

Introduction to Information Security [CSP 458 - 02]

Lab-2: Secret Key Encryption [100 points]

Anusha Sonte Parameshwar [A20518694]

Overview:

The learning objective of this lab is to get familiar with the concepts in the secret-key encryption by performing the tasks listed below:

Lab Tasks:

1.	Frequency Analysis	2
2.	Encryption using Different Ciphers and Modes	6
2.1	Using cipher type AES-128-CBC	6
2.2	Using cipher type AES-128-CFB	7
2.3	Using cipher type BF-CBC	7
2.4	Using cipher type AES-128-ECB	8
2.5	Using cipher type AES-128-OFB	9
3.	Encryption Mode: ECB vs CBC	10
3.1	Using ECB Mode	10
3.2	Using CBC Mode	11
3.3	Testing with the picture of our choice	12
3.3.1	Using ECB Mode	13
3.3.2	Using CBC Mode	14
4.	Padding	15
4.1	Using ECB Mode	16
4.2	Using CBC Mode	17
4.3	Using CFB Mode	19
4.4	Using OFB Mode	21
5.	Error propagation: Corrupted Ciphertext	23
5.1	Using AES-128 ECB Mode	23
5.2	Using AES-128 CBC Mode	25
5.3	Using AES-A28 CFB Mode	26
5.4	Using AES-A28 OFB Mode	28
6.	Initial Vector (IV)	30
6.1.	Uniqueness of the IV	30
6.1.1	Using two different IVs	30
6.1.2	Using the same IV	32
6.2.	Common mistakes: Use the same IV	33
6.3.	Common mistakes: Use a predictable IV	34
7.	Programming using the Crypto library [Extra Credit]	35

Task 1: Frequency Analysis

Goal: The goal of this task is to find out the original text from the mono-alphabetic cipher text using frequency analysis technique.

Steps Performed:

- Initially, I created a file named ‘Ciphertext.txt’ and copied the given ciphertext into this file.

```
[anushasp@Anushas-MacBook-Air Task1 % cat Ciphertext.txt
hfcnkopw ahyplhp ya wznysgj hxzlvylv oxp qfwgs qyox lpq spdpqfncploa xznnplylv pdpwj szj z wyvfwfka pskhzo
yfl hfceylylv oxp oxpfwj fu ylufwczoyfl zls hfcnkcozoyfl qyox xlzafl ajaopca zls afuoqzwp spayvl ya oxp ip
j of akhhpa za flp fu oxp fgspao hfcnkopw ahyplhp spnzwocploa yl oxp hxyhzvf zwpz oxp ha spnzwocplo zo yy
o xza z gflv xyaofwj fu cppoylv oxya hxzggplvp oxwfkvx mkzgyoj pskhzojyfl yl aczgg hgzaawffc pldywfclcploa z
gflv qyox ylopwlaxyn zls wpapzwhx fnnfwoklyoypa yl ylskaowj zls lzoyflzg gzefwzofwypa
yyo aoksploa qfwi qyox fkw uzhkgoj fl qfwgshgzaa wpapzwhx yl zwpa oxzo ylhgksp szoz ahyplhp syaowyekops a
jaopca ylufwczoyfl wpowpdzg hfcnkopw lpoqfwiylv yl opggyvpl o ylufwczoyfl ajaopca zls zgvfwyoxca
oxp spnzwocplo fuupwa ezhxpgfw fu ahyplhp czaopw fu ahyplhp nwfpupaayflzg czaopw zls nxs spvwppa ngka vwzsk
zop hpwoyuhzopa zhhpgrwzops hfkwapa zls lfslspvwpp aoksj nzwooycp aoksploa hzl ozip pdplylv hgzaapa zls gf
lvsyaolhp aoksploa hzl pwz czaopwa spvwppa flgylp aoksploa wzop fkw opzhxylv za zcflyv oxp epao zo oxp kl
ydpwayoj zls fkw uzhkgoj xzdp qfl lkcpwfka opzhxylv zqzwsa
oxp aphwpo aploplhp ya vffs bfe vkja
anushasp@Anushas-MacBook-Air Task1 % ]
```

- Analyzing the most used English three letter words. The most used three letter word is ‘THE’, so replacing ‘oxp’ from ciphertext with ‘THE’ using the tr command and creating the output file as ‘Plaintext.txt’.

```
[anushasp@Anushas-MacBook-Air Task1 % tr 'oxp' 'THE' <Ciphertext.txt> Plaintext.txt
[anushasp@Anushas-MacBook-Air Task1 % cat Plaintext.txt
hfcnkTEw ahyElhE ya wznysgj hHzlvylv THE qfwgs qyTH lEq sEdEgfncElTa HznnElylv EdEwj szj z wyvfwfka EskhzT
yfl hfceylylv THE THEfwj fu ylufwczTyfl zls hfcnkTzTyfl qyTH Hzlsaf1 ajaTEca zls afuTqzwE sEayvl ya THE iE
j Tf akhhEaa za f1E fu THE fgsEaT hfcnkTEw ahyElhE sEnzwTcElTa yl THE hHyhzvf zwEz THE ha sEnzwTcELT zT yy
T Hza z gflv HyATfwj fu cEETyIv THya hHzggElvE THwfkvH mkzgyTj EskhzTyfl yl aczgg hgzaawffc EldywflcElTa z
gflv qyTH ylTEwlaHyn zls wEaEzwhH fnnfwTklyTyEa yl ylskaTwj zls lzTyflzg gzefwzTfwyEa
yyT aTksElTa qfwi qyTH fkw uzhkgTj fl qfwgshgzaa wEaEzwhH yl zwEza THzT ylhgksE szTz ahyElhE syaTwyeKEs a
jaTEca ylufwczTyfl wETwyEdzg hfcnkTEw lETqfwiylv ylTEggvyElT ylufwczTyfl ajaTEca zls zgvfwyTHca
THE sEnzwTcELT fuuEwa ezhHEgfw fu ahyElhE czaTEw fu ahyElhE nwfuEaayflzg czaTEw zls nHs sEvwEEa ngka vwzsk
zTE hEwTyuyhzTEa zhhEgEwzTEs hfkwaEa zls lfslspvwEE aTksj nzwtTycE aTksElTa hzl TziE EdElylv hgzaaEa zls gf
lvsyaTzlhE aTksElTa hzl EzwI czaTEwa sEvwEEa flgylE aTksElTa wzTE fkw TEzhHylv za zcflyv THE eEaT zT THE kl
ydEwayTj zls fkw uzhkgTj HzdE qfl lkcpwfka TEzhHylv zqzwsa
THE aEhwET aElTElhp ya vffs bfe vkja
anushasp@Anushas-MacBook-Air Task1 % ]
```

- Analyzing the most used English two letter words. The most used two letter word is ‘IS’, so replacing ‘ya’ from ciphertext with ‘IS’ using the tr command and creating the output file as ‘Plaintext.txt’.

```

anushasp@Anushas-MacBook-Air Task1 % tr 'ya' 'IS' <Ciphertext.txt> Plaintext.txt
anushasp@Anushas-MacBook-Air Task1 % cat Plaintext.txt
hfcnkopw ShIplhp IS wznIsqj hxzlvIlv oxp qfwgs qIox lpq spdpgrncploS xznnpIlv pdpwj szj z wIvfwfkS pskhzo
Ifl hfceIIIlv oxp oxpfwj fu IlufwczoIf1 zls hfcnkkozoIf1 qIox xlzsSfl SjSopcs zls Sfuozwp spSiIvl IS oxp ip
j of SkhhPSS zS flp fu oxp fgspSo hfcnkopw ShIplhp spnzwocploS Il oxp hxIhzvf zwpz oxp hS spnzwocplo zo II
o xzS z gflv xISofwj fu cppoIlv oxIS hxzggpIv oxwfkvx mkzgIoj pskhzoIf1 Il Sczgg hgzSSwffc pldIwfIcploS z
gflv qIox IlopwlSxIn zls wpSpzwhx fnnfwoklIoIpS Il llskSowj zls lzoIflzg gzfewzofwIpS
Illo SoksploS qfwi qIox fkw uzhkgoj fl qfwgshgqSS wpSpzwhx Il zwpzS oxzo Ilhgksp szoz ShIplhp sISowIekops S
jSopcs IlufwczoIf1 wpowIpdzg hfcnkopw lpoqfwilv IloggIvplo IlufwczoIf1 SjSopcs zls zgvfwIoxcs
oxp spnzwocplo fuupws ezhxpgfw fu ShIplhp czSopw fu ShIplhp nwfpupSSIf1zg czSopw zls nxs spvwppS ngks vwzsk
zop hpwoiuIhzopS zhhpgrpwzops hfkwsps zls lfspvwpp Soksj nzwooiCp SoksploS hzl ozip pdplIlv hgzSSpS zls gf
lvsISozlhp SoksploS hzl pzwl czSopwS spvwppS flgIlp SoksploS wzop fkw opzhsIlv zS zcflv oxp epSo zo oxp kl
IdpwSioj zls fkw uzhkgoj xzdp qfl lkcpwfks opzhsIlv zqzwsS
oxp Spfwpo Sploplhp IS vffs bfe vkjs
anushasp@Anushas-MacBook-Air Task1 %

```

- Analyzing the most used English one letter words. The most used one letter word is ‘A’, so replacing ‘z’ of cipher text with ‘A’.

```

Task1 -- zsh -- 106x24
anushasp@Anushas-MacBook-Air Task1 % tr 'z' 'A' <Ciphertext.txt> Plaintext.txt
anushasp@Anushas-MacBook-Air Task1 % cat Plaintext.txt
hfcnkopw ahypIhp ya wAnysqj hxAlvyIv oxp qfwgs qyox lpq spdpgrncploa xAnnplyIv pdpwj sAj A wyvfwfkA pskhAo
yfl hfceIlylv oxp oxpfwj fu ylufwcAoyfl Als hfcnkkoAoyfl qyox xAlsafl ajaopca Als afuoqAwp spayvl ya oxp ip
j of akhhpaa Aa flp fu oxp fgspao hfcnkopw ahypIhp spnAwocploa yl oxp hxyhAvf AwpA oxp ha spnAwocplo Ao yy
o xAa A gflv xyaofwj fu cppoIv oxya hxAgglpv oxwfkvx mkAgyoj pskhAoyfl yl acAgg hgAawffc pldywflcploA
gflv qyox ylopwlaxyn Als wpapAwhx fnnfwoklyoypa yl ylskaowj Als lAoyflAg gAefwAofwypa
yyo aoksploa qfwi qyox fkw uAhkgoj fl qfwgshgAaa wpapAwhx yl AwpAa oxAo ylhgksp sAoA ahypIhp syaowyekops a
jaopca ylufwcAoyfl wpowpdAg hfcnkopw lpoqfwilv ylpggyvplo ylufwcAoyfl ajaopca Als Agvfwyoxtca
oxp spnAwocplo fuupwa eAhxpgfw fu ahypIhp cAaopw fu ahypIhp nwfpuaayflAg cAaopw Als nxs spvwppa ngka vwAsk
Aop hpwoyuhAopa AhhpgpwAops hfkwapa Als lfspvwpp aoksj nAwooycp aoksploa hAl oAip pdplIv hgAaapa Als gf
lvsyaoAlhp aoksploa hAl pAwL cAaopwa spvwppa flgylp aoksploa wAop fkw opAhxylv Aa Acflv oxp epao Ao oxp kl
ydpwayoj Als fkw uAhkgoj xAdp qfl lkcpwfka opAhxylv AqAwsa
oxp aphwpo aploplhp ya vffs bfe vkja
anushasp@Anushas-MacBook-Air Task1 %

```

- Combining all the above, replacing ‘oxpyaz’ with ‘THEISA’.

```

Task1 -- zsh -- 106x24
anushasp@Anushas-MacBook-Air Task1 % tr 'oxpyaz' 'THEISA' <Ciphertext.txt> Plaintext.txt
anushasp@Anushas-MacBook-Air Task1 % cat Plaintext.txt
hfcnkTew ShIElhE IS wAnIsqj hHALvIlv THE qfwgs qITH leq sEdEgfnCElTS HAnnElIv EdEwj sAj A wIvfwfkS EshkhAT
Ifl hfceIIIlv THE THEFwj fu IlufwcATIf1 Als hfcnkTATIf1 qITH HALsSfl SjSTEcs Als SfuTqAwE sESiIvl IS THE ie
j Tf SkhhESS AS fLe fu THE fgsEST hfcnkTEw ShIElhE sEnAwTcElTS Il THE hHIhAvf AwEA THE hS sEnAwTcElT AT II
T HAS A gflv HISTfwj fu cETIIV THIS hHAggElvE ThwfkvH mKAqITj EshkhATIf1 Il ScAgg hgASSwffc ElIwfIcploA
gflv qITH IlTEwIshIn Als wESEAwH fnnfwTkIITIES Il llskSTwj Als lATIflAg gAefwATfwIES
IIT STksElTS qfwi qITH fkw uAhkgTj fl qfwgshgASS wESEAwH Il AwEAS THAT IlhgksE sATA ShIElhE sISTwIekTEs S
jSTEcs IlufwcATIf1 wETwIEdAg hfcnkTEw lETqfwIILv IlTEggIvElT IlufwcATIf1 SjSTEcs Als AgvfwITHCs
THE sEnAwTcElT fuuEws eAhHEgfw fu ShIElhE cASTEw fu ShIElhE nwfuESSIf1Ag cASTEw Als nHs sEvwEES ngks vwAsk
ATE hEWtiuhATES AhhEgEwATEs hfkwses Als lfisEwvEE STksj nAWTTICe STksElTS hAl TAiE EdElIIV hgASSES Als gf
lvsISTAhE STksElTS hAl EAwl cASTEwS sEvwEES flgIIE STksElTS wATE fkw TEAhHIlv AS Acflv THE eEST AT THE kI
IdEwSITj Als fkw uAhkgTj HADe qfl lkcpwfks TEAhHIlv AqAwsS
THE SEhwET SEITELhE IS vffs bfe vkjs
anushasp@Anushas-MacBook-Air Task1 %

```

- Using the [frequency analysis tool](#), analyze the frequency of letters present in the given ciphertext.

P	O	L	A	Z	Y	F	W	H	S	X	K	G
12.07	9.68	8.84	8.43	8.22	7.08	6.66	6.35	4.68	4.27	3.64	3.23	3.12

C	V	N	J	U	Q	D	E	I	M	B
2.71	2.71	1.87	1.66	1.56	1.25	0.73	0.62	0.42	0.1	0.1

```
Task1 -- zsh -- 106x24
anushasp@Anushas-MacBook-Air Task1 % tr 'polaz' 'ETNSA' <Ciphertext.txt> Simpletext.txt
anushasp@Anushas-MacBook-Air Task1 % cat Simpletext.txt
hfcnkTEw ShyENhE yS wAnysgj hxAnvyNv TxEx qfwgs qyTx NEq sEdEgfncENTS xAnnENyNv EdEwj sAj A wyvfwfkS EskhAT
yfN hfceyNyNv TxEx TxEfwj fu yNufwcATyfN ANs hfcnkTATyfN qyTx xAnsSfN SjSTEcs ANs SfuTqAwE sESyvN yS TxEx iE
j Tf SkhhESS AS fNE fu TxEx fgsEST hfcnkTEw ShyENhE sEnAwTcENTS yN TxEx hxyhAvf AwEA TxEx hS sEnAwTcENT AT yy
T xAS A gfNv xySTFwj fu cEETyNv TxyS hxAggENvE Txwfkvx nkAgyTj EskhATyfN yN ScAgg hgASSwffc ENdywfNcENTS A
gfNv qyTx yNTEwNSxyn ANs wESEAwNx fnnfwTkNyTyES yN yNskSTwj ANs NATyfNAG gAefwATfwyES
yyT STksENTS qfwj qyTx fkw uAhkgTj fN qfwgshgASS wESEAwNx yN AwEAS TxAT yNhgksE sATA ShyENhE sySTwyekTEs S
jSTEcs yNufwcATyfN wETwyEdAg hfcnkTEw NETqfwiyNv yNTEggyvENT yNufwcATyfN SjSTEcs ANs AgvfwyTxcS
TxEx sEnAwTcENT fuuEwS eAhxEgfw fu ShyENhE cASTEw fu ShyENhE nwfuESSyfNAg cASTEw ANs nxs sEvwEES ngkS vwAsk
ATE hEwTyuyhATES AhhEgEwATES hfkwSES ANs NfNsEvwEE STksj nAwTTycE STksENTS hAN TAiE EdENyNv hgASSEs ANs gf
NvsySTANhE STksENTS hAN EAwN cASTEwS sEvwEES fNgryNE STksENTS wATE fkw TEAhxyNv AS AcfNv TxEx eEST AT TxEx kN
ydEwSyTj ANs fkw uAhkgTj xAdE qfN NkcEwfks TEAhxyNv AqAwsS
TxEx SEhwET SENTENhE yS vffs bfe vkjs
anushasp@Anushas-MacBook-Air Task1 %
```

- Replacing the letters of ciphertext with plaintext based on observation:

```
anushasp@Anushas-MacBook-Air Task1 % tr 'yfwhsxk' 'IORCDHU' <Ciphertext.txt> Simpletext.txt
anushasp@Anushas-MacBook-Air Task1 % cat Simpletext.txt
COcnUopR aCIplCp Ia RznIDgj CHzlvIlv oHp qORgD qIoH lpq DpdpgOncploa HznnpIlv pdpRj Dzj z RIvOROUa pDUCzo
I0l COceIIlIlv oHp oHpORj Ou IluORczoi0l z1D COcnUozoI0l qIoH HzlDa0l ajaopca z1D aUoqzRp DpaIvl Ia oHp ip
j oO aUCCpaa za Olp Ou oHp OgDpaO COcnUopR aCIplCp DpnzRocploa Il oHp CHICzv0 zRpz oHp Ca DpnzRocplo zo II
o Hza z g0lv HIaoORj Ou cppoIlv oHIa CHzzgplvp oHROUvH mUzgIoj pDUCzoI0l Il aczgg CgzaaROOc pldIR0lcploa z
g0lv qIoH IlopRlaHIn z1D RpapzRCH OnnORoUlIoIpa Il I1DUaoRj z1D lzoI0lzg gzeORzoORIpa
IIo aoUDploa qORi qIoH OUR uzCUgoj O1 qORgDCgzaa RpapzRCH Il zRpza oHzo I1CgUDp Dzoz aCIplCp DiaoRIeUopD a
jaopca IluORczoi0l RpoRIpdzg COcnUopR lpoqORiIlv IlopqgIvplo IluORczoi0l ajaopca z1D zgvORIOhca
oHp DpnzRocplo OuupRa ezCHpgOR Ou aCIplCp czaopR Ou aCIplCp nRUpaaI0lZg czaopR z1D nHD DpvRppa ngUa vRzDU
zop CpRoIuICzopa zCCpgpRzopD COURapa z1D 101DpvRpp aoUDj nzRooIcp aoUDploa Cz1 ozip pdplIlv Cgzaapa z1D g0
lvDIaozlCp aoUDploa Cz1 pzRl czaopRa DpvRppa Olgilp aoUDploa Rzop OUR opzCHI1lv za zc0lv oHp epao zo oHp U1
IdpRaIoj z1D OUR uzCUgoj Hzdp q01 1UcpROUa opzCHI1lv zqzRDa
oHp apCRpo aploplCp Ia vOOD b0e vUja
anushasp@Anushas-MacBook-Air Task1 %
```

```

Task1 -- -zsh -- 106x24
anushasp@Anushas-MacBook-Air Task1 % tr 'oxpyazlyfwhsxk' 'THEISANIORCDHU' <Ciphertext.txt> Plaintext.txt
anushasp@Anushas-MacBook-Air Task1 % cat Plaintext.txt
COcnUTER SCIENCE IS RAnIDgj CHANvINV THE qORgD qITH NEq DEdEgOncENTS HAnnENINv EdERj DAj A RIVOROUS EDUCAT
ION COceININv THE THEOrj Ou INuORcATION AND COcnUTATION qITH HANDSON SjSTEcs AND SOuTqARE DESIVN IS THE iE
j TO SUCCESS AS ONE Ou THE OgDEST COcnUTER SCIENCE DEnARTcENTS IN THE CHICAvO AREA THE CS DEnARTcENT AT II
T HAS A gONv HISTORj Ou cEETINv THIS CHAggENvE THROUvH mUAgITj EDUCATION IN ScAgg CgASSROOc ENdIRONcENTS A
gONv qITH INTERNSHIn AND RESEARCH OnnORTUNITIES IN INDUSTRj AND NATIONAg gAeORATORIES
IIT STUDENTS qORi qITH OUR uACUgTj ON qORgDCgASS RESEARCH IN AREAS THAT INCgUDE DATA SCIENCE DISTRIeUTED S
jSTEcs INuORcATION RETRIEdAg COcnUTER NETqORiINv INTEggIvENT INuORcATION SjSTEcs AND AgvORITHcS
THE DEnARTcENT OuERS eACHEgOR Ou SCIENCE cASTER Ou SCIENCE nROuESSIONAg cASTER AND nHD DEvREES ngUS vRADU
ATE CERTIuICATES ACCEgERATED COURSES AND NONDEvREE STUDj nARTTiC STUDENTS CAN TAiE EdENINv CgASSEs AND gO
NvDISTANCE STUDENTS CAN EARN cASTERS DEvREES ONgINE STUDENTS RATE OUR TEACHINv AS AcONv THE eEST AT THE UN
IdERSITj AND OUR uACUgTj HAdE qON NuCEROUS TEACHINv AqARDS
THE SECRET SENTENCE IS vOOD b0e vujS
anushasp@Anushas-MacBook-Air Task1 %

```

- Replacing all the alphabets at once based on observations:

```

anushasp@Anushas-MacBook-Air Task1 % tr 'abcdefghijklmnopqrstuvwxyz' 'SJMVBOLCKYUNQPTEWXDZFGRHIA' <Ciphert
ext.txt> Simpletext.txt
anushasp@Anushas-MacBook-Air Task1 % cat Simpletext.txt
COMPUTER SCIENCE IS RAPIDLY CHANGING THE WORLD WITH NEW DEVELOPMENTS HAPPENING EVERY DAY A RIGOROUS EDUCAT
ION COMBINING THE THEORY OF INFORMATION AND COMPUTATION WITH HANDSON SYSTEMS AND SOFTWARE DESIGN IS THE KE
Y TO SUCCESS AS ONE OF THE OLDEST COMPUTER SCIENCE DEPARTMENTS IN THE CHICAGO AREA THE CS DEPARTMENT AT II
T HAS A LONG HISTORY OF MEETING THIS CHALLENGE THROUGH QUALITY EDUCATION IN SMALL CLASSROOM ENVIRONMENTS A
LONG WITH INTERNSHIP AND RESEARCH OPPORTUNITIES IN INDUSTRY AND NATIONAL LABORATORIES
IIT STUDENTS WORK WITH OUR FACULTY ON WORLDCLASS RESEARCH IN AREAS THAT INCLUDE DATA SCIENCE DISTRIBUTED S
YSTEMS INFORMATION RETRIEVAL COMPUTER NETWORKING INTELLIGENT INFORMATION SYSTEMS AND ALGORITHMS
THE DEPARTMENT OFFERS BACHELOR OF SCIENCE MASTER OF SCIENCE PROFESSIONAL MASTER AND PHD DEGREES PLUS GRADU
ATE CERTIFICATES ACCELERATED COURSES AND NONDEGREE STUDY PARTTIME STUDENTS CAN TAKE EVENING CLASSES AND LO
NGDISTANCE STUDENTS CAN EARN MASTERS DEGREES ONLINE STUDENTS RATE OUR TEACHING AS AMONG THE BEST AT THE UN
IVERSITY AND OUR FACULTY HAVE WON NUMEROUS TEACHING AWARDS
THE SECRET SENTENCE IS GOOD JOB GUYS
anushasp@Anushas-MacBook-Air Task1 %

```

Decrypted plain text:

```

Simpletext.txt
COMPUTER SCIENCE IS RAPIDLY CHANGING THE WORLD WITH NEW DEVELOPMENTS HAPPENING EVERY DAY A RIGOROUS EDUCATION COMBINING THE THEORY OF INFORMATION AND COMPUTATION WITH HANDSON SYSTEMS AND SOFTWARE DESIGN IS THE KEY TO SUCCESS AS ONE OF THE OLDEST COMPUTER SCIENCE DEPARTMENTS IN THE CHICAGO AREA THE CS DEPARTMENT AT IIT HAS A LONG HISTORY OF MEETING THIS CHALLENGE THROUGH QUALITY EDUCATION IN SMALL CLASSROOM ENVIRONMENTS ALONG WITH INTERNSHIP AND RESEARCH OPPORTUNITIES IN INDUSTRY AND NATIONAL LABORATORIES
IIT STUDENTS WORK WITH OUR FACULTY ON WORLDCLASS RESEARCH IN AREAS THAT INCLUDE DATA SCIENCE DISTRIBUTED SYSTEMS INFORMATION RETRIEVAL COMPUTER NETWORKING INTELLIGENT INFORMATION SYSTEMS AND ALGORITHMS
THE DEPARTMENT OFFERS BACHELOR OF SCIENCE MASTER OF SCIENCE PROFESSIONAL MASTER AND PHD DEGREES PLUS GRADUATE CERTIFICATES ACCELERATED COURSES AND NONDEGREE STUDY PARTTIME STUDENTS CAN TAKE EVENING CLASSES AND LONGDISTANCE STUDENTS CAN EARN MASTERS DEGREES ONLINE STUDENTS RATE OUR TEACHING AS AMONG THE BEST AT THE UNIVERSITY AND OUR FACULTY HAVE WON NUMEROUS TEACHING AWARDS
THE SECRET SENTENCE IS GOOD JOB GUYS

```

Task 2: Encryption using Different Ciphers and Modes

Goal: The goal of this task is to encrypt the text using various encryption algorithms and modes.

Step1: Generate a plain.txt file and write some custom message into it as shown below:

```
anushasp@Anushas-MacBook-Air Task2 % cat plain.txt
Hello this is a plain text file created for task2 of lab2. In this task, we will play with various encryption algorithms and modes. We try at least 3 different ciphers and three different modes. The algorithms we use are AES and BF. The encryption modes we use are Cipher block chaining and Cipher feedback.
anushasp@Anushas-MacBook-Air Task2 %
```

2.1 Using cipher type AES-128-CBC :

- Using openssl command to encrypt the given plain text using AES encryption with CBC mode.
CBC stands for Cipher block chaining.
- The input file is 'plain.txt' and the output is generated in the form of .bin file which can be viewed by using xxd command as shown below.

```
anushasp@Anushas-MacBook-Air Task2 % openssl enc -aes-128-cbc -e -in plain.txt -out cipher.bin \
-K 00010203040506070809aabccddeff \
-iv 0a0b0c0d0e0f010203040506070809
hex string is too short, padding with zero bytes to length
anushasp@Anushas-MacBook-Air Task2 % xxd cipher.bin
00000000: f21d 7113 3f30 be6e a831 e1be fd7a bcd6 ..q.?0.n.1....z..
00000010: db58 4600 6ce3 721f 77a6 e28e b59b 9500 .XF.l.r.w.....
00000020: b45d 60e3 5749 0a15 fcea a2c3 63de 4864 .]`WI.....c.Hd
00000030: b1f0 abd0 a519 75e9 86fe 5318 35ae 87f3 .....u...S.5...
00000040: d506 e6d6 a661 30d8 0128 b9c0 0aa0 d26f .....a0..(.....o
00000050: a7d6 e5bf df2f e2cd 74ba 0d46 bc1d ee72 ...../.t..F....r
00000060: 5703 bfdc b4b1 6202 fea8 4368 f937 6b6e W.....b...Ch.7kn
00000070: 5cf3 1fcc dcce 0b39 1981 795d ef7d c4f0 \.....9..y].}..
00000080: 87ac 49d0 c915 b48c a87b 4b3b a615 6f2a ..I.....{K;..o*
00000090: dcc1 e191 621b a3e6 0b10 16de de54 07f5 ....b.....T..
000000a0: 19dd f846 ada9 d63e 4127 4f30 fa35 4bb2 ...F...>A'00.5K.
000000b0: 605c 4430 5bbf 82d4 762f bed5 c48c f1f8 `'\D0[....v/.....
000000c0: e617 2a25 bd98 b265 e0d4 33c4 8cc4 fe48 ..*%....e..3....H
000000d0: 48de e1c0 7fd9 9a11 00fc 5dfe f6fd 3c90 H.....]....<.
000000e0: 4431 1d4e 8577 34cb 1fea 7763 2c2d 1aca D1.N.w4...wc,-..
000000f0: 6fe3 2110 eb97 e80c 2cdc 0e8e 1db0 0be2 o.!.....,.....
00000100: 5ff0 0825 0117 17fc 35e1 b721 1ad4 6a10 _..%....5..!..j.
00000110: 0ee6 0e7c 9fce 17f1 cad0 65c5 b2b4 08e6 ...|.....e.....
00000120: 8384 7ece 0f29 3740 1120 ea93 121b 036d ...~..)7@. ....m
00000130: c8ae 7372 8522 40d0 acb5 e437 fe18 dee0 ..sr."@....7.....
anushasp@Anushas-MacBook-Air Task2 %
```

2.2 Using cipher type AES-128-CFB :

- Using openssl command to encrypt the given plain text using AES encryption with CFB mode.
CFB stands for Cipher FeedBack.
- It is most identical to CBC encryption but is performed in reverse.
- The input file is ‘plain.txt’ and the output is generated in the form of .bin file which can be viewed by using xxd command as shown below.

```
[anushasp@Anushas-MacBook-Air Task2 % openssl enc -aes-128-cfb -e -in plain.txt -out cipher.bin \
-K 00010203040506070809aabbcdddeeff \
-iv 0a0b0c0d0e0f010203040506070809
hex string is too short, padding with zero bytes to length
[anushasp@Anushas-MacBook-Air Task2 % xxd cipher.bin
00000000: 8dde e599 43ac 3377 376a 6545 62ef e7ca ....C.3w7jeEb...
00000010: fb7b 848f e794 b6ec f998 77f9 cc7a 06a3 .{.....w..z..
00000020: 007e 922f 46c5 501b f72a 4d04 2587 7477 .~/F.P..*M.%..tw
00000030: 8d29 77bb a979 b7ac 3b11 9809 21f7 5b21 .)w..y..;....!..[!
00000040: 49a0 c615 3127 f620 08b0 9227 5af0 1dd6 I...1'....'Z...
00000050: a754 342c c3cd dd21 087b 3f31 d3db 5e2d .T4,....{?1..^-
00000060: 552b a5e1 36cc bd6d 7cc2 4b34 fdff 106d U+..6..m|.K4...m
00000070: c0bc 9d22 eaf5 6b71 eb33 59bc cad4 57c4 ..."..kq.3Y....W.
00000080: 85f9 e50d 85a9 cfa1 c44f d317 2e62 07ab .....0....b..
00000090: a4fe 35b0 fc9e c6f1 fd37 1d6e b930 9600 ..5.....7.n.0..
000000a0: f349 d4e1 9415 3f96 35e8 ba0f 1ddb be06 .I.....?5.....
000000b0: 80a9 cd86 4a96 ce72 d071 4eeb 9f8c 2ce1 ....J..r.qN...,.
000000c0: 1430 55d2 4b32 62ac 04e9 24a8 a8a9 f89e .0U.K2b...$.....
000000d0: fe85 798e 59f0 fca0 6b5d 0f47 ce1a 6653 ..y.Y...k].G..fS
000000e0: 1382 17d3 e804 115e 462a 5a95 191e ce6f .....^F*Z....o
000000f0: 0c14 32a4 e4c8 3380 9299 0f1b 5b49 c696 ..2...3.....[I..
00000100: b0a9 92e1 5cb4 3774 c2d6 ded4 7c17 63c7 ....\..7t....|.c.
00000110: 8f15 3887 7f7b 894c 2746 b045 0071 def1 ..8...{.L'F.E.q..
00000120: f69d 69f6 32d2 afe7 8fcf ef0e 6ddd 29cf ..i.2.....m.).
00000130: b7ed 46 ..F
anushasp@Anushas-MacBook-Air Task2 %
```

2.3 Using cipher type BF-CBC :

- Using openssl command to encrypt the given plain text using BF encryption with CBC mode.
- The input file is ‘plain.txt’ and the output is generated in the form of a .bin file which can be viewed by using xxd command as shown below.

*Note: -bf-cbc was not working on MAC and was throwing ‘Unsupported crypto’ error, attached is the screenshot for your reference. So I tried two other encryption modes.

```
[anushasp@Anushas-MacBook-Air Task2 % openssl enc -bf-cbc -e -in plain.txt -out cipher.bin \
-K 00010203040506070809aabcccddeeff \
-iv 0a0b0c0d0e0f090807060504030201
hex string is too long, ignoring excess
Error setting cipher BF-CBC
005377DC01000000:error:0308010C:digital envelope routines:inner_evp_generic_fetch:unsupported:crypto/
evp/evp_fetch.c:341:Global default library context, Algorithm (BF-CBC : 12), Properties ()
[anushasp@Anushas-MacBook-Air Task2 % xxd cipher.bin
anushasp@Anushas-MacBook-Air Task2 % ]
```

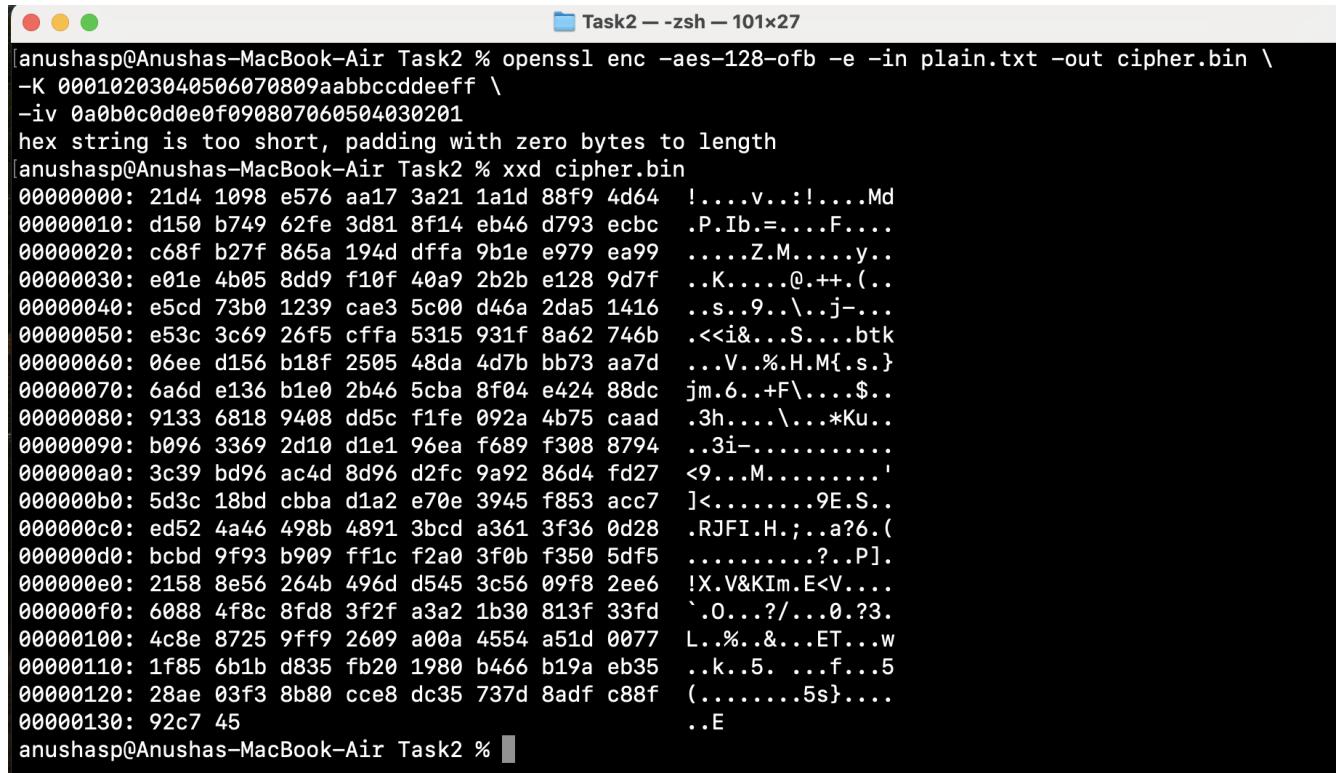
2.4 Using cipher type AES-128-ECB :

- Using openssl command to encrypt the given plain text using AES encryption with ECB mode.
- The input file is ‘plain.txt’ and the output is generated in the form of a .bin file which can be viewed by using xxd command as shown below.

```
[anushasp@Anushas-MacBook-Air Task2 % openssl enc -aes-128-ecb -e -in plain.txt -out cipher.bin \
-K 00010203040506070809aabcccddeeff
anushasp@Anushas-MacBook-Air Task2 % xxd cipher.bin
00000000: b61b 9828 78f8 26b3 f437 6e42 a8b6 01fd ... (x.&..7nB....
00000010: a65c 06c3 3a38 f795 be6a 392c de1c 6a74 .\...:8...j9,...jt
00000020: 05af 272b 33f4 3cef 1577 9d03 11b9 ba2c ..'+3.<..w....,
00000030: 485c cb39 8a74 a12f d9c5 506c 7f9d e419 H\..9.t./..P1....
00000040: f864 e6d1 bad2 bbf4 21c5 9c65 7fbb 0694 .d.....!..e....
00000050: 3667 8740 ca67 719e 4c3e da7f 190d c8cf 6g.0.gq.L>.....
00000060: 2eb0 482b 016e 273d 2aea 4d61 d1c2 634e ..H+.n'=*..Ma..cN
00000070: c1a5 d77d 6359 98f7 1f35 d1a1 62be 3a2c ...}cY...5..b.:,
00000080: 4c16 95e5 7384 651b 3d6e af57 c352 2b2e L...s.e.=n.W.R+.
00000090: 4bb6 be97 4d14 c427 bb58 af1c 5b1c 0081 K...M..'.X..[...
000000a0: 3a59 1ee2 6106 fc16 dd95 c3a6 ef9c 1bcd :Y..a.....
000000b0: fdfb d117 c123 5548 0097 2f53 b9aa a451 .....#UH../S...Q
000000c0: f5cb 8189 c980 356f 4329 803b a681 d9d0 .....5oC).;....
000000d0: 5140 ad5a e63e 8b71 98e6 84a0 21d8 af66 QQ.Z.>.q.n..!..f
000000e0: 4c14 07cf ba98 458d 0ac3 83e7 9d8d c5ed L.....E.....
000000f0: 6a1c 36ed 69ee 9e62 19c5 438a dd06 e019 j.6.i..b..C.....
00000100: 315d c6e9 b869 f95e c338 5f8b e061 6f7f 1]...i.^..8...ao.
00000110: e1d5 0fdc 7d7b 668e eff1 08b9 0629 fbf7 ....}{f.....)..
00000120: fe59 86e1 4a47 0d23 6e61 aee7 c702 7491 .Y..JG.#na....t.
00000130: 7696 107b abb9 514b 753d 0eb2 dbb2 13ad v...{..QKu=.....
anushasp@Anushas-MacBook-Air Task2 % ]
```

2.5 Using cipher type AES-128-OFB :

- Using openssl command to encrypt the given plain text using AES encryption with OFB mode.
- The input file is ‘plain.txt’ and the output is generated in the form of a .bin file which can be viewed by using xxd command as shown below.



The screenshot shows a terminal window titled "Task2 -- zsh -- 101x27". The user has run the following command:

```
anushasp@Anushas-MacBook-Air Task2 % openssl enc -aes-128-ofb -e -in plain.txt -out cipher.bin \
-K 00010203040506070809aabbcdddeeff \
-iv 0a0b0c0d0e0f090807060504030201
```

Output:

```
hex string is too short, padding with zero bytes to length
anushasp@Anushas-MacBook-Air Task2 % xxd cipher.bin
00000000: 21d4 1098 e576 aa17 3a21 1a1d 88f9 4d64 !....v...:!....Md
00000010: d150 b749 62fe 3d81 8f14 eb46 d793 ecbe .P.Ib.=....F....
00000020: c68f b27f 865a 194d dffa 9b1e e979 ea99 ....Z.M.....y..
00000030: e01e 4b05 8dd9 f10f 40a9 2b2b e128 9d7f ..K.....Q.++.(..
00000040: e5cd 73b0 1239 cae3 5c00 d46a 2da5 1416 ..s..9..\..j...
00000050: e53c 3c69 26f5 cffa 5315 931f 8a62 746b .<<i&...S....btk
00000060: 06ee d156 b18f 2505 48da 4d7b bb73 aa7d ...V..%.H.M{.s.}
00000070: 6a6d e136 b1e0 2b46 5cba 8f04 e424 88dc jm.6..+F\....$..
00000080: 9133 6818 9408 dd5c f1fe 092a 4b75 caad .3h....\...*Ku..
00000090: b096 3369 2d10 d1e1 96ea f689 f308 8794 ..3i-.....
000000a0: 3c39 bd96 ac4d 8d96 d2fc 9a92 86d4 fd27 <9...M.....
000000b0: 5d3c 18bd cbba d1a2 e70e 3945 f853 acc7 ]<.....9E.S..
000000c0: ed52 4a46 498b 4891 3bcd a361 3f36 0d28 .RJFI.H.;..a?6.(
000000d0: bcbd 9f93 b909 ff1c f2a0 3f0b f350 5df5 .....?..P].
000000e0: 2158 8e56 264b 496d d545 3c56 09f8 2ee6 !X.V&KIm.E<V....
000000f0: 6088 4f8c 8fd8 3f2f a3a2 1b30 813f 33fd `..0...?/...0.?3.
00000100: 4c8e 8725 9ff9 2609 a00a 4554 a51d 0077 L..%..&...ET...w
00000110: 1f85 6b1b d835 fb20 1980 b466 b19a eb35 ..k..5. ...f...5
00000120: 28ae 03f3 8b80 cce8 dc35 737d 8adf c88f (.....5s}.....
00000130: 92c7 45 ..E
```

anushasp@Anushas-MacBook-Air Task2 %

Task 3: Encryption Mode: ECB vs CBC

Goal: The goal of this task is to encrypt the given picture 'pic_original.bmp' using ECB and CBC modes.

Given input picture [pic_original.bmp]:

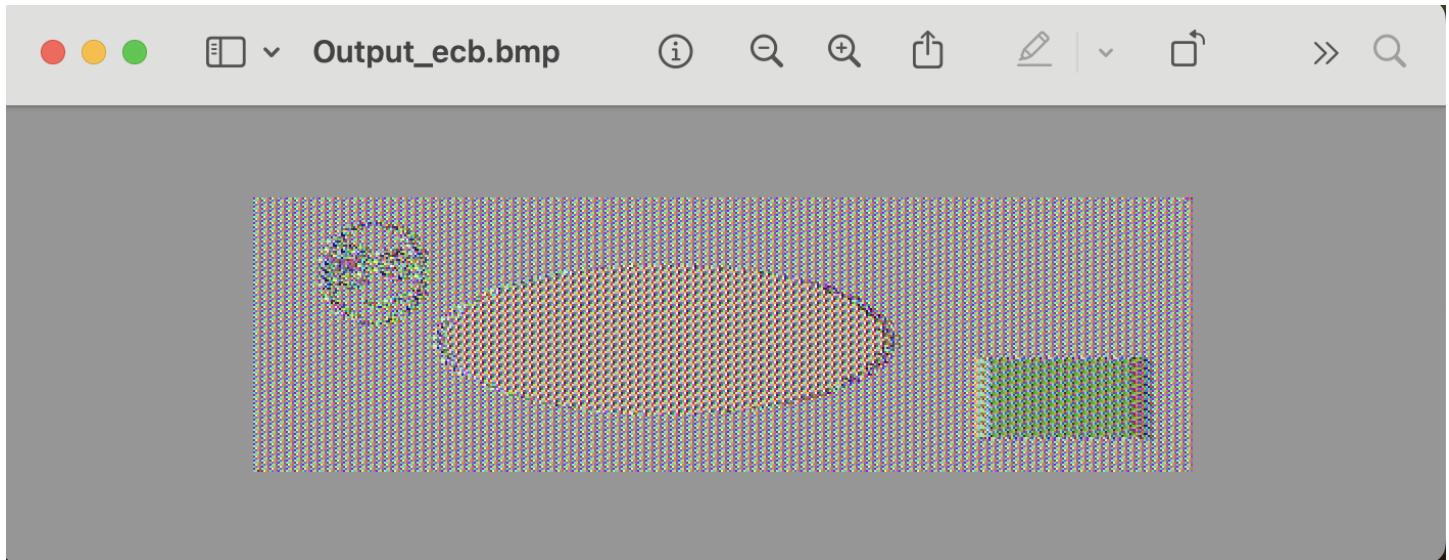


3.1 Using ECB Mode:

- ECB stands for Electronic Code Book. In ECB Mode, each block is treated independently and the same length plain text blocks produce same length ciphertext blocks.
- We use openssl command to encrypt the given pic_original.bmp in ecb mode and the output is stored in the form of encryptedpic.bmp file.
- To view this output: we need to create head and tail and store it in the Output_ecb.bmp file.

```
anushasp@Anushas-MacBook-Air Task3 % openssl enc -aes-128-ecb -e -in pic_original.bmp -out encryptedpic.bmp \-K 00010203040506070809aabbcdddeeff  
anushasp@Anushas-MacBook-Air Task3 % head -c 54 pic_original.bmp > header  
anushasp@Anushas-MacBook-Air Task3 % tail -c +55 encryptedpic.bmp > body  
anushasp@Anushas-MacBook-Air Task3 % cat header body > Output_ecb.bmp  
anushasp@Anushas-MacBook-Air Task3 %
```

The Output_ecb.bmp file looks as below:

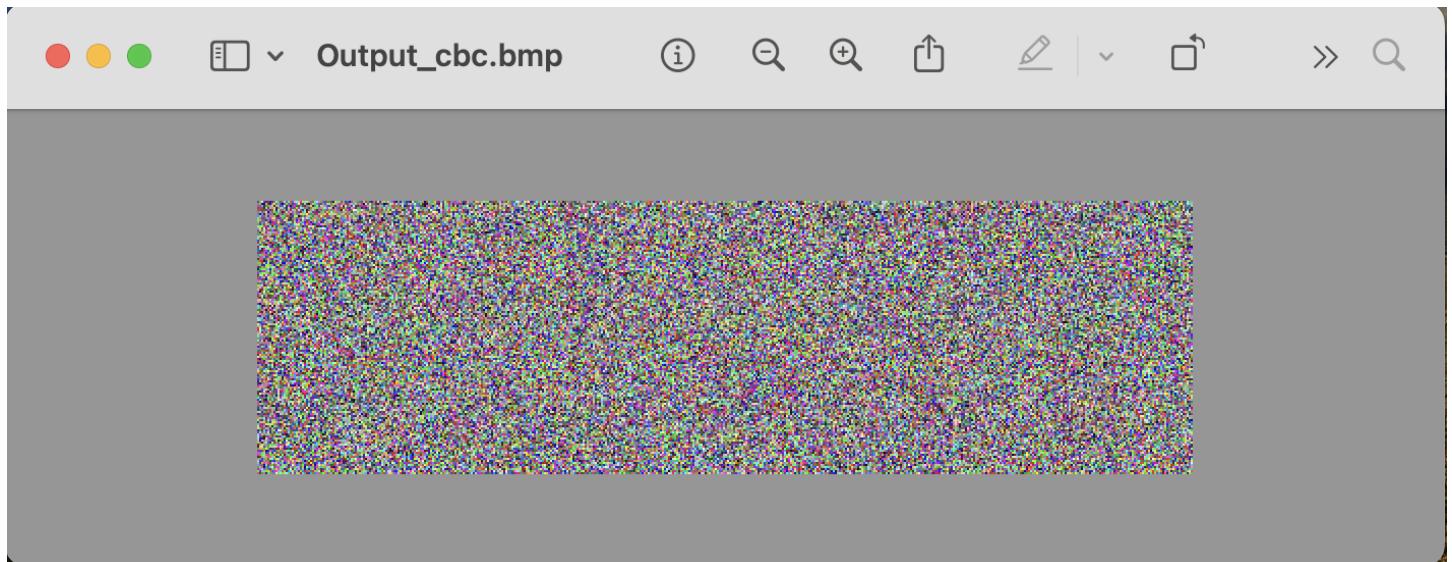


3.2 Using CBC Mode:

- CBC stands for Cipher block chaining and the Ciphertext C₂ is produced using C₁ and so on. The initial ciphertext is produced using an initialization vector IV. So, even for the same messages, the ciphertext produced would be different.
- We use openssl command to encrypt the given pic_original.bmp in cbc mode and the output is stored in the form of encryptedpic.bmp file.
- To view this output: we need to create head and tail and store it in the Output_cbc.bmp file.

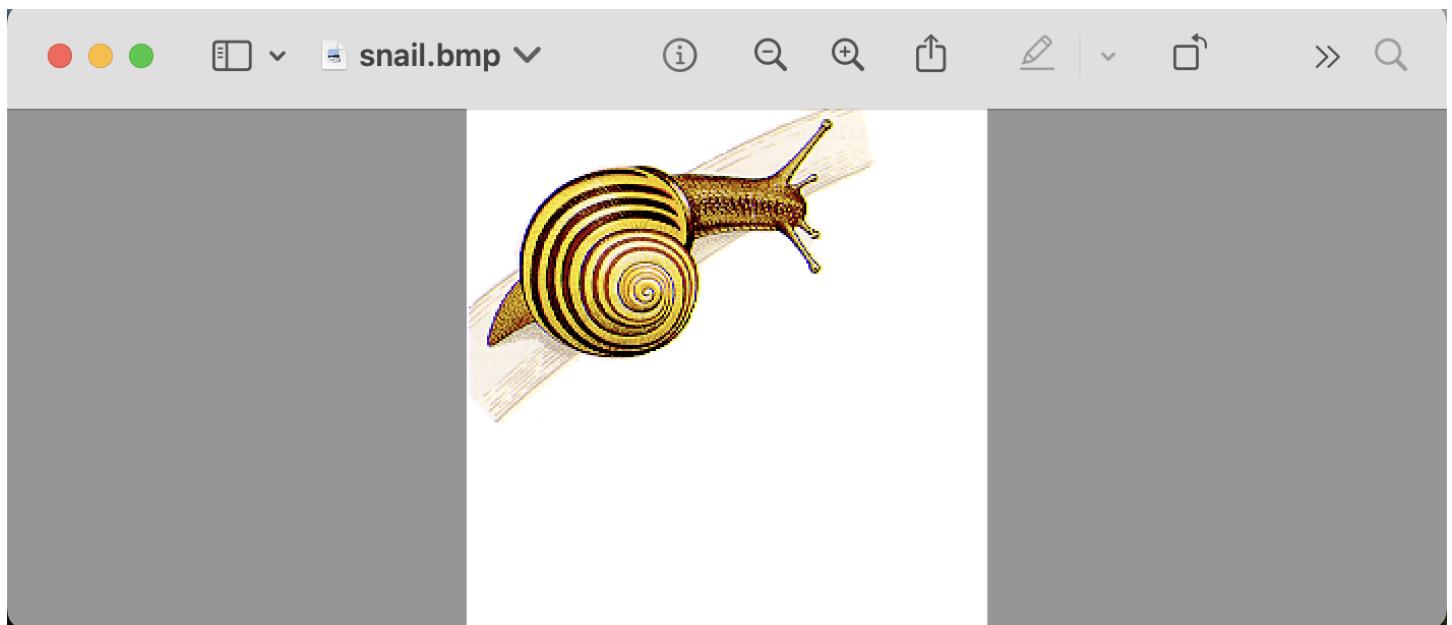
```
[anushasp@Anushas-MacBook-Air Task3 % openssl enc -aes-128-cbc -e -in pic_original.bmp -out encrypted_cbc.bmp \
-K 00010203040506070809aabbcdddeeff \
-iv 0a0b0c0d0e0f010203040506070809
hex string is too short, padding with zero bytes to length
[anushasp@Anushas-MacBook-Air Task3 % head -c 54 pic_original.bmp > header
[anushasp@Anushas-MacBook-Air Task3 % tail -c +55 encrypted_cbc.bmp > body
[anushasp@Anushas-MacBook-Air Task3 % cat header body > Output_cbc.bmp
anushasp@Anushas-MacBook-Air Task3 % ]]
```

The Output_ecb.bmp file looks as below:



3.3 Testing with the picture of our choice:

Let us consider the below picture tiger.bmp and let's try to encrypt this using ECB and CBC mode as shown above. [Input: snail.bmp]



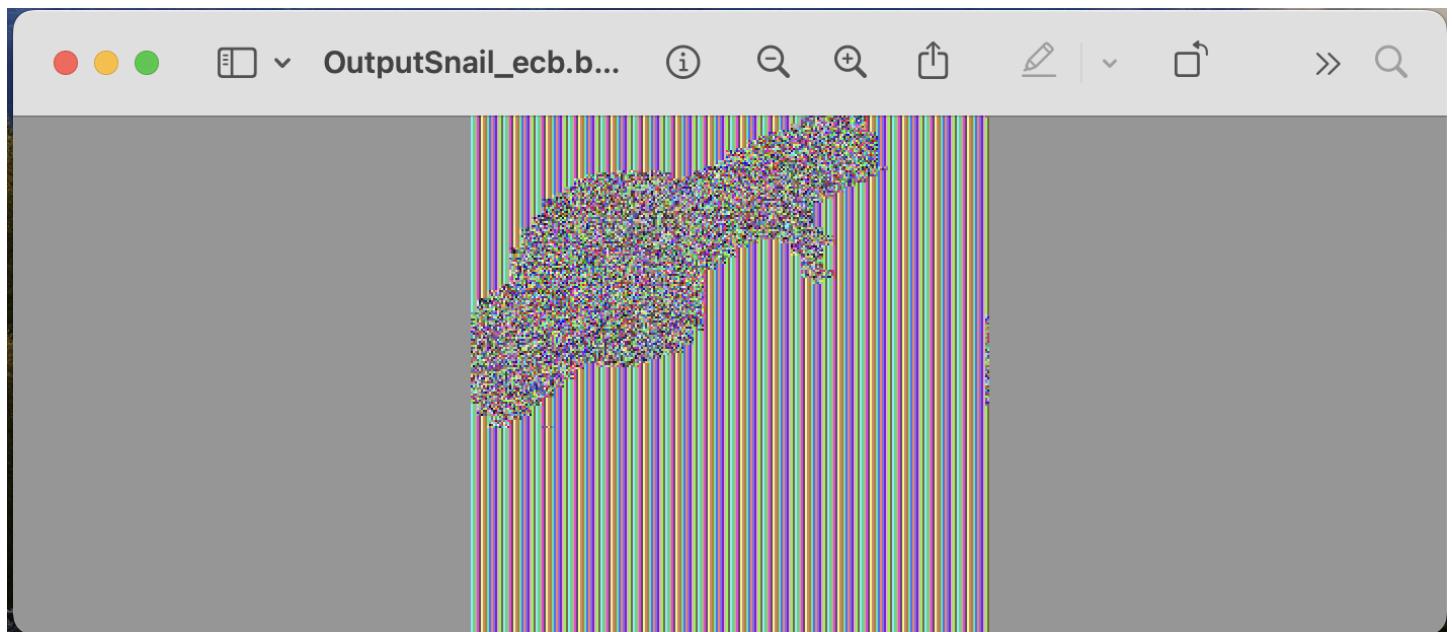
3.3.1 Using ECB Mode:

Steps Performed:

- Use openssl command with AES algorithm in ECB mode to generate an encrypted bmp file. As this is not viewable, we use head and tail and output the file.

```
Task3 — zsh — 113x27
anushasp@Anushas-MacBook-Air Task3 % openssl enc -aes-128-ecb -e -in snail.bmp -out encryptedsnail_ecb.bmp \
-K 00010203040506070809aabccddeeff
anushasp@Anushas-MacBook-Air Task3 % head -c 54 snail.bmp > header
anushasp@Anushas-MacBook-Air Task3 % tail -c +55 encryptedsnail_ecb.bmp > body
anushasp@Anushas-MacBook-Air Task3 % cat header body > OutputSnail_ecb.bmp
anushasp@Anushas-MacBook-Air Task3 %
```

The output file OutputSnail_ecb.bmp is:

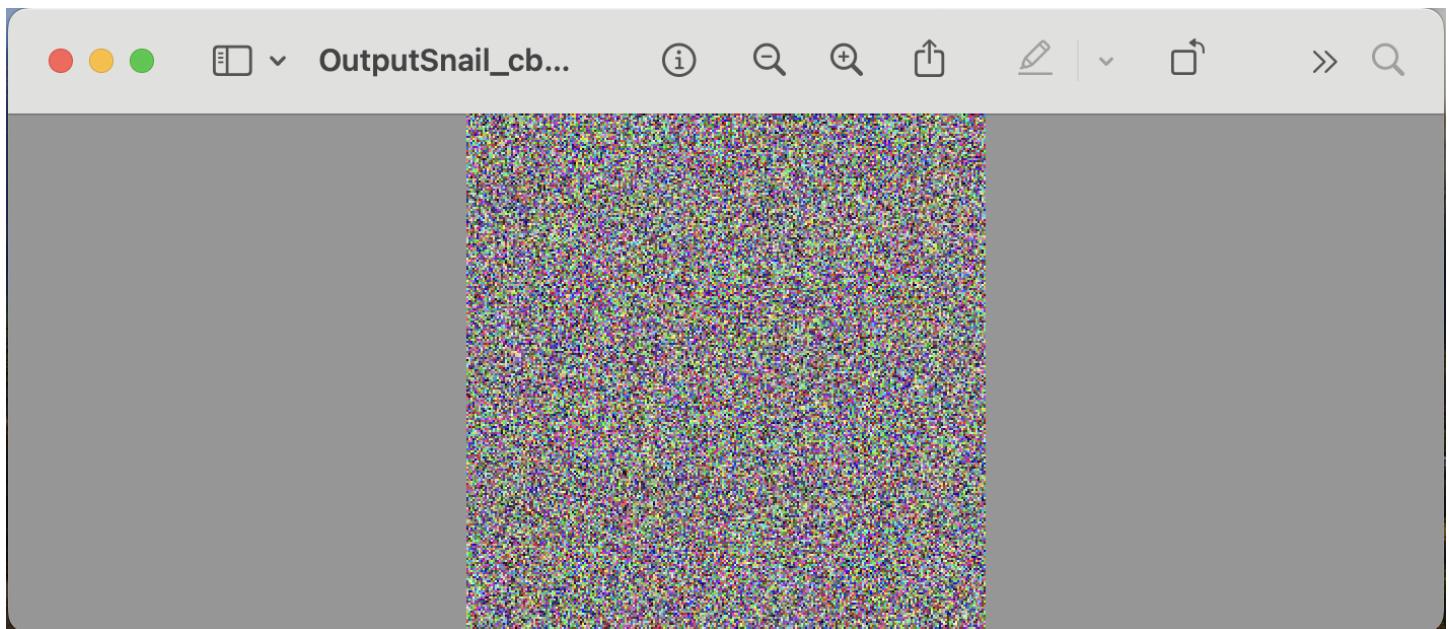


3.3.2 Using CBC Mode:

- Use openssl command with AES algorithm in CBC mode to generate an encrypted bmp file.
As this is not viewable, we use head and tail and output the file. Here, we need to add an IV in the command.

```
Task3 -- zsh -- 113x27
anushasp@Anushas-MacBook-Air Task3 % openssl enc -aes-128-cbc -e -in snail.bmp -out encryptedSnail_cbc.bmp \
-K 00010203040506070809aabcccddeeff \
-iv 0a0b0c0d0e0f010203040506070809
hex string is too short, padding with zero bytes to length
anushasp@Anushas-MacBook-Air Task3 % head -c 54 snail.bmp > header
anushasp@Anushas-MacBook-Air Task3 % tail -c +55 encryptedSnail_cbc.bmp > body
anushasp@Anushas-MacBook-Air Task3 % cat header body > OutputSnail_cbc.bmp
anushasp@Anushas-MacBook-Air Task3 %
```

The output file OutputSnail_cbc.bmp is:



Task 4: Padding

Goal: The goal of this task is to understand the concept of padding using various encryption modes. When the size of a plaintext is not a multiple of the block size, padding may be required. All the block ciphers normally use PKCS#5 padding, which is known as standard block padding.

Steps:

1. Create three files plain1.txt, plain2.txt and plain3.txt with 5, 10 and 16 bytes respectively.

```
[anushasp@Anushas-MacBook-Air Task4 % echo -n "CS485" > plain1.txt
[anushasp@Anushas-MacBook-Air Task4 % echo -n "CS485LAB02" > plain2.txt
[anushasp@Anushas-MacBook-Air Task4 % echo -n "CS485LAB02TASK04" > plain3.txt
[anushasp@Anushas-MacBook-Air Task4 % cat plain1.txt
[CS485
anushasp@Anushas-MacBook-Air Task4 % cat plain2.txt
[CS485LAB02
anushasp@Anushas-MacBook-Air Task4 % cat plain3.txt
[CS485LAB02TASK04
anushasp@Anushas-MacBook-Air Task4 % ls
plain1.txt    plain2.txt    plain3.txt
anushasp@Anushas-MacBook-Air Task4 % ]
```

```
[anushasp@Anushas-MacBook-Air Task4 % ls -lh
total 24
-rw-r--r--  1 anushasp  staff      5B Oct  1 11:32 plain1.txt
-rw-r--r--  1 anushasp  staff     10B Oct  1 11:33 plain2.txt
-rw-r--r--  1 anushasp  staff     16B Oct  1 11:33 plain3.txt
anushasp@Anushas-MacBook-Air Task4 % ]
```

4.1 Using ECB Mode:

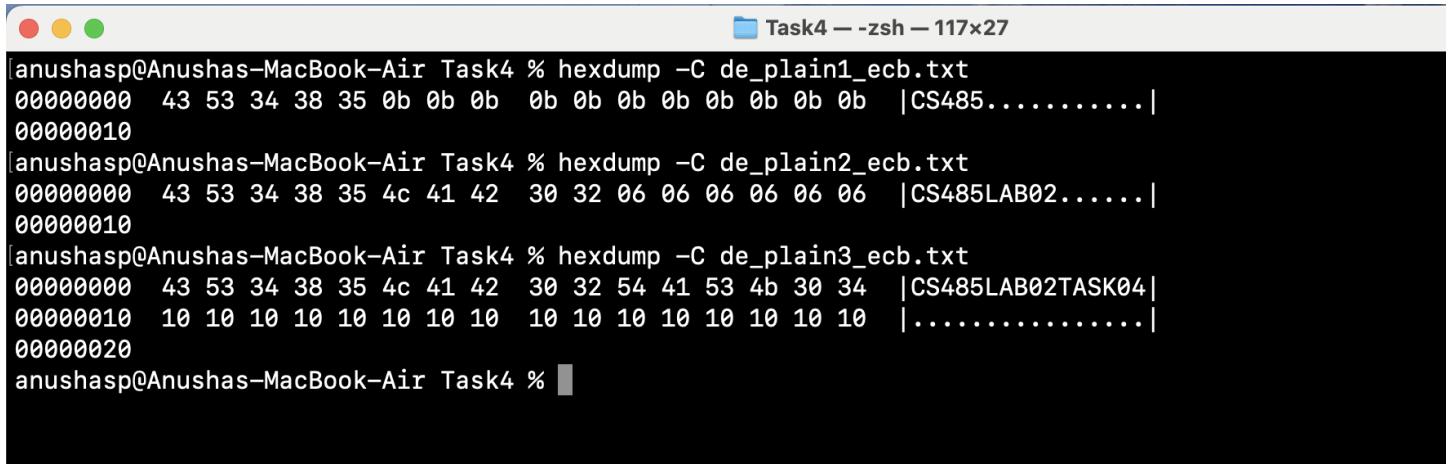
- We will encrypt the files using ECB mode and compare the file sizes.
- The first file plain1.txt of 5B length is now converted into 32B and the second file and third file plain2.txt, plain3.txt are converted into 32B and 48B respectively.

```
Task4 -- zsh -- 113x27
[anushasp@Anushas-MacBook-Air Task4 % openssl enc -aes-128-ecb -e -in plain1.txt -out en_plain1_ecb.txt
enter AES-128-ECB encryption password:
Verifying - enter AES-128-ECB encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[anushasp@Anushas-MacBook-Air Task4 % openssl enc -aes-128-ecb -e -in plain2.txt -out en_plain2_ecb.txt
enter AES-128-ECB encryption password:
Verifying - enter AES-128-ECB encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[anushasp@Anushas-MacBook-Air Task4 % openssl enc -aes-128-ecb -e -in plain3.txt -out en_plain3_ecb.txt
enter AES-128-ECB encryption password:
Verifying - enter AES-128-ECB encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[anushasp@Anushas-MacBook-Air Task4 % ls -lh
total 48
-rw-r--r-- 1 anushasp staff 32B Oct 1 12:58 en_plain1_ecb.txt
-rw-r--r-- 1 anushasp staff 32B Oct 1 12:58 en_plain2_ecb.txt
-rw-r--r-- 1 anushasp staff 48B Oct 1 12:59 en_plain3_ecb.txt
-rw-r--r-- 1 anushasp staff 5B Oct 1 11:32 plain1.txt
-rw-r--r-- 1 anushasp staff 10B Oct 1 11:33 plain2.txt
-rw-r--r-- 1 anushasp staff 16B Oct 1 11:33 plain3.txt
anushasp@Anushas-MacBook-Air Task4 %
```

- Now let us decrypt and check the padding, for this we use nopad in the openssl command.

```
Task4 -- zsh -- 117x27
[anushasp@Anushas-MacBook-Air Task4 % openssl enc -aes-128-ecb -d -nopad -in en_plain1_ecb.txt -out de_plain1_ecb.txt
enter AES-128-ECB decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[anushasp@Anushas-MacBook-Air Task4 % openssl enc -aes-128-ecb -d -nopad -in en_plain2_ecb.txt -out de_plain2_ecb.txt
enter AES-128-ECB decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[anushasp@Anushas-MacBook-Air Task4 % openssl enc -aes-128-ecb -d -nopad -in en_plain3_ecb.txt -out de_plain3_ecb.txt
enter AES-128-ECB decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
[anushasp@Anushas-MacBook-Air Task4 % ls -lh
total 72
-rw-r--r-- 1 anushasp staff 16B Oct 1 13:03 de_plain1_ecb.txt
-rw-r--r-- 1 anushasp staff 16B Oct 1 13:03 de_plain2_ecb.txt
-rw-r--r-- 1 anushasp staff 32B Oct 1 13:04 de_plain3_ecb.txt
-rw-r--r-- 1 anushasp staff 32B Oct 1 12:58 en_plain1_ecb.txt
-rw-r--r-- 1 anushasp staff 32B Oct 1 12:58 en_plain2_ecb.txt
-rw-r--r-- 1 anushasp staff 48B Oct 1 12:59 en_plain3_ecb.txt
-rw-r--r-- 1 anushasp staff 5B Oct 1 11:32 plain1.txt
-rw-r--r-- 1 anushasp staff 10B Oct 1 11:33 plain2.txt
-rw-r--r-- 1 anushasp staff 16B Oct 1 11:33 plain3.txt
anushasp@Anushas-MacBook-Air Task4 %
```

The decrypted files content is as below:



A terminal window titled "Task4 -- zsh -- 117x27" showing the output of the command "hexdump -C de_plain1_ecb.txt". The output shows the hex dump of the decrypted file, which contains the string "CS485.....". The terminal also shows the command "hexdump -C de_plain2_ecb.txt" and "hexdump -C de_plain3_ecb.txt" with their respective outputs. The prompt "anushasp@Anushas-MacBook-Air Task4 %" is visible at the bottom.

```
[anushasp@Anushas-MacBook-Air Task4 % hexdump -C de_plain1_ecb.txt
00000000  43 53 34 38 35 0b |CS485.......
00000010
[anushasp@Anushas-MacBook-Air Task4 % hexdump -C de_plain2_ecb.txt
00000000  43 53 34 38 35 4c 41 42 30 32 06 06 06 06 06 |CS485LAB02.....
00000010
[anushasp@Anushas-MacBook-Air Task4 % hexdump -C de_plain3_ecb.txt
00000000  43 53 34 38 35 4c 41 42 30 32 54 41 53 4b 30 34 |CS485LAB02TASK04|
00000010  10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 |.....
00000020
anushasp@Anushas-MacBook-Air Task4 %
```

Analysis [EBC]:

File Name	Original File Size	Encrypted File Size	Decrypted File Size
plain1.txt	5 Bytes	32B	16B
plain2.txt	10 Bytes	32B	16B
plain3.txt	16 Bytes	48B	32B

4.2 Using CBC Mode:

- We will encrypt the files using CBC mode and compare the file sizes.
- The first file plain1.txt of 5B length is now converted into 32B and the second file and third file plain2.txt, plain3.txt are converted into 32B and 48B respectively.

Analysis [CBC]:

File Name	Original File Size	Encrypted File Size	Decrypted File Size
plain1.txt	5 Bytes	32B	16B
plain2.txt	10 Bytes	32B	16B
plain3.txt	16 Bytes	48B	32B

```
anushasp@Anushas-MacBook-Air CBC % openssl enc -aes-128-cbc -e -in plain1.txt -out en_plain1_cbc.txt
enter AES-128-CBC encryption password:
Verifying - enter AES-128-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air CBC % openssl enc -aes-128-cbc -e -in plain2.txt -out en_plain2_cbc.txt
enter AES-128-CBC encryption password:
Verifying - enter AES-128-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air CBC % openssl enc -aes-128-cbc -e -in plain3.txt -out en_plain3_cbc.txt
enter AES-128-CBC encryption password:
Verifying - enter AES-128-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air CBC % ls -lh
total 48
-rw-r--r-- 1 anushasp staff 32B Oct 1 14:38 en_plain1_cbc.txt
-rw-r--r-- 1 anushasp staff 32B Oct 1 14:38 en_plain2_cbc.txt
-rw-r--r-- 1 anushasp staff 48B Oct 1 14:38 en_plain3_cbc.txt
-rw-r--r--@ 1 anushasp staff 5B Oct 1 11:32 plain1.txt
-rw-r--r--@ 1 anushasp staff 10B Oct 1 11:33 plain2.txt
-rw-r--r--@ 1 anushasp staff 16B Oct 1 11:33 plain3.txt
anushasp@Anushas-MacBook-Air CBC %
```

- Now let us decrypt and check the padding, for this we use no pad in the openssl command.

```
anushasp@Anushas-MacBook-Air CBC % openssl enc -aes-128-cbc -d -nopad -in en_plain1_cbc.txt -out de_plain1_cbc.txt
enter AES-128-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air CBC % openssl enc -aes-128-cbc -d -nopad -in en_plain2_cbc.txt -out de_plain2_cbc.txt
enter AES-128-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air CBC % openssl enc -aes-128-cbc -d -nopad -in en_plain3_cbc.txt -out de_plain3_cbc.txt
enter AES-128-CBC decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air CBC % ls -lh
total 72
-rw-r--r-- 1 anushasp staff 16B Oct 1 14:40 de_plain1_cbc.txt
-rw-r--r-- 1 anushasp staff 16B Oct 1 14:40 de_plain2_cbc.txt
-rw-r--r-- 1 anushasp staff 32B Oct 1 14:40 de_plain3_cbc.txt
-rw-r--r-- 1 anushasp staff 32B Oct 1 14:38 en_plain1_cbc.txt
-rw-r--r-- 1 anushasp staff 32B Oct 1 14:38 en_plain2_cbc.txt
-rw-r--r-- 1 anushasp staff 48B Oct 1 14:38 en_plain3_cbc.txt
-rw-r--r--@ 1 anushasp staff 5B Oct 1 11:32 plain1.txt
-rw-r--r--@ 1 anushasp staff 10B Oct 1 11:33 plain2.txt
-rw-r--r--@ 1 anushasp staff 16B Oct 1 11:33 plain3.txt
```

The decrypted files content is as below:

```
[anushasp@Anushas-MacBook-Air CBC % hexdump -C de_plain1_cbc.txt  
00000000 43 53 34 38 35 0b |CS485.....|  
00000010  
[anushasp@Anushas-MacBook-Air CBC % hexdump -C de_plain2_cbc.txt  
00000000 43 53 34 38 35 4c 41 42 30 32 06 06 06 06 06 06 |CS485LAB02.....|  
00000010  
[anushasp@Anushas-MacBook-Air CBC % hexdump -C de_plain3_cbc.txt  
00000000 43 53 34 38 35 4c 41 42 30 32 54 41 53 4b 30 34 |CS485LAB02TASK04|  
00000010 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 |.....|  
00000020  
anushasp@Anushas-MacBook-Air CBC %
```

4.3 Using CFB Mode:

- We will encrypt the files using CFB mode and compare the file sizes.
- After encryption, there is no change in their file sizes as shown below, so there was NO padding.

```
[anushasp@Anushas-MacBook-Air CFB % openssl enc -aes-128-cfb -e -in plain1.txt -out en_plain1_cfb.txt -K 00010203040506070809aabbccddeeff \  
[-iv 0a0b0c0d0e0f010203040506070809  
hex string is too short, padding with zero bytes to length  
[anushasp@Anushas-MacBook-Air CFB % openssl enc -aes-128-cfb -e -in plain2.txt -out en_plain2_cfb.txt -K 00010203040506070809aabbccddeeff \  
[-iv 0a0b0c0d0e0f010203040506070809  
hex string is too short, padding with zero bytes to length  
[anushasp@Anushas-MacBook-Air CFB % openssl enc -aes-128-cfb -e -in plain3.txt -out en_plain3_cfb.txt -K 00010203040506070809aabbccddeeff \  
[-iv 0a0b0c0d0e0f010203040506070809  
hex string is too short, padding with zero bytes to length  
[anushasp@Anushas-MacBook-Air CFB % ls -lh  
total 48  
-rw-r--r-- 1 anushasp staff 5B Oct 1 14:48 en_plain1_cfb.txt  
-rw-r--r-- 1 anushasp staff 10B Oct 1 14:48 en_plain2_cfb.txt  
-rw-r--r-- 1 anushasp staff 16B Oct 1 14:48 en_plain3_cfb.txt  
-rw-r--r--@ 1 anushasp staff 5B Oct 1 11:32 plain1.txt  
-rw-r--r--@ 1 anushasp staff 10B Oct 1 11:33 plain2.txt  
-rw-r--r--@ 1 anushasp staff 16B Oct 1 11:33 plain3.txt  
anushasp@Anushas-MacBook-Air CFB %
```

- Now let us decrypt and check the padding, for this we use no pad in the openssl command.

```

anushasp@Anushas-MacBook-Air CFB % openssl enc -aes-128-cfb -d -nopad -in en_plain1_cfb.txt -out de_plain1_cfb.txt -K 00010203040506070809aabcccddeeff \
-aniv 0a0b0c0d0e0f010203040506070809
hex string is too short, padding with zero bytes to length
anushasp@Anushas-MacBook-Air CFB % openssl enc -aes-128-cfb -d -nopad -in en_plain2_cfb.txt -out de_plain2_cfb.txt -K 00010203040506070809aabcccddeeff \
-aniv 0a0b0c0d0e0f010203040506070809
hex string is too short, padding with zero bytes to length
anushasp@Anushas-MacBook-Air CFB % openssl enc -aes-128-cfb -d -nopad -in en_plain3_cfb.txt -out de_plain3_cfb.txt -K 00010203040506070809aabcccddeeff \
-aniv 0a0b0c0d0e0f010203040506070809
hex string is too short, padding with zero bytes to length
anushasp@Anushas-MacBook-Air CFB % ls -lh
total 72
-rw-r--r-- 1 anushasp staff 5B Oct 1 14:49 de_plain1_cfb.txt
-rw-r--r-- 1 anushasp staff 10B Oct 1 14:49 de_plain2_cfb.txt
-rw-r--r-- 1 anushasp staff 16B Oct 1 14:50 de_plain3_cfb.txt
-rw-r--r-- 1 anushasp staff 5B Oct 1 14:48 en_plain1_cfb.txt
-rw-r--r-- 1 anushasp staff 10B Oct 1 14:48 en_plain2_cfb.txt
-rw-r--r-- 1 anushasp staff 16B Oct 1 14:48 en_plain3_cfb.txt
-rw-r--r--@ 1 anushasp staff 5B Oct 1 11:32 plain1.txt
-rw-r--r--@ 1 anushasp staff 10B Oct 1 11:33 plain2.txt
-rw-r--r--@ 1 anushasp staff 16B Oct 1 11:33 plain3.txt
anushasp@Anushas-MacBook-Air CFB %

```

The decrypted files content is as below:

```

anushasp@Anushas-MacBook-Air CFB % ls
de_plain1_cfb.txt en_plain1_cfb.txt plain1.txt
de_plain2_cfb.txt en_plain2_cfb.txt plain2.txt
de_plain3_cfb.txt en_plain3_cfb.txt plain3.txt
anushasp@Anushas-MacBook-Air CFB % hexdump -C de_plain1_cfb.txt
00000000 43 53 34 38 35 |CS485|
00000005
anushasp@Anushas-MacBook-Air CFB % hexdump -C de_plain2_cfb.txt
00000000 43 53 34 38 35 4c 41 42 30 32 |CS485LAB02|
0000000a
anushasp@Anushas-MacBook-Air CFB % hexdump -C de_plain3_cfb.txt
00000000 43 53 34 38 35 4c 41 42 30 32 54 41 53 4b 30 34 |CS485LAB02TASK04|
00000010
anushasp@Anushas-MacBook-Air CFB %

```

Analysis [CBC]:

File Name	Original File Size	Encrypted File Size	Decrypted File Size
plain1.txt	5 Bytes	5 Bytes	5 Bytes
plain2.txt	10 Bytes	10 Bytes	10 Bytes
plain3.txt	16 Bytes	16 Bytes	16 Bytes

4.4 Using OFB Mode:

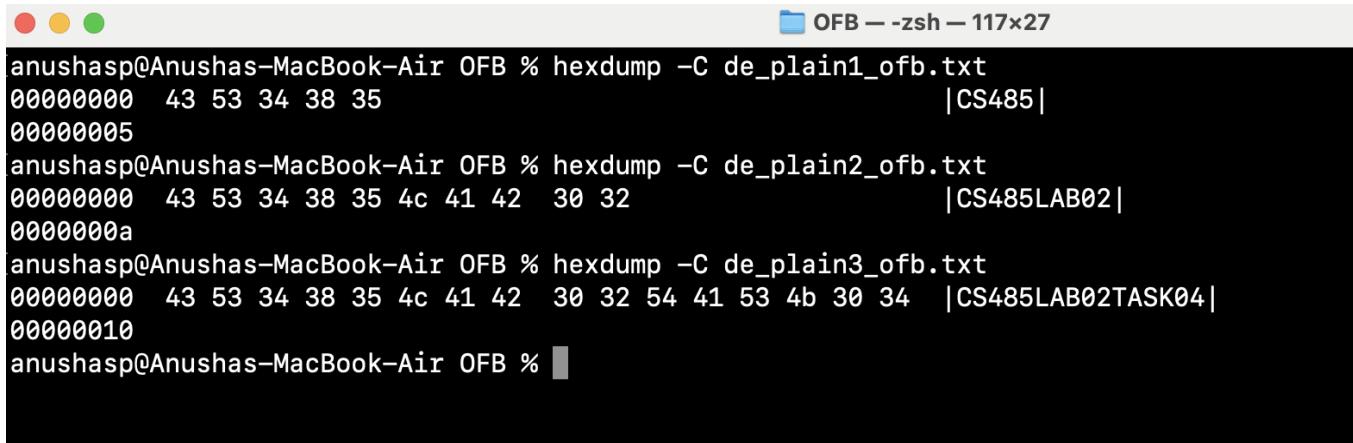
- We will encrypt the files using OFB mode and compare the file sizes.
- After encryption, there is no change in their file sizes as shown below, so there was NO padding.

```
anushasp@Anushas-MacBook-Air OFB % ls
plain1.txt      plain2.txt      plain3.txt
anushasp@Anushas-MacBook-Air OFB % openssl enc -aes-128-ofb -e -in plain1.txt -out en_plain1_ofb.txt
[enter AES-128-OFB encryption password:
Verifying - enter AES-128-OFB encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air OFB % openssl enc -aes-128-ofb -e -in plain2.txt -out en_plain2_ofb.txt
[enter AES-128-OFB encryption password:
Verifying - enter AES-128-OFB encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air OFB % openssl enc -aes-128-ofb -e -in plain3.txt -out en_plain3_ofb.txt
[enter AES-128-OFB encryption password:
Verifying - enter AES-128-OFB encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air OFB % ls -lh
total 48
-rw-r--r--  1 anushasp  staff   21B Oct  1 15:00 en_plain1_ofb.txt
-rw-r--r--  1 anushasp  staff   26B Oct  1 15:00 en_plain2_ofb.txt
-rw-r--r--  1 anushasp  staff   32B Oct  1 15:01 en_plain3_ofb.txt
-rw-r--r--@ 1 anushasp  staff    5B Oct  1 11:32 plain1.txt
-rw-r--r--@ 1 anushasp  staff   10B Oct  1 11:33 plain2.txt
-rw-r--r--@ 1 anushasp  staff   16B Oct  1 11:33 plain3.txt
anushasp@Anushas-MacBook-Air OFB %
```

- Now let us decrypt and check the padding, for this we use no pad in the openssl command.

```
anushasp@Anushas-MacBook-Air OFB % openssl enc -aes-128-ofb -d -nopad -in en_plain1_ofb.txt -out de_plain1_ofb.txt
[enter AES-128-OFB decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air OFB % openssl enc -aes-128-ofb -d -nopad -in en_plain2_ofb.txt -out de_plain2_ofb.txt
[enter AES-128-OFB decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air OFB % openssl enc -aes-128-ofb -d -nopad -in en_plain3_ofb.txt -out de_plain3_ofb.txt
[enter AES-128-OFB decryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air OFB % ls -lh
total 72
-rw-r--r--  1 anushasp  staff    5B Oct  1 15:03 de_plain1_ofb.txt
-rw-r--r--  1 anushasp  staff   10B Oct  1 15:03 de_plain2_ofb.txt
-rw-r--r--  1 anushasp  staff   16B Oct  1 15:03 de_plain3_ofb.txt
-rw-r--r--  1 anushasp  staff   21B Oct  1 15:00 en_plain1_ofb.txt
-rw-r--r--  1 anushasp  staff   26B Oct  1 15:00 en_plain2_ofb.txt
-rw-r--r--  1 anushasp  staff   32B Oct  1 15:01 en_plain3_ofb.txt
-rw-r--r--@ 1 anushasp  staff    5B Oct  1 11:32 plain1.txt
-rw-r--r--@ 1 anushasp  staff   10B Oct  1 11:33 plain2.txt
-rw-r--r--@ 1 anushasp  staff   16B Oct  1 11:33 plain3.txt
anushasp@Anushas-MacBook-Air OFB %
```

The decrypted files content is as below:



OFB — -zsh — 117x27

```
anushasp@Anushas-MacBook-Air OFB % hexdump -C de_plain1_ofb.txt
00000000  43 53 34 38 35          |CS485|
00000005
anushasp@Anushas-MacBook-Air OFB % hexdump -C de_plain2_ofb.txt
00000000  43 53 34 38 35 4c 41 42  30 32          |CS485LAB02|
0000000a
anushasp@Anushas-MacBook-Air OFB % hexdump -C de_plain3_ofb.txt
00000000  43 53 34 38 35 4c 41 42  30 32 54 41 53 4b 30 34  |CS485LAB02TASK04|
00000010
anushasp@Anushas-MacBook-Air OFB %
```

Analysis [OFB]:

File Name	Original File Size	Encrypted File Size	Decrypted File Size
plain1.txt	5 Bytes	21 Bytes	5 Bytes
plain2.txt	10 Bytes	26 Bytes	10 Bytes
plain3.txt	16 Bytes	32 Bytes	16 Bytes

Conclusion:

For ECB and CBC there was padding but for CFB and OFB there was NO padding.

ECB and CBC modes need to be paired with a deterministic padding mode such as PKCS#5 or PKCD#7 compatible padding to operate on messages of any size.

OFB and CFB modes do not require padding. In OFB and CFB modes, the encryption and decryption processes are identical.

Task 5: Error propagation: Corrupted Cipher Text

Goal: The goal of this task is to understand the error propagation property of various encryption modes.

Step 1: Create a plaintext file of at least 1000 bytes long.

```
anushasp@Anushas-MacBook-Air ECB % cat plaintext.txt
To understand the error propagation property of various encryption modes, we would like to do the following exercise:
1. Create a text file that is at least 1000 bytes long.
2. Encrypt the file using the AES-128 cipher.
3. Unfortunately, a single bit of the 55th byte in the encrypted file got corrupted. You can achieve this corruption using the bless hex editor.
4. Decrypt the corrupted ciphertext file using the correct key and IV.
Please answer the following question: How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task. Please provide justification.
Most of the encryption modes require an initial vector (IV). Properties of an IV depend on the cryptographic scheme used. If we are not careful in selecting IVs, the data encrypted by us may not be secure at all, even though we are using a secure encryption algorithm and mode. The objective of this task is to help students understand the problems if an IV is not selected properly.
anushasp@Anushas-MacBook-Air ECB %
```

5.1 Using AES-128 ECB Mode:

- The plaintext is divided into fixed-sized blocks (usually 128 bits or 16 bytes in the case of AES).
- Each block is treated independently during encryption.
- The same encryption key is used to encrypt each block of plaintext.
- Encrypt the above plaintext file using AES algorithm and ECB encryption mode.

```
anushasp@Anushas-MacBook-Air ECB % openssl enc -aes-128-ecb -e -in plaintext.txt -out encrypted_ecb.txt
[enter AES-128-ECB encryption password:
[Verifying - enter AES-128-ECB encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air ECB %
```

- Corrupt the 55th bit using HEX Fiend as shown below.

The screenshot shows the HEX Fiend application interface. At the top, there's a toolbar with three colored circles (red, yellow, green) and a title bar labeled "encrypted_ecb.txt". The main area displays a grid of hex values and their corresponding ASCII characters. A cursor is positioned over the byte at address 300, which is highlighted in blue. The bottom status bar indicates "Byte 55 selected out of 1184 bytes".

Address	Hex	ASCII
000	53616C74	53 E6 C7 45
020	65645FF5	65 E4 5F F5
040	75E92569	75 E9 25 69
060	3E8A72F4	3E 8A 72 F4
080	F337AAAD	F3 37 AA AD
0A0	FF3996E4	FF 39 96 E4
0C0	F927441B	F9 27 44 1B
0E0	5B99B596	5B 99 B5 96
100	020 AF749EF0	02 0A F7 49 E F0
120	8DF31DC7	8D F3 1D C7
140	40335CA0	40 33 5C A0
160	3A33AD67	3A 33 AD 67
180	326C6C83	32 6C 6C 83
200	B80ABABA	B8 0A BABA
220	E5688EDB	E5 68 8E DB
240	8E107D29	8E 10 7D 29
260	93E17DB7	93 E1 7D B7
280	471CC9E0	47 1C C9 E0
300	CC9EBD6B	CC 9E BD 6B
320	CA77C1E4	CA 77 C1 E4
340	284BE29E	28 4B E2 9E
360	4228B862	42 28 B8 62
380	FBA5124A	F B A5 12 4A
400	452E46CC	45 2E 46 CC
420	C96D5DE1	C9 6D 5D E1
440	0F04ABF9	0F 04 AB F9
460	25FCEED7	25 FCE ED 7
480	731807FA	73 18 07 FA
500	0367C8A6	03 67 C8 A6
520	94F8A5C8	94 F8 A5 C8
540	662DD862	66 2D D8 62
560	951CDF65	95 1C DF 65
580	F3734E5F	F3 73 4E 5F
600	C6252CF3	C6 25 2C F3
620	6E9266CD	6E 92 66 CD
640	073A9ECA	07 3A 9E CA
660	732776C4	73 27 76 C4
680	4B740CBF	4B 74 0C BF
700	80691868	80 69 18 68
720	A8C5D173	A8 C5 D1 73
740	0A0 C51C4D42	0A 0C 51 C4 D4 2
760	7F803181	7F 80 31 81
780	BB0364F9	BB 03 64 F9
800	4F050F07	4F 05 0F 07
820	951A9ACE	95 1A 9A CE
840	57BFCE30	57 BF CE 30
860	44840A28	44 84 0A 28
880	D396E9FD	D3 96 E9 FD
900	C378D128	C3 78 D1 28
920	8EEA2323	8E EA 23 23
940	E5B3887C	E5 B3 88 7C
960	9C34DFA6	9C 34 DF A6
980	5184411F	51 84 41 1F
1000	E0FFA2DC	E0 FF A2 DC
1020	85C64CFC	85 C6 4C FC
1040	EBAFC0C6	E B A F C0 C6
1060	7309B87B	73 09 B8 7B
1080	353D24BE	35 3D 24 BE
1100	E9DE54BF	E9 DE 54 BF
1120	1C9A25DB	1C 9A 25 DB
1140	61626F5F	61 62 6F 5F
1160	C855F594	C8 55 F5 94
1180	216194A9	21 61 94 A9
1200	712D6A80	71 2D 6A 80
1220	FD377184	FD 37 71 84
1240	2D36D78D	2D 36 D7 8D
1260	7E5849D5	7E 58 49 D5
1280	418F2094	41 8F 20 94
1300	6C37A064	6C 37 A0 64
1320	07CC5F21	07 CC 5F 21
1340	22BC8FF0	22 BC 8F F0
1360	EF67780D	EF 67 78 0D
1380	C13A47DD	C1 3A 47 DD
1400	8B52A93A	8B 52 A9 3A
1420	2CABB1BE	2C AB B1 BE
1440	E9BBCD4A	E9 BB CD 4A
1460	AE24520D	AE 24 52 0D
1480	99CDC7E6	99 CD C7 E6
1500	7C205C5B	7C 20 5C 5B
1520	7D5B9F75	7D 5B 9F 75
1540	BB6484C6	BB 64 84 C6
1560	1CD767D7	1C D7 67 D7
1580	7476FDB0	74 76 FD B0
1600	8E51D6DE	8E 51 D6 DE
1620	8871BAE2	88 71 BA E2
1640	F8266303	F8 26 63 03
1660	ECEB42CC	ECEB 42 CC
1680	72D5DF30	72 D5 DF 30
1700	876743D7	87 67 43 D7
1720	3FB0C8F7	3F B0 C8 F7
1740	18A21A43	18 A2 1A 43
1760	D1F3F3B1	D1 F3 F3 B1
1780	B4F324F7	B4 F3 24 F7
1800	28920C8C	28 92 0C 8C
1820	0E3484E8	0E 34 84 E8
1840	9DB1C6FA	9D B1 C6 FA
1860	55A2295A	55 A2 29 5A
1880	78029E2D	78 02 9E 2D
1900	666F652D	66 6F 65 2D
1920	ABC37AB1	ABC3 7A B1
1940	B16ED370	B1 6E D3 70
1960	C8880F67	C8 88 0F 67
1980	5D674E15	5D 67 4E 15
2000	7AC0C847	7A C0 C8 47
2020	A138A18E	A1 38 A1 8E
2040	C29F6F31	C2 9F 6F 31
2060	93D0509D	93 D0 50 9D
2080	8397D3C2	83 97 D3 C2
2100	8DE29E45	8D E2 9E 45
2120	CBFC5EE1	CB FC 5E E1
2140	1D074F34	1D 07 4F 34
2160	FCFB58FD	FC FB 58 FD
2180	0756FB47	07 56 FB 47
2200	997A0284	99 7A 02 84
2220	60D44BB6	60 D4 4B B6
2240	5230DCCB	52 30 DCCB
2260	DC4C4BB1	DC 4C 4B B1
2280	84B2B1F9	84 B2 B1 F9
2300	E5213348	E5 21 33 48
2320	328BD39C	32 8B D3 9C
2340	25C73EEB	25 C7 3E EB
2360	5F77CCBB	5F 77 CC BB
2380	FD70475E	FD 70 47 5E
2400	53CBCEA6	53 CB CE A6
2420	3BC2D10F	3B C2 D1 0F
2440	174677ED	17 46 77 ED
2460	841D4B41	84 1D 4B 41
2480	D4AD3431	D4 AD 34 31
2500	394353CF	39 43 53 CF
2520	5D239A06	5D 23 9A 06
2540	2F8B1660	2F 8B 16 60
2560	E7B81DBB	E7 B8 1D BB
2580	00CACFE3	00 CAC F E3
2600	95DB12D3	95 DB 12 D3
2620	790EB2EA	79 0E B2 EA
2640	7D0C8DC3	7D 0C 8D C3
2660	DD0F1032	DD 0F 10 32
2680	B5E60CBD	B5 E6 0C BD
2700	0EA9ECEC	0E A9 ECEC
2720	0A525FE8	0A 52 5F E8
2740	996396FC	99 63 96 FC
2760	1C93C101	1C 93 C1 01
2780	A8C251B2	A8 C2 51 B2
2800	3685445F	36 85 44 5F
2820	5C3FF52A	5C 3F F5 2A
2840	5ABF5235	5A BF 52 35
2860	FA5383AC	FA 53 83 AC
2880	19670DED	19 67 0D ED
2900	7366DAF7	73 66 DA F7
2920	2114F87D	21 14 F8 7D
2940	F32244F4	F3 22 44 F4
2960	A1B3CD88	A1 B3 CD 88
2980	B6795EF8	B6 79 5E F8
3000	36F1957C	36 F1 95 7C
3020	8CE7D22D	8C E7 D2 2D
3040	8090D787	80 90 D7 87
3060	AAB54014	AAB5 40 14
3080	B85D107A	B8 5D 10 7A
3100	481CE316	48 1C E3 16
3120	8EB62461	8E B6 24 61
3140	5C19472F	5C 19 47 2F
3160	527FAAFF	52 7F A A F F
3180	2FBFC9075	2F BC F9 07 5
3200	9CB3F23E	9C B3 F2 3E
3220	8341BB73	83 41 BB 73
3240	CD62F924	CD 62 F9 24
3260	77F7AA20	77 F7 AA 20
3280	C360B972	C3 60 B9 72
3300	2AC4FDC9	2A C4 FDC 9
3320	95C68ED7	95 C6 8E D7
3340	5EAF75FB	5E AF 75 FB
3360	F6FC18F1	F6 FC 18 F1
3380	C92BCD59	C9 2B CD 59
3400	97C8A3DB	97 C8 A3 DB
3420	613B8B1E	61 3B 8B 1E
3440	1DBC8B62	1D BC 8B 62
3460	BC739834	BC 73 98 34
3480	FE0BBAC8	FE 0B BA C8
3500	09D0FCDE	09 D0 F C D E
3520	D451E25A	D4 51 E2 5A
3540	F8455226	F8 45 52 26
3560	53D989CE	53 D9 89 CE
3580	6E053FEF	6E 05 3F EF
3600	DDD392A4	DD D3 92 A4
3620	4DBDA3F7	4D BD A3 F7
3640	86C9E0E5	86 C9 E0 E5
3660	BECB9B58	BECB 9B 58
3680	05B54622	05 B5 46 22
3700	5B3D473E	5B 3D 47 3E
3720	F4D7B4EF	F4 D7 B4 E F
3740	B19FC250	B1 9F C2 50
3760	0B9E9C0D	0B 9E 9C 0D

- Now decrypt the file after corruption and check the output.

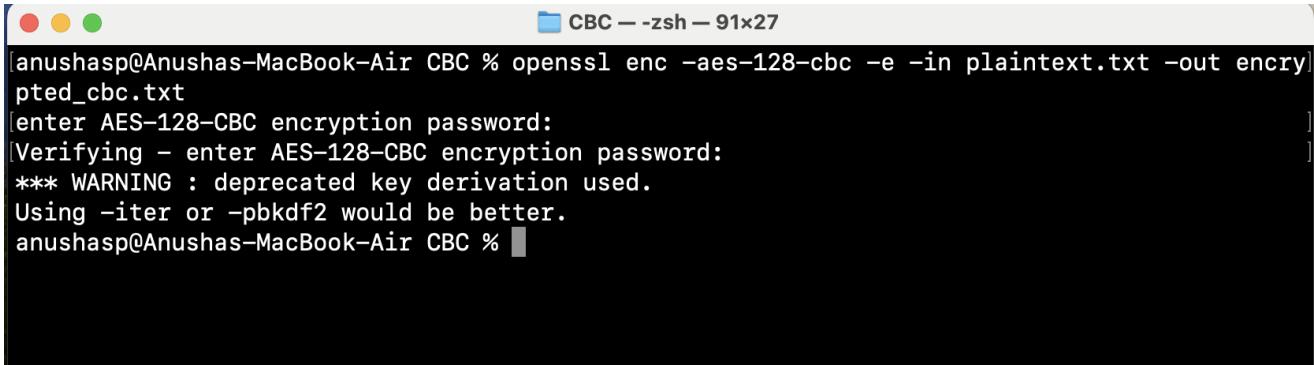
The screenshot shows the decrypted_ecb.txt file in the HEX Fiend application. The file content is mostly illegible due to corruption, appearing as a series of random hex digits. The bottom status bar indicates "Byte 55 selected out of 1184 bytes".

Observation:

In ECB mode of encryption, each block of plain text is encrypted separately. So, on corrupting the 55th byte only the block containing that corrupted byte will be affected. This is one of the drawbacks of ECB mode, as it does not provide any inherent error propagation or diffusion, making it less secure and susceptible to patterns in the plaintext. [So impacted number of blocks = 1]

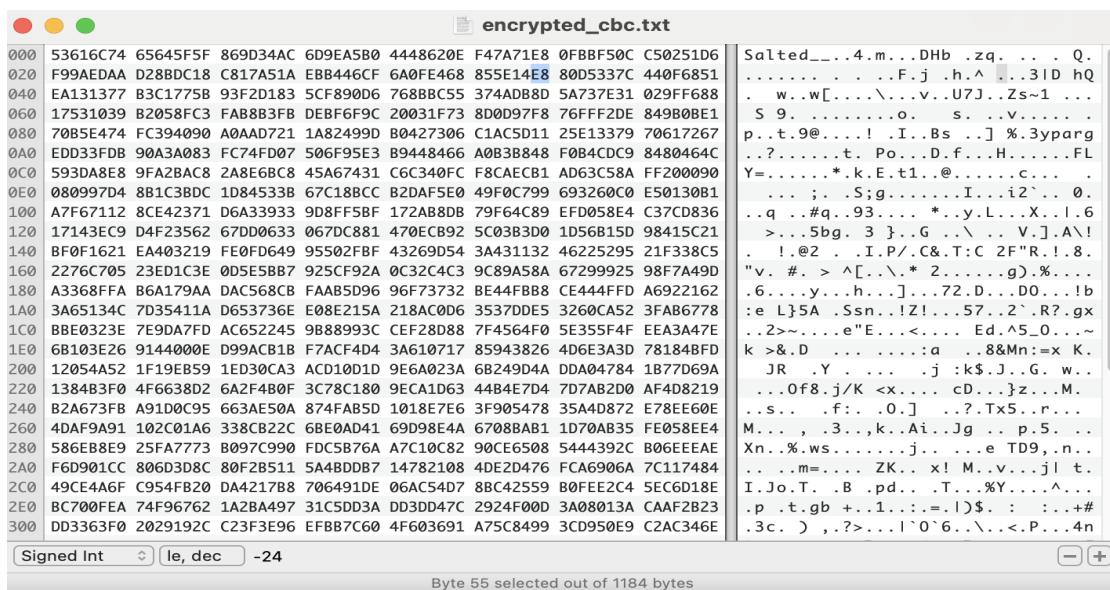
5.2 Using AES-128 CBC Mode:

- In CBC mode, each plaintext block is XORed (bitwise exclusive OR) with the previous ciphertext block before encryption.
- This chaining of blocks adds an element of feedback into the encryption process.
- Let us encrypt the above plaintext file using AES algorithm and CBC encryption mode.



```
[anushasp@Anushas-MacBook-Air CBC % openssl enc -aes-128-cbc -e -in plaintext.txt -out encrypted_cbc.txt
[enter AES-128-CBC encryption password:
[Verifying - enter AES-128-CBC encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air CBC % ]
```

- Corrupt the 55th bit using HEX Fiend as shown below



- Now decrypt the file after corruption and check the output.

```
To understand the error propagation in various encryption modes, we would like to do the following exercise:
1. Create a text file that is at least 1000 bytes long.
2. Encrypt the file using the AES-128 cipher.
3. Unfortunately, a single bit of the 55th byte in the encrypted file got corrupted. You can achieve this corruption using the bless hex editor.
4. Decrypt the corrupted ciphertext file using the correct key and IV.
Please answer the following question: How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task.
Please provide justification.
Most of the encryption modes require an initial vector (IV). Properties of an IV depend on the cryptographic scheme used. If we are not careful in selecting IVs, the data encrypted by us may not be secure at all, even though we are using a secure encryption algorithm and mode. The objective of this task is to help students understand the problems if an IV is not selected properly.
```

Observation:

In CBC (Cipher Block Chaining) mode of encryption, if one byte is corrupted, the impact will be localized to the block containing the corrupted byte and the subsequent block during decryption. One damaged block propagates to two blocks as we use the first cipher text in generating the second and so on. [So impacted = 2 blocks]

5.3 Using AES-A28 CFB Mode:

- CFB mode operates in a manner that resembles a stream cipher, generating a stream of pseudorandom bits that are XORed with the plaintext to produce the ciphertext.
- Let us encrypt the above plaintext file using AES algorithm and CFB encryption mode.

```
anushasp@Anushas-MacBook-Air CFB % openssl enc -aes-128-cfb -e -in plaintext.txt -out encrypted_cfb.txt
enter AES-128-CFB encryption password:
Verifying - enter AES-128-CFB encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air CFB %
```

- Corrupt the 55th bit using HEX Fiend as shown below:

The screenshot shows the 'encrypted_cfb.txt' file in HEX Fiend. The file consists of two main sections: hex data and ASCII text. The hex data is a long sequence of bytes, and the ASCII text includes file paths, file names, and some descriptive text. The byte at index 55 is highlighted in blue. The status bar at the bottom indicates 'Signed Int' and 'le, dec 104'. The bottom right corner of the window has a small icon.

- Now decrypt the file after corruption and check the output.

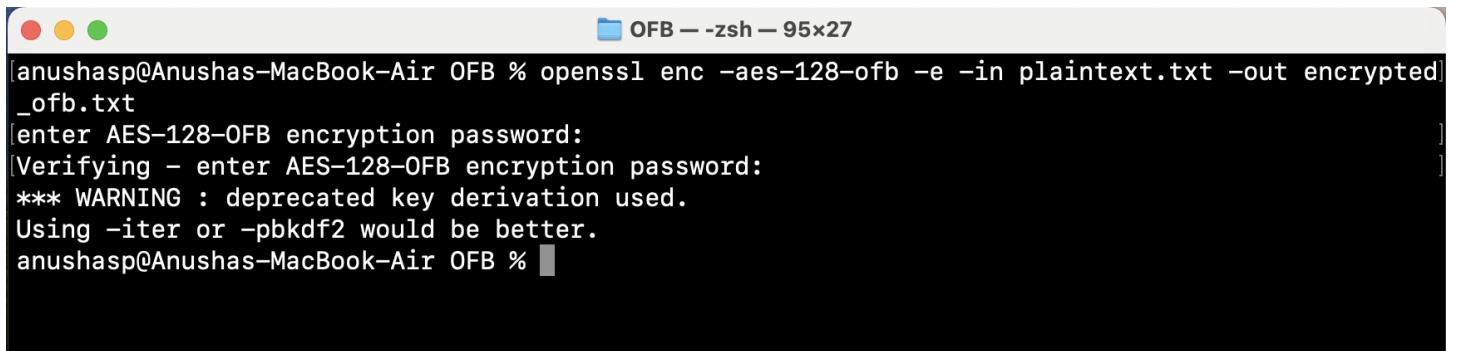
The screenshot shows the 'decrypted_cfb.txt' file in HEX Fiend. The file contains a single block of ASCII text. The text discusses error propagation in CFB mode and provides instructions for a task. It includes numbered steps, a question, and a justification section. The text is mostly in English with some German words like 'Übertragung' and 'Fehlerausbreitung'.

Observation:

In CFB mode, if one byte is corrupted, the impact is typically limited to the block containing the corrupted byte and the subsequent blocks during decryption. [So impact ~ 2 blocks]

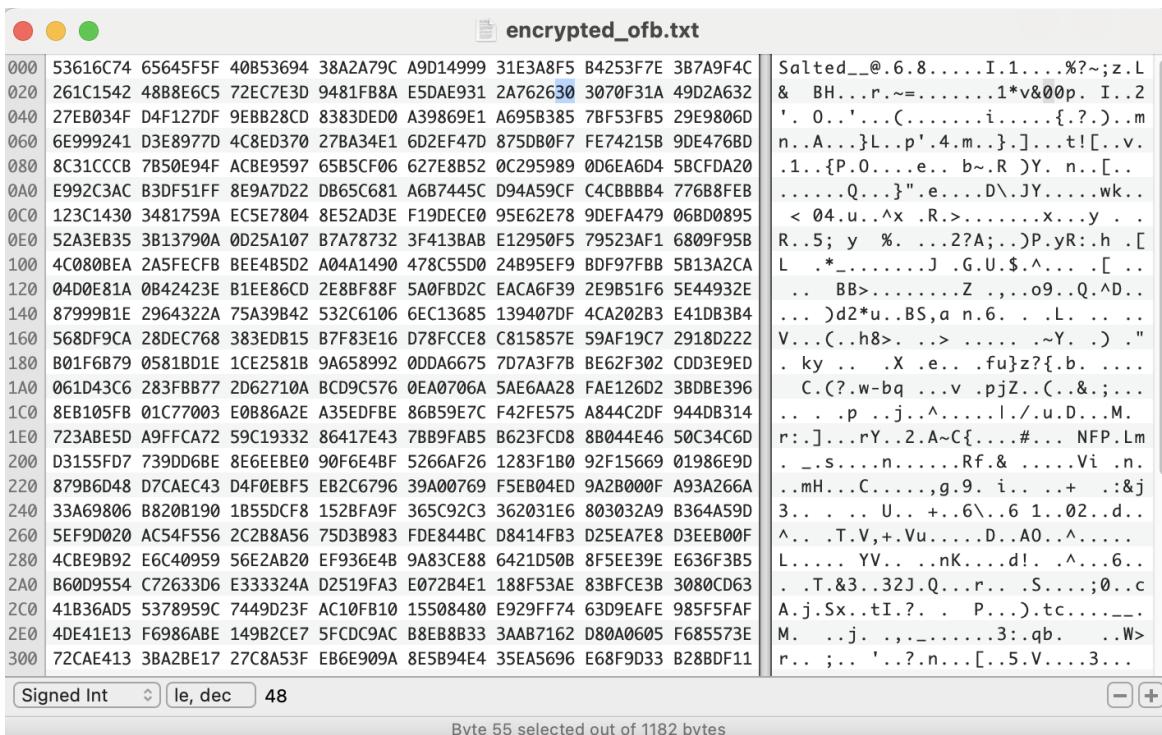
5.4 Using AES-128 OFB Mode:

- OFB mode operates in a manner that resembles a stream cipher. It generates a pseudorandom bit stream, which is then XORed with the plaintext to produce the ciphertext.
- Let us encrypt the above plaintext file using AES algorithm and OFB encryption mode.



```
[anushasp@Anushas-MacBook-Air OFB % openssl enc -aes-128-ofb -e -in plaintext.txt -out encrypted_ofb.txt
[enter AES-128-OFB encryption password:
[Verifying - enter AES-128-OFB encryption password:
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
anushasp@Anushas-MacBook-Air OFB %
```

- Corrupt the 55th bit using HEX Fiend as shown below



- Now decrypt the file after corruption and check the output.

 decrypted_ofb.txt

To understand the error propagation ~~property~~ of various encryption modes, we would like to do the following exercise:

1. Create a text file that is at least 1000 bytes long.
2. Encrypt the file using the AES-128 cipher.
3. Unfortunately, a single bit of the 55th byte in the encrypted file got corrupted. You can achieve this corruption using the bless hex editor.
4. Decrypt the corrupted ~~ciphertext~~ file using the correct key and IV.

Please answer the following question: How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task.

Please provide justification.

Most of the encryption modes require an initial vector (IV). Properties of an IV depend on the cryptographic scheme used. If we are not careful in selecting IVs, the data encrypted by us may not be secure at all, even though we are using a secure encryption algorithm and mode. The objective of this task is to help students understand the problems if an IV is not selected properly.

Observation:

In OFB mode, the error does not propagate beyond the corrupted byte. It is designed for streaming, making it suitable for applications where error recovery is important.

So impact = only corrupted byte!

Summary:

Mode	Observation	Impact
ECB	Only the block containing the corrupted byte is affected.	1 block
CBC	Current block and subsequent block.	2 blocks
CFB	Current block and subsequent block.	2 blocks
OFB	Only the corrupted byte is affected.	1 byte

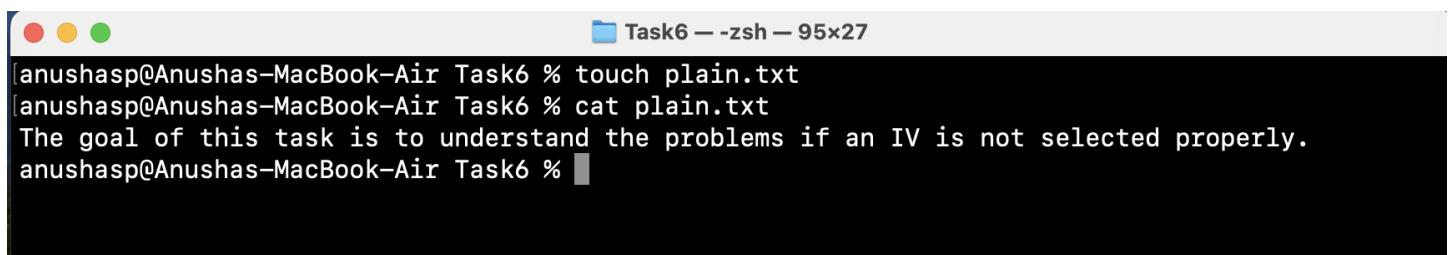
Task 6: Initial Vector (IV)

Goal: The goal of this task is to understand the problems if an IV is not selected properly.

6.1. Uniqueness of the IV:

Initialization Vectors (IVs) are used in various cryptographic algorithms, such as block ciphers in Cipher Block Chaining (CBC) mode, to add randomness and prevent patterns in the ciphertext.

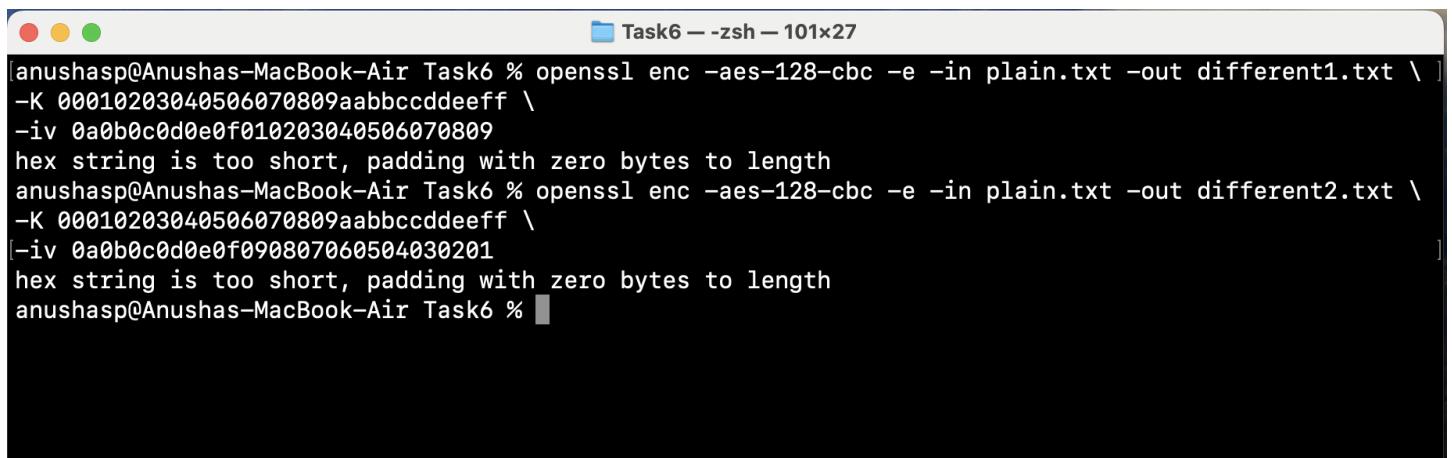
Step 1: Creating the plaintext file plain.txt



```
[anushasp@Anushas-MacBook-Air Task6 % touch plain.txt
[anushasp@Anushas-MacBook-Air Task6 % cat plain.txt
The goal of this task is to understand the problems if an IV is not selected properly.
anushasp@Anushas-MacBook-Air Task6 % ]
```

6.1.1 Using two different IVs:

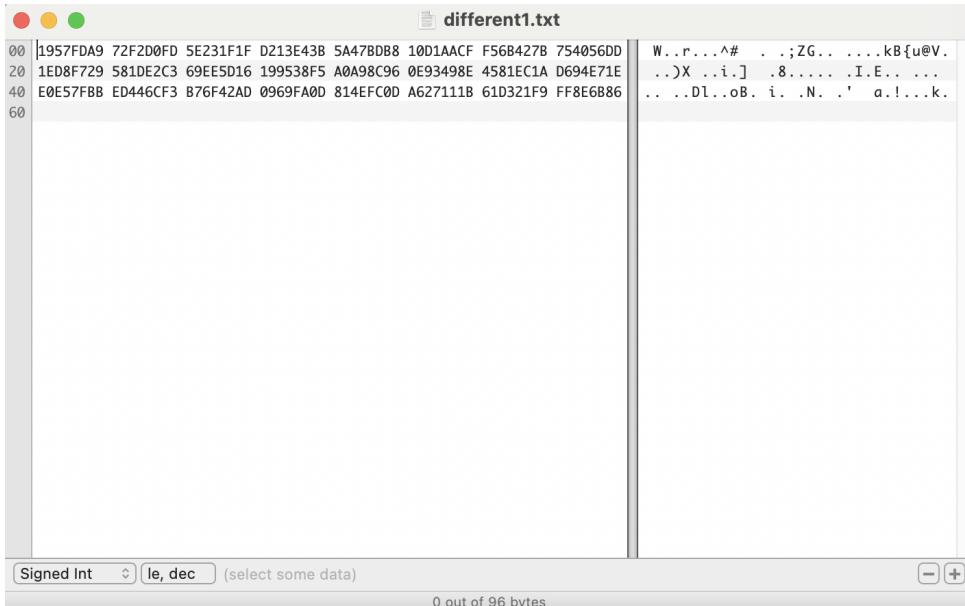
- Encrypting the same plain.txt using two different IVs.



```
[anushasp@Anushas-MacBook-Air Task6 % openssl enc -aes-128-cbc -e -in plain.txt -out different1.txt \
-K 00010203040506070809aabcccddeeff \
-iv 0a0b0c0d0e0f010203040506070809
hex string is too short, padding with zero bytes to length
anushasp@Anushas-MacBook-Air Task6 % openssl enc -aes-128-cbc -e -in plain.txt -out different2.txt \
-K 00010203040506070809aabcccddeeff \
-iv 0a0b0c0d0e0f090807060504030201
hex string is too short, padding with zero bytes to length
anushasp@Anushas-MacBook-Air Task6 % ]
```

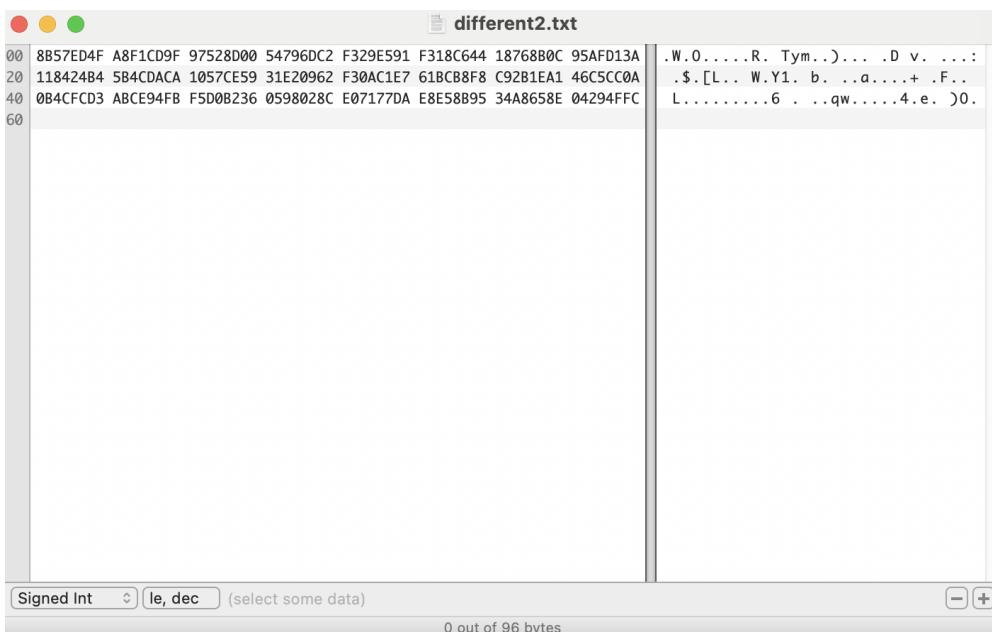
Output files:

different1.txt



The screenshot shows a hex editor window titled "different1.txt". The left pane displays memory starting at address 00 with data: 1957FDA9 72F2D0FD SE231F1F D213E43B 5A47BDB8 10D1AACF F56B427B 754056DD, followed by several lines of hex data. The right pane shows the corresponding ASCII representation, which is mostly a mix of random characters like 'W', 'r', 'Z', 'G', 'k', 'B', etc., with some punctuation and symbols interspersed.

different2.txt

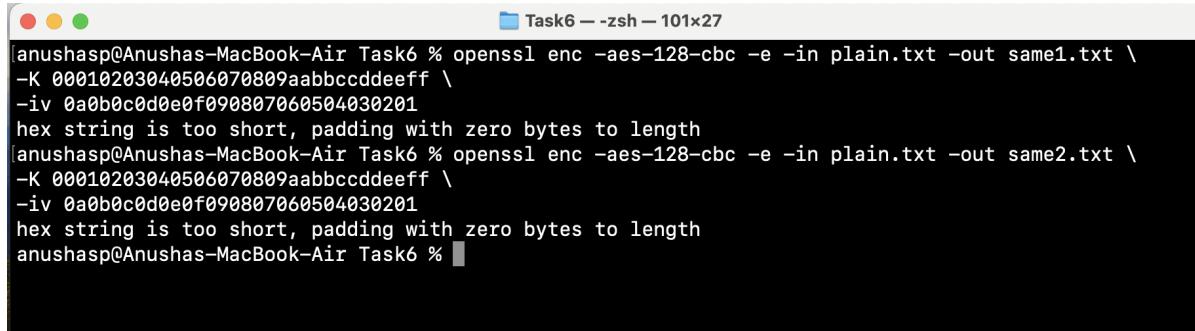


The screenshot shows a hex editor window titled "different2.txt". The left pane displays memory starting at address 00 with data: 8857ED4F A8F1CD9F 97528D00 54796DC2 F329E591 F318C644 18768B0C 95AFD13A, followed by several lines of hex data. The right pane shows the corresponding ASCII representation, which is mostly a mix of random characters like 'W', 'O', 'R', 'T', 'Y', '1', 'b', 'a', 'F', 'L', '6', 'q', 'w', 'e', '0', etc.

Observation: The ciphertext generated for the same plain text is different when different IV is used.

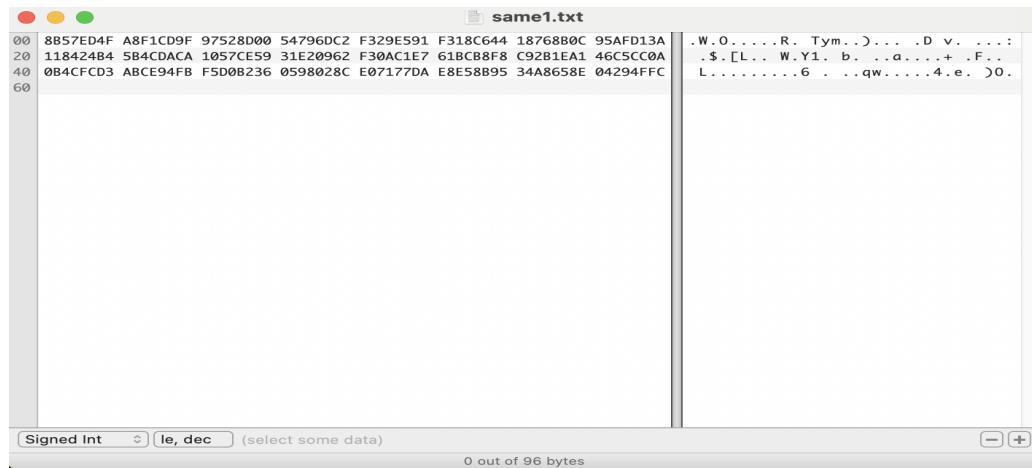
6.1.2 Using the same IV:

- Encrypting the same plain.txt using the same IV twice.



```
[anushasp@Anushas-MacBook-Air Task6 % openssl enc -aes-128-cbc -e -in plain.txt -out same1.txt \
-K 00010203040506070809aabbcdddeeff \
-iv 0a0b0c0d0e0f090807060504030201
hex string is too short, padding with zero bytes to length
[anushasp@Anushas-MacBook-Air Task6 % openssl enc -aes-128-cbc -e -in plain.txt -out same2.txt \
-K 00010203040506070809aabbcdddeeff \
-iv 0a0b0c0d0e0f090807060504030201
hex string is too short, padding with zero bytes to length
anushasp@Anushas-MacBook-Air Task6 %
```

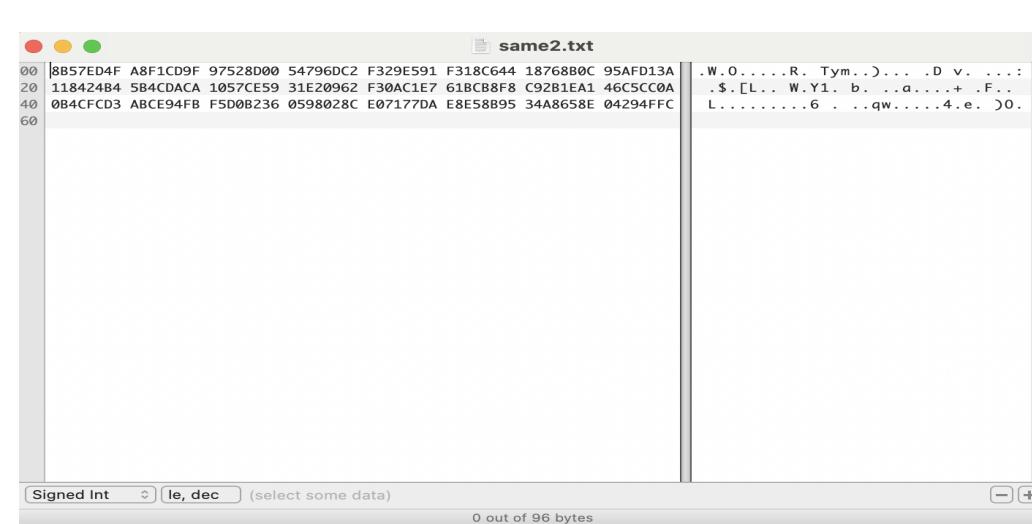
Output files:



same1.txt

Hex	ASCII
00 8B57ED4F A8F1CD9F 97528D00 54796DC2 F329E591 F318C644 18768B0C 95AFD13A	.W.O.....R. Tym..>... .D v.
20 118424B4 5B4CDACA 1057CE59 31E20962 F30AC1E7 61BCB8F8 C92B1EA1 46C5CC0A	.\$. [L.. W.Y1. b. ..a....+ .F..
40 0B4CFCD3 ABCE94FB F5D0B236 0598028C E07177DA E8E58B95 34A8658E 04294FFC	L.....6 .. .qw....4.e. >O.
60	

Signed Int (le, dec) (select some data) 0 out of 96 bytes



same2.txt

Hex	ASCII
00 8B57ED4F A8F1CD9F 97528D00 54796DC2 F329E591 F318C644 18768B0C 95AFD13A	.W.O.....R. Tym..>... .D v.
20 118424B4 5B4CDACA 1057CE59 31E20962 F30AC1E7 61BCB8F8 C92B1EA1 46C5CC0A	.\$. [L.. W.Y1. b. ..a....+ .F..
40 0B4CFCD3 ABCE94FB F5D0B236 0598028C E07177DA E8E58B95 34A8658E 04294FFC	L.....6 .. .qw....4.e. >O.
60	

Signed Int (le, dec) (select some data) 0 out of 96 bytes

Observation: The ciphertext generated for the same plain text is exactly the same when the same IV is used.

6.2. Common mistakes: Use the same IV

Here the goal is to find how much of P2 can be revealed if P1, C1 and C2 are known.

We know,

Plaintext (P1): This is a known message!

Ciphertext (C1): a469b1c502c1cab966965e50425438e1bb1b5f9037a4c15913

Plaintext (P2): (unknown to you)

Ciphertext (C2): bf73bcd3509299d566c35b5d450337e1bb175f903fafc15913

Let us convert P1 into hexadecimal format:

P1 : 54 68 69 73 20 69 73 20 61 20 6B 6E 6F 77 6E 20 6D 65 73 73 61 67 65 21 0A

In OFB Mode, P1 (XOR) C1 = P2 (XOR) C2

So, P2 = P1 (XOR) C1 (XOR) C2

On calculation $P2 = 4F726465723A204C61756E63682061206D697373696C65210A$

On conversion to ASCII, text generated = “Order: Launch a missile!”

Observation:

In OFB mode, if an attacker has access to a plaintext (P1) and its corresponding ciphertext (C1) and the IV is always the same, they can compute the keystream used to encrypt P1 by XORing P1 with C1. Once they have the keystream, they can XOR it with C2 to obtain the plaintext P2, even if P2 is unknown.

In CFB mode, if the same IV is used, a similar vulnerability exists. An attacker who knows P1 and C1 can compute the keystream in the same way as in OFB mode and use it to decrypt C2 and reveal the entire content of P2.

In summary, the primary difference in the context of the known-plaintext attack is that using the same IV in OFB mode does not reveal information about other encrypted messages, while in CFB mode, it can potentially reveal information about other blocks encrypted with the same IV, depending on the IV reuse patterns and the encryption algorithm used.

6.3. Common mistakes: Use a predictable IV

We know,

Encryption method: 128-bit AES with CBC mode.

Key (in hex): 00112233445566778899aabccddeeff (known only to Bob)

Ciphertext (C1): bef65565572cceee2a9f9553154ed9498 (known to both)

IV used on P1 (known to both)

(in ascii): 1234567890123456

(in hex) : 31323334353637383930313233343536

Next IV (known to both)

(in ascii): 1234567890123457

(in hex) : 31323334353637383930313233343537

Bob sent an encrypted message and Eve knows its content is either 'Yes' or 'No'.

Our goal is to construct a message P2 and ask Bob to encrypt it and give you the ciphertext. And use this opportunity to figure out whether the actual content of P1 is Yes or No .

We can find out P2 by using:

$$P1 = IV1 \text{ (XOR)} IV2 \text{ (XOR)} P2$$

Let us assume that P2 = 'Yes'

P2 (in hex): 59 65 73

After XOR operations.

On conversion to ASCII, text generated = “Yes”

Observation:

In conclusion, through a chosen-plaintext attack in the context of AES-CBC encryption, it is possible to make an educated guess about the content of P1. By carefully crafting a new plaintext message P2 and observing the changes in the ciphertext when P2 is encrypted, one can infer the likely content of P1 based on whether the ciphertext changes significantly or not.

Task 7: Programming using the Crypto library [Extra Credit]

Goal: The goal of this task is to write a program to find the key that is used for the encryption when plaintext and ciphertext are given.

- AES-128-CBC cipher is used for encryption.
- The key used to encrypt this plaintext is an English word shorter than 16 characters; the word can be found from a typical English dictionary. Since the word has less than 16 characters (i.e. 128 bits), pound signs (#: hexadecimal value is 0x23) are appended to the end of the word to form a key of 128 bits.

The plaintext, ciphertext, and IV are listed in the following:

- Plaintext (total 21 characters): This is a secret tool
- Ciphertext (in hex format):
ece6753e938f8f903cabbbe12d395bf5f7eae38ad918a2d3e1c3a832476d5c7a
- IV (in hex format): 010203040506070809000a0b0c0d0e0f

Steps Performed:

- Download the words.txt file from the internet.
- Create a .txt for storing the plaintext called plaintext.txt of 21 characters with given data.
- Program for finding the key:
- Run using the below command by including the lcrypto library.
`gcc -o findKey findKey.c -lcrypto`
- After that we will verify if the key generated is correct using the openssl command by encrypting the given plaintext with the generated key and given key.
- The above command will generate a ciphertext file and using hex fiend we can compare the contents of the generated file with the given ciphertext in hex format to verify if the key is correct.

Program: [findKey.c]

Git link: https://github.com/anushasonte/CS458_InformationSecurity/blob/main/Lab2/findKey.c

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

void hex_string_to_bytes(const char *hex_string, unsigned char *bytes, size_t hex_string_len) {
    for (size_t i = 0; i < hex_string_len / 2; i++) {
        sscanf(&hex_string[2 * i], "%2hhx", &bytes[i]);
    }
}

int main() {
    // Define the provided plaintext, ciphertext, and IV
    const char *plaintext = "This is a secret tool";
    const char *ciphertext_hex
    = "ece6753e938f8f903cabbbe12d395bf5f7eae38ad918a2d3e1c3a832476d5c7a";
    const char *iv_hex = "010203040506070809000a0b0c0d0e0f";

    // Load the English word list from a file (words.txt)
    FILE *wordlist = fopen("words.txt", "r");
    if (wordlist == NULL) {
        perror("Failed to open wordlist file");
        return 1;
    }

    char word[16];
    while (fgets(word, sizeof(word), wordlist)) {
        // Remove newline characters from the word
        word[strcspn(word, "\n")] = '\0';
```

```

// Initialize OpenSSL's EVP context
EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
EVP_CIPHER_CTX_init(ctx);

// Convert IV and ciphertext from hex strings to bytes
unsigned char iv[16];
unsigned char ciphertext[128];
hex_string_to_bytes(iv_hex, iv, 32);
hex_string_to_bytes(ciphertext_hex, ciphertext, 128);

// Pad the word with pound signs (#) to form a 128-bit key
unsigned char key[16];
memset(key, '#', sizeof(key));

// Copy the word into the key, ensuring it doesn't exceed the key size
size_t word_len = strlen(word);
if (word_len > 16) {
    word_len = 16;
}
memcpy(key, word, word_len);

// Set up the decryption parameters
EVP_DecryptInit_ex(ctx, EVP_aes_128_cbc(), NULL, key, iv);

// Decrypt the ciphertext
unsigned char decrypted[128];
int decrypted_len;
EVP_DecryptUpdate(ctx, decrypted, &decrypted_len, ciphertext, 128);
EVP_DecryptFinal_ex(ctx, decrypted + decrypted_len, &decrypted_len);

```

```

// Check if the decrypted plaintext matches the given plaintext
if (strncmp((char *)decrypted, plaintext, strlen(plaintext)) == 0) {
    printf("Found the key: %s\n", word);
    break;
}

EVP_CIPHER_CTX_free(ctx);
}

fclose(wordlist);
return 0;
}

```

Output:

```

anushasp@Anushas-MacBook-Air Task7 % gcc -o findKey findKey.c $LDFLAGS $CPPFLAGS -lcrypto
ld: warning: dylib (/opt/homebrew/opt/openssl@1.1/lib/libcrypto.dylib) was built for newer macOS version (14.0) than being linked (13.3)
anushasp@Anushas-MacBook-Air Task7 %
Found the key: safety
anushasp@Anushas-MacBook-Air Task7 %

```

The Key found = 'safefy'

Steps to run the program:

- Download the program ‘findKey.c’ using below gitlink
https://github.com/anushasonte/CS458_InformationSecurity/blob/main/Lab2/findKey.c
- Download the words.txt file using the below gitlink
https://github.com/anushasonte/CS458_InformationSecurity/blob/main/Lab2/words.txt
- Put both the above files in the same folder.
- Use below commands:
 gcc -o findKey findKey.c -lcrypto
 ./findKey

Verification:

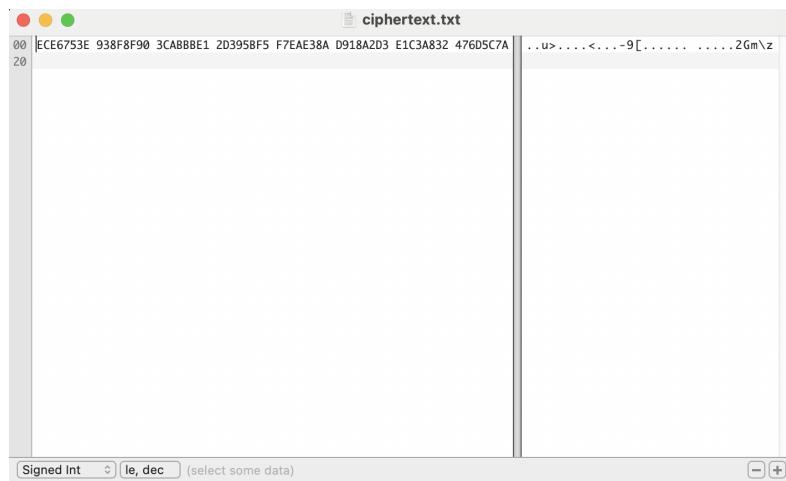
- Let us try to encrypt the given plain text using the key found and see if the ciphertext generated would be the same as given in the input.

Key (in hex format): 736166657479232323232323232323

IV (in hex format): 010203040506070809000a0b0c0d0e0f

```
anushasp@Anushas-MacBook-Air Task7 % openssl enc -e -aes-128-cbc -in plaintext.txt -out ciphertext.txt -K 73616665747923232323232323232323 -iv 010203040506070  
809000a0b0c0d0e0f  
anushasp@Anushas-MacBook-Air Task7 %
```

Opening the generated ciphertext.txt using hexfiend.



We can see that the hexadecimal format of the ciphertext is exactly the same as that of the key! Hence, the key generated by the program is correct and

Key = safety

(in hex): 73616665747923232323232323232323

All the files generated as part of this lab for all tasks can be found below:

https://github.com/anushasonte/CS458_InformationSecurity/blob/main/Lab2/Lab2.zip