

Task 1: Frequency Analysis

Running the given ciphertext through the frequency parser at the provided link gives us the following letter counts:

n : 488	z : 95
y : 373	l : 90
v : 348	g : 83
x : 291	b : 83
u : 280	r : 82
q : 276	e : 76
m : 264	d : 59
h : 235	f : 49
t : 183	s : 19
i : 166	j : 5
p : 156	k : 5
a : 116	o : 4
c : 104	w : 1

Attempting to replace the most common letters in the ciphertext with the most common english letters does not present readable text, meaning the frequency of letters in the ciphertext does not correspond directly with that of English as a whole. However, the parser also provides the most common trigrams in our ciphertext “ytn” and “vup”. The most common trigrams in english are “the” and “and”. Since both of our common cipher trigrams appear as separate words in the ciphertext, this is encouraging.

We guess that $x \Rightarrow O$, and $q \Rightarrow S$ because q commonly appears at the end of words. From here, enough words take shape that we can begin to guess letters based on the context of individual words.

```
THE OSaAhS TzhN ON SzNDAd lHmaH SEECs AgOzT hmrHT AbTEh THmS iONr STHANrE
ALAhDS Thme THE gArrEh bEEiS imSE A NONARENAhMAN TOO

THE ALAhDS hAaE lAS gOOSEnDED gd THE DEcmSE Ob HAhFed lEmNSTEmN AT mTS OzTSET
AND THE AeeAhENT mceiOSmON Ob HmS bmic aOceAND AT THE END AND mT lAS SHAeED gd
THE EcEhRENAe Ob cETOO TmcES ze giAasrOlN eOimTmaS AhcaANDd AaTmfMSc AND
A NATmONAI aONfEHsATmON AS ghMEb AND cAD AS A bEfEh DhEAc AgOzT LHETHEh THEhE
OzrHT TO gE A ehESmDENT lMnbhED THE SEASON DmDNT ozST SEEC EkThA iONr mT lAS
EkThA iONr gEaAzSE THE OSaAhS lEHc cOfED TO THE bmhST lEEsEND mN cAhAh TO
AfOMd aONbimaTmNr lMTH THE aiOSmNr aEHecOND Ob THE lMNTeh OidcemaS THANsS
edEONraHANr

ONE gmr jzESTmON SzhhOzNDmNr THmS dEAhS AaADEcd ALAhDS mS HOL Oh mb THE
aEHecOND lmi ADDhESS cETOO ESaEamAiId AbTEh THE rOiDEN riOGES lHmaH gEAce
A ozgmiANT aOcmNrOzT eAhTd bOh TmcES ze THE cOfEcENT SeEAhHEADED gd
eOLEhbzi HOiidLOOD lOCEN LHO HEieED hAmSE cmilmONS Ob DOiAhS TO bmrHT SEkzAi
HAhASSCENT AhOzND THE aOzNThd

SmrNAimNr THEmh SzeeOhT rOiDEN riOGES ATTENDEES SLATHED THEcSEifES mN giAas
SeOhTED iAeEi emNS AND SOZNDED Obb AgOzT SEkmST eOLEh mcgAiANaES bhOc THE hED
aAheET AND THE STArE ON THE Amh E lAS aAiIED OzT AgOzT eAd mNEjzmTd AbTEh
mTS bOhcEh ANAhOh aATT SADIeh jzmT ONaE SHE iEAhNED THAT SHE lAS cASmNr bAh

Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS
```

Eventually we get:

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

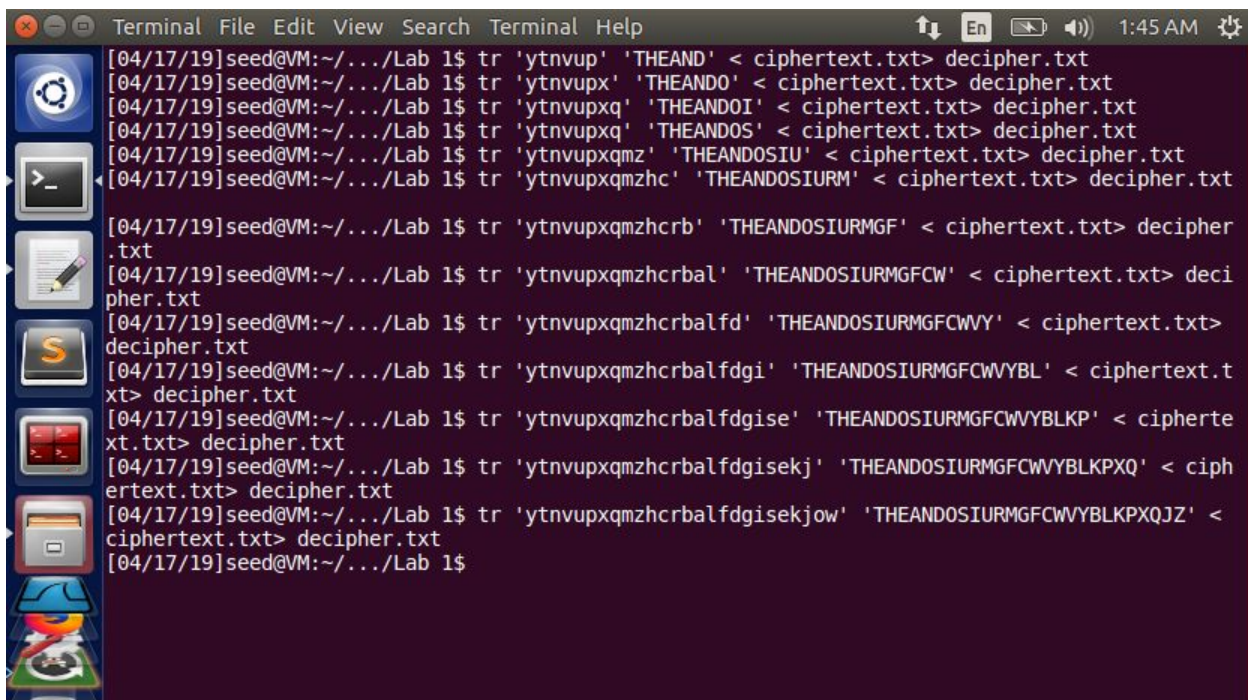
THE AWARDS RACE WAS BOOKENDED BY THE DEMISE OF HARVEY WEINSTEIN AT ITS OUTSET
AND THE APPARENT IMPLOSION OF HIS FILM COMPANY AT THE END AND IT WAS SHAPED BY
THE EMERGENCE OF METOO TIMES UP BLACKGOWN POLITICS ARMCANDY ACTIVISM AND
A NATIONAL CONVERSATION AS BRIEF AND MAD AS A FEVER DREAM ABOUT WHETHER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

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

SIGNALING THEIR SUPPORT GOLDEN GLOBES ATTENDEES SWATHED THEMSELVES IN BLACK
SPORTED LAPEL PINS AND SOUNDED OFF ABOUT SEXIST POWER IMBALANCES FROM THE RED
CARPET AND THE STAGE ON THE AIR E WAS CALLED OUT ABOUT PAY INEQUITY AFTER
ITS FORMER ANCHOR CATT SADLER QUIT ONCE SHE LEARNED THAT SHE WAS MAKING FAR
```

Plain Text ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS

Here you can see the process of guessing letters to substitute, two at a time:



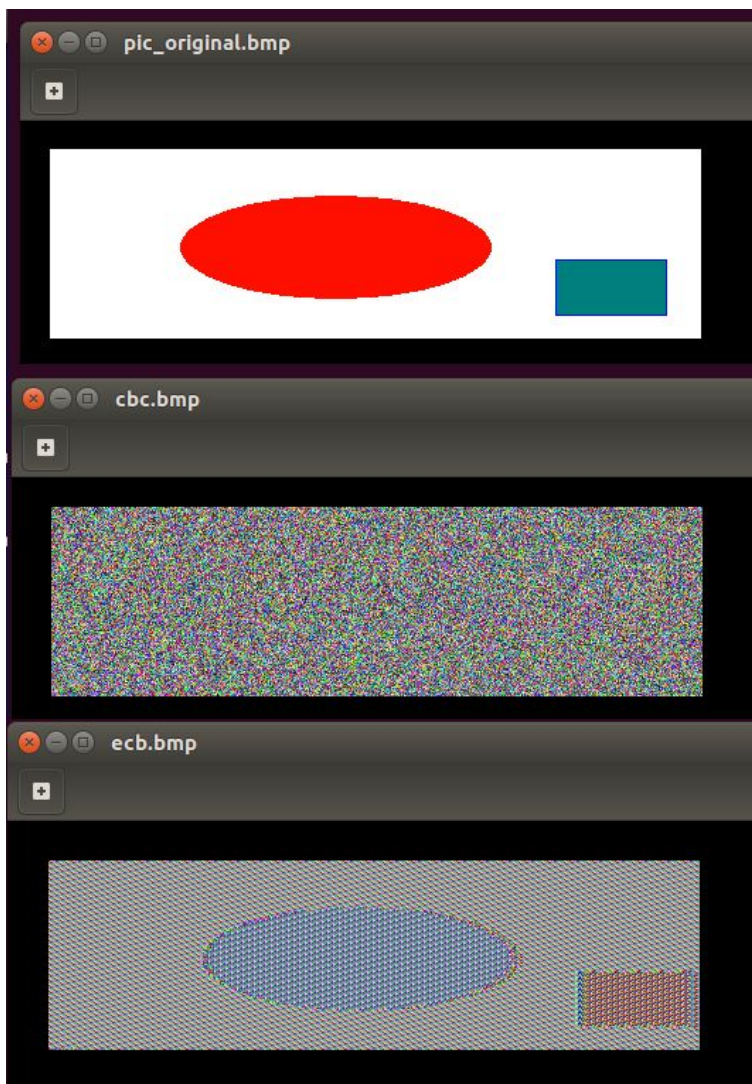
```
Terminal File Edit View Search Terminal Help 1:45 AM
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvup' 'THEAND' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvupx' 'THEANDO' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvupxq' 'THEANDOI' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvupxq' 'THEANDOS' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvupxqmz' 'THEANDOSIU' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvupxqmzhc' 'THEANDOSIURM' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvupxqmzhcbrb' 'THEANDOSIURMGF' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvupxqmzhcbrbal' 'THEANDOSIURMGFCW' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvupxqmzhcbrbalfd' 'THEANDOSIURMGFCWVY' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvupxqmzhcbrbalfdgi' 'THEANDOSIURMGFCWVYBL' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvupxqmzhcbrbalfdgise' 'THEANDOSIURMGFCWVYBLKP' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvupxqmzhcbrbalfdgisekj' 'THEANDOSIURMGFCWVYBLKPXQ' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$ tr 'ytnvupxqmzhcbrbalfdgisekjow' 'THEANDOSIURMGFCWVYBLKPXQJZ' < ciphertext.txt> decipher.txt
[04/17/19]seed@VM:~/.../Lab 1$
```


Task 2: Encryption using Different Ciphers and Modes

```
Terminal
[04/17/19]seed@VM:~/.../Task2$ openssl enc -des-ede3-cbc -e -in plain.txt -out cipher.bin -K 0011223344556677888
9aabbccddeeff -iv 0102030405060708
[04/17/19]seed@VM:~/.../Task2$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher2.bin -K 0011223344556677888
9aabbccddeeff -iv 0102030405060708lear
non-hex digit
invalid hex iv value
[04/17/19]seed@VM:~/.../Task2$ openssl enc -aes-128-ctr -e -in plain.txt -out cipher3.bin -K 0011223344556677888
9aabbccddeeff -iv 0102030405060708
[04/17/19]seed@VM:~/.../Task2$ ls
cipher2.bin cipher3.bin cipher.bin plain.txt
[04/17/19]seed@VM:~/.../Task2$
```

Using openssl enc command we can choose what cipher and mode of operation we want. An input file and output file is specified and we can choose our own key and IV too.

Task 3: Encryption Mode – ECB vs. CBC



```
[04/17/19]seed@VM:~/.../Task3$ openssl enc -aes-128-cbc -e -in pic_original.bmp -out pic_cbc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[04/17/19]seed@VM:~/.../Task3$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out pic_ecb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
warning: iv not use by this cipher
[04/17/19]seed@VM:~/.../Task3$ head -c 54 pic_original.bmp > header
[04/17/19]seed@VM:~/.../Task3$ cat header pic_ecb.bin > ecb.bmp
[04/17/19]seed@VM:~/.../Task3$ cat header pic_cbc.bin > cbc.bmp
[04/17/19]seed@VM:~/.../Task3$ eog ecb.bmp
[04/17/19]seed@VM:~/.../Task3$ eog cbc.bmp
```

ECB is deterministic meaning that the same block of input will produce the same block of output. For example, we can see areas that are white in the original image are encrypted the same way. With images like these, we are able to make out the borders of different objects. If this mode of operation was used to try to hide the image, it would be a failure as we can make out the two shapes, one being an oval and the other being a square. The positioning of the shapes are not affected at all either.

Task 4: Padding

```
[04/17/19]seed@VM:~/.../Task4$ openssl enc -aes-128-ecb -e -in helloworld.txt -out helloworld_ecb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
warning: iv not use by this cipher
[04/17/19]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -e -in helloworld.txt -out helloworld_cbc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[04/17/19]seed@VM:~/.../Task4$ openssl enc -aes-128-cfb -e -in helloworld.txt -out helloworld_cfb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[04/17/19]seed@VM:~/.../Task4$ openssl enc -aes-128-ofb -e -in helloworld.txt -out helloworld_ofb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[04/17/19]seed@VM:~/.../Task4$ ls -al
total 28
drwxrwxr-x 2 seed seed 4096 Apr 17 03:24 .
drwxr-xr-x 6 seed seed 4096 Apr 17 03:20 ..
-rw-rw-r-- 1 seed seed 48 Apr 17 03:25 helloworld_cbc.bin
-rw-rw-r-- 1 seed seed 32 Apr 17 03:25 helloworld_cfb.bin
-rw-rw-r-- 1 seed seed 48 Apr 17 03:25 helloworld_ecb.bin
-rw-rw-r-- 1 seed seed 32 Apr 17 03:25 helloworld_ofb.bin
-rw-rw-r-- 1 seed seed 32 Apr 17 03:21 helloworld.txt
[04/17/19]seed@VM:~/.../Task4$
```

CBC and ECB modes have padding. CFB and OFB modes do not have padding because they encrypt by only one bit or byte at a time. Characters are a single bytes so no padding is needed.

```
[04/17/19]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -e -in f1.txt -out f1.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[04/17/19]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -e -in f2.txt -out f2.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[04/17/19]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -e -in f3.txt -out f3.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[04/17/19]seed@VM:~/.../Task4$ ls -al
total 52
drwxrwxr-x 2 seed seed 4096 Apr 17 03:47 .
drwxr-xr-x 6 seed seed 4096 Apr 17 03:20 ..
-rw-rw-r-- 1 seed seed 16 Apr 17 03:47 f1.bin
-rw-rw-r-- 1 seed seed 5 Apr 17 03:41 f1.txt
-rw-rw-r-- 1 seed seed 16 Apr 17 03:47 f2.bin
-rw-rw-r-- 1 seed seed 10 Apr 17 03:41 f2.txt
-rw-rw-r-- 1 seed seed 32 Apr 17 03:47 f3.bin
-rw-rw-r-- 1 seed seed 16 Apr 17 03:41 f3.txt
```

We can see that f1.txt (5 bytes) and f2.txt (10 bytes) after encryption both became 16 bytes. We also see that f3.txt which was 16 bytes to before encryption ended up with a size of 32 bytes. This leads us to believe that padding occurs in blocks of 16 bytes.

```

[04/17/19]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -d -in f1.bin -out f1_dec.txt -K 0011223344
5566778889aabbccddeeff -iv 0102030405060708 -nopad
[04/17/19]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -d -in f2.bin -out f2_dec.txt -K 0011223344
5566778889aabbccddeeff -iv 0102030405060708 -nopad
[04/17/19]seed@VM:~/.../Task4$ openssl enc -aes-128-cbc -d -in f3.bin -out f3_dec.txt -K 0011223344
5566778889aabbccddeeff -iv 0102030405060708 -nopad
[04/17/19]seed@VM:~/.../Task4$ hexdump -C f1_dec.txt
00000000 31 32 33 34 35 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b |12345.....|
00000010
[04/17/19]seed@VM:~/.../Task4$ hexdump -C f2_dec.txt
00000000 31 32 33 34 35 36 37 38 39 30 06 06 06 06 06 06 |1234567890.....|
00000010
[04/17/19]seed@VM:~/.../Task4$ hexdump -C f3_dec.txt
00000000 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 |1234567890123456|
00000010 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 |.....|
00000020

```

f1_dec.txt had repeating '0b's, which is hex for 11. Since the padding is 16 bytes and the file is only 5 bytes, the repeating numbers tell us how many bytes are left until a multiple of 16 is reached (which is out padding size). We can see the same for f2 and f3.

Task 5: Error Propagation – Corrupted Cipher Text

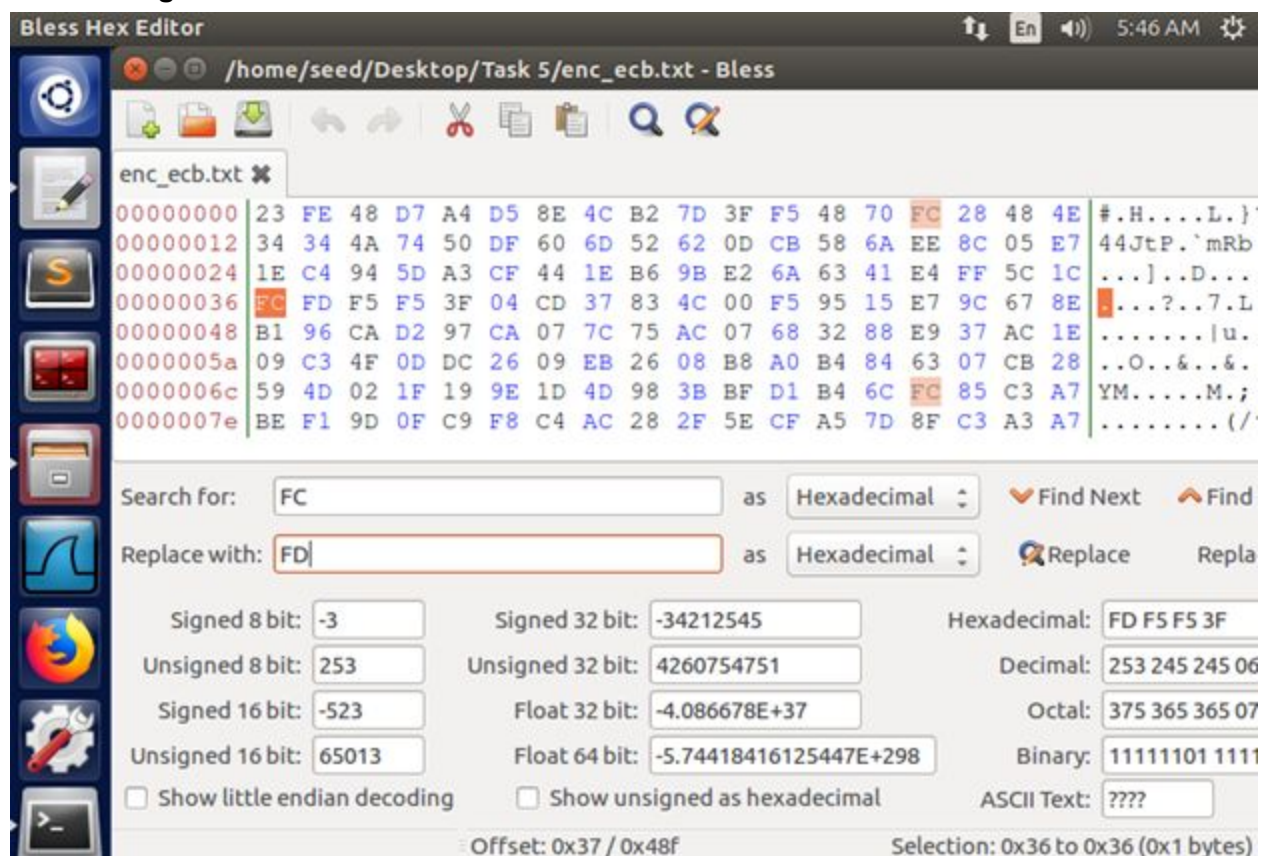
Decrypting w/ Encryption Mode ECB

Before Conduction Task Prediction: With the assumption that to obtain a plain text block, you need the corresponding cipher block, if the cipher block is corrupted, then the respective plain text block will also be corrupted.

Command to encrypt >1000 byte file (plainTextFile.txt) → enc_ecb.txt

```
[04/17/19]seed@VM:~/../Task 5$ openssl enc -aes-128-ecb -e -in plainTextFile.txt -out enc_ecb.txt -K 00112233445566778889aabbccddeeff
```

The 4th row, 1st column is the 55th byte because each row is 18 bytes. Replaced FC with FD to change 1 bit.



Command to decrypt corrupted encrypted text file → dec_ecb.txt

```
[04/17/19]seed@VM:~/../Task 5$ openssl enc -aes-128-ecb -d -in enc_ecb.txt -out dec_e
cb.txt -K 00112233445566778889aabbccddeeff
```

Result of decrypting the corrupted ciphertext

```
thishasabunchofsentences space words thishasabun[09][14]ù¹[09][02]0}[09][02]0ÔXRçu¥Bace
wordsthishasabunchofsentences space wordsthishasabunchofsentences space
wordsthishasabunchofsentences space wordsthishasabunchofsentences space
wordsthishasabunchofsentences space wordsthishasabunchofsentences space
wordsthishasabunchofsentences space wordsthishasabunchofsentences space
wordsthishasabunchofsentences space wordsthishasabunchofsentences space
wordsthishasabunchofsentences space wordsthishasabunchofsentences space
wordsthishasabunchofsentences space wordsthishasabunchofsentences space
wordsthishasabunchofsentences space wordsthishasabunchofsentences space
wordsthishasabunchofsentences space wordsthishasabunchofsentences space
wordsthishasabunchofsentences space wordsthishasabunchofsentences space words
thishasabunchofsentences space wordsthishasabunchofsentences space
wordsthishasabunchofsentences space wordsthishasabunchofsentences space
wordsthishasabunchofsentences space wordsthishasabunchofsentences space words
```

Conclusion:

Since each plaintext block is encrypted separately and each ciphertext block is decrypted separately, you can see how the corrupted ciphertext block effect it's respective plaintext block and nothing else.

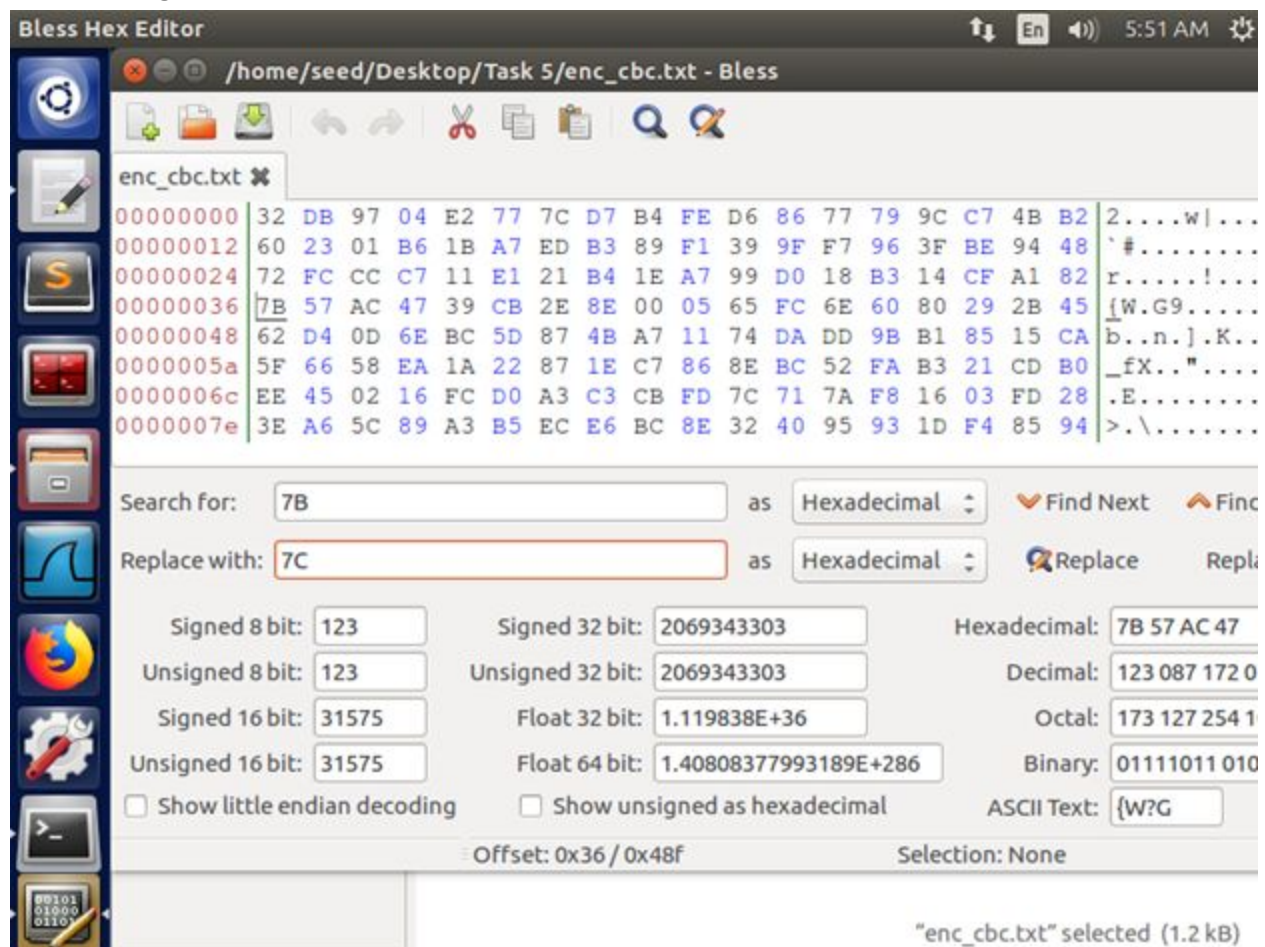
Decrypting w/ Encryption Mode CBC

Before Conduction Task Prediction: If a single bit in the cipher bit is damaged, only the two received blocks of plaintexts will be damaged.

Command to encrypt >1000 byte file (plainTextFile.txt) → enc_cbc.txt

```
[04/17/19]seed@VM:~/../Task 5$ openssl enc -aes-128-cbc -e -in plainTextFile.txt -out enc_cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

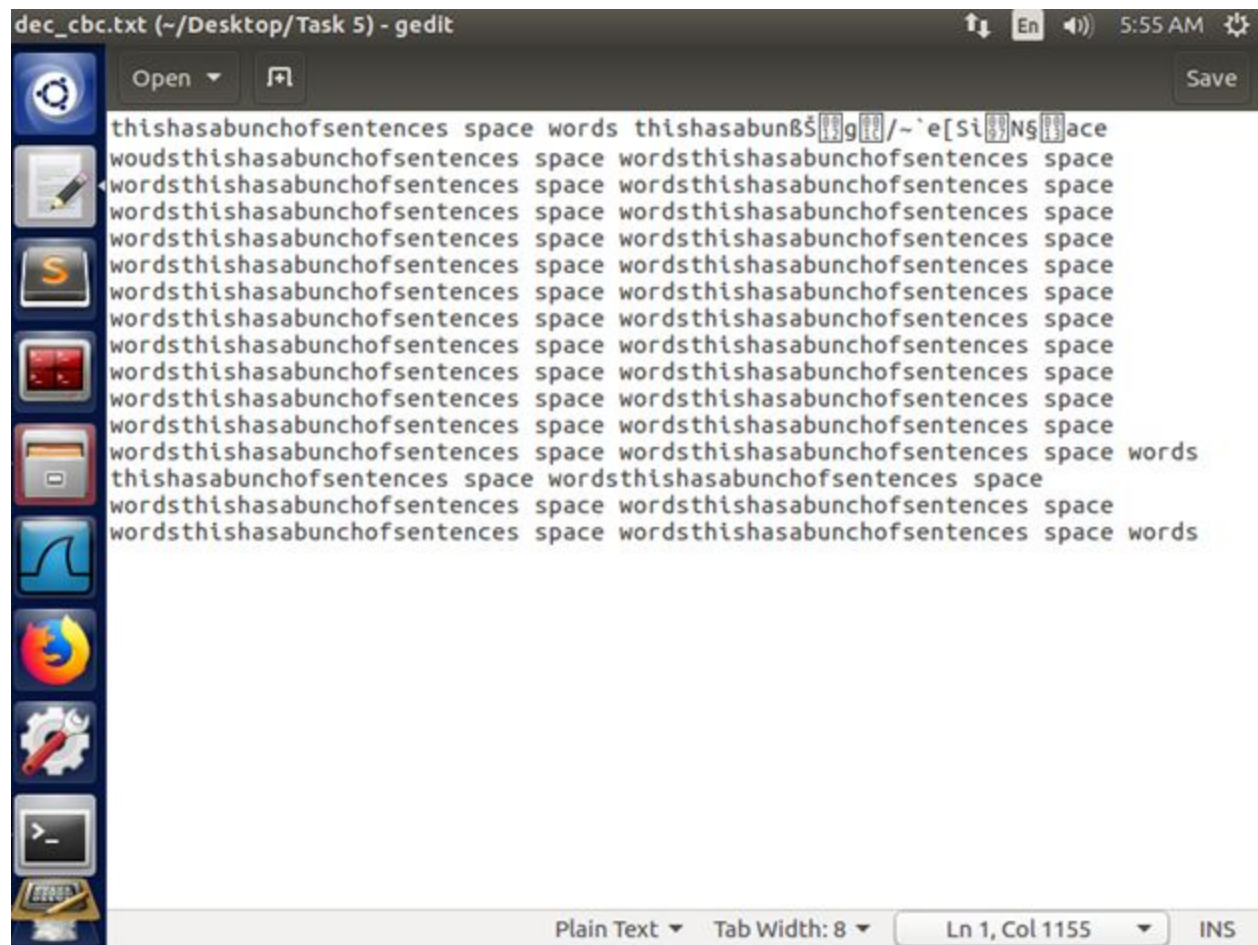
The 4th row, 1st column is the 55th byte because each row is 18 bytes. Replaced 7B with 7C to change 1 bit.



Command to decrypt corrupted encrypted text file → dec_cbc.txt

```
[04/17/19]seed@VM:~/../Task 5$ openssl enc -aes-128-cbc -d -in enc_cbc.txt -out dec_cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

Result of decrypting the corrupted ciphertext



Conclusion:

You can witness with the result two blocks of plaintext being impacted due to the single corrupted block of ciphertext since the encryption algorithm consists of a single block of ciphertext being used to decrypt into two blocks of plaintext.

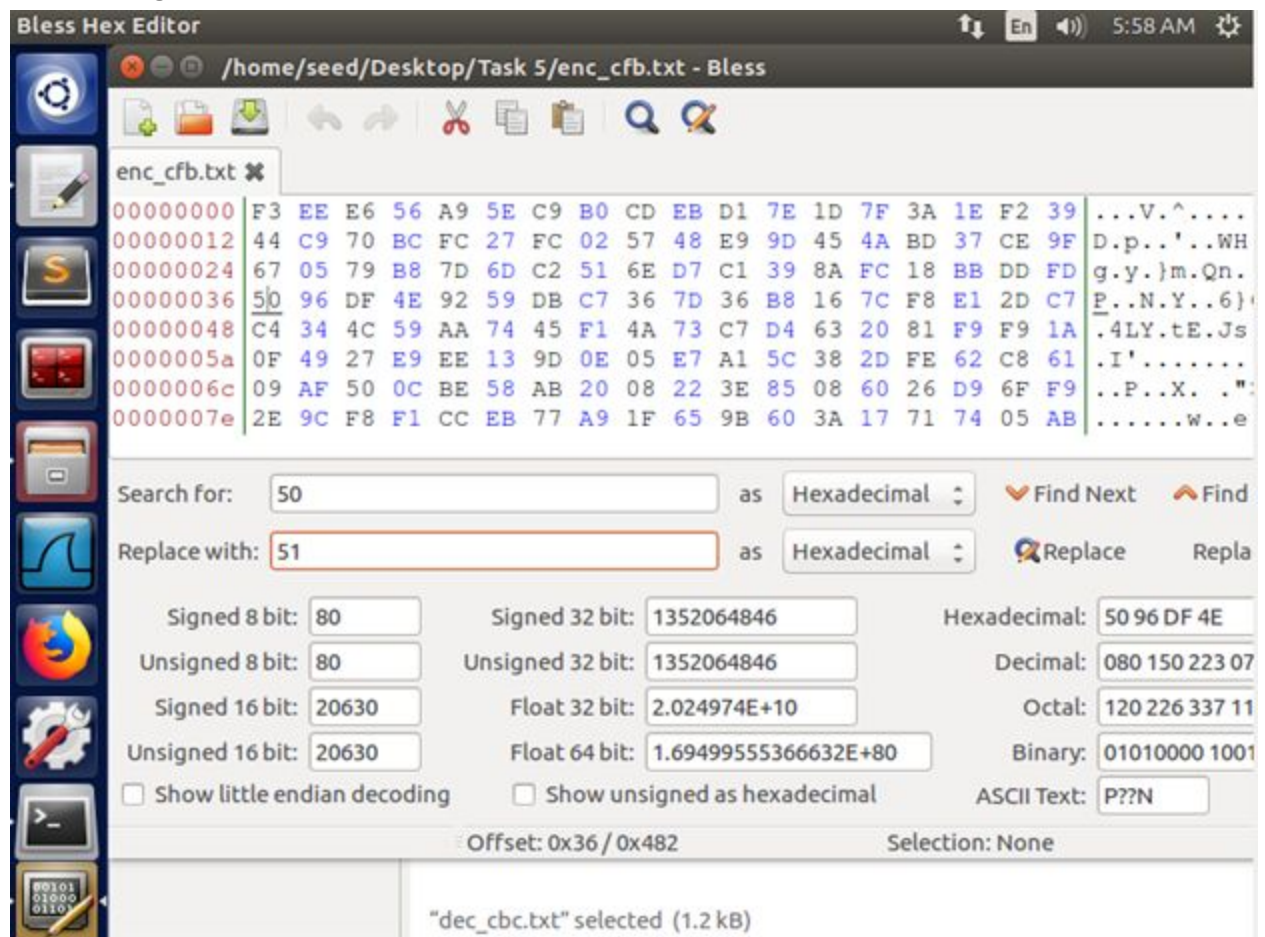
Decrypting w/ Encryption Mode CFB

Before Conduction Task Prediction: Similar to CFB, if a single bit in the cipher bit is damaged, only the two received blocks of plaintexts will be damaged.

Command to encrypt >1000 byte file (plainTextFile.txt) → enc_cfb.txt

```
[04/17/19]seed@VM:~/../Task 5$ openssl enc -aes-128-cfb -e -in plainTextFile.txt -out enc_cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

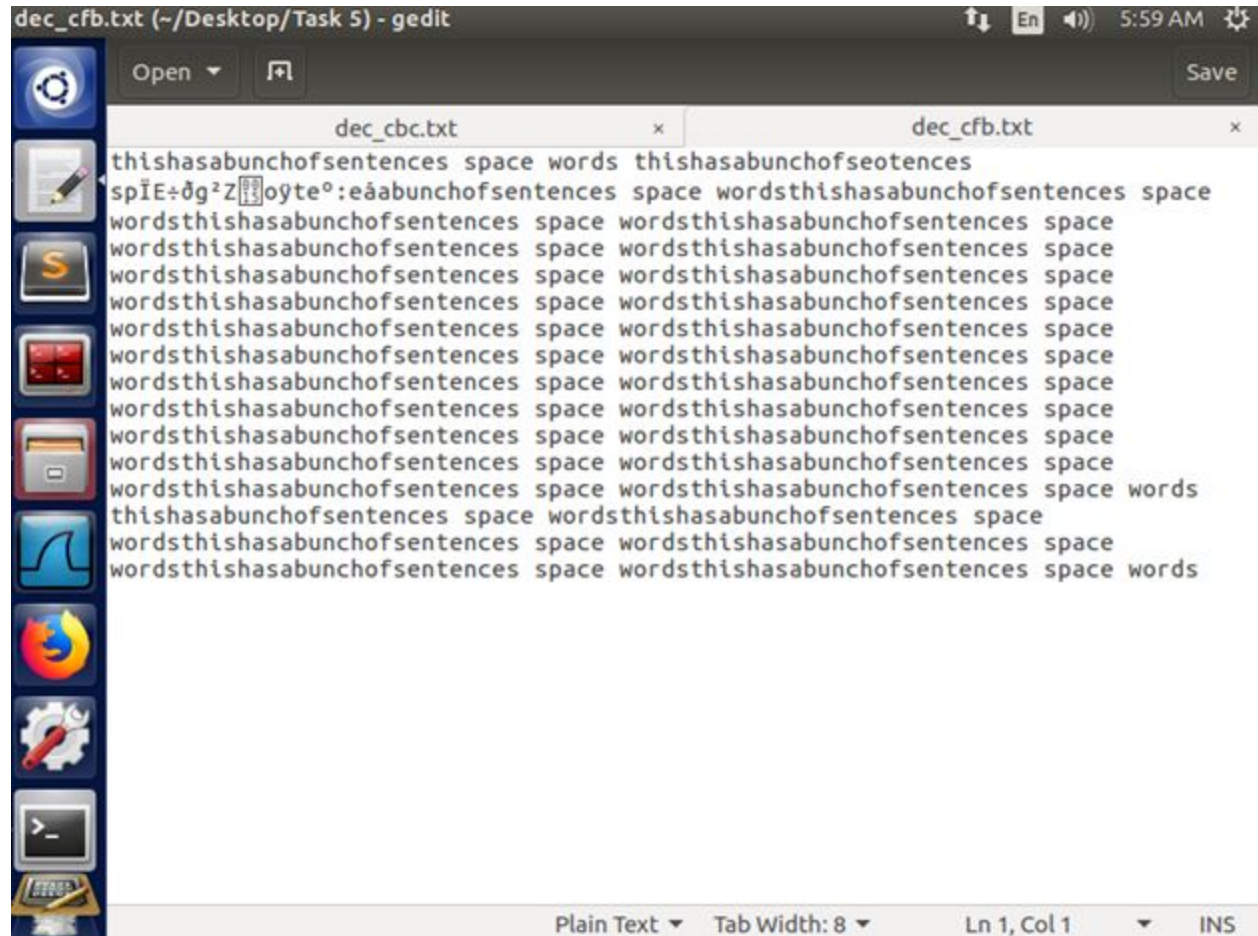
The 4th row, 1st column is the 55th byte because each row is 18 bytes. Replaced 50 with 51 to change 1 bit.



Command to decrypt corrupted encrypted text file → dec_cfb.txt

```
[04/17/19]seed@VM:~/.../Task 5$ openssl enc -aes-128-cfb -d -in enc_cfb.txt -out dec_cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

Result of decrypting the corrupted ciphertext



Conclusion:

Similar to cbc, you can see how two blocks of plaintext are impacted due to a single block of corrupted ciphertext.

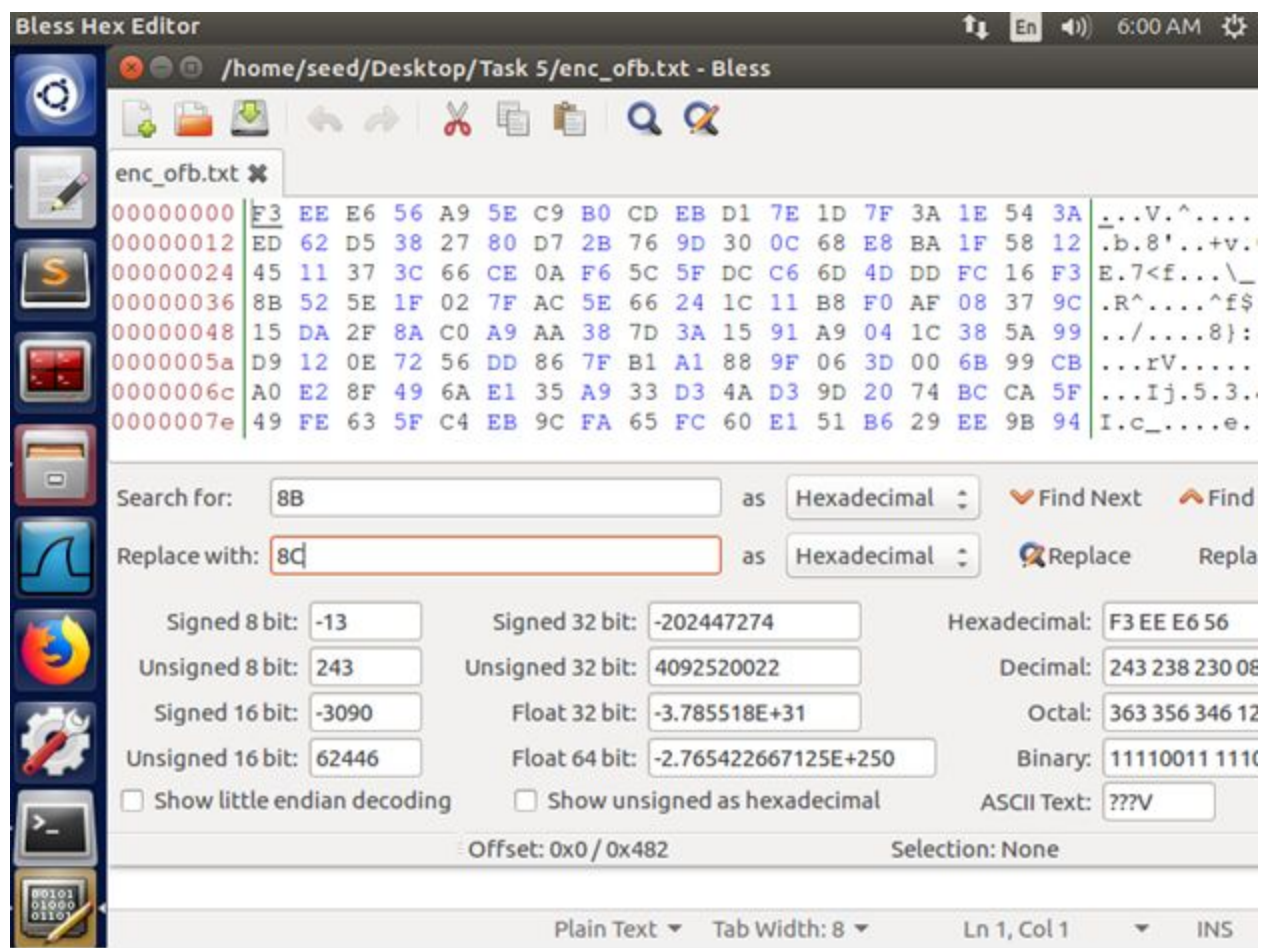
Decrypting w/ Encryption Mode OFB

Before Conduction Task Prediction: If a single bit is corrupted in either block of ciphertext or plaintext, then the respective block will be corrupted as well.

Command to encrypt >1000 byte file (plainTextFile.txt) → enc_ofb.txt

```
[04/17/19]seed@VM:~/.../Task 5$ openssl enc -aes-128-ofb -e -in plainTextFile.txt -out enc_ofb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

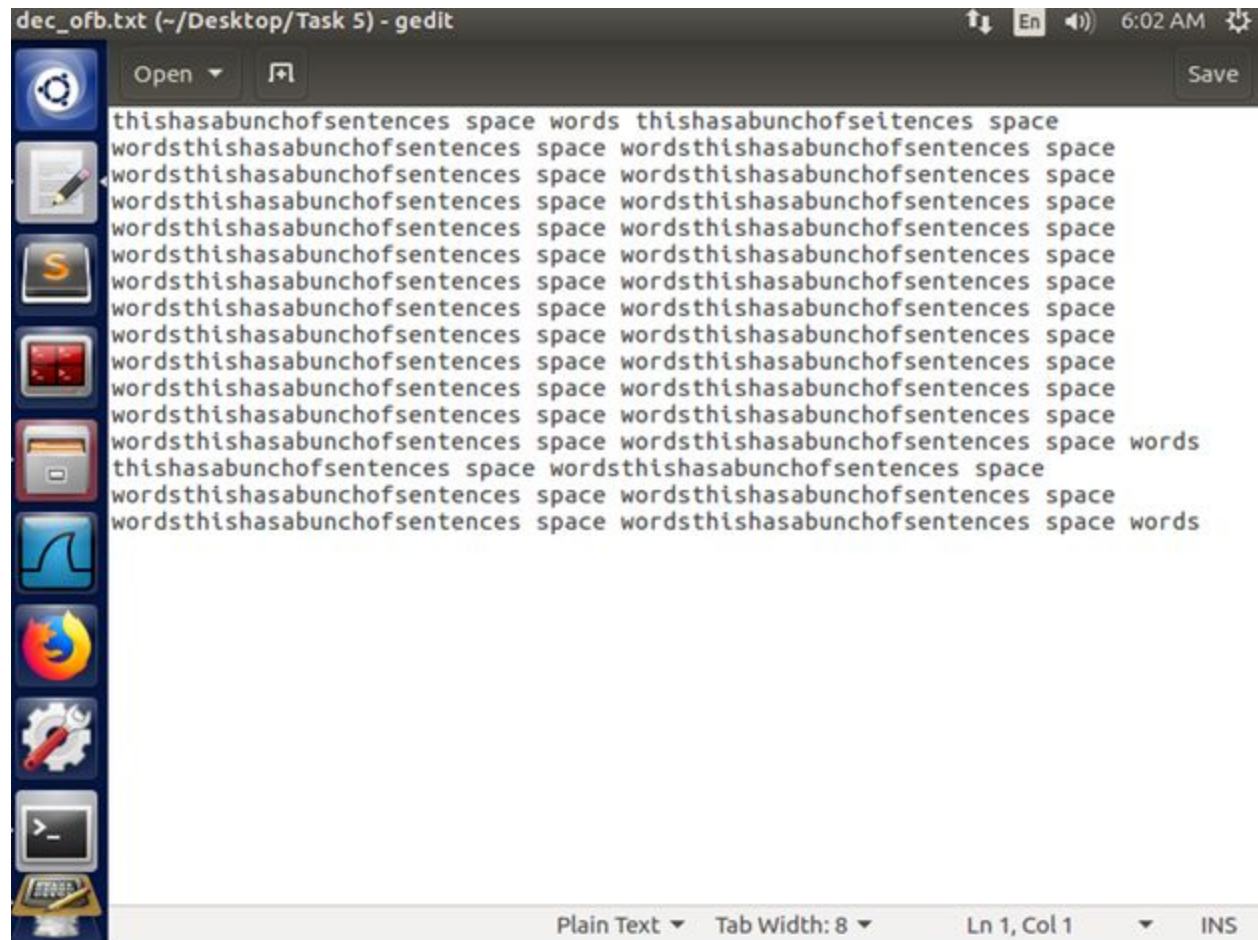
The 4th row, 1st column is the 55th byte because each row is 18 bytes. Replaced 8B with 8C to change 1 bit.



Command to decrypt corrupted encrypted text file → dec_ofb.txt

```
[04/17/19]seed@VM:~/.../Task 5$ openssl enc -aes-128-ofb -d -in enc_ofb.txt -out dec_ofb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

Result of decrypting the corrupted ciphertext



```
dec_ofb.txt (~/Desktop/Task 5) - gedit
thishasabunchofsences space words thishasabunchofsences space
wordsthishasabunchofsences space wordsthishasabunchofsences space
wordsthishasabunchofsences space wordsthishasabunchofsences space
wordsthishasabunchofsences space wordsthishasabunchofsences space
wordsthishasabunchofsences space wordsthishasabunchofsences space
wordsthishasabunchofsences space wordsthishasabunchofsences space
wordsthishasabunchofsences space wordsthishasabunchofsences space
wordsthishasabunchofsences space wordsthishasabunchofsences space
wordsthishasabunchofsences space wordsthishasabunchofsences space
wordsthishasabunchofsences space wordsthishasabunchofsences space words
thishasabunchofsences space wordsthishasabunchofsences space
wordsthishasabunchofsences space wordsthishasabunchofsences space
wordsthishasabunchofsences space wordsthishasabunchofsences space words
Plain Text Tab Width: 8 Ln 1, Col 1 INS
```

Conclusion:

Only the respective single block of plaintext was impacted since in the decryption algorithm consists of a single connection between the respective single block of plaintext and ciphertext.

Task 6: Initial Vector (IV)

6.1: We encrypted a plaintext multiple times as instructed. All three of the encrypted output files were different, despite two of them having the same IV and key. These two utilized the same IV:

```
Salted_ \90\C9@ \AA6 \AF
 \9EV \8A\F7\AS\C11%
<; \F6 \F3\80\E4Q \82\81\8F%K \F4\8F\ER\9C \90k" \E6\ERe\F5\E7 \AD\B9\97Yl}
 \D8RK9 \E1\F \Q* \D5 \99 > \89\A5\C89 \B0H* \9E\AC\BA\DD\FB- \A9\C1~$ \C7r\9E(\A4\00%
 \E9NA*r \E \BC \89 \8AY \80\80 \D4} \88\92! \E1^N\AF\FA\93\90pD \E0i~ \ACW}@
 \FA
```

```
Salted_ " N?æ Öwİ ù%û·Æ uif]ÿpİÄ.Nx€ Šäp4vwëGw- uÚÉó Û'ò=@ç C /}
äö«>îiāñ iOÄ7 Lâ]æMò "Üx ±0Ö.°âu. yç&r=D , |ÄùÒc R!%k ú\B¥ò°u
±A«²p|
```

This one used a separate one:

```
Salted_ Z \8B\A8\F6\E4yg\EE
HJ^% \DB\E7\F0\98 \D6\F2 \9E\B2\F4\B2\DC) % \E8\A2~{Mu \8E\FB\C4qN?U \C1\92\F3\CE
 \ERw07( \E5a\F0\AB\80/7 \B7\8F\C1\EA \EFn \B7\BA\98 \E6;k6 \C8\E57 \0Ak3 \98\D2? \AE \BF
 \F5\8F\C3 \89KM \C8\06Kb&B z \F7 6 \A0 \F7\B5\8A
 \Cdt \A0 \DF \F4\9Cs \00\A8 \EB8 \A \E0
```

As can be seen in the terminal log:

```
[04/18/19]seed@VM:~/.../Lab 1$ openssl enc -aes128 -k f1f1f1 -iv 12345 -in in1.txt -out out1.t
xt
[04/18/19]seed@VM:~/.../Lab 1$ openssl enc -aes128 -k f1f1f1 -iv 12345 -in in1.txt -out out2.t
xt
[04/18/19]seed@VM:~/.../Lab 1$ openssl enc -aes128 -k f1f1f1 -iv 12345 -in in1.txt -out out2.t
xt
[04/18/19]seed@VM:~/.../Lab 1$ openssl enc -aes128 -k f1f1f1 -iv 67890 -in in1.txt -out out3.t
xt
```

There appears to be some pattern between the two with the same IV in that the second one does not have illegal characters. Perhaps this gives away some kind of information about the text?

6.2: In OFB mode, the plaintext is only XOR'd with the encrypted block at the end, meaning we can take the hex of the plaintext and XOR it again with ciphertext to get the blocks. Then once again, we XOR the blocks with the unknown ciphertext to get the second plain text. This would not work the same for CFB mode, because each block of plaintext is used to encrypt the next, so the encryption blocks are not the same between ciphertexts. However, the first block would be revealed, which may be enough to help crack the rest.

546869732069732061206b6e6f776e206d65737361676521 : plaintext 1
XOR
A469b1c502c1cab966965e50425438e1bb1b5f9037a4c159 : ciphertext 1
EQUALS
f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478 : encryption blocks

f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478 : encryption blocks
XOR
Bf73bcd3509299d566c35b5d450337e1bb175f903fafc159 : ciphertext 2
EQUALS
4f726465723a204c61756e63682061206d697373696c6521 : plaintext 2

The plaintext then translates to “**Order: Launch a missile!**”

6.3:

- We couldn't manage to find a way to represent the solution in actual commands because we couldn't find a way to apply multiple XOR's to get the ciphertext we wanted. To explain conceptually how we would've performed the *chosen-plaintext attack* requires the understanding how we could take advantage of knowing the next IV and how bitwise XOR works.

What we can do is send Bob to encrypt

$$(IV_{second} \oplus IV_{first} \oplus "yes")$$

Then we can apply another XOR resulting in this...

$$IV_{second} \oplus (IV_{second} \oplus IV_{first} \oplus "yes")$$

And with the knowledge of take XOR of the same bit strings results in 0...

$$\cancel{IV_{second}} \oplus (\cancel{IV_{second}} \oplus IV_{first} \oplus "yes")$$

So what we have left is

$$(IV_{first} \oplus "yes")$$

- So with this, if the resulting ciphertext Eve gets back is the same as the ciphertext that is known, then we know the plaintext is "yes", otherwise, if the ciphertext is not known, then we know that the only other option is "no".

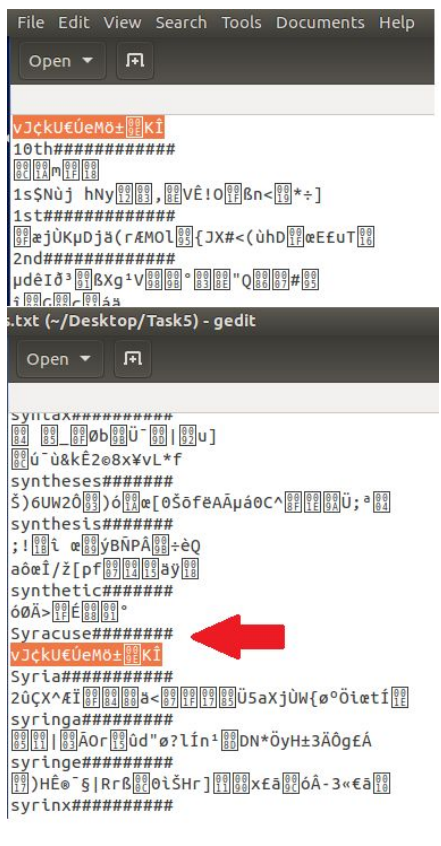
This is our attempt to apply FIRST, the first iv, and after apply the second iv twice, but resulted in something that was much longer than the original ciphertext.

```
[04/19/19]seed@VM:~/.../Task 6 (copy)$ openssl enc -aes-128-cbc -e -in yes.txt -out firstIV_yes.txt -K 00112233445566778899aabbccddeeff -iv 31323334353637383930313233343536
[04/19/19]seed@VM:~/.../Task 6 (copy)$ openssl enc -aes-128-cbc -e -in firstIV_yes.txt -out firstIV_yes_afteroneIV2.txt -K 00112233445566778899aabbccddeeff -iv 31323334353637383930313233343537
[04/19/19]seed@VM:~/.../Task 6 (copy)$ openssl enc -aes-128-cbc -e -in firstIV_yes_afteroneIV2.txt -out firstIV_yes_aftertwoIV2.txt -K 00112233445566778899aabbccddeeff -iv 31323334353637383930313233343537
```

Task 7: Programming using the Crypto Library

```
[04/19/19]seed@VM:~/.../Task5$ gcc -o keyfind keyfind.c -lcrypto
[04/19/19]seed@VM:~/.../Task5$ ./keyfind > allwords.txt
[04/19/19]seed@VM:~/.../Task5$
```

Compile the program using gcc then redirect the output (using '>') to an empty text file. Here the output was redirected to "allwords.txt".

 <p>The screenshot shows a text editor window titled ".txt (~/Desktop/Task5) - gedit". The text inside is a list of words, many of which are partially obscured by redaction boxes. The word "Syracuse#####" is highlighted in orange. A red arrow points to this line from the right-hand side of the image.</p>	<p>Saving the output to a text file allows for search functions. In the program, the known cipher is printed to the first line. Simply highlighting the first line and searching for it in the text file gives an instant result. Above the match is the input key associated with it. In this case, it happens to be "Syracuse#####".</p>
---	--

```
[04/19/19]seed@VM:~/.../Task5$ echo -n "Syracuse#####" > key
[04/19/19]seed@VM:~/.../Task5$ xxd -p key
537972616375736523232323232323
[04/19/19]seed@VM:~/.../Task5$
```

The key is then converted to hex.


```
[04/19/19]seed@VM:~/.../Task5$ xxd file.txt
00000000: 5468 6973 2069 7320 6120 746f 7020 7365  This is a top se
00000010: 6372 6574 2e                                cret.
[04/19/19]seed@VM:~/.../Task5$ openssl enc -aes-128-cbc -e -in file.txt -out file_enc.b
in -K 537972616375736523232323232323 -iv aabbccddeeff00998877665544332211
[04/19/19]seed@VM:~/.../Task5$ xxd file_enc.bin
00000000: 764a a26b 55a4 da65 4df6 b19e 4bce 00f4  vJ.kU..eM...K...
00000010: ed05 e093 46fb 0e76 2583 cb7d a2ac 93a2  ....F..v%..}....
[04/19/19]seed@VM:~/.../Task5$
```

```
Plaintext (total 21 characters): This is a top secret.
Ciphertext (in hex format): 764aa26b55a4da654df6b19e4bce00f4
                             ed05e09346fb0e762583cb7da2ac93a2
IV (in hex format):         aabbccddeeff00998877665544332211
```

Using the openssl command on original text with the obtained key (in hex) and given IV, it is encrypted into “file_enc.bin”. The xxd command is used on the encrypted file which displayed the entire file in hex. The given ciphertext and the hex values of the new file are compared, it is an exact match. This means the key really was “Syracuse#####”.

Keyfind.c code:

```
//Eduard Klimenko
//Plaintext (total 21 characters): This is a top secret.
//Ciphertext (in hex format): 764aa26b55a4da654df6b19e4bce00f4ed05e09346fb0e762583cb7da2ac93a2
//IV (in hex format): aabbccddeeff00998877665544332211

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <openssl/evp.h>
#include <openssl/aes.h>

unsigned char* encrypt(FILE *ifp, FILE *ofp, unsigned char* key, unsigned char* temp_iv);
void copy_string(char *target, char *source);

int main(void) {
    FILE *fin = fopen("file.txt", "rb"); //input file
    FILE *fout = fopen("file_enc.bin", "wb"); //output file
    FILE *words = fopen("words.txt", "r"); //word list
    FILE *inc = fopen("cipher", "rb"); //given ciphertext
    FILE *ivfile = fopen("iv", "rb"); //given IV

    // arrays for input word, key, iv, and cipher
    char word[50];
    unsigned char key[16];
    unsigned char iv[16];
    unsigned char cipher[32];

    // reads in the cipher and iv
    fscanf(inc, "%s", cipher);
    printf("%s\n", cipher); //prints given cipher on the first line
    fscanf(ivfile, "%s", iv);

    // goes through each word in the word list
    while (fscanf(words, "%s", word) != EOF) {
        copy_string(key, word); //copies into key string
        printf("%s\n", key);
    }
}
```

```

        printf("%s\n", encrypt(fin,fout,key,iv));
    }

    fclose(fin);
    fclose(fout);
    fclose(words);
    fclose(inc);
    fclose(ivfile);
    return 0;
}

// encrypts a given file
unsigned char* encrypt(FILE *ifp, FILE *ofp, unsigned char* temp_key, unsigned char* temp_iv)
{
    //Get file size
    fseek(ifp, 0L, SEEK_END);
    int fsize = ftell(ifp);
    //set back to normal
    fseek(ifp, 0L, SEEK_SET);

    int outLen1 = 0; int outLen2 = 0;
    unsigned char *indata = malloc(fsize);
    unsigned char *outdata = malloc(fsize*2);
    //unsigned char key[16] = "example#####";
    unsigned char iv[] = {170,187,204,221,238,255,0,153,136,119,102,85,68,51,34,17}; // given IV

    //Read File
    fread(indata,sizeof(char),fsize, ifp);//Read Entire File

    //Set up encryption
    EVP_CIPHER_CTX ctx;
    EVP_EncryptInit(&ctx,EVP_aes_128_cbc(), temp_key, iv);
    EVP_EncryptUpdate(&ctx,outdata,&outLen1,indata,fsize);
    EVP_EncryptFinal(&ctx,outdata + outLen1,&outLen2);
    //fwrite(outdata,sizeof(char),outLen1 + outLen2,ofp);
    EVP_CIPHER_CTX_cleanup(&ctx);
    return outdata;
}

// helper function to add #'s as padding
void copy_string(char *target, char *source) {
    int lencounter = 0;
    while (*source && lencounter < 16) {
        *target = *source;
        source++;
        target++;
        lencounter++;
    }
    while (lencounter < 16){
        *target = '#';
        target++;
        lencounter++;
    }
    *target = '\\0';
}

```