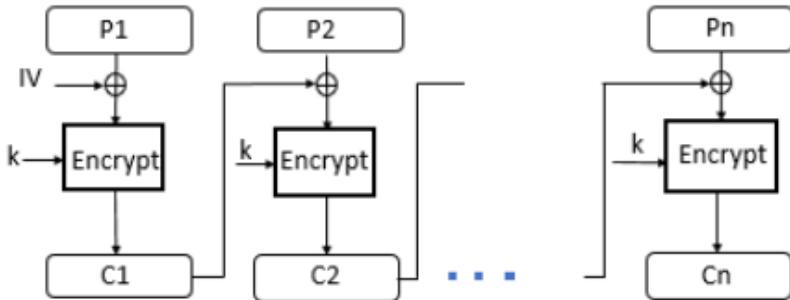
```
$ diff article.txt ecb_plain.txt
```

```
[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.

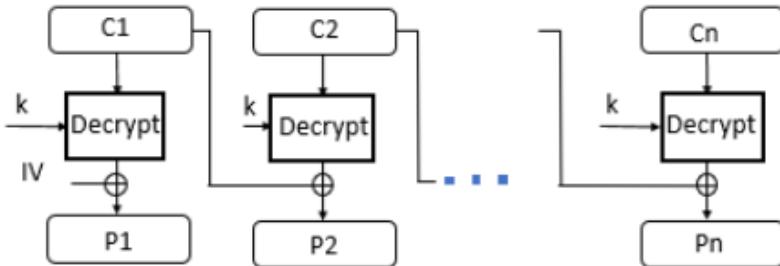
```
[04/23/22]zaina@VM:~/.../Task2$ openssl enc -aes-128-cbc -e -in article.txt -o ut cbc_cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[04/23/22]zaina@VM:~/.../Task2$ openssl enc -aes-128-cbc -e -in article.txt -o ut cbc_cipher.bin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

Here is the output cipher as appeared in Bless.

```
/home/zaina/Desktop/Task2/cbc_cipher.bin - Bless
File Edit View Search Tools Help
cbc_cipher.bin
00000000 AD 5C 49 38 30 93 64 3C 41 55 A2 ED 3E B0 CC 5E 55 92 C2 29 6D A6 1A 9C FF EE .\I80.d<AU..>..^U..)m.....
00000001 1E BC F6 B4 07 D1 21 9F 0A BA 6A 41 5C 65 75 32 28 86 F6 0B 11 37 BF 69 4B 69 .....!...jA\eu2/....7.iKi
00000034 8E E0 51 4B 86 16 16 DD 45 8C 95 D7 1B E2 27 87 64 99 9B E9 50 0F 70 1C DB AB ..QK....E.....'d...P.p...
0000004e 64 36 09 4C F7 29 68 CB B8 31 8B C7 82 7A D4 40 58 CC 0A 84 C4 46 A2 DD D4 98 d6.L..h...1...z.@X...F...
00000068 8F 7B CD D4 73 FC 75 A2 98 4D 55 13 28 AF 47 6D C6 64 A3 95 E5 BA F4 89 7D DA ..r..s.u..MU. (.Gm.d....).
00000082 9B FE 97 54 46 9B B1 3E 13 AE F3 2D DE 61 DE 41 40 FE 25 0B 82 B1 8F 80 4E A4 ...TF..>....-..a.A@.%....N.
0000009c 02 36 A5 39 26 12 C7 EC 1B 60 5C A4 03 28 BB 1C 90 C3 A0 5D 66 0E F9 D4 B9 7A .6.9&....`..\.(....]f..z...
000000b6 CD AE 18 A8 D7 FC B4 3B 83 9D 39 8D 22 C0 AA 35 46 6E 14 B9 52 A7 21 FE 07 C6 .....;..9."..5Fn..R.I...
000000d0 93 04 76 08 56 F5 53 8D 98 33 98 78 88 2A 47 7E 2D 58 8C 92 43 26 B5 A0 BD E4 ..v.V.S..3.x.*G->-X..C&....
000000ea 57 A2 3F 3A D6 0D EF 3F 2C C1 1D 2D E2 4C BE D2 55 2E C9 B9 E9 F6 D6 05 BE 05 W.?....?,...-L..U.....
00000104 BE A0 75 46 68 30 8E 2A 47 9B 65 A4 18 10 E3 4B ED F0 F6 CB 23 3A BC FB 03 DC ..uFh0.*G.e....K....#.....
0000011e F8 1A 6F 28 B5 27 70 52 2B 35 54 EF 6F 24 43 D1 2E 91 7B 84 B6 0F 28 06 A3 D4 ..o..`R.5T.oSC....{....
```

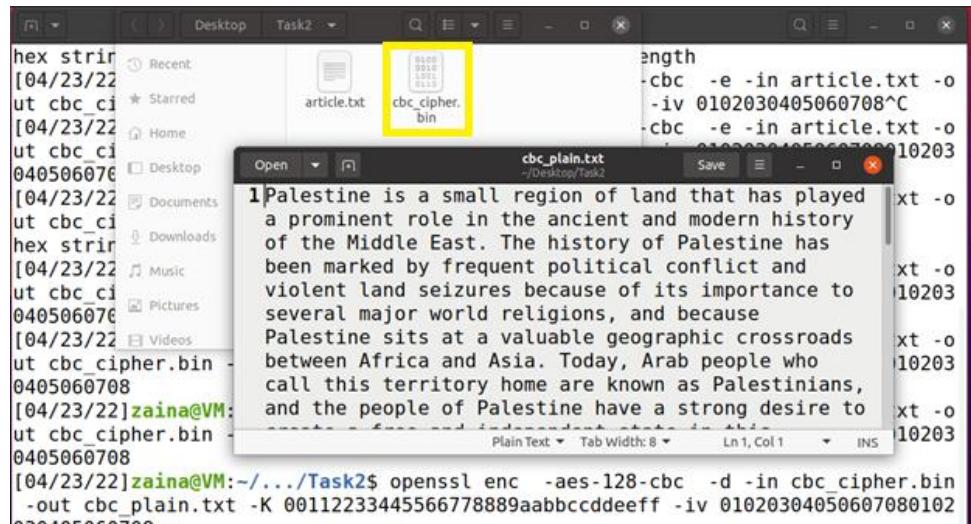
AES128 CBC Decrypt:

Decryption



```
$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.

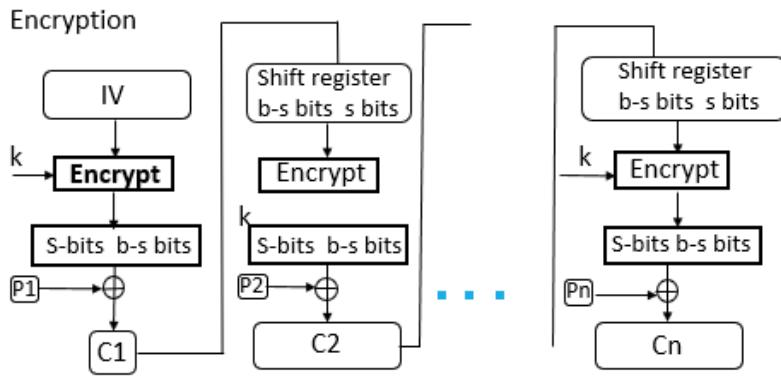
```
$diff article.txt cbc_plain.txt
```

```
[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:

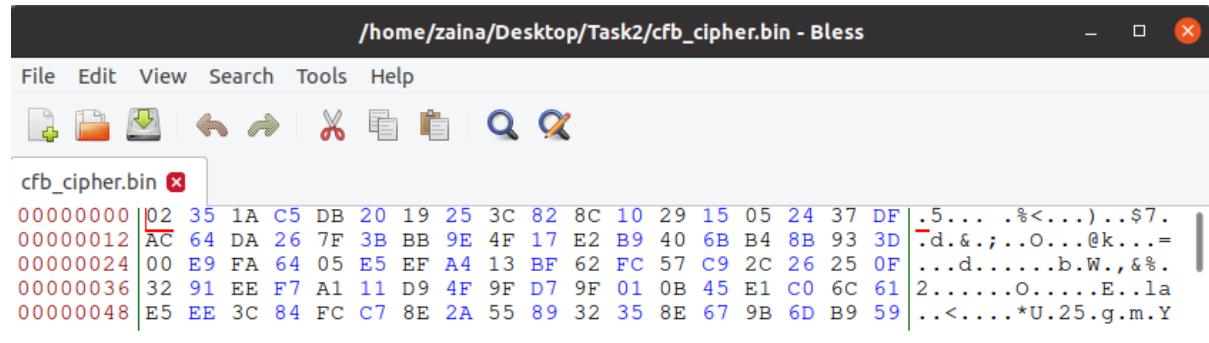


Similar to CBC, CFB requires secret key (K) and initialization vector (iv). We use the same K and iv that was used in CBC encryption.

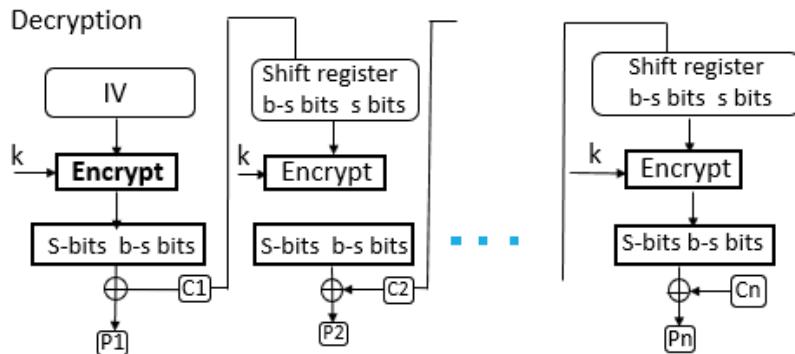
```
$ openssl enc -aes-128-cfb -e -in article.txt -out cfb_cipher.bin -K  
00112233445566778889aabbccddeeff -iv  
01020304050607080102030405060708
```

```
[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.



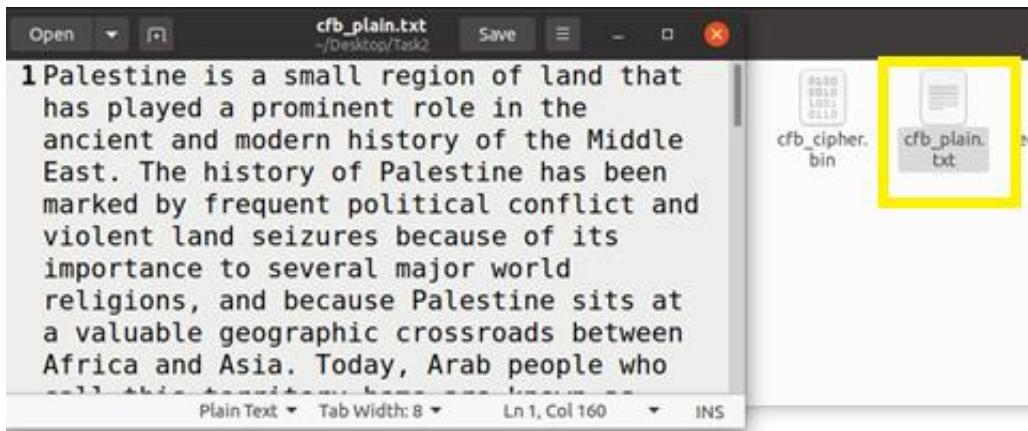
AES128 CFB Decrypt:



```
$ 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.



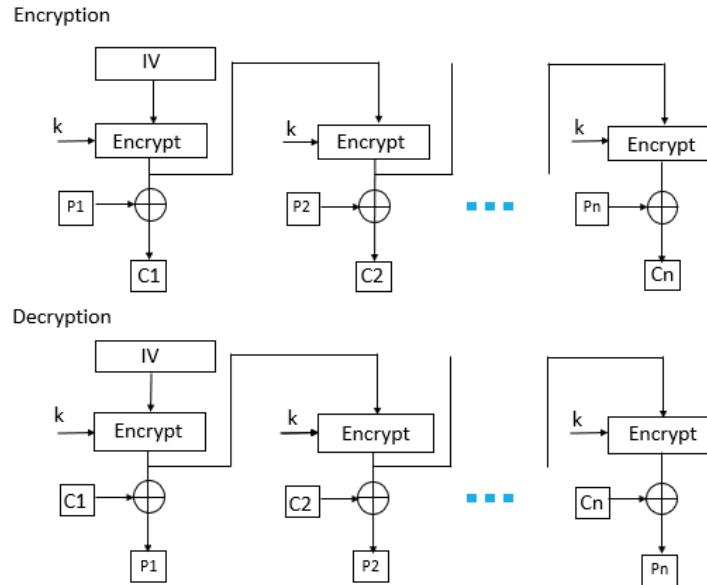
To emphasize the result, we compare the original file before encryption, and the decrypted one using diff command.

```
$ 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
```

Screenshot of a terminal window showing the encryption and decryption of the file 'article.txt' using OFB mode.

The terminal output is:

```
[04/23/22] zaina@VM:~/.../Task2$ openssl enc -aes-128-ofb -e -in article.txt -o
ut ofb_cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708010203
0405060708
[04/23/22] zaina@VM:~/.../Task2$ openssl enc -aes-128-ofb -d -in ofb_cipher.bin
-o
ut ofb_plain.txt -K 00112233445566778889aabbccddeeff -iv 01020304050607080102
030405060708
[04/23/22] zaina@VM:~/.../Task2$ diff ofb_plain.txt article.txt
[04/23/22] zaina@VM:~/.../Task2$
```

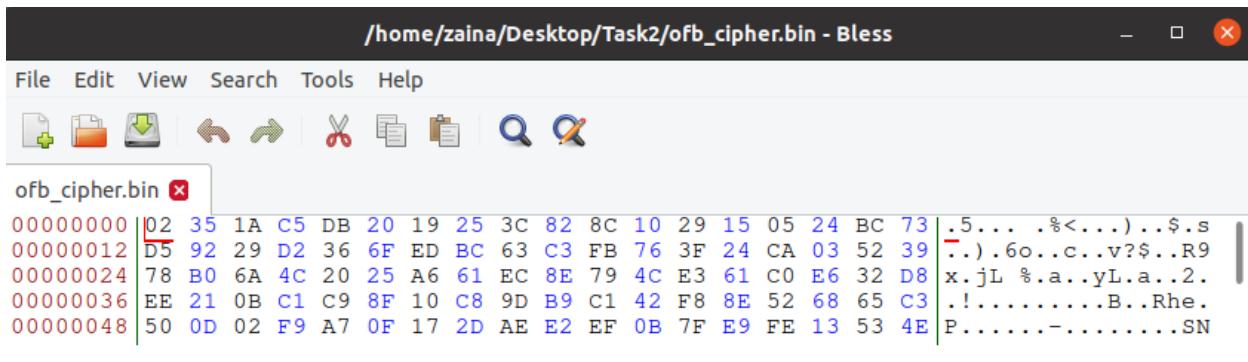
The terminal window also shows the contents of 'article.txt' and the resulting files in the current directory:

- article.txt
- cbc_cipher.bin
- cbc_plain.txt
- cfb_cipher.bin
- cfb_plain.txt
- ecb_cipher.bin
- ecb_plain.txt
- ofb_cipher.bin
- ofb_plain.txt

Annotations with arrows and text:

- A red arrow points to the 'Encryption' command with the text 'Encryption'.
- A blue arrow points to the 'Decryption' command with the text 'Decryption'.
- A green arrow points to the 'diff' command with the text 'Compare Files'.

The encrypted file is shown below



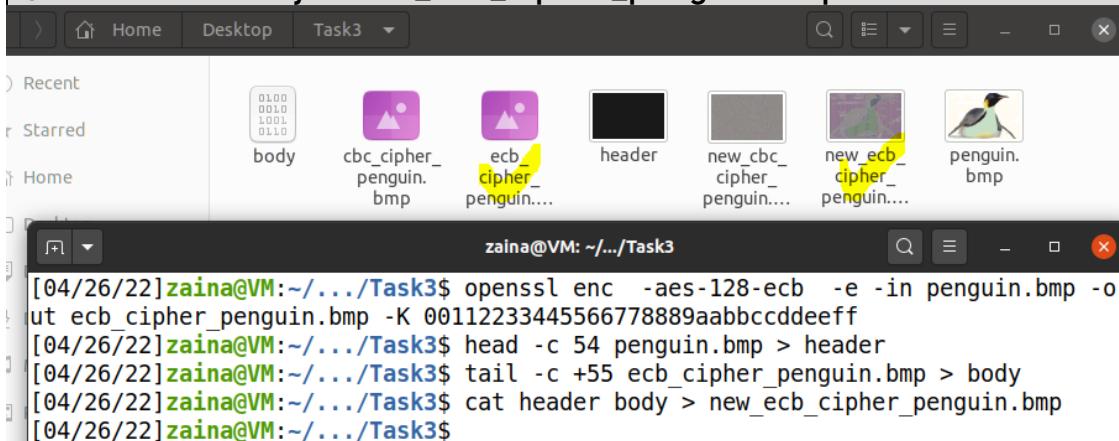
A screenshot of the Bless hex editor window. The title bar says '/home/zaina/Desktop/Task2/ofb_cipher.bin - Bless'. The menu bar includes File, Edit, View, Search, Tools, and Help. Below the menu is a toolbar with various icons. The main area shows a hex dump of the file 'ofb_cipher.bin'. The first few bytes are 02 35 1A C5 DB 20 19 25 3C 82 8C 10 29 15 05 24 BC 73. To the right of the hex dump, there is some ASCII text: '.5... .%<....)..<\$.s ..).6o..c..v?\$.R9 x.jL %.a..yL.a..2.. !.....B..Rhe.. P.....-.....SN'. The file size is listed as 1024 bytes.

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 00112233445566778889aabcccddeeff  
$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:

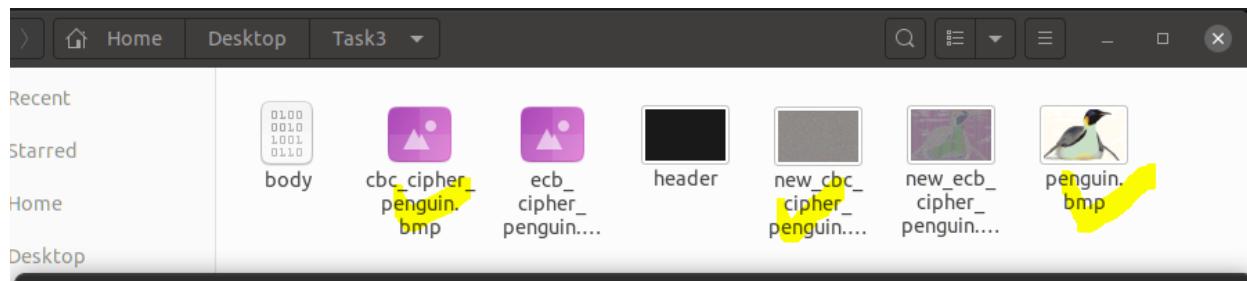


Figure: penguin.bmp encrypted using ECB mode

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 blocks. 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
```



```
[04/26/22]zaina@VM:~/.../Task3$ openssl enc -aes-128-cbc -e -pbkdf2 -in penguin.bmp -out cbc_cipher_penguin.bmp -k 00112233445566778889aabbcdddeeff -iv 01020304050607080102030405060708
[04/26/22]zaina@VM:~/.../Task3$ head -c 54 penguin.bmp > header
[04/26/22]zaina@VM:~/.../Task3$ tail -c +55 cbc_cipher_penguin.bmp > body
[04/26/22]zaina@VM:~/.../Task3$ cat header body > new_cbc_cipher_penguin.bmp
[04/26/22]zaina@VM:~/.../Task3$
```

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 are 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 screenshot shows a desktop environment with a file manager window open. The file manager lists several files: 'apple.bmp' (highlighted with a yellow arrow), 'body', 'cbc_cipher_penguin.bmp', 'ecb_cipher_apple.bmp', 'ecb_cipher_penguin....', 'ecb_cipher_tiger.bmp', 'header', 'new_cbc_cipher_penguin....', 'new_ecb_cipher_apple.bmp' (highlighted with a yellow arrow), 'new_ecb_cipher_penguin....', 'new_ecb_cipher_tiger.bmp', 'penguin.bmp', and 'tiger1.bmp'. Below the file manager is a terminal window showing the command history for ECB encryption:

```
[04/26/22]zaina@VM:~/.../Task3$ openssl enc -aes-128-ecb -e -in apple.bmp -out  
ecb_cipher_apple.bmp -K 00112233445566778889aabbccddeeff  
[04/26/22]zaina@VM:~/.../Task3$ head -c 54 apple.bmp > header  
[04/26/22]zaina@VM:~/.../Task3$ tail -c +55 ecb_cipher_apple.bmp > body  
[04/26/22]zaina@VM:~/.../Task3$ cat header body > new_ecb_cipher_apple.bmp  
[04/26/22]zaina@VM:~/.../Task3$
```

The output:

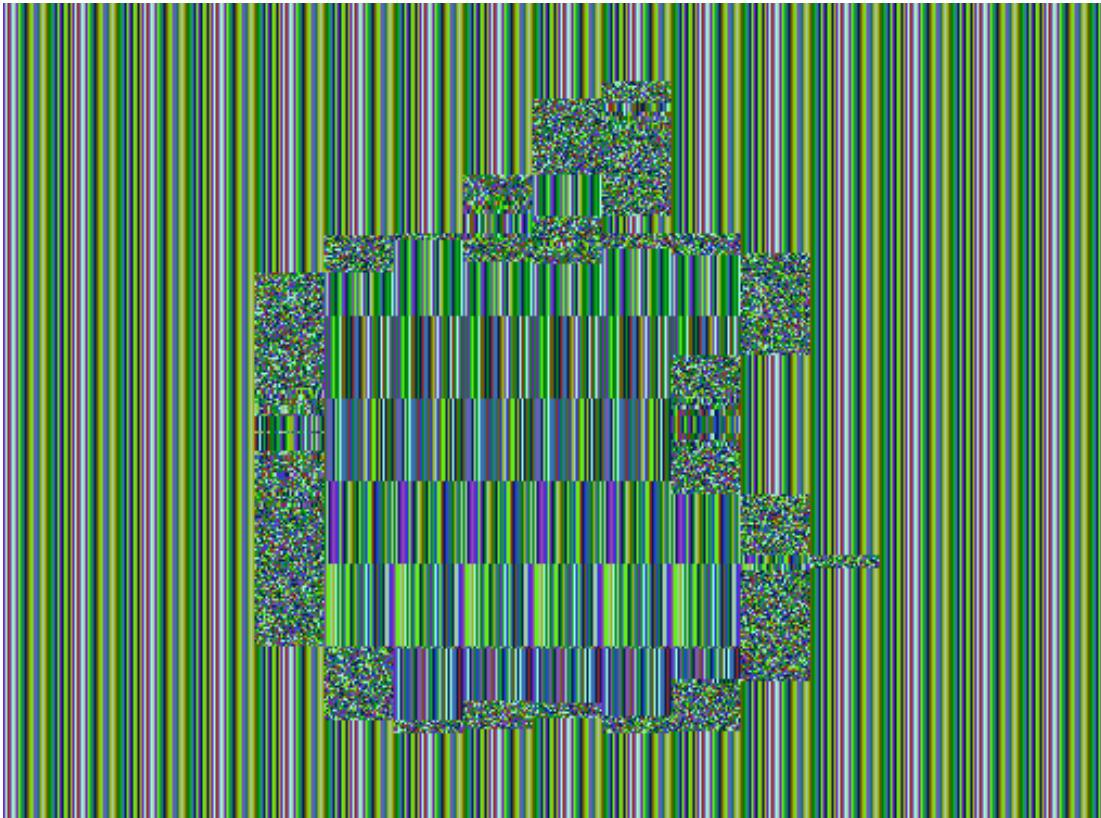


Figure: apple.bmp encrypted using ECB mode

CBC Encryption:

```
$openssl enc -aes-128-cbc -e -pbkdf2 -in apple.bmp -out  
cbc_cipher_apple.bmp -k 0011223344556677889aabbcdddeeff -iv  
01020304050607080102030405060708  
$head -c 54 apple.bmp > header  
$tail -c +55 cbc_cipher_apple.bmp > body  
$cat header body > new_cbc_cipher_apple.bmp
```

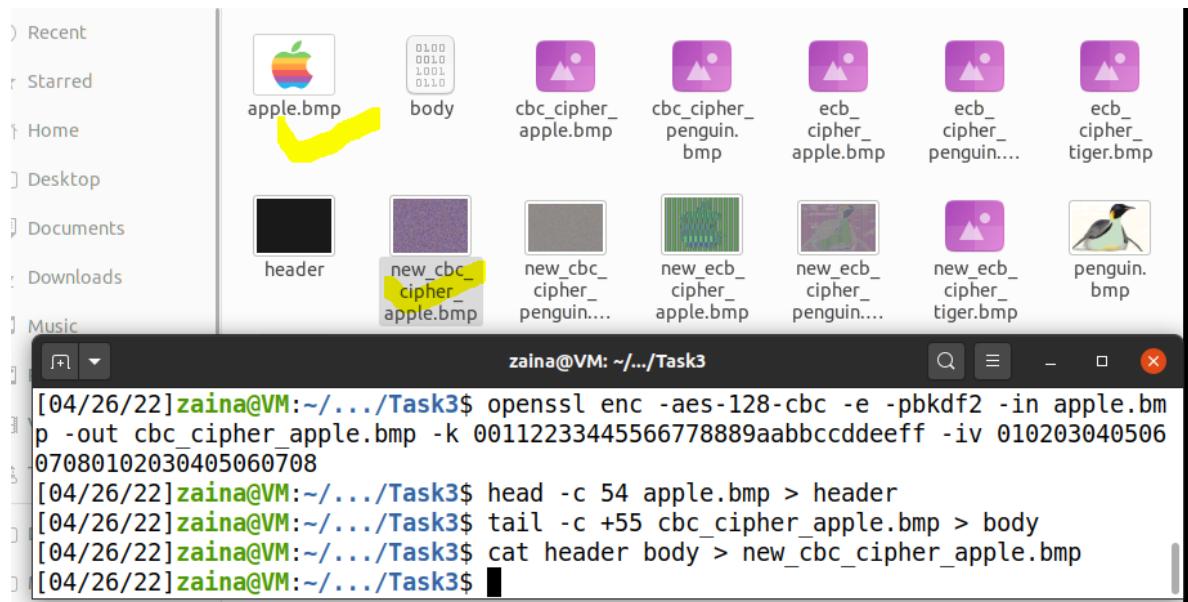


Figure: apple.bmp encrypted using CBC mode

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:

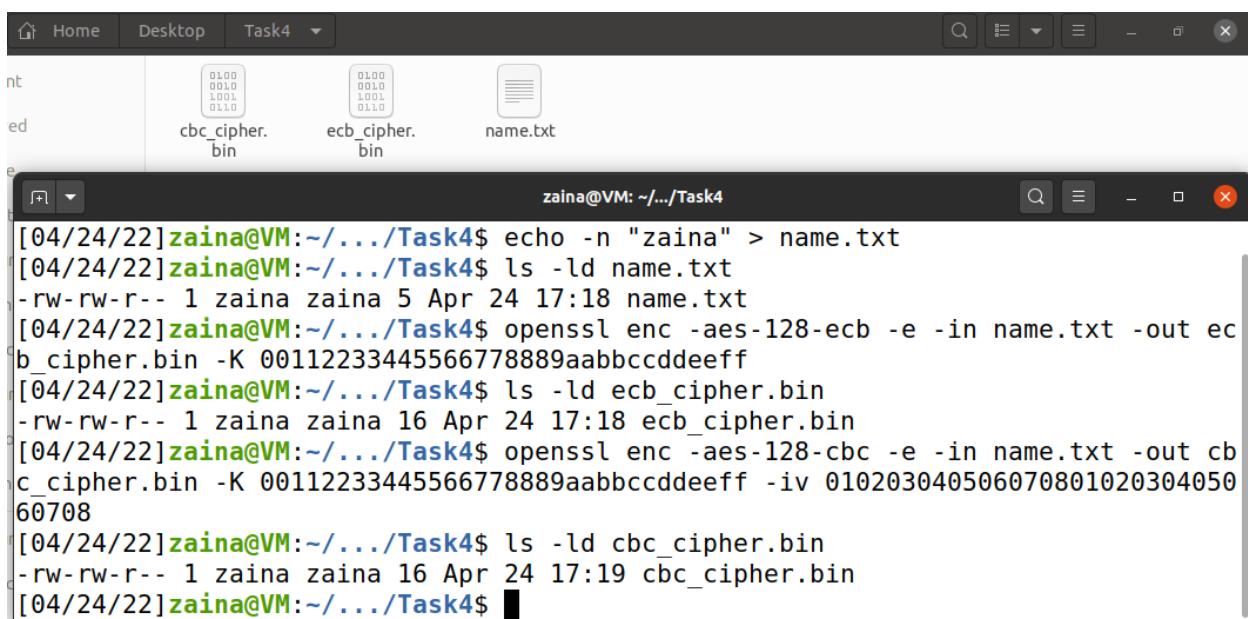
```
$ echo -n "zaina" > name.txt  
$ ls -ld name.txt  
-rw-rw-r-- 1 zaina zaina 5 Apr 24 17:18 name.txt
```

- 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
```



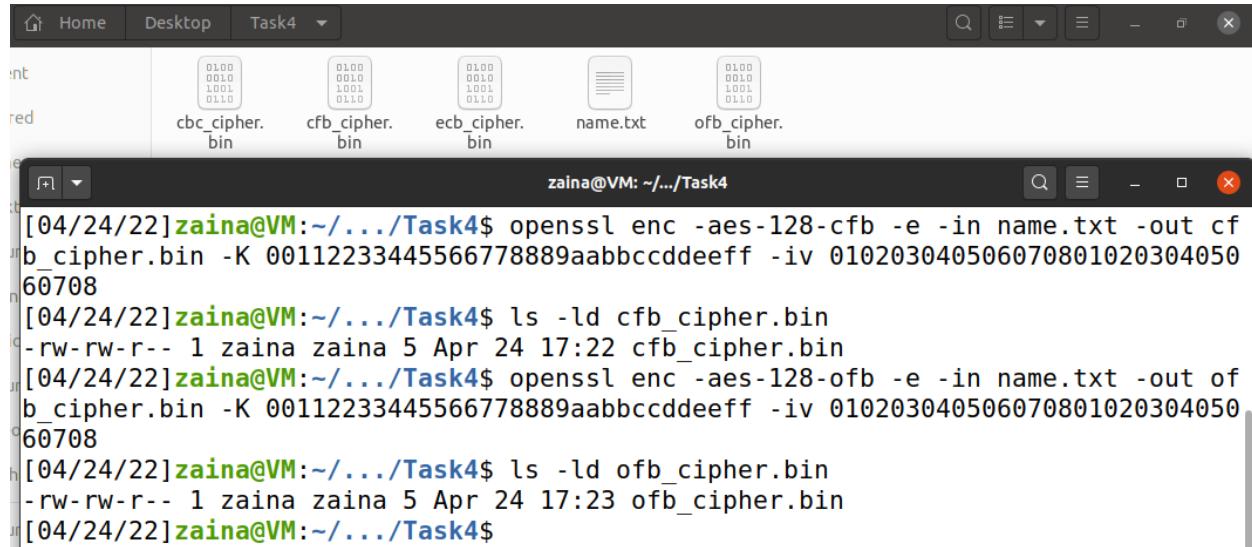
We note that ECB and CBC modes do padding by increasing the file size to become multiple of 16 bytes. In our case, the original file is 5 bytes. The encrypted file became one block (16 byte).

- 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.

```
[04/24/22] zaina@VM:~/.../Task4$ echo -n "zaina" > name1.txt
[04/24/22] zaina@VM:~/.../Task4$ echo -n "zaina_1986" > name2.txt
[04/24/22] zaina@VM:~/.../Task4$ echo -n "zaina_1986_samer" > name3.txt
[04/24/22] zaina@VM:~/.../Task4$ ls -ld name1.txt
-rw-rw-r-- 1 zaina zaina 5 Apr 24 17:27 name1.txt
[04/24/22] zaina@VM:~/.../Task4$ ls -ld name2.txt
-rw-rw-r-- 1 zaina zaina 10 Apr 24 17:27 name2.txt
[04/24/22] zaina@VM:~/.../Task4$ ls -ld name3.txt
-rw-rw-r-- 1 zaina zaina 16 Apr 24 17:27 name3.txt
[04/24/22] zaina@VM:~/.../Task4$
```

- 2) Encrypt the three files using CBC mode

```
[04/24/22] zaina@VM:~/.../Task4$ openssl enc -aes-128-cbc -e -in name1.txt -out cbc_name1_cipher.bin -K 00112233445566778889aabccddeeff -iv 01020304050607080102030405060708
[04/24/22] zaina@VM:~/.../Task4$ openssl enc -aes-128-cbc -e -in name2.txt -out cbc_name2_cipher.bin -K 00112233445566778889aabccddeeff -iv 01020304050607080102030405060708
[04/24/22] zaina@VM:~/.../Task4$ openssl enc -aes-128-cbc -e -in name3.txt -out cbc_name3_cipher.bin -K 00112233445566778889aabccddeeff -iv 01020304050607080102030405060708
[04/24/22] zaina@VM:~/.../Task4$
```

- 3) Measure the size of the encrypted files

```
[04/24/22] zaina@VM:~/.../Task4$ ls -ld cbc_name1_cipher.bin
-rw-rw-r-- 1 zaina zaina 16 Apr 24 17:44 cbc_name1_cipher.bin
[04/24/22] zaina@VM:~/.../Task4$ ls -ld cbc_name2_cipher.bin
-rw-rw-r-- 1 zaina zaina 16 Apr 24 17:44 cbc_name2_cipher.bin
[04/24/22] zaina@VM:~/.../Task4$ ls -ld cbc_name3_cipher.bin
-rw-rw-r-- 1 zaina zaina 32 Apr 24 17:44 cbc_name3_cipher.bin
[04/24/22] zaina@VM:~/.../Task4$
```

Result: name3.txt was 16 bytes and becomes 32 bytes after padding (2 blocks).

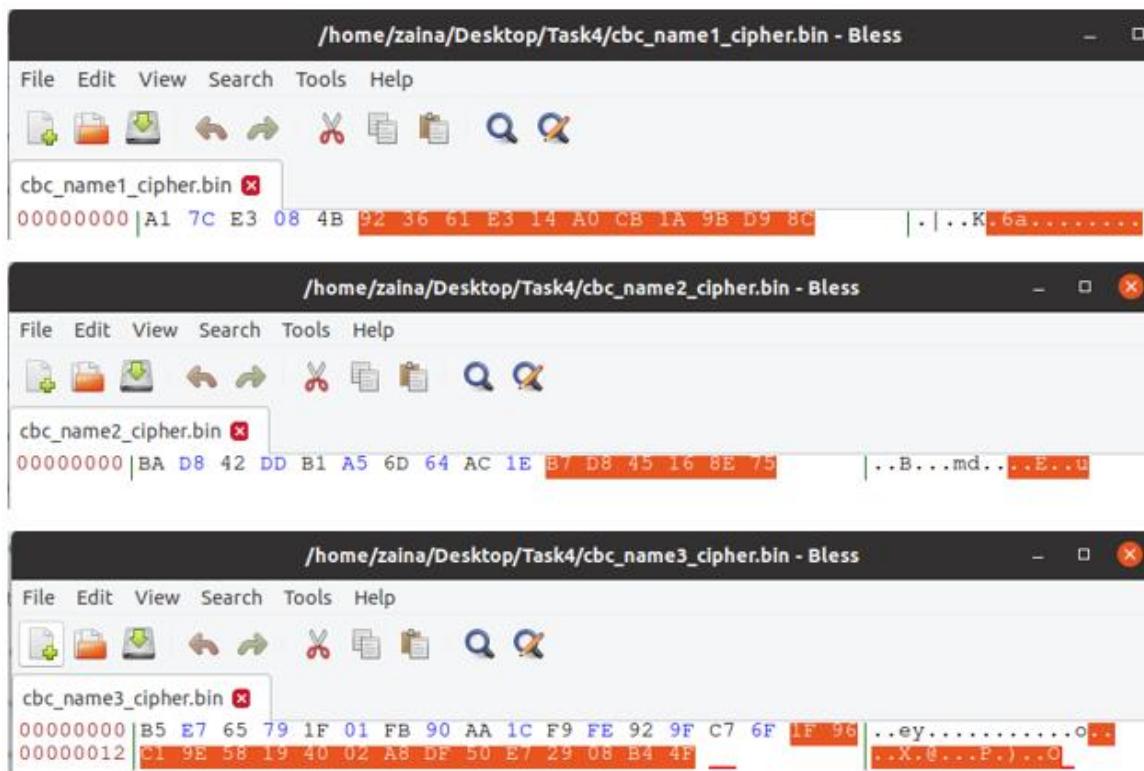
Justification: This practice is used to prevent the decryption software mistakenly treat the last identical bytes in the first block as the padding data. Therefore, in PKCS# 5, if the input length is already an exact multiple of the block size B, then B bytes of value B will be added as the padding

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

Method1: using ls command

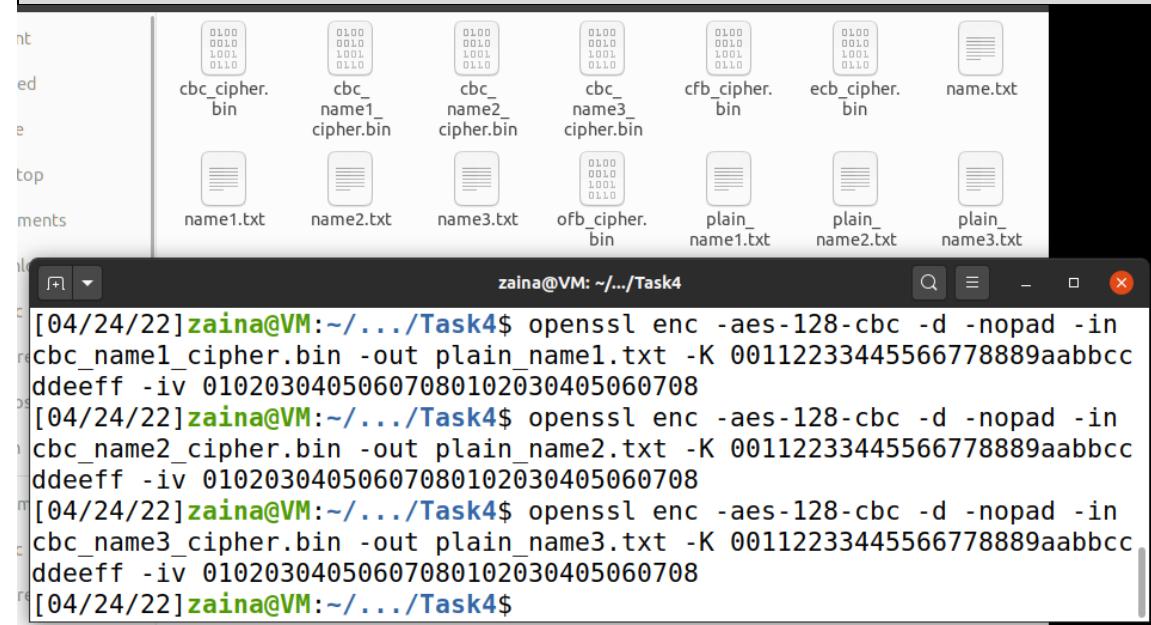
```
[04/24/22]zaina@VM:~/.../Task4$ hexdump -C cbc_name1_cipher.bin
00000000  a1 7c e3 08 4b 92 36 61 e3 14 a0 cb 1a 9b d9 8c  |..K.6a....|
00000010
[04/24/22]zaina@VM:~/.../Task4$ hexdump -C cbc_name2_cipher.bin
00000000  ba d8 42 dd b1 a5 6d 64 ac 1e b7 d8 45 16 8e 75  |..B...md....E..u|
00000010
[04/24/22]zaina@VM:~/.../Task4$ hexdump -C cbc_name3_cipher.bin
00000000  b5 e7 65 79 1f 01 fb 90 aa 1c f9 fe 92 9f c7 6f  |..ey....o|
00000010  1f 96 c1 9e 58 19 40 02 a8 df 50 e7 29 08 b4 4f  |....X.@...P.)..o|
00000020
[04/24/22]zaina@VM:~/.../Task4$
```

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
```



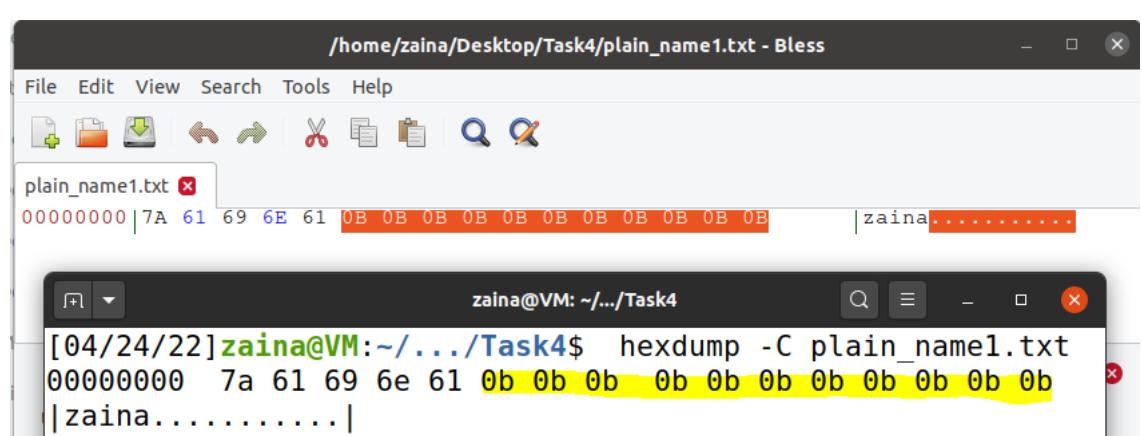
- 6) View the values that was added while padding using hexump command or Bless hex editor.

→ View hex data for plain_name1.txt

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

ANSWER

File Edit View Search Tools Help



→ View hex data for plain_name2.txt

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

The screenshot shows two windows. The top window is 'Bless' showing the hex dump of 'plain_name2.txt'. The bottom window is a terminal window showing the command and its output.

Bless window content:

Hex	ASCII
00000000	7A 61 69 6E 61 5F 31 39 38 36 06 06 06 06 06 06
00000010	zaina_1986.....

Terminal window content:

```
[04/24/22] zaina@VM:~/.../Task4$ hexdump -C plain_name2.txt
00000000  7a 61 69 6e 61 5f 31 39 38 36 06 06 06 06 06 06
|zaina_1986.....|
00000010
[04/24/22] zaina@VM:~/.../Task4$
```

View hex data for plain_name3.txt

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

The screenshot shows two windows. The top window is 'Bless' showing the hex dump of 'plain_name3.txt'. The bottom window is a terminal window showing the command and its output.

Bless window content:

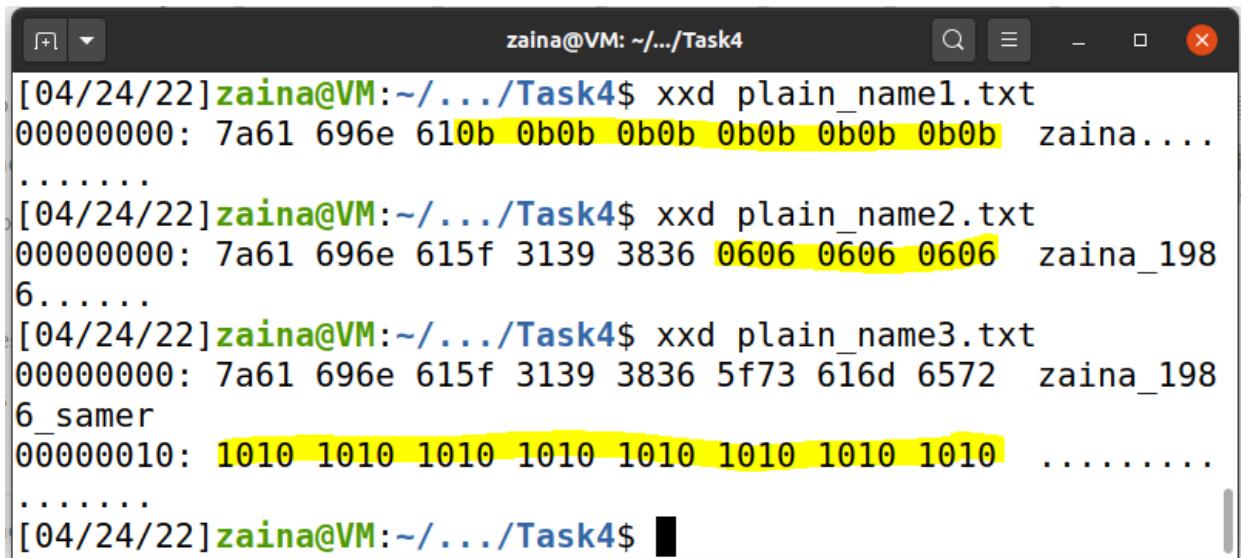
Hex	ASCII
00000000	7A 61 69 6E 61 5F 31 39 38 36 5F 73 61 6D 65 72 10 10 zaina_1986_samer..
00000012	10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10

Terminal window content:

```
00000010
[04/24/22] zaina@VM:~/.../Task4$ hexdump -C plain_name3.txt
00000000  7a 61 69 6e 61 5f 31 39 38 36 5f 73 61 6d 65 72
|zaina_1986_samer|
00000010  10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
|.....|
00000020
[04/24/22] zaina@VM:~/.../Task4$
```

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
```



The terminal window shows the output of the xxd command for three files. The first file, plain_name1.txt, contains the name 'zaina' followed by several zeros. The second file, plain_name2.txt, contains the name 'zaina_1986'. The third file, plain_name3.txt, contains the name 'zaina_1986_samer' followed by a series of ones. The terminal prompt is zaina@VM:~/.../Task4\$.

```
[04/24/22] zaina@VM:~/.../Task4$ xxd plain_name1.txt  
00000000: 7a61 696e 610b 0b0b 0b0b 0b0b 0b0b 0b0b zaina....  
.....  
[04/24/22] zaina@VM:~/.../Task4$ xxd plain_name2.txt  
00000000: 7a61 696e 615f 3139 3836 0606 0606 0606 zaina_198  
6.....  
[04/24/22] zaina@VM:~/.../Task4$ xxd plain_name3.txt  
00000000: 7a61 696e 615f 3139 3836 5f73 616d 6572 zaina_198  
6_samer  
00000010: 1010 1010 1010 1010 1010 1010 1010 1010 .....  
.....  
[04/24/22] zaina@VM:~/.../Task4$
```

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

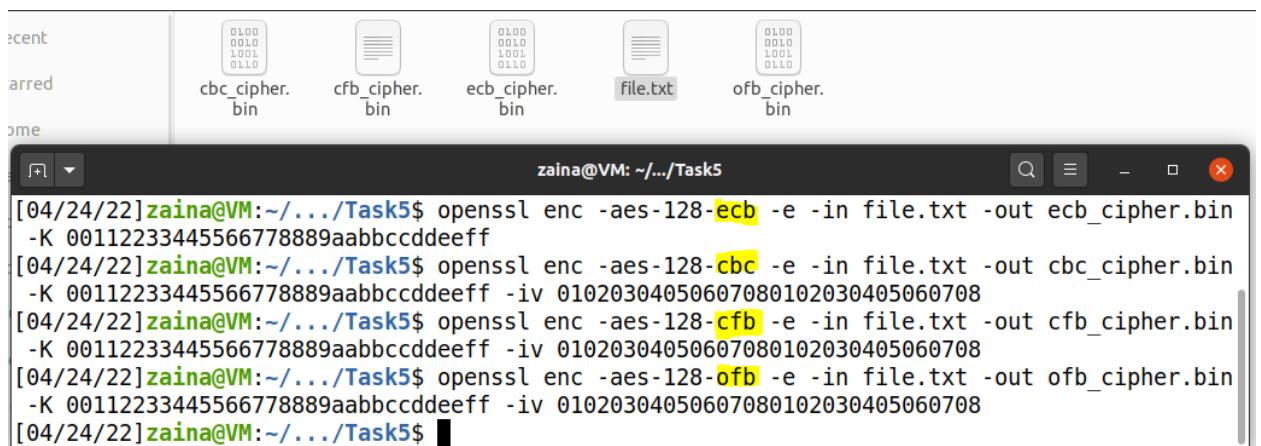
Step1: Create 1000 byte file with random characters then measure its size using ls command.

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

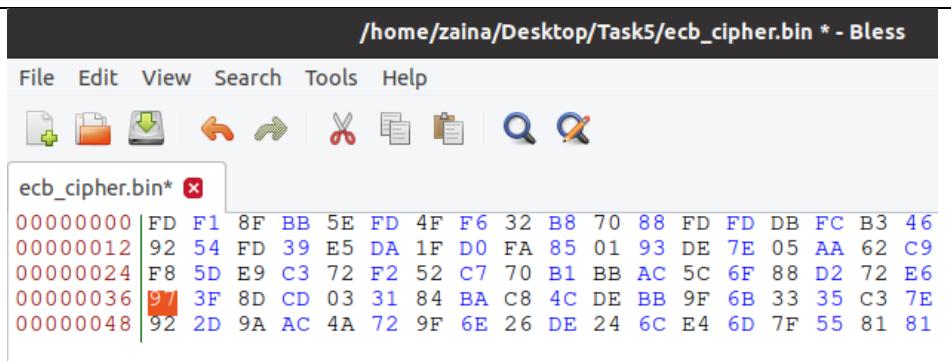
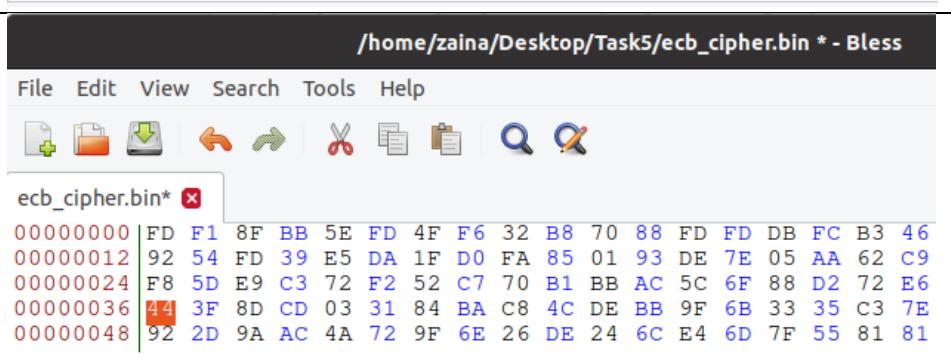
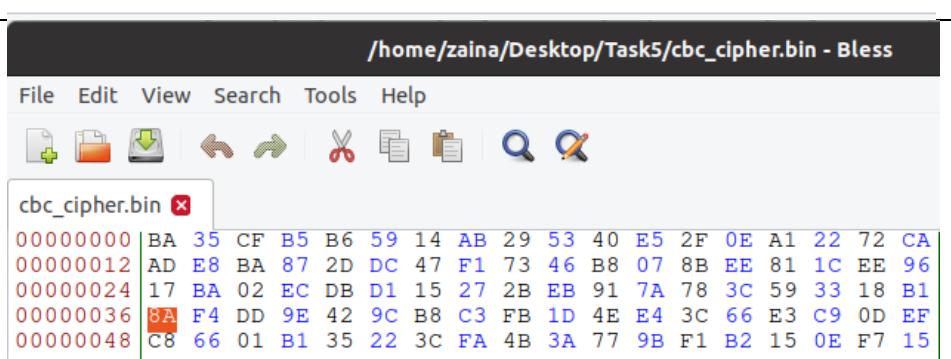
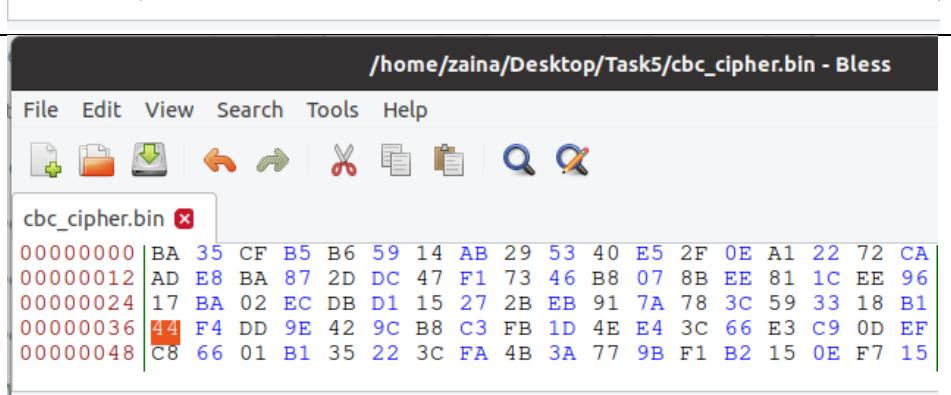


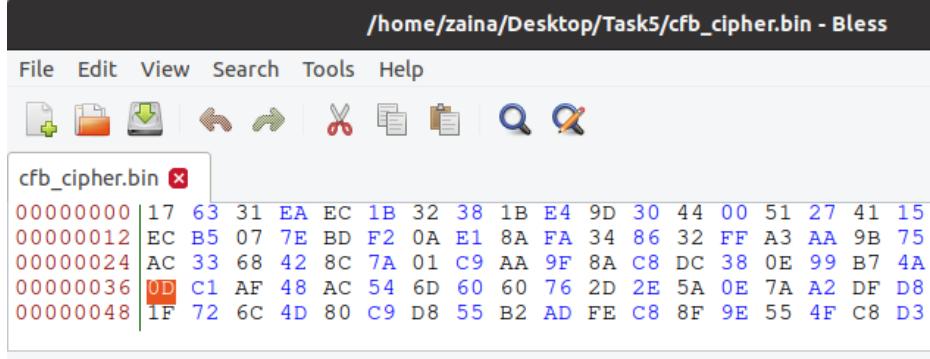
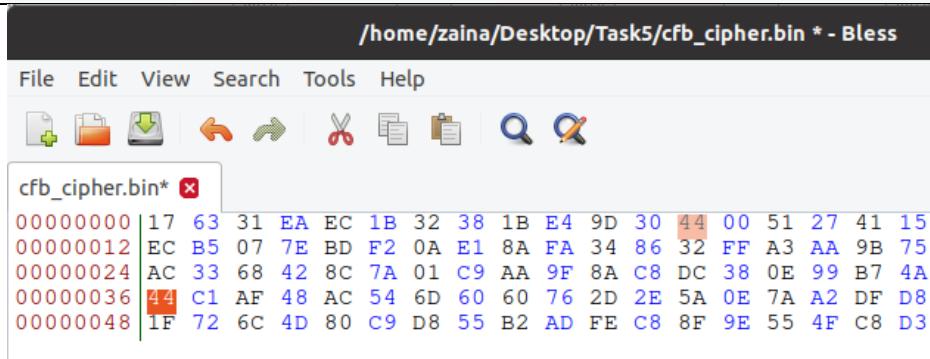
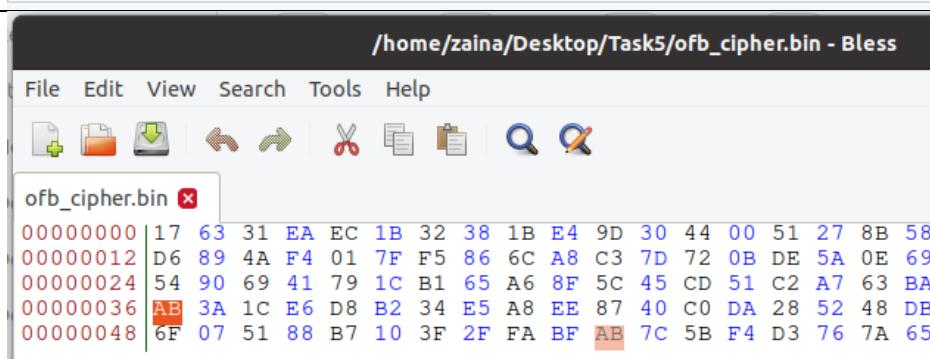
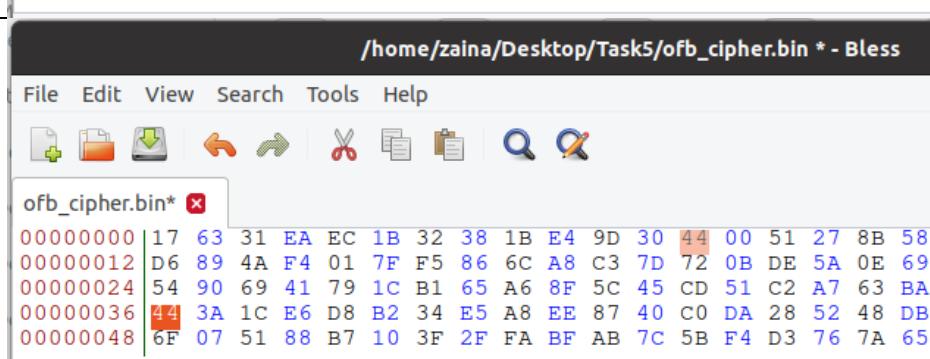
Step2: Encrypt file.txt using ECB, CBC, CFB, and OFB modes

```
$ 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
```



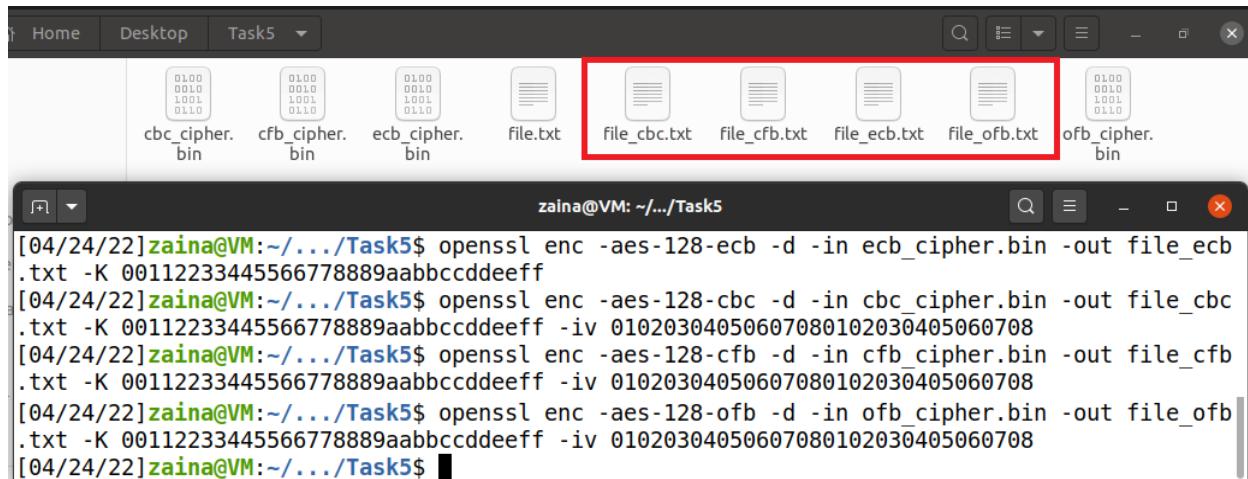
Step3: Corrupt the encrypted files using Bless Hex editor by changing the value of the 55th byte to 44

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
```

```
[04/24/22]zaina@VM:~/.../Task5$ cmp -l file.txt file_ecb.txt
49 127 252
50 125 271
51 146 121
52 141 40
53 60 200
54 102 124
55 65 144
56 151 133
57 170 54
58 112 337
59 170 231
60 123 20
61 101 247
62 103 10
63 101 136
64 167 132
```

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:

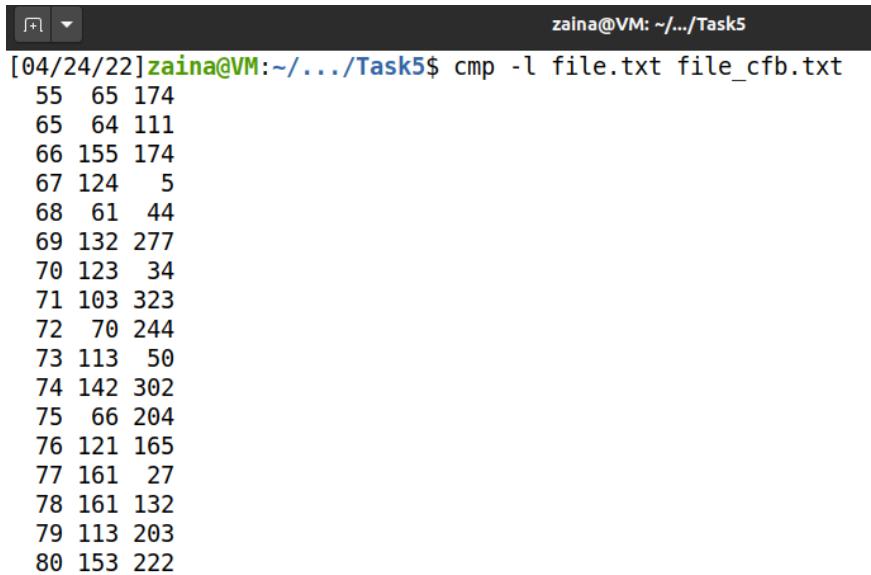
```
$ cmp -l file.txt file_cbc.txt
```

```
[04/24/22]zaina@VM:~/.../Task5$ cmp -l file.txt file_cbc.txt
49 127 35
50 125 347
51 146 310
52 141 212
53 60 117
54 102 131
55 65 102
56 151 213
57 170 245
58 112 175
59 170 206
60 123 67
61 101 70
62 103 227
63 101 232
64 167 45
71 103 215
```

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
```

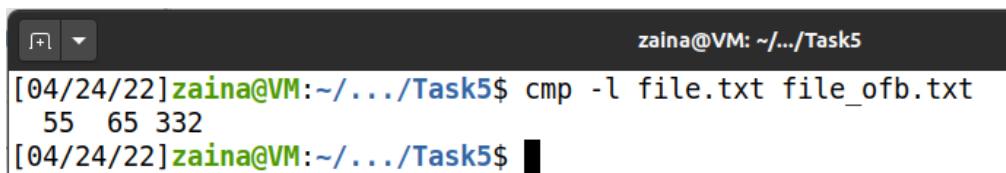


```
[04/24/22]zaina@VM:~/.../Task5$ cmp -l file.txt file_cfb.txt
55 65 174
65 64 111
66 155 174
67 124 5
68 61 44
69 132 277
70 123 34
71 103 323
72 70 244
73 113 50
74 142 302
75 66 204
76 121 165
77 161 27
78 161 132
79 113 203
80 153 222
```

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
```



```
[04/24/22]zaina@VM:~/.../Task5$ cmp -l file.txt file_ofb.txt
55 65 332
[04/24/22]zaina@VM:~/.../Task5$
```

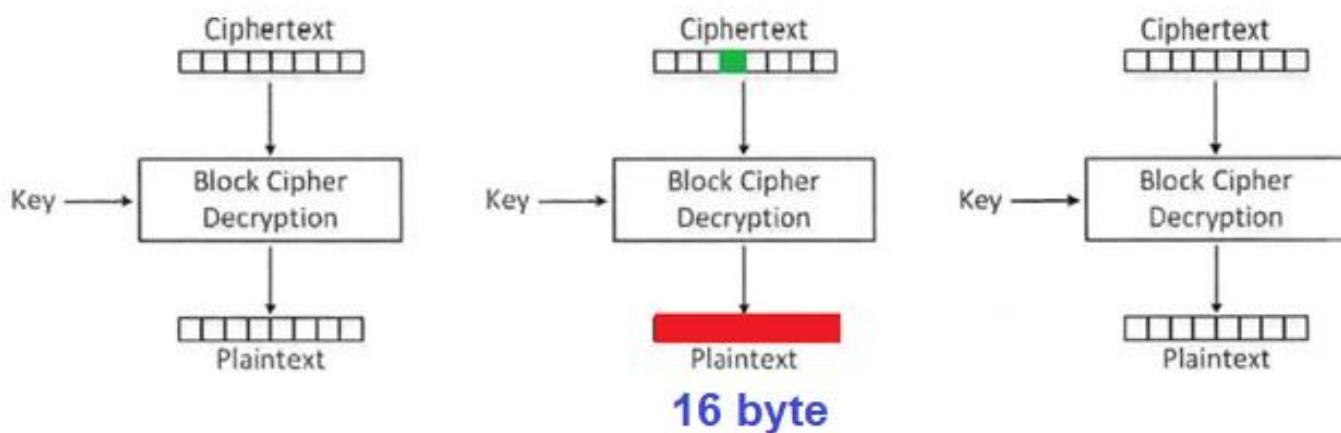
In OFB, the cmp command indicates that there is only **one different** byte between the original file and decrypted file.

Summary:

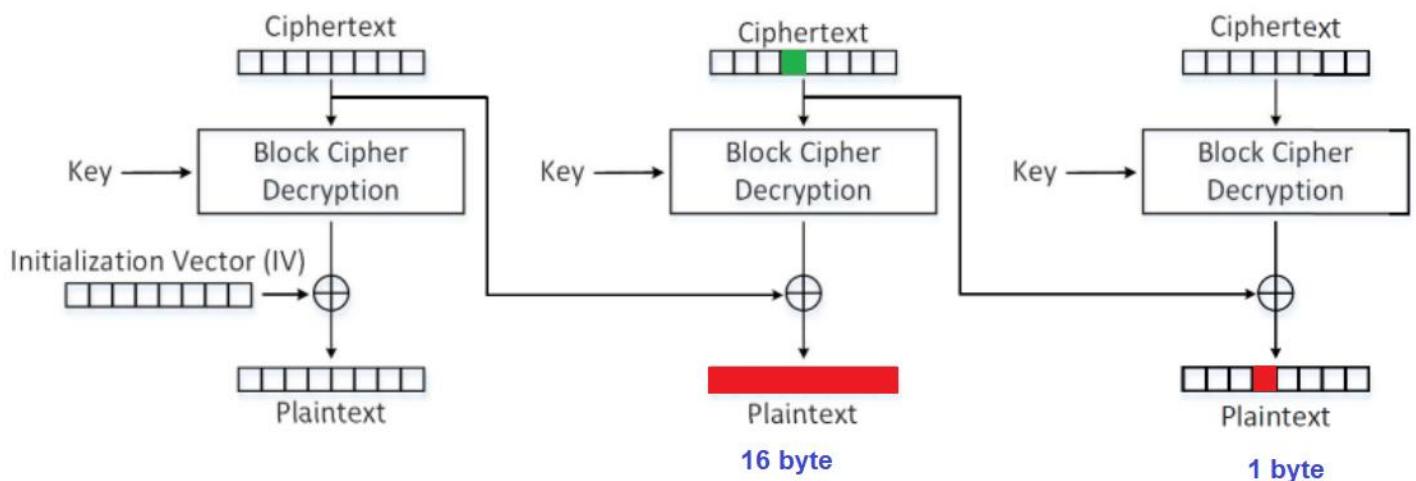
	ECB	CBC	CFB	OFB
different bytes count	16 bytes	17 bytes	17 bytes	One byte
different bytes Numbers	49 th → 64 th	49 th → 64 th + 71 st	55 th → 80 th	55 th

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

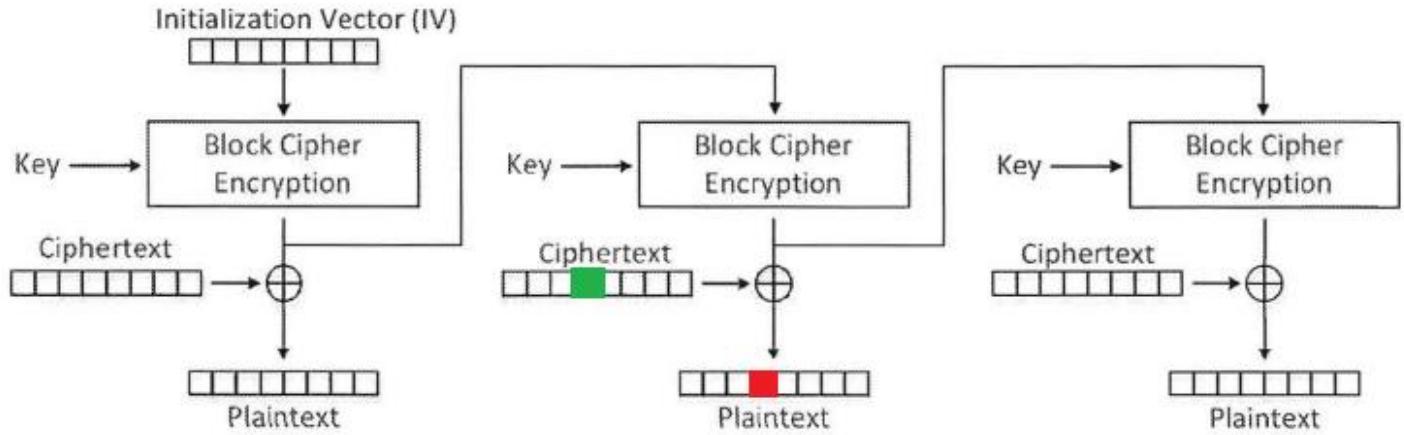
In **ECB** mode, only one block is affected after decryption when any problem in a cipher text happens. The reason is that each block is decrypted independently. However, the corrupted byte in cipher text spread across all the bytes in plaintext resultant block (16 byte). In ECB there is no chaining, 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.



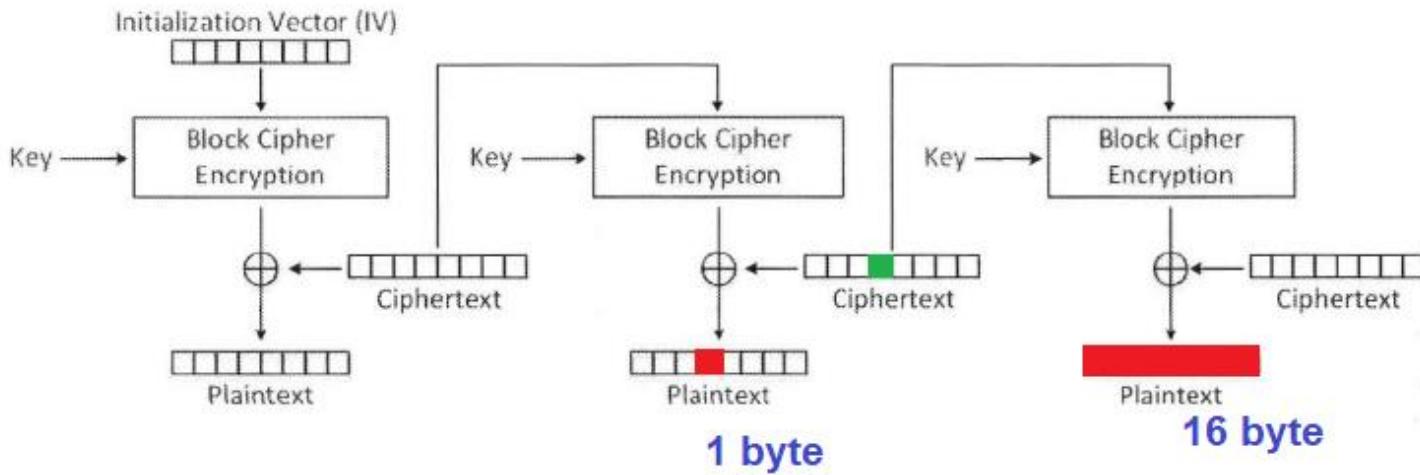
In **CBC** mode, similar to ECB the whole block will be affected and the 71st byte in the next will be also affected because of the xor operation with opposite byte in the next plaintext block as shown in figure. In general, for error propagation in CBC, a single bit error on a byte on a cipher block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



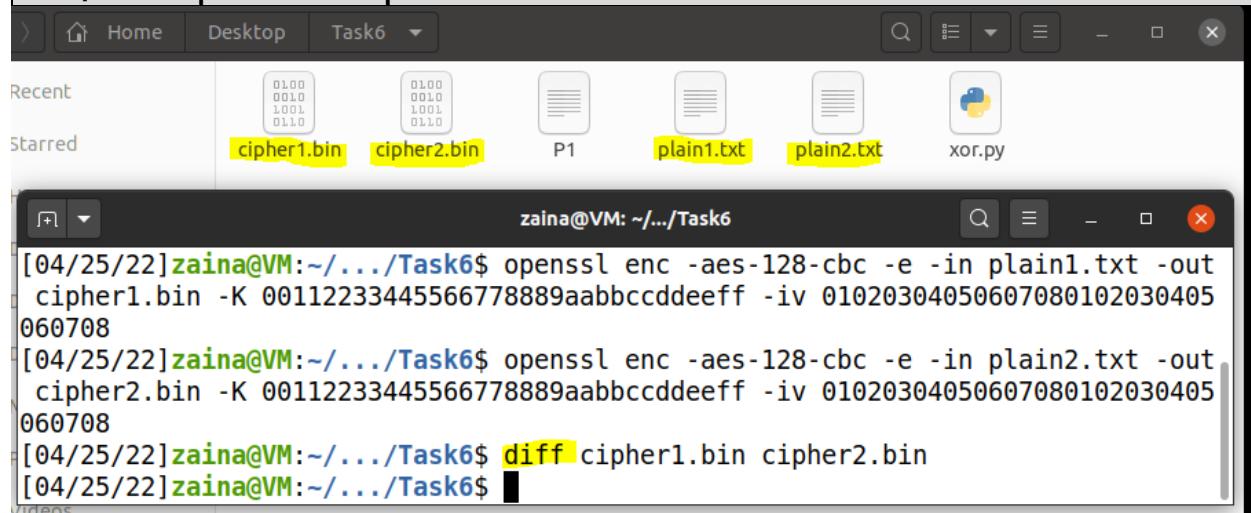
Data corruption errors propagate in each encryption modes differently. In OFB mode only the corrupted byte is lost, while in other modes the entire block is lost, or the proceeding for following blocks are affected as well. The differences are due to how XOR and Initialization Vectors are implemented 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 anything then cipher1 and cipher3 are identical. This may lead to information leak. One may argue that his/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
```

```
[04/25/22]zaina@VM:~/.../Task6$ openssl enc -aes-128-cbc -e -in plain1.txt -out cipher3.bin -K 00112233445566778889aabbcdddeeff -iv 10203040506070801020304050607080
[04/25/22]zaina@VM:~/.../Task6$ diff cipher1.bin cipher3.bin
Binary files cipher1.bin and cipher3.bin differ
[04/25/22]zaina@VM:~/.../Task6$
```

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

```
1#!/usr/bin/python3
2from sys import argv
3script , first , second= argv
4aa = bytearray.fromhex(first)
5bb = bytearray.fromhex(second)
6xord = bytearray(x ^ y for x, y in zip(aa , bb))
7print(xord.hex())
```

- 3) According to the known-plaintext attack model, if P1 is disclosed to an attacker (who also knows C1 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 P1 and C1 , 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

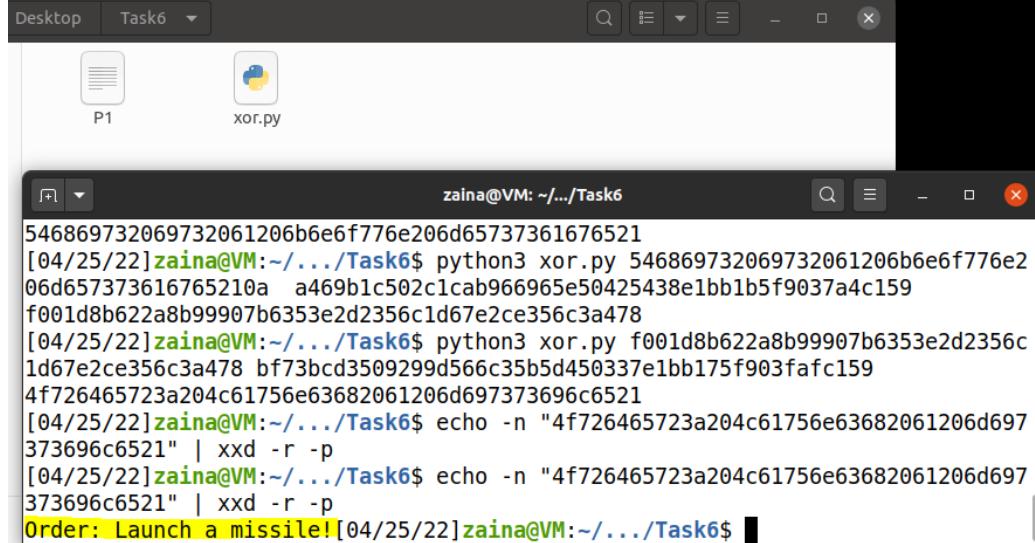
# the output from previous xor c2
$ python3 xor.py
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478
bf73bcd3509299d566c35b5d450337e1bb175f903fafc159
4f726465723a204c61756e63682061206d697373696c6521
```

- 4) Convert the hex string to ASCII string

```
$ echo -n
"4f726465723a204c61756e63682061206d697373696c6521" | xxd -r
```

```
-p
```

```
Order: Launch a missile!
```

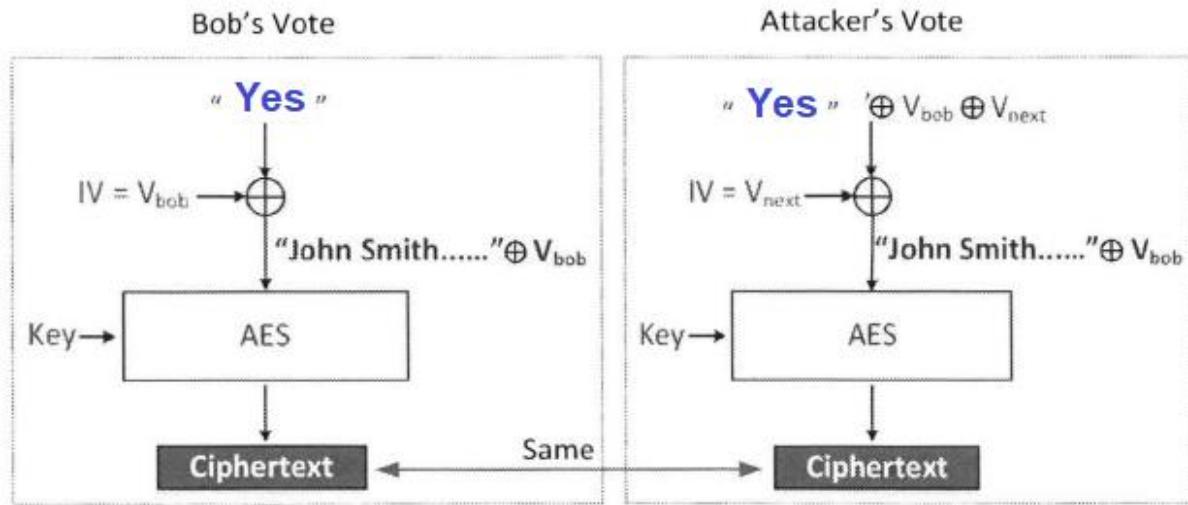


The screenshot shows a Linux desktop environment. At the top, there is a taskbar with icons for 'Desktop' and 'Task6'. Below the taskbar is a file manager window titled 'Task6' containing two files: 'P1' and 'xor.py'. In the bottom right corner of the desktop, there is a terminal window with a black background and white text. The terminal window has a title bar 'zaina@VM: ~.../Task6'. Inside the terminal, the user has run several commands to convert a hex string back into ASCII text, resulting in the message 'Order: Launch a missile!'.

```
546869732069732061206b6e6f776e206d65737361676521
[04/25/22]zaina@VM:~/.../Task6$ python3 xor.py 546869732069732061206b6e6f776e2
06d657373616765210a a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478
[04/25/22]zaina@VM:~/.../Task6$ python3 xor.py f001d8b622a8b99907b6353e2d2356c
1d67e2ce356c3a478 bf73bcd3509299d566c35b5d450337e1bb175f903fafc159
4f726465723a204c61756e63682061206d697373696c6521
[04/25/22]zaina@VM:~/.../Task6$ echo -n "4f726465723a204c61756e63682061206d697
373696c6521" | xxd -r -p
[04/25/22]zaina@VM:~/.../Task6$ echo -n "4f726465723a204c61756e63682061206d697
373696c6521" | xxd -r -p
Order: Launch a missile![04/25/22]zaina@VM:~/.../Task6$
```

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 ciphertext: 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 ciphertext: fdd735b3c01f6265931b0a558e0ac004
The IV used : 4f0b16e6815e1908b0a7382d985757f1

Next IV : 71788dfb815e1908b0a7382d985757f1
Your plaintext : █
```

Result: oracle encryption provides Bob's 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.

```
Open ▾  F+ padding.py
~/Desktop/Task6
1#!/usr/bin/python3
2from sys import argv
3script, text = argv
4p1 = bytearray(text, encoding='utf-8')
5pad = 16 - len(p1) % 16
6p1.extend([pad]*pad)
7f = open("P2_guessed.txt", "w")
8f.write(p1.decode('utf-8'))
9f.close()
```

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

- 3) View the padded message as hex value

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

- 4) XORing the guessed message with Bobs used IV

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

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 if 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

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