Task 1: Frequency Analysis Against Monoalphabetic Substitution Cipher

- Step 1: In this step, I learned how the ‘tr’ command can convert upper case to lower case from plaintext, removed anything but the letters and spaces between words. The purpose of this step is to show how to prepare plaintext for a symmetric encryption (seen in first screenshot).
- Step 2: In this step, I used a python function to create a random assortment of the alphabet to use as my encryption key (seen in first screenshot).

```
[09/24/19]JCogswell@Attacker:~/.../Lab3$ tr [:upper:] [:lower:] < article.txt > lowercase.txt
bash: article.txt: No such file or directory
[09/24/19]JCogswell@Attacker:~/.../Lab3$ tr [:upper:] [:lower:] < ciphertext.txt > lowercase.txt
[09/24/19]JCogswell@Attacker:~/.../Lab3$ tr -cd '[a-z][\n][:space:]' < lowercase.txt > plaintext.txt
[09/24/19]JCogswell@Attacker:~/.../Lab3$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> s = "abcdefghijklmnopqrstuvwxyz"
>>> list = random.sample(s, len(s))
>>> ''.join(list)
'hkdluxfjtcrgnaybswqmoeizvp'
>>> 
```

- Step 3: In this step I applied the encryption using the ‘tr’ command and my newly created encryption key. Note: I only encrypted letters, not the space and return characters. I did this a few times because as I was learning how this worked, I ended up messing up the provided encrypted file by encrypting it again with my own key. I then realized that step 3 was not meant to have us apply our encryption to the provided encrypted file. This step covers how to encrypt a plaintext article using my key.

```
[09/24/19]JCogswell@Attacker:~/.../Lab3$ tr 'abcdefghijklmnopqrstuvwxyz' 'hkdluxfjtcrgnaybswqmoeizvp' < pl
aintext.txt > ciphertext.txt
[09/24/19]JCogswell@Attacker:~/.../Lab3$ tr 'hkdluxfjtcrgnaybswqmoeizvp' < plaintext.txt > ciphertxt.txt
tr: missing operand after ‘hkdluxfjtcrgnaybswqmoeizvp’
Two strings must be given when translating.
Try 'tr --help' for more information.
[09/24/19]JCogswell@Attacker:~/.../Lab3$ tr 'hkdluxfjtcrgnaybswqmoeizvp' 'abcdefghijklmnopqrstuvwxyz' < pl
aintext.txt > ciphertxt.txt
[09/24/19]JCogswell@Attacker:~/.../Lab3$ tr 'hkdluxfjtcrgnaybswqmoeizvp' 'abcdefghijklmnopqrstuvwxyz' < ci
phertxt.txt > plaintext.txt
[09/24/19]JCogswell@Attacker:~/.../Lab3$ tr 'hkdluxfjtcrgnaybswqmoeizvp' 'abcdefghijklmnopqrstuvwxyz' < ci
phertext.txt > ciphertxt.txt
[09/24/19]JCogswell@Attacker:~/.../Lab3$ 
```

Now, the provided encrypted file does not come with a key. Therefore, I need to use the ‘grep’ command to count the amount of times a specified character or combination of characters appears. I can use this information to figure out what the likely plaintext character is for each of the encrypted characters. For example, the only one-letter words in the English language are ‘I’ and ‘A’ so I can use this information to figure out which characters are A and which ones are I. Additionally, the word ‘the’ is widely used so I can search for a three letter combination of characters and use grep to count the occurrences of that specific combination of three characters. As shown in the screenshot, ‘ytn’ occurs 53 times so I can make the assumption that y=T, t=H, and n=E. I use the ‘tr’ command to swap these encrypted characters for my choice of plaintext characters.

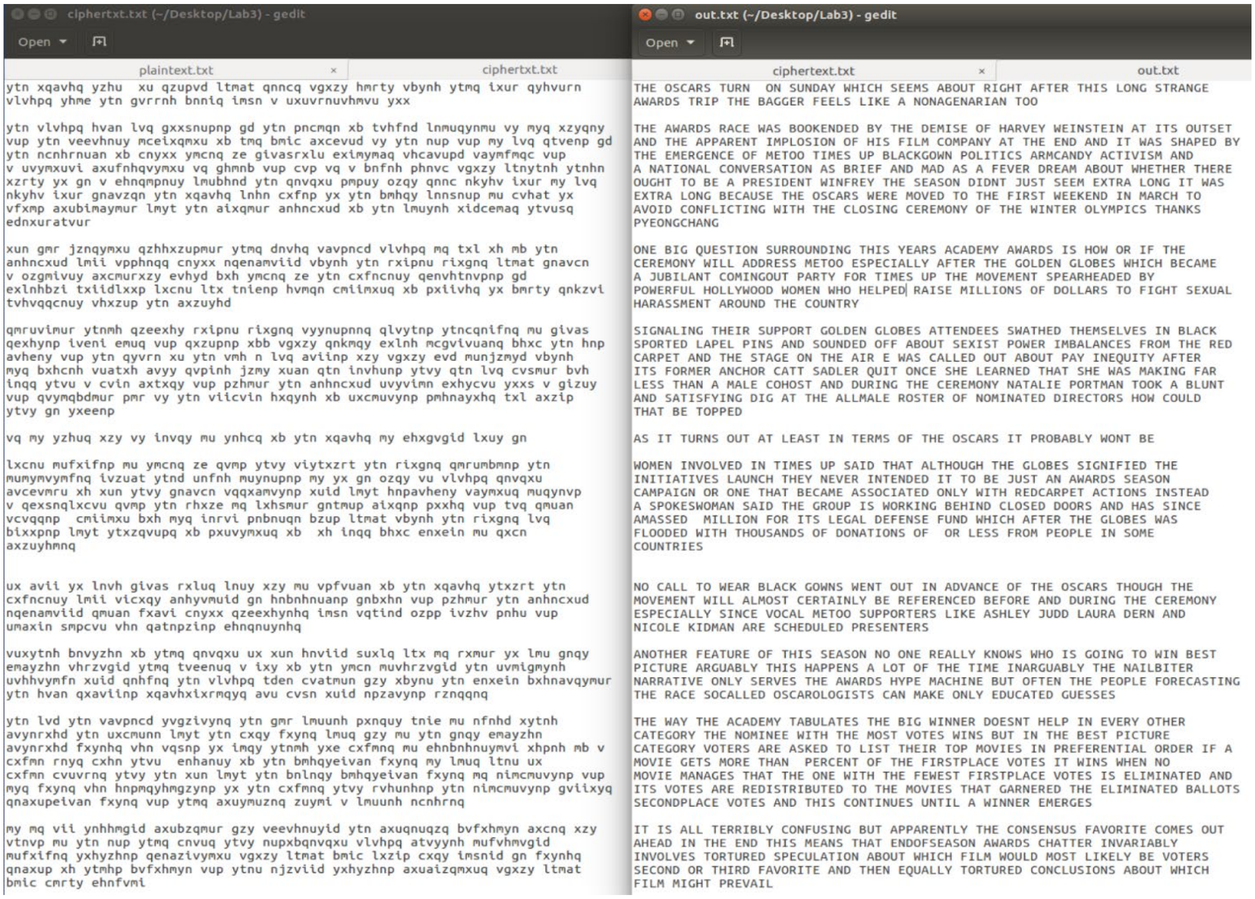
I can reload the file which contains my changes and see if it starts making any sense. After doing this a few times with very common words, it becomes very easy to figure out which encrypted characters belong to which English characters. I did this several times and figured out the entire encryption key and converted the encrypted article to the plaintext article that it was created from.

```
Terminal

[09/25/19]JCogswell@Attacker:~/.../Lab3$ grep -c 'ytn' ciphertxt.txt
53
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytn' 'THE' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ grep -c 'a' ciphertxt.txt
57
[09/25/19]JCogswell@Attacker:~/.../Lab3$ grep -c 'v' ciphertxt.txt
69
[09/25/19]JCogswell@Attacker:~/.../Lab3$ ty 'v' 'A' ciphertxt.txt
ty: command not found
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'v' 'A' ciphertxt.txt
tr: extra operand 'ciphertxt.txt'
Try 'tr --help' for more information.
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'v' 'A' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ grep -c 'Aup' ciphertxt.txt
0
[09/25/19]JCogswell@Attacker:~/.../Lab3$ grep -c 'vup' ciphertxt.txt
25
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'up' 'ND' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvup' 'THEAND' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupx' 'THEANDO' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxh' 'THEANDOR' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhm' 'THEANDORI' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ grep -c 'q' ciphertxt.txt
67
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmq' 'THEANDORIS' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ grep -c 'z' ciphertxt.txt
52
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmqz' 'THEANDORISC' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmq' 'THEANDORIS' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmqz' 'THEANDORISU' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmqza' 'THEANDORISUC' < ciphertxt.txt > out.t
xt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmqzad' 'THEANDORISUCY' < ciphertxt.txt > out
.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmqzadlrb' 'THEANDORISUCYWGF' < ciphertxt.txt
 > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmqzadlrbcgi' 'THEANDORISUCYWGFMBL' < ciphert
xt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmqzadlrbcgise' 'THEANDORISUCYWGFMBLKP' < cip
hertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmqzadlrbcgisef' 'THEANDORISUCYWGFMBLKPV' < c
iphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmqzadlrbcgisefjo' 'THEANDORISUCYWGFMBLKPVQJ'
 < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmqzadlrbcgisefjok' 'THEANDORISUCYWGFMBLKPVQJ
X' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tr 'ytnvupxhmqzadlrbcgisefjokw' 'THEANDORISUCYWGFMBLKPVQ
JXZ' < ciphertxt.txt > out.txt
[09/25/19]JCogswell@Attacker:~/.../Lab3$
```

Left: encrypted file.

Right: unencrypted file.


ciphertxt.txt (~/Desktop/Lab3) - gedit

Open

plaintext.txt × ciphertxt.txt

ytn xqavhq yzhu xu qzupvd ltmat qnncq vgxzy hmrty vbynh ytmq ixur qyhvurn
vlvhpq yhme ytn gvrrnh bnniq imsn v uxuvrnuvhmvu yxx

ytn vlvhpq hvan lvq gxxsnupnp gd ytn pncmqn xb tvhfnd lnmuqynmu vy myq xzyqny
vup ytn veevhnuy mceixqmxu xb tmq bmic axcevud vy ytn nup vup my lvq qtvenp gd
ytn ncnhrnuan xb cnyxx ymcnq ze givasrxlu eximymaq vhcavupd vaymfmqc vup
v uvymxuvi axufnhqvymxu vq ghmnb vup cvp vq v bnfnh phnvc vgxzy ltnytnh ytnhn
xzrty yx gn v ehnqmpnuy lmubhnd ytn qnvqxu pmpuy ozqy qnnc nkyhv ixur my lvq
nkyhv ixur gnavzqn ytn xqavhq lnhn cxfnp yx ytn bmhqy lnnsnup mu cvhat yx
vfxmp axubimaymur lmyt ytn aixqmur anhncxud xb ytn lmuynh xidcemaq ytvusq
ednxuratvur

xun gmr jznqymxu qzhhxzupmur ytmq dnvhq vavpncd vlvhpq mq txl xh mb ytn
anhncxud lmii vpphnqq cnyxx nqenamviid vbynh ytn rxipnu rixgnq ltmat gnavcn
v ozgmivuy axcmurxzy evhyd bxh ymcnq ze ytn cxfncnuy qenvhtnvpnp gd
exlnhbzi txiidlxxp lxcnu ltx tnienp hvmqn cmiimxuq xb pxiivhq yx bmrty qnkzvi
tvhvqqcnuy vhxzup ytn axzuyhd

qmruvimur ytnmh qzeexhy rxipnu rixgnq vyynupnnq qlvytnp ytncqnifnq mu givas
qexhynp iveni emuq vup qxzupnp xbb vgxzy qnkmqy exlnh mcgvivuanq bhxc ytn hnp
avheny vup ytn qyvrn xu ytn vmh n lvq aviinp xzy vgxzy evd munjzmyd vbynh
myq bxhcnh vuatxh avyy qvpinh jzmy xuan qtn invhunp ytvy qtn lvq cvsmur bvh
inqq ytvu v cvin axtxqy vup pzhmur ytn anhncxud uvyvimn exhycvu yxxs v gizuy
vup qvymqbdmur pmr vy ytn viicvin hxqynh xb uxcmuvynp pmhnayxhq txl axzip
ytvy gn yxeenp

vq my yzhuq xzy vy invqy mu ynhcq xb ytn xqavhq my ehxgvgid lxuy gn

lxcnu mufxifnp mu ymcnq ze qvmp ytvy viytxzrt ytn rixgnq qmrumbmnp ytn
mumymvymfnq ivzuat ytnd unfnh muynupnp my yx gn ozqy vu vlvhpq qnvqxu
avcevmru xh xun ytvy gnavcn vqqxamvynp xuid lmyt hnpavheny vaymxuq muqynvp
v qexsnqlxcvu qvmp ytn rhxze mq lxhsmur gntmup aixqnp pxxhq vup tvq qmuan
vcvqqnp cmiimxu bxh myq inrvi pnbnuqn bzup ltmat vbynh ytn rixgnq lvq
bixxpnp lmyt ytxzqvupq xb pxuvymxuq xb xh inqq bhxc enxein mu qxcn
axzuyhmnq

ux avii yx lnvh givas rxluq lnuy xzy mu vpfvuan xb ytn xqavhq ytxzrt ytn
cxfncnuy lmii vicxqy anhyvmuid gn hnbnhnuanp gnbxhn vup pzhmur ytn anhncxud
nqenamviid qmuan fxavi cnyxx qzeexhynhq imsn vqtind ozpp ivzhv pnhu vup
umaxin smpcvu vhn qatnpzinp ehnqnuynhq

vuxytnh bnvyzhn xb ytmq qnvqxu ux xun hnviid suxlq ltx mq rxmur yx lmu gnqy
emayzhn vhrzvgid ytmq tveenuq v ixy xb ytn ymcn muvhrzvgid ytn uvmigmynh
uvhhvymfn xuid qnhfnq ytn vlvhpq tden cvatmun gzy xbynu ytn enxein bxhnavqymur
ytn hvan qxaviinp xqavhxixrmqyq avu cvsn xuid npzavynp rznqqnq

ytn lvd ytn vavpncd yvgzivynq ytn gmr lmuunh pxnquy tnie mu nfnhd xytnh
avynrxhd ytn uxcmunn lmyt ytn cxqy fxynq lmuq gzy mu ytn gnqy emayzhn
avynrxhd fxynhq vhn vqsnp yx imqy ytnmh yxe cxfmnq mu ehnbnhnuymvi xhpnh mb v
cxfmn rnyq cxhn ytvu enhanuy xb ytn bmhqyeivan fxynq my lmuq ltnu ux
cxfmn cvuvrnq ytvy ytn xun lmyt ytn bnlnqy bmhqyeivan fxynq mq nimcmuvynp vup
myq fxynq vhn hnpmqyhmgzynp yx ytn cxfmnq ytvy rvhunhnp ytn nimcmuvynp gviixyq
qnaxupeivan fxynq vup ytmq axuymuznq zuymi v lmuunh ncnhrnq

my mq vii ynhhmgid axubzqmur gzy veevhnuyid ytn axuqnuqzq bvfxhmyn axcnq xzy
vtnvp mu ytn nup ytmq cnvuq ytvy nupxbqnvqxu vlvhpq atvyynh mufvhmvgid
mufxifnq yxhyzhnp qenazivymxu vgxzy ltmat bmic lxzip cxqy imsnid gn fxynhq
qnaxup xh ytmhp bvfxhmyn vup ytnu njzviid yxhyzhnp axuaizqmxuq vgxzy ltmat
bmic cmrty ehnfvmi

out.txt (~/Desktop/Lab3) - gedit

Open

ciphertext.txt × out.txt

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

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

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

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

AS IT TURNS OUT AT LEAST IN TERMS OF THE OSCARS IT PROBABLY WONT BE

WOMEN INVOLVED IN TIMES UP SAID THAT ALTHOUGH THE GLOBES SIGNIFIED THE
INITIATIVES LAUNCH THEY NEVER INTENDED IT TO BE JUST AN AWARDS SEASON
CAMPAIGN OR ONE THAT BECAME ASSOCIATED ONLY WITH REDCARPET ACTIONS INSTEAD
A SPOKESWOMAN SAID THE GROUP IS WORKING BEHIND CLOSED DOORS AND HAS SINCE
AMASSED MILLION FOR ITS LEGAL DEFENSE FUND WHICH AFTER THE GLOBES WAS
FLOODED WITH THOUSANDS OF DONATIONS OF OR LESS FROM PEOPLE IN SOME
COUNTRIES

NO CALL TO WEAR BLACK GOWNS WENT OUT IN ADVANCE OF THE OSCARS THOUGH THE
MOVEMENT WILL ALMOST CERTAINLY BE REFERENCED BEFORE AND DURING THE CEREMONY
ESPECIALLY SINCE VOCAL METOO SUPPORTERS LIKE ASHLEY JUDD LAURA DERN AND
NICOLE KIDMAN ARE SCHEDULED PRESENTERS

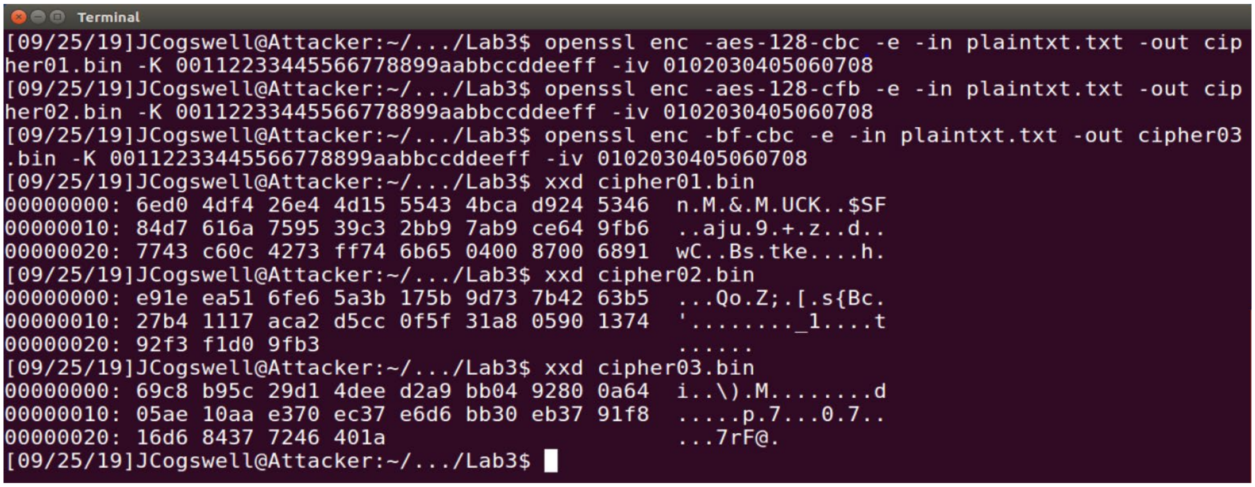
ANOTHER FEATURE OF THIS SEASON NO ONE REALLY KNOWS WHO IS GOING TO WIN BEST
PICTURE ARGUABLY THIS HAPPENS A LOT OF THE TIME INARGUABLY THE NAILBITER
NARRATIVE ONLY SERVES THE AWARDS HYPE MACHINE BUT OFTEN THE PEOPLE FORECASTING
THE RACE SOCALLED OSCAROLOGISTS CAN MAKE ONLY EDUCATED GUESSES

THE WAY THE ACADEMY TABULATES THE BIG WINNER DOESNT HELP IN EVERY OTHER
CATEGORY THE NOMINEE WITH THE MOST VOTES WINS BUT IN THE BEST PICTURE
CATEGORY VOTERS ARE ASKED TO LIST THEIR TOP MOVIES IN PREFERENTIAL ORDER IF A
MOVIE GETS MORE THAN PERCENT OF THE FIRSTPLACE VOTES IT WINS WHEN NO
MOVIE MANAGES THAT THE ONE WITH THE FEWEST FIRSTPLACE VOTES IS ELIMINATED AND
ITS VOTES ARE REDISTRIBUTED TO THE MOVIES THAT GARNERED THE ELIMINATED BALLOTS
SECONDPLACE VOTES AND THIS CONTINUES UNTIL A WINNER EMERGES

IT IS ALL TERRIBLY CONFUSING BUT APPARENTLY THE CONSENSUS FAVORITE COMES OUT
AHEAD IN THE END THIS MEANS THAT ENDOFSEASON AWARDS CHATTER INVARIABLY
INVOLVES TORTURED SPECULATION ABOUT WHICH FILM WOULD MOST LIKELY BE VOTERS
SECOND OR THIRD FAVORITE AND THEN EQUALLY TORTURED CONCLUSIONS ABOUT WHICH
FILM MIGHT PREVAIL

Task 2: Encryption using Different Ciphers and Modes

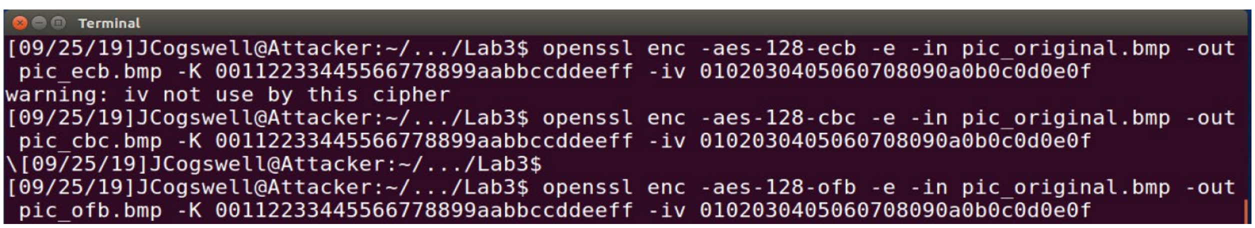
In this task, I used openssl to apply different encryption algorithms to a text file containing the words "Hello World My name is James Cogswell." I did this three times with different algorithms and the output from each encryption went to its own .bin binary file. I then used the 'xxd' command to create a hex dump of the .bin files.



```
Terminal
[09/25/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -e -in plaintxt.txt -out cip
her01.bin -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[09/25/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cfb -e -in plaintxt.txt -out cip
her02.bin -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[09/25/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -bf-cbc -e -in plaintxt.txt -out cipher03
.bin -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[09/25/19]JCogswell@Attacker:~/.../Lab3$ xxd cipher01.bin
00000000: 6ed0 4df4 26e4 4d15 5543 4bca d924 5346  n.M.&.M.UCK..$SF
00000010: 84d7 616a 7595 39c3 2bb9 7ab9 ce64 9fb6  ..aju.9.+.z..d..
00000020: 7743 c60c 4273 ff74 6b65 0400 8700 6891  wC..Bs.tke....h.
[09/25/19]JCogswell@Attacker:~/.../Lab3$ xxd cipher02.bin
00000000: e91e ea51 6fe6 5a3b 175b 9d73 7b42 63b5  ...Qo.Z;.[.s{Bc.
00000010: 27b4 1117 aca2 d5cc 0f5f 31a8 0590 1374  '........_1....t
00000020: 92f3 f1d0 9fb3                           ......
[09/25/19]JCogswell@Attacker:~/.../Lab3$ xxd cipher03.bin
00000000: 69c8 b95c 29d1 4dee d2a9 bb04 9280 0a64  i..\).M........d
00000010: 05ae 10aa e370 ec37 e6d6 bb30 eb37 91f8  .....p.7...0.7..
00000020: 16d6 8437 7246 401a                      ...7rF@.
[09/25/19]JCogswell@Attacker:~/.../Lab3$
```

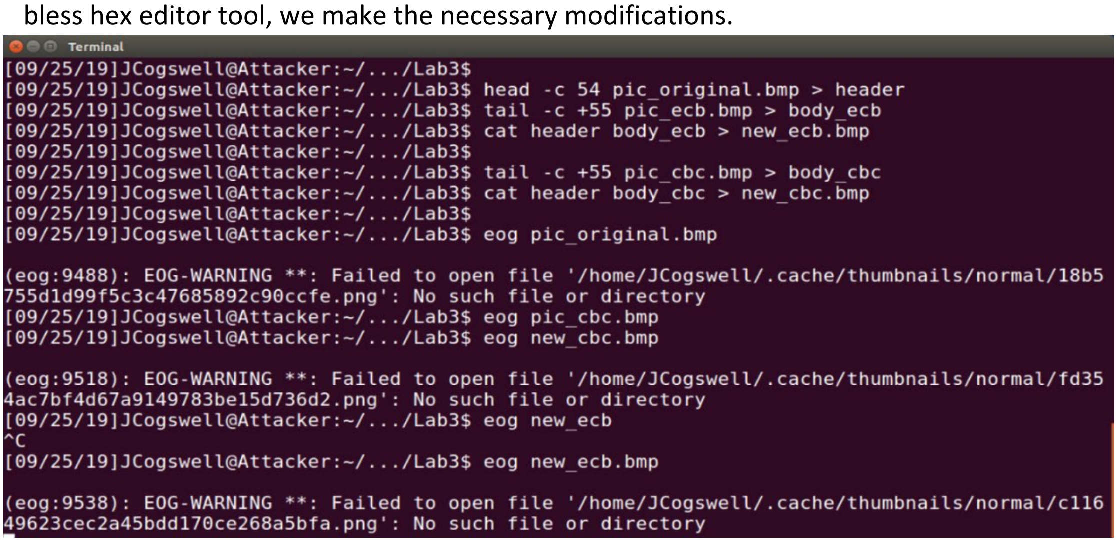
Task 3: Encryption Mode – ECB vs. CBC

In this task, we encrypted a file called 'pic_original.bmp' using openssl and two different types of AES-128 bit: ECB (electronic code book) and CBC (cipher block chaining). Additionally, I encrypted it with OFB but didn't end up using that in this task.



```
Terminal
[09/25/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-ecb -e -in pic_original.bmp -out
 pic_ecb.bmp -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
warning: iv not use by this cipher
[09/25/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -e -in pic_original.bmp -out
 pic_cbc.bmp -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
\[09/25/19]JCogswell@Attacker:~/.../Lab3$
[09/25/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-ofb -e -in pic_original.bmp -out
 pic_ofb.bmp -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
```

In .bmp files the first 54 bytes contain the header information about the picture and if we encrypt it without changing the header information, it won't be treated as a legitimate .bmp file. Therefore, we replace the header of the encrypted picture with the header of the original picture. Using the bless hex editor tool, we make the necessary modifications.



```
Terminal
[09/25/19]JCogswell@Attacker:~/.../Lab3$
[09/25/19]JCogswell@Attacker:~/.../Lab3$ head -c 54 pic_original.bmp > header
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tail -c +55 pic_ecb.bmp > body_ecb
[09/25/19]JCogswell@Attacker:~/.../Lab3$ cat header body_ecb > new_ecb.bmp
[09/25/19]JCogswell@Attacker:~/.../Lab3$
[09/25/19]JCogswell@Attacker:~/.../Lab3$ tail -c +55 pic_cbc.bmp > body_cbc
[09/25/19]JCogswell@Attacker:~/.../Lab3$ cat header body_cbc > new_cbc.bmp
[09/25/19]JCogswell@Attacker:~/.../Lab3$
[09/25/19]JCogswell@Attacker:~/.../Lab3$ eog pic_original.bmp

(eog:9488): EOG-WARNING **: Failed to open file '/home/JCogswell/.cache/thumbnails/normal/18b5
755d1d99f5c3c47685892c90ccfe.png': No such file or directory
[09/25/19]JCogswell@Attacker:~/.../Lab3$ eog pic_cbc.bmp
[09/25/19]JCogswell@Attacker:~/.../Lab3$ eog new_cbc.bmp

(eog:9518): EOG-WARNING **: Failed to open file '/home/JCogswell/.cache/thumbnails/normal/fd35
4ac7bf4d67a9149783be15d736d2.png': No such file or directory
[09/25/19]JCogswell@Attacker:~/.../Lab3$ eog new_ecb
^C
[09/25/19]JCogswell@Attacker:~/.../Lab3$ eog new_ecb.bmp

(eog:9538): EOG-WARNING **: Failed to open file '/home/JCogswell/.cache/thumbnails/normal/c116
49623cec2a45bdd170ce268a5bfa.png': No such file or directory
```

We then use the SEED-provided image viewer, namely *eog*, to display the encrypted pictures. The question in the lab instructions says “Can you derive any useful information about the original picture from the encrypted picture? Please explain your observations.”

Observations: Note: I opened the original image just to help explain my point. As we can see from the three images, the CBC image (new_cbc.bmp) image is of no use to us in trying to figure out what the original image is supposed to show. The ECB image (new_ecb.bmp) shows outlines of the oval and rectangle, so we already know a lot about the original photo. The reason there’s a difference between the CBC and ECB encrypted images is because CBC scrambles plaintext prior to each encryption step. I believe this is why the ECB encrypted image seems to have several lines and the CBC is looks like a television with no cable connection.

Task 4: Padding

In this task, the purpose was to demonstrate how padding works for block ciphers. The mode I chose for this is CBC because it uses padding, but I will describe which modes do and do not require padding at the end of this task. The first thing I did was create new files called f1.txt, f2.txt, and f3.txt and contain 5B, 10B, and 16B of data, respectively. The size of these files are shown in the screenshot where I used “ls -l f*.txt” and it says 5, 10, and 16.

```
Terminal
[09/26/19]JCogswell@Attacker:~/.../Lab3$ echo -n "12345" > f1.txt
[09/26/19]JCogswell@Attacker:~/.../Lab3$ echo -n "1234567890" > f2.txt
[09/26/19]JCogswell@Attacker:~/.../Lab3$ echo -n "1234567890abcdef" > file3.txt
[09/26/19]JCogswell@Attacker:~/.../Lab3$ rm file3.txt
[09/26/19]JCogswell@Attacker:~/.../Lab3$ echo -n "1234567890abcdef" > f3.txt
[09/26/19]JCogswell@Attacker:~/.../Lab3$
[09/26/19]JCogswell@Attacker:~/.../Lab3$ ls -l f*.txt
-rw-rw-r-- 1 JCogswell JCogswell  5 Sep 26 14:03 f1.txt
-rw-rw-r-- 1 JCogswell JCogswell 10 Sep 26 14:03 f2.txt
-rw-rw-r-- 1 JCogswell JCogswell 16 Sep 26 14:03 f3.txt
-rw-rw-r-- 1 JCogswell JCogswell 10 Sep 26 14:01 file2.txt
[09/26/19]JCogswell@Attacker:~/.../Lab3$ rm file2.txt
[09/26/19]JCogswell@Attacker:~/.../Lab3$ ls -l f*.txt
-rw-rw-r-- 1 JCogswell JCogswell  5 Sep 26 14:03 f1.txt
-rw-rw-r-- 1 JCogswell JCogswell 10 Sep 26 14:03 f2.txt
-rw-rw-r-- 1 JCogswell JCogswell 16 Sep 26 14:03 f3.txt
[09/26/19]JCogswell@Attacker:~/.../Lab3$
```

Next, I encrypted each file using AES 128-bit CBC mode and named the encrypted files encf1.bin, encf2.bin, and encf3.bin for f1.txt, f2.txt, and f3.txt, respectively. The key and initialization vector used are in the screenshot. Now, when I type "ls -l encf*.bin" it shows different size files (32B for encf1 and encf2 but 48B for encf3). This is because of the padding added to the file during encryption. When we decrypt these files and save them as decf1.txt, decf2.txt, and decf3.txt, we can use xxd *filename* to get a hex dump of the file which allows us to see the padding of the file. Notice the dots – this is the padding for the file.

```
Terminal
[09/26/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -e -in f1.txt -out encf1.bin
 -k 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
[09/26/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -e -in f2.txt -out encf2.bin
 -k 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
[09/26/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -e -in f3.txt -out encf3.bin
 -k 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
[09/26/19]JCogswell@Attacker:~/.../Lab3$
[09/26/19]JCogswell@Attacker:~/.../Lab3$ ls -l encf*
-rw-rw-r-- 1 JCogswell JCogswell 32 Sep 26 14:06 encf1.bin
-rw-rw-r-- 1 JCogswell JCogswell 32 Sep 26 14:06 encf2.bin
-rw-rw-r-- 1 JCogswell JCogswell 48 Sep 26 14:06 encf3.bin
[09/26/19]JCogswell@Attacker:~/.../Lab3$
[09/26/19]JCogswell@Attacker:~/.../Lab3$
[09/26/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -d -in encf1.bin -out decf1.
txt -nopad -k 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
[09/26/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -d -in encf2.bin -out decf2.
txt -nopad -k 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
[09/26/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -d -in encf3.bin -out decf3.
txt -nopad -k 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f
[09/26/19]JCogswell@Attacker:~/.../Lab3$
[09/26/19]JCogswell@Attacker:~/.../Lab3$ xxd decf1.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345...........
[09/26/19]JCogswell@Attacker:~/.../Lab3$ xxd decf2.txt
00000000: 3132 3334 3536 3738 3930 0606 0606 0606  1234567890......
[09/26/19]JCogswell@Attacker:~/.../Lab3$ xxd decf3.txt
00000000: 3132 3334 3536 3738 3930 6162 6364 6566  1234567890abcdef
00000010: 1010 1010 1010 1010 1010 1010 1010 1010  ................
[09/26/19]JCogswell@Attacker:~/.../Lab3$
```

From what I found playing around with it a little bit, the CFB and OFB modes do not require padding. This means that the ciphertext is the same exact length as the plaintext. Some cipher modes require padding because the length of the plaintext must be an exact multiple of the specified block length. For instance, 3DES requires a block length of 8 bytes so the plaintext isn't exactly 8 bytes then padding must be added for the block length requirement to be met. The mode used is up to the user or requirements of the system they're using.

Task 5: Error Propagation – Corrupted Cipher Text

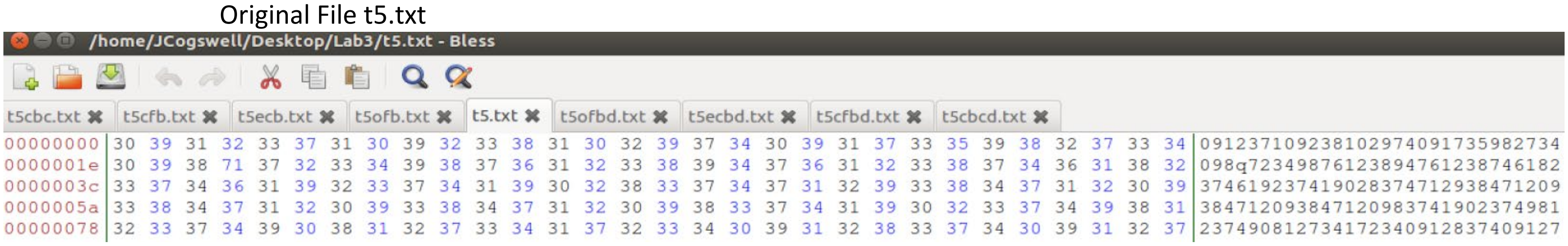
In this task, the purpose is to learn the error propagation property of each encryption mode. To answer the first question before beginning the task…

Expectations: I don't quite know exactly what to expect, but I would imagine that CFB and OFB will produce the same plaintext files as the original file since they don't need use padding. Since 1000 bytes is not evenly divisible by 16, the other modes will have different plaintext after decryption.

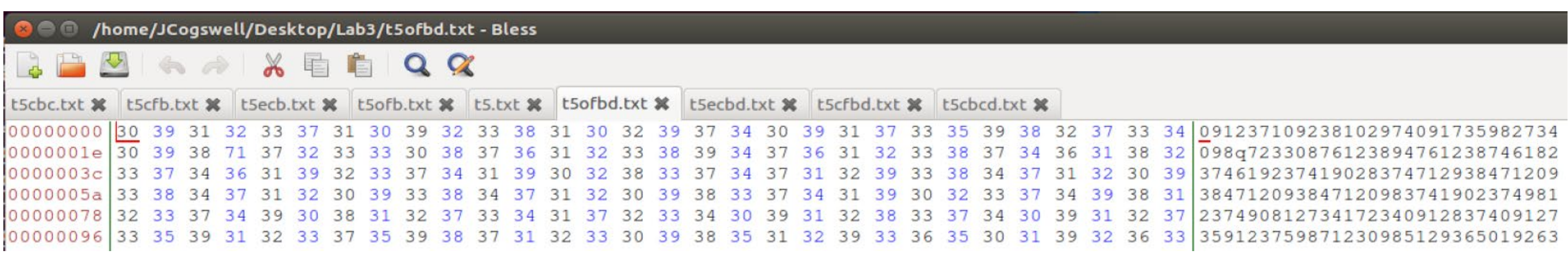
On to the task….

First, I created a text file that was 1000 bytes long with random numbers and letters. Next, I encrypted the text file with AES-128 encryption and CBC, CFB, OFB and ECB modes. I then used the Bless hex editor and changed the 55th byte of each encrypted file to FF, then decrypted each of those files and compared their decrypted plain text against the original unencrypted file.

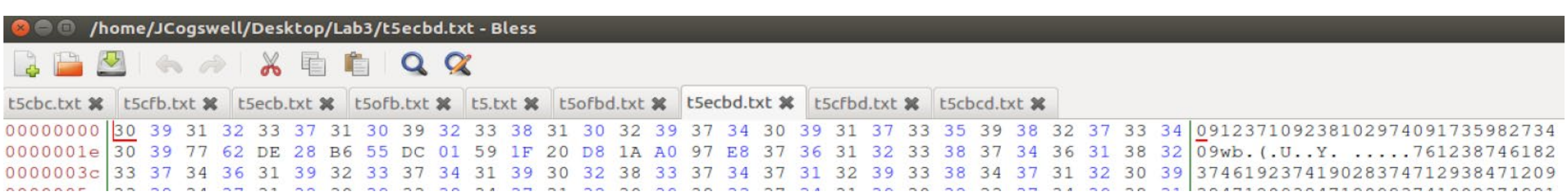
Original File t5.txt



OFB decrypted file. **No changes to decrypted file.**



ECB decrypted file. (**18 Bytes Changed**)



CFB decrypted file (**16 Bytes Changed**)

```
/home/JCogswell/Desktop/Lab3/t5cfbd.txt - Bless
t5cbc.txt  t5cfb.txt  t5ecb.txt  t5ofb.txt  t5.txt  t5ofbd.txt  t5ecbd.txt  t5cfbd.txt  t5cbcd.txt
00000000 30 39 31 32 33 37 31 30 39 32 33 38 31 30 32 39 37 34 30 39 31 37 33 35 39 38 32 37 33 34 091237109238102974091735982734
0000001e 30 39 38 71 37 32 33 51 33 38 37 36 31 32 33 38 39 34 B8 12 2D 3E 6B 83 6A E2 0D 38 82 91 098q723Q3876123894..->k.j..8..
0000003c 6D 64 07 B9 31 39 32 33 37 34 31 39 30 32 38 33 37 34 37 31 32 39 33 38 34 37 31 32 30 39 md..192374190283747129384712O9
0000005a 33 38 34 37 31 32 30 39 33 38 34 37 31 32 30 39 38 33 37 34 31 39 30 32 33 37 34 39 38 31 384712093847120983741902374981
```

CBC decrypted file (**18 Bytes Changed**)

```
/home/JCogswell/Desktop/Lab3/t5cbcd.txt - Bless
t5cbc.txt  t5cfb.txt  t5ecb.txt  t5ofb.txt  t5.txt  t5ofbd.txt  t5ecbd.txt  t5cfbd.txt  t5cbcd.txt
00000000 30 39 31 32 33 37 31 30 39 32 33 38 31 30 32 39 37 34 30 39 31 37 33 35 39 38 32 37 33 34 091237109238102974091735982734
0000001e 30 39 8F EE E3 9E 59 66 F8 D2 BD 89 C3 21 0C 0E 7D 01 37 36 31 32 33 E7 3B 34 36 31 38 32 09....Yf.....!..}.76123.;46182
0000003c 33 37 34 36 31 39 32 33 37 34 31 39 30 32 38 33 37 34 37 31 32 39 33 38 34 37 31 32 30 39 374619237419028374712938471209
0000005a 33 38 34 37 31 32 30 39 33 38 34 37 31 32 30 39 38 33 37 34 31 39 30 32 33 37 34 39 38 31 384712093847120983741902374981
```

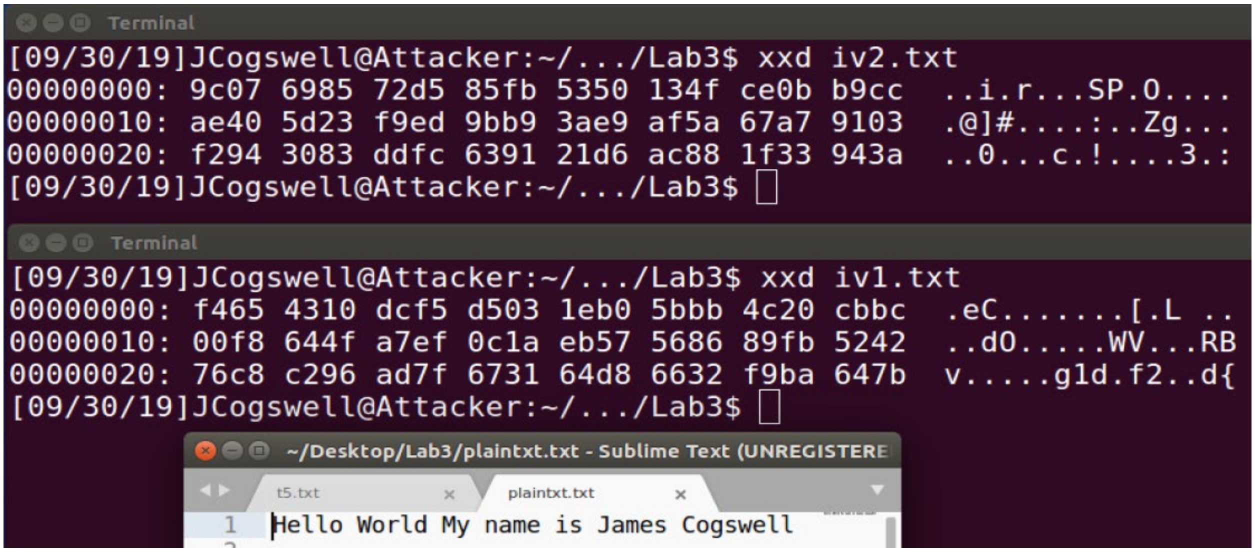
Observation: I was mostly correct in my initial thought. I said that only OFB and CFB will produce unchanged plaintext files after decrypting the corrupted encrypted files. OFB seems to be unchanged and CFB was changed by 16 bytes which is significant. The other two modes I was correct about. I'm not quite sure why the CFB changed by 16 bytes. It seems to me that if OFB was unchanged then CFB should have been unchanged as well.

__Task 6: Initial Vector (IV):__ The purpose of this task is to teach us the importance of selecting IVs carefully so that we can achieve a secure encryption since the security is not guaranteed simply by using a secure encryption algorithm and mode.

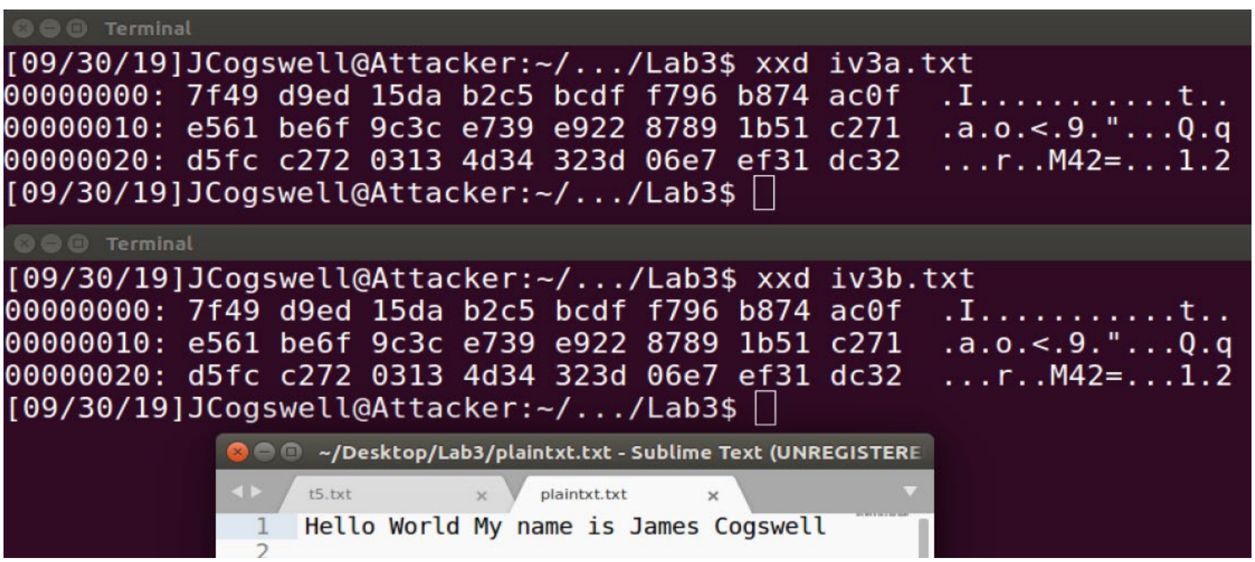
Task 6.1 – The purpose of this part is to show us the value and necessity of having a *unique* IV – meaning no IV can be used twice with the same key. I encrypted the same file (plaintxt.txt) four times – first I encrypted it with two different IVs, then I encrypted it using the same IV twice. The commands used are shown below.

```
Terminal
[09/30/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -e -in plaintxt.txt -out iv1
.txt -K 00112233445566778899aabbccddeeff -iv 0123456789abcdef
[09/30/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -e -in plaintxt.txt -out iv2
.txt -K 00112233445566778899aabbccddeeff -iv f0e98d76c54b32a10
[09/30/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -e -in plaintxt.txt -out iv3
a.txt -K 00112233445566778899aabbccddeeff -iv qwerty9876543210
non-hex digit
invalid hex iv value
[09/30/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -e -in plaintxt.txt -out iv3
a.txt -K 00112233445566778899aabbccddeeff -iv a01b23c45d67e89f0
[09/30/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -aes-128-cbc -e -in plaintxt.txt -out iv3
b.txt -K 00112233445566778899aabbccddeeff -iv a01b23c45d67e89f0
[09/30/19]JCogswell@Attacker:~/.../Lab3$
```

As seen in the screenshot below, iv1.txt and iv2.txt (the two that were encrypted using different IVs) have completely different contents.



This screenshot (below) shows the two files (iv3a.txt and iv3b.txt) that were encrypted using the same IV. Notice how the contents are identical. This is a major security issue.



Using unique IVs every time the same key is used is extremely important in protecting the security of that key. If they get the key, then cracking the files encrypted with that key becomes trivial by comparison. IVs need to be random for every key!

Task 6.2 – This task is designed to show something similar to what I mentioned at the end of the previous task: we can decrypt other files if we have access to the plaintext and ciphertext of one file and only the ciphertext of the other file if they're using the same IV. According to the provided lab guide, an attacker can XOR the plaintext and the ciphertext to obtain the encryption function. If the key and the IV are the same, then that function can be used to decrypt other files with the same IV and key combination.

P1 ⊕ C1 = O = P2 ⊕ C2

C2 ⊕ (P1 ⊕ C1) = P2

Using an ASCII text to Hex converter, I show the plaintext P1 and its equivalent in Hex. Then I calculate the output O using an XOR calculator.

P1 = This is a known message! = 546869732069732061206b6e6f776e206d65737361676521

C1 = a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159

O = P1 ⊕ C1 = f001d8b622a8b99907b6353e2d2356c1d67e2ce356c3a478

Now, XOR the output with C2, convert the Hex to ASCII and you have your text P2 decrypted.

C2 = bf73bcd3509299d566c35b5d450337e1bb175f903fafc159

P2 = C2 ⊕ O = 4f726465723a204c61756e63682061206d697373696c6521

(Hex to ASCII Conversion of P2) = Order: Launch a missile!

If we were to use CFB instead of OFB in this problem, we would only be able to decrypt the plaintext of the first block. The reason for this is because CFB uses the ciphertext it produces after encrypting the first block to encrypt the next block and so on. Therefore, the ciphertext that we would be able to decrypt using this method would only be the plaintext of the first block.

Task 6.3 – This part of task 6 demonstrates the importance of unpredictability of IVs, or randomly generated IVs. We will see what happens if the IV is predictable in this task. The scenario here is that Bob sent an encrypted message to Eve using a key that is only known to Bob and the C1 as well as the IV used on P1 are known to both (IV for P1 ends in 536). Lastly, Eve knows that she can *predict* the next IV (ending in 537). Eve cannot decrypt the message but knows that the message contains either the words 'Yes' or 'No'. This is how Eve can see the contents of P1:

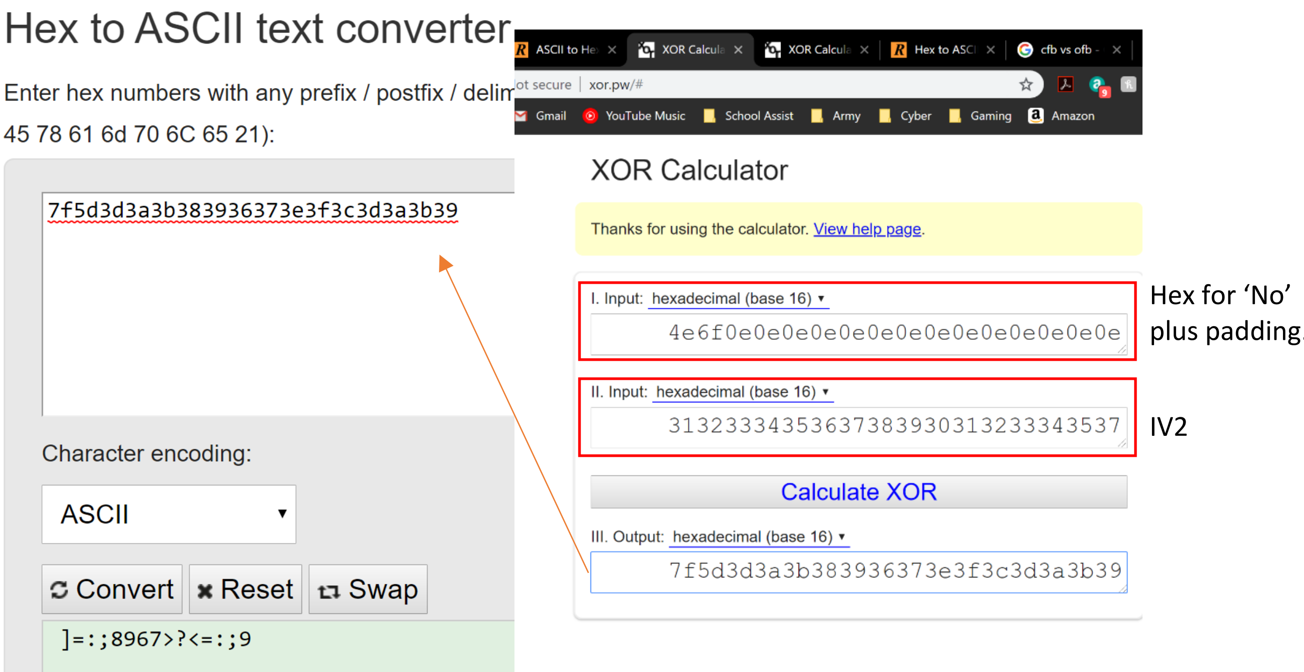
1: Use an ASCII to Hex converter to get the hex values of the words 'Yes' and 'No'

2: Use an XOR calculator to XOR IV2 (ends in 537) with the hex values of 'Yes' and 'No'. Make sure to add padding of size 13B to hex 'Yes' and of size 14B to hex 'No' for the XOR to work properly.

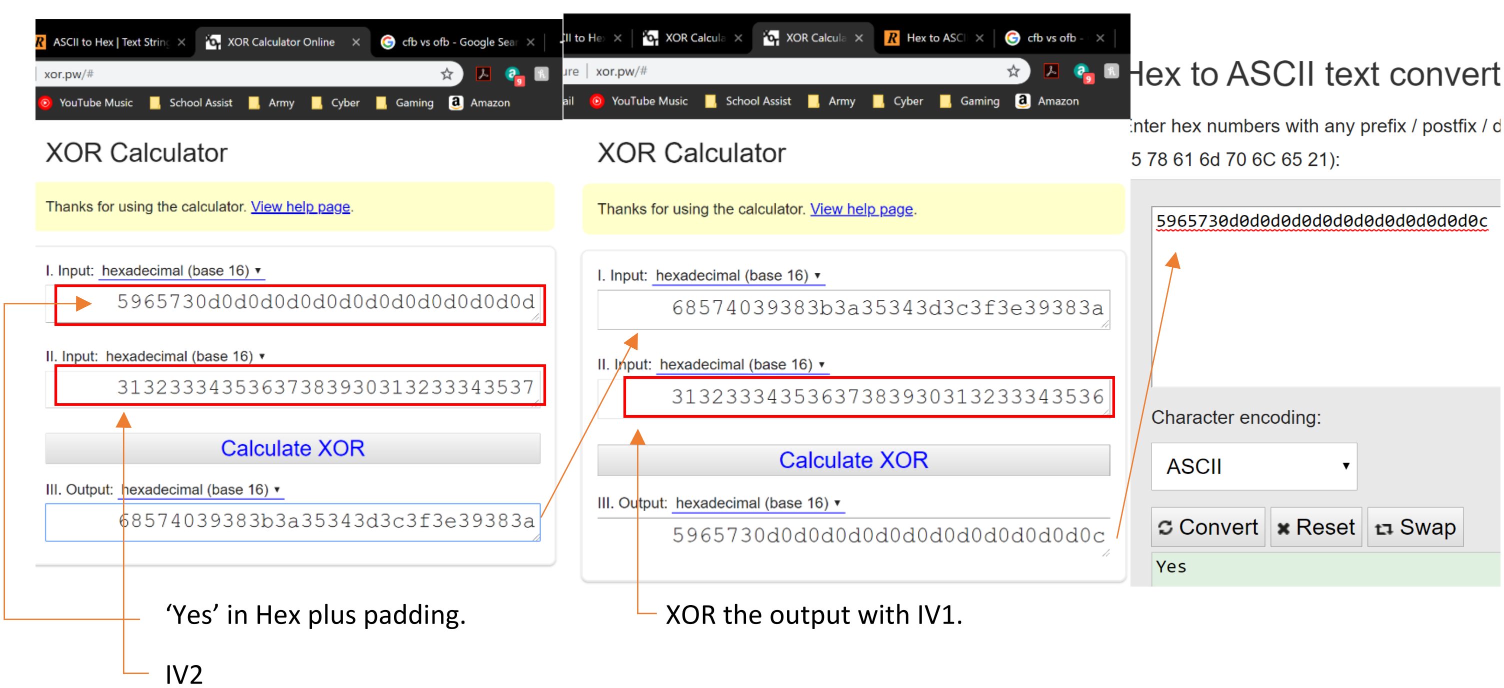
3: Use the output and XOR it with IV2 (ends in 536). This produces an output which will be the same as P1 if you guessed the correct message content.

I showed both 'Yes' and 'No'. See the screenshots below:

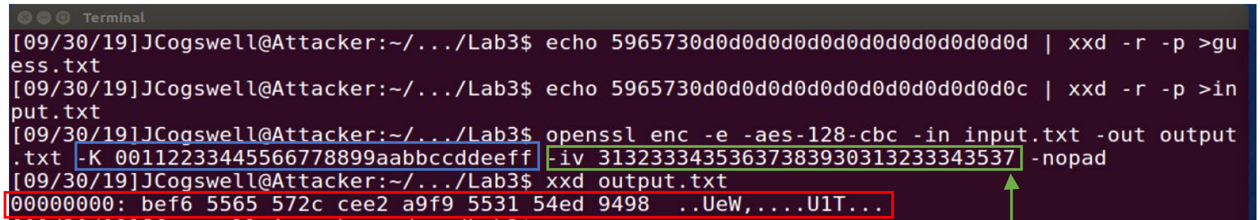
Using 'No' as my first guess… Converting our output to ASCII does not produce the word 'No'.



Using 'Yes' as my first guess… Converting our output to ASCII *does* produce the word 'Yes'.



The above screenshots prove that the original message was ‘Yes’ and explains how we can decrypt any ciphertext that continues this pattern. Below I encrypted the message using the key (we’re acting like Bob encrypted it since we wouldn’t know the key, technically) and IV2 and showed that the newly encrypted message is identical to the message shown in the lab instructions.



```
Terminal
[09/30/19]JCogswell@Attacker:~/.../Lab3$ echo 5965730d0d0d0d0d0d0d0d0d0d0d0d0d | xxd -r -p >guess.txt
[09/30/19]JCogswell@Attacker:~/.../Lab3$ echo 5965730d0d0d0d0d0d0d0d0d0d0d0d0c | xxd -r -p >input.txt
[09/30/19]JCogswell@Attacker:~/.../Lab3$ openssl enc -e -aes-128-cbc -in input.txt -out output.txt -K 00112233445566778899aabbccddeeff -iv 31323334353637383930313233343537 -nopad
[09/30/19]JCogswell@Attacker:~/.../Lab3$ xxd output.txt
00000000: bef6 5565 572c cee2 a9f9 5531 54ed 9498  ..UeW,....U1T...
```

IV2

```
Encryption method: 128-bit AES with CBC mode.

Key (in hex):     00112233445566778899aabbccddeeff (known only to Bob)
Ciphertext (C1): bef65565572ccee2a9f9553154ed9498 (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
```