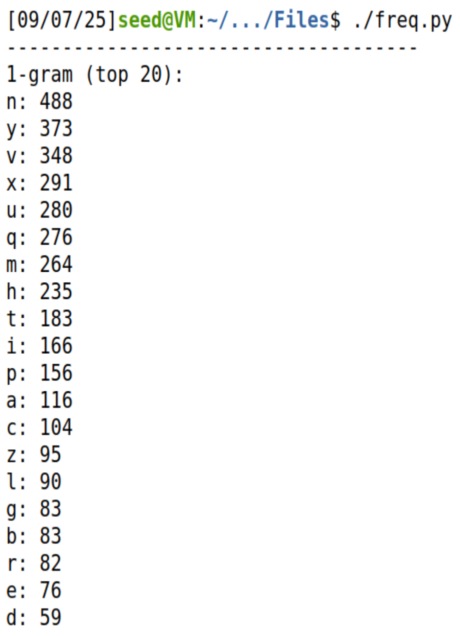
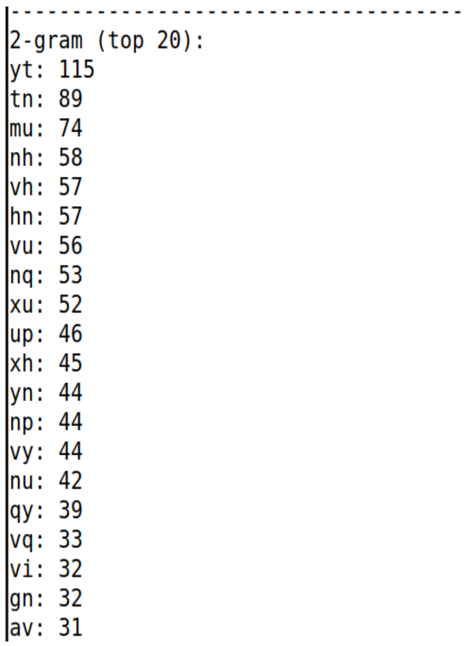
Timothy Yip

Secret-Key Encryption Lab

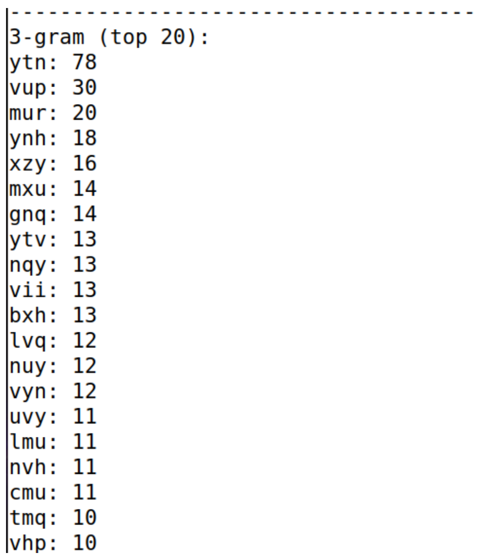
Task 1: Frequency Analysis



```
[09/07/25]seed@VM:~/.../Files$ ./freq.py
-----------------------------------
1-gram (top 20):
n: 488
y: 373
v: 348
x: 291
u: 280
q: 276
m: 264
h: 235
t: 183
i: 166
p: 156
a: 116
c: 104
z: 95
l: 90
g: 83
b: 83
r: 82
e: 76
d: 59
```



```
-----------------------------------
2-gram (top 20):
yt: 115
tn: 89
mu: 74
nh: 58
vh: 57
hn: 57
vu: 56
nq: 53
xu: 52
up: 46
xh: 45
yn: 44
np: 44
vy: 44
nu: 42
qy: 39
vq: 33
vi: 32
gn: 32
av: 31
```



```
-----------------------------------
3-gram (top 20):
ytn: 78
vup: 30
mur: 20
ynh: 18
xzy: 16
mxu: 14
gnq: 14
ytv: 13
nqy: 13
vii: 13
bxh: 13
lvq: 12
nuy: 12
vyn: 12
uvy: 11
lmu: 11
nvh: 11
cmu: 11
tmq: 10
vhp: 10
```

Analysis:
The most common trigram from ./freq.py/ was ytn, which appeared 78 times. The most common English trigram is “the” so we can map:

“y” → “t”
“t” → “h”
“n” → “e”

We can cross check this with “yt” (115 times) and “tn” (89 times) to the 2 most common English bigrams “th” and “he”.
Next, the second most common trigram from ./freq.py/ is vup, which appeared 30 times. The second most common English trigram is “and” so we can map:

“v” → “a”
“u” → “n”
“p” → “d”

We can cross check this with “vu” (56 times) and “up” (46 times) to another 2 English bigrams “an” and “nd”.

We can continue with mapping “m” → “i” as the bigram “mu” appeared 74 times and as we mapped previously, “u” → “n”, we can create another common English bigram in “in”.

Following that, we can map “x” → “o” as the bigram “xu” appeared 52 times and as we mapped previously, “u” → “n”, we can create another common English bigram in “on”.

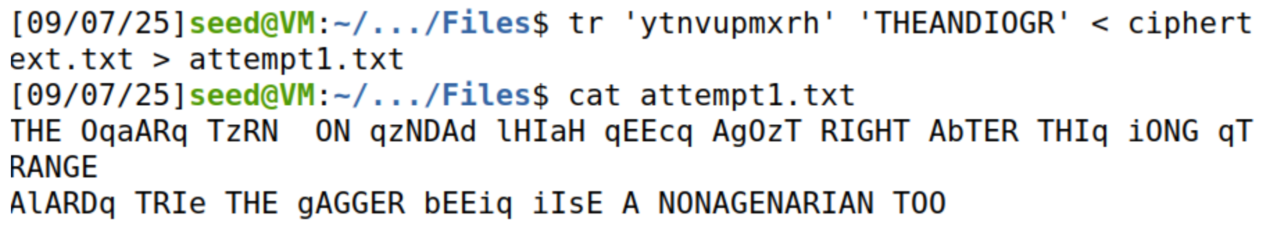
Using previous mappings, we can conclude that “r” → “g” as the trigram “mur” appeared 20 times. With “m” → “i” and “u” → “n”, we have “ing”, the most common three-letter ending in English.

Similarly, we can decisively say that “h” → “r” as the bigram “nh” appeared 58 times and “hn” appeared 57 times. With the previous mapping of “n” → “e”, we can see that we form 2 very common letter combinations in English with “er” and “re”.

With all these maps done, we can find a partial message using the command
tr 'ytnvupmxrh' 'THEANDIOGR' < ciphertext.txt > attempt1.txt

Attempt 2:

Now that we did the first attempt, we can try to find misspelled words that would fit in.



```
[09/07/25]seed@VM:~/.../Files$ tr 'ytnvupmxrh' 'THEANDIOGR' < ciphert
ext.txt > attempt1.txt
[09/07/25]seed@VM:~/.../Files$ cat attempt1.txt
THE OqaARq TzRN  ON qzNDAd lHIaH qEEcq AgOzT RIGHT AbTER THIq iONG qT
RANGE
AlARDq TRIe THE gAGGER bEEiq iIsE A NONAGENARIAN TOO
```

From this we can see that in the word “tzrn” can be “turn”, “abter” can be “after”, “thiq” can be “this”, and “qtrange” can be “strange”.

From this we have mapped:

Timothy Yip

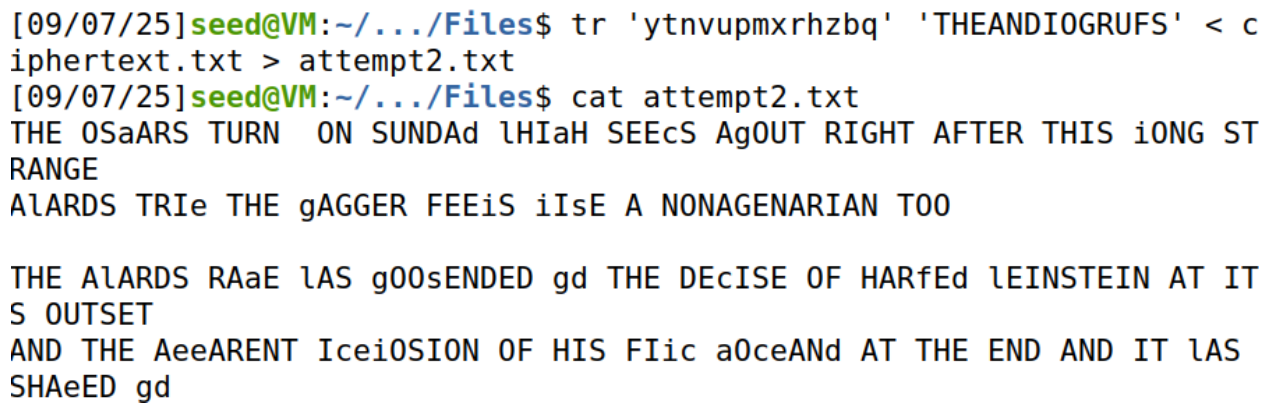
“z” → “u”

“b” → ‘f”

“q” → “s”

Now we can use the command:

tr 'ytnvupmxrhzbq' 'THEANDIOGRUFS' < ciphertext.txt > attempt2.txt



```
[09/07/25]seed@VM:~/.../Files$ tr 'ytnvupmxrhzbq' 'THEANDIOGRUFS' < c
iphertext.txt > attempt2.txt
[09/07/25]seed@VM:~/.../Files$ cat attempt2.txt
THE OSaARS TURN  ON SUNDAd lHIaH SEEcS AgOUT RIGHT AFTER THIS iONG ST
RANGE
AlARDS TRIe THE gAGGER FEEiS iIsE A NONAGENARIAN TOO

THE AlARDS RAaE lAS gOOsENDED gd THE DEcISE OF HARfEd lEINSTEIN AT IT
S OUTSET
AND THE AeeARENT IceiOSION OF HIS FIic aOceANd AT THE END AND IT lAS
SHAeED gd
```

Attempt 3:

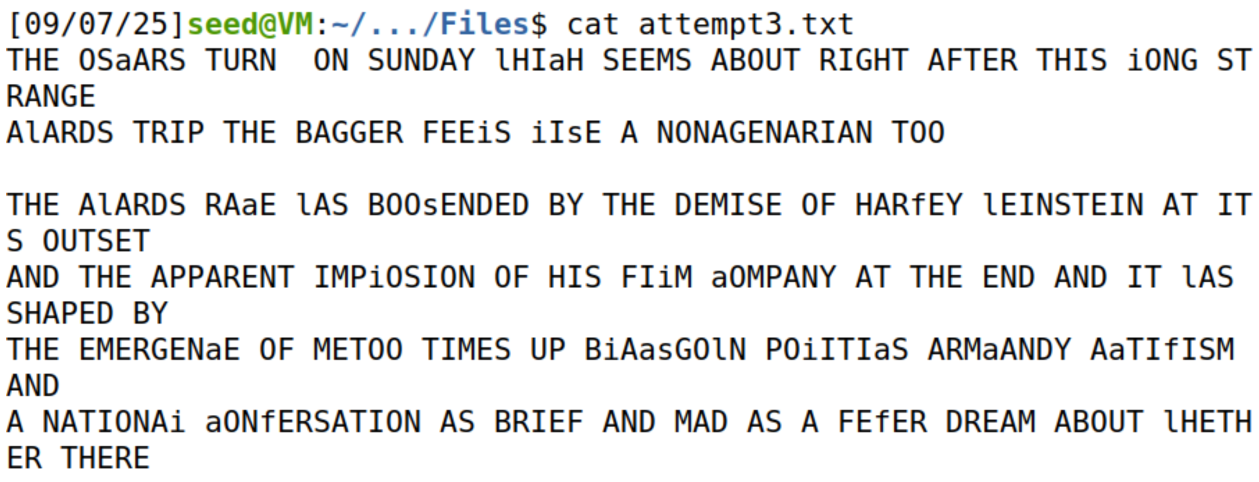
Similarly from attempt 2, we can find misspelled words, for example:

“sundad” → “sunday”	“d” → “y”
“seecs” → “seems”	“c” → “m”
“agout” → “about”	“g” → “b”
“aeearent” → “apparent”	“e” → “p”

Now we can use the command:

tr 'ytnvupmxrhzbqdcge' 'THEANDIOGRUFSYMBP' < ciphertext.txt > attempt3.txt

Timothy Yip



```
[09/07/25]seed@VM:~/.../Files$ cat attempt3.txt
THE OSaARS TURN  ON SUNDAY lHIaH SEEMS ABOUT RIGHT AFTER THIS iONG ST
RANGE
AlARDS TRIP THE BAGGER FEEiS iIsE A NONAGENARIAN TOO

THE AlARDS RAaE lAS BOOsENDED BY THE DEMISE OF HARfEY lEINSTEIN AT IT
S OUTSET
AND THE APPARENT IMPiOSION OF HIS FIiM aOMPANY AT THE END AND IT lAS
SHAPED BY
THE EMERGENaE OF METOO TIMES UP BiAasGOlN POiITIaS ARMaANDY AaTIfISM
AND
A NATIONAi aONfERSATION AS BRIEF AND MAD AS A FEfER DREAM ABOUT lHETH
ER THERE
```

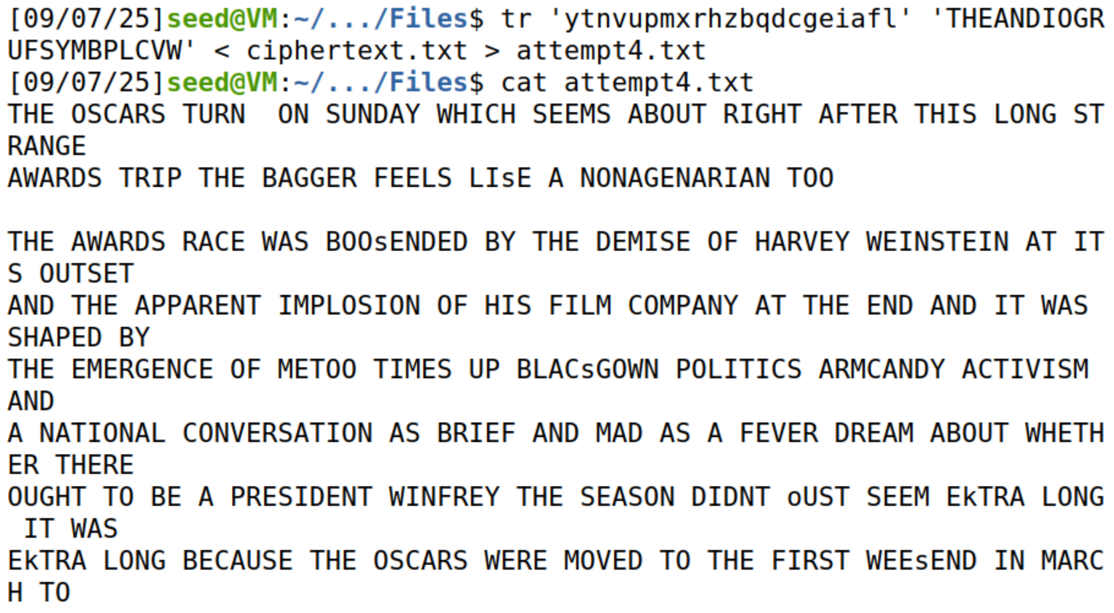
Attempt 4:

Similarly from attempt 2, we can find misspelled words, for example:

“iong”, “feeis”, “impiosion”, “fiim”	“i” → “l”
“aompany”, “emergenae”	“a” → “c”
“aatifism” (assuming from previous “a” → “c”)	“f” → “v”
“lhether”	“l” → “w”

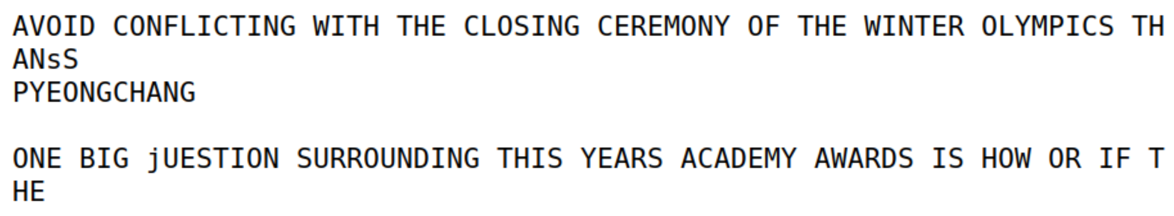
Now we can use the command:

tr 'ytnvupmxrhzbqdcgeiafl' 'THEANDIOGRUFSYMBPLCVW' < ciphertext.txt > attempt4.txt



```
[09/07/25]seed@VM:~/.../Files$ tr 'ytnvupmxrhzbqdcgeiafl' 'THEANDIOGR
UFSYMBPLCVW' < ciphertext.txt > attempt4.txt
[09/07/25]seed@VM:~/.../Files$ cat attempt4.txt
THE OSCARS TURN  ON SUNDAY WHICH SEEMS ABOUT RIGHT AFTER THIS LONG ST
RANGE
AWARDS TRIP THE BAGGER FEELS LIsE A NONAGENARIAN TOO

THE AWARDS RACE WAS BOOsENDED BY THE DEMISE OF HARVEY WEINSTEIN AT IT
S OUTSET
AND THE APPARENT IMPLOSION OF HIS FILM COMPANY AT THE END AND IT WAS
SHAPED BY
THE EMERGENCE OF METOO TIMES UP BLACsGOWN POLITICS ARMCANDY ACTIVISM
AND
A NATIONAL CONVERSATION AS BRIEF AND MAD AS A FEVER DREAM ABOUT WHETH
ER THERE
OUGHT TO BE A PRESIDENT WINFREY THE SEASON DIDNT oUST SEEM EkTRA LONG
 IT WAS
EkTRA LONG BECAUSE THE OSCARS WERE MOVED TO THE FIRST WEEsEND IN MARC
H TO
```

Timothy Yip



```
AVOID CONFLICTING WITH THE CLOSING CEREMONY OF THE WINTER OLYMPICS TH
ANsS
PYEONGCHANG

ONE BIG jUESTION SURROUNDING THIS YEARS ACADEMY AWARDS IS HOW OR IF T
HE
```

FINAL ATTEMPT (DECIPHERED):

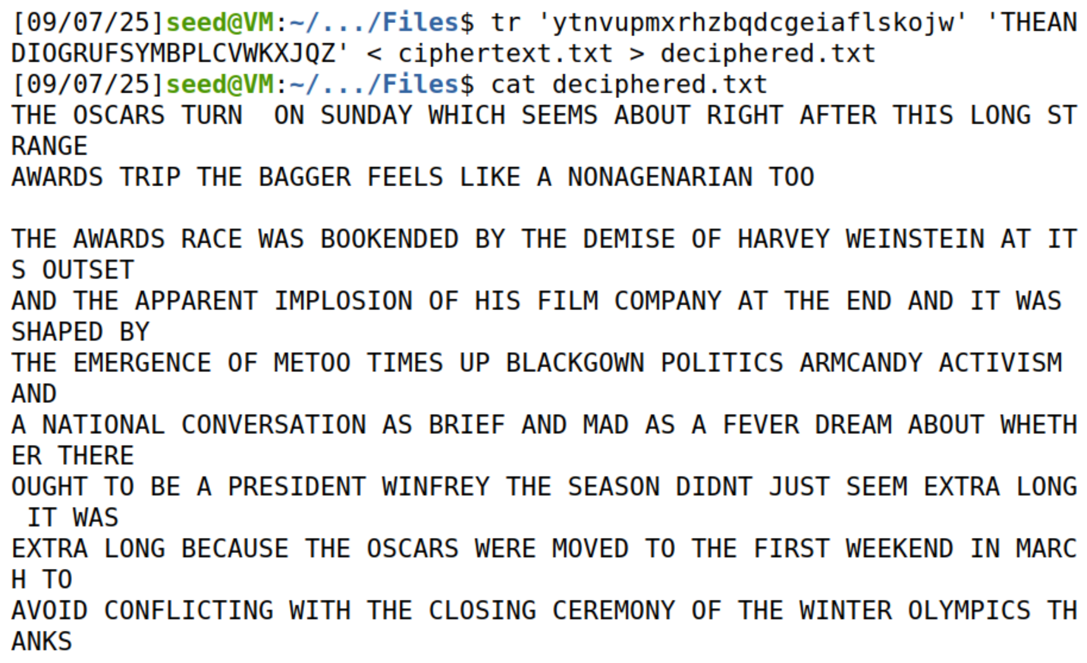
Finally, we have something that can be readable and we deduce the final mappings of the ciphertext. For example:

"lise" → "like"	"s" → "k"
"ektra" → "extra"	"k" → "x"
"oust" → "just"	"o" → "j"
"juestion" → "question"	"j" → "q"

And the last letter missing is from both to finalize the mappings: "w" → 'z"

Now we can use the command:

tr 'ytnvupmxrhzbqdcgeiaflskojw' 'THEANDIOGRUFSYMBPLCVWKXJQZ' < ciphertext.txt > deciphered.txt



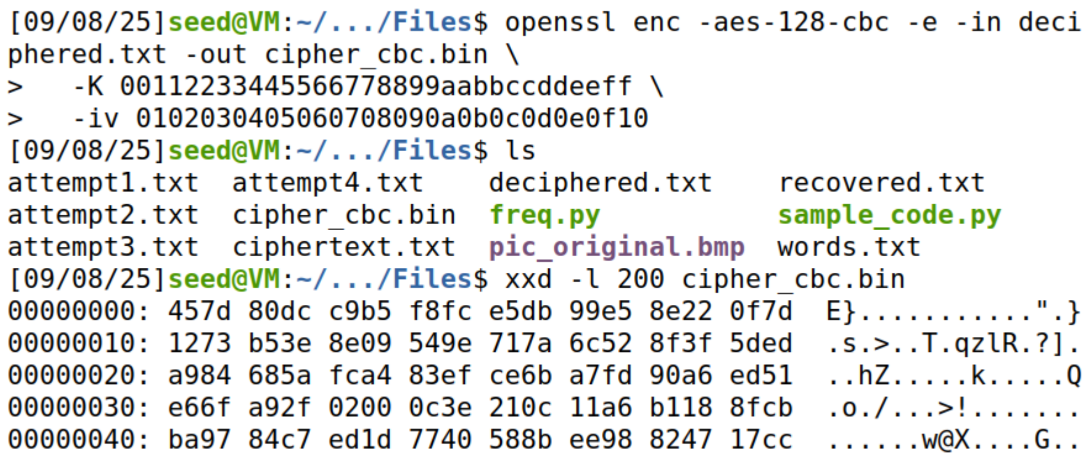
```
[09/07/25]seed@VM:~/.../Files$ tr 'ytnvupmxrhzbqdcgeiaflskojw' 'THEAN
DIOGRUFSYMBPLCVWKXJQZ' < ciphertext.txt > deciphered.txt
[09/07/25]seed@VM:~/.../Files$ cat deciphered.txt
THE OSCARS TURN  ON SUNDAY WHICH SEEMS ABOUT RIGHT AFTER THIS LONG ST
RANGE
AWARDS TRIP THE BAGGER FEELS LIKE A NONAGENARIAN TOO

THE AWARDS RACE WAS BOOKENDED BY THE DEMISE OF HARVEY WEINSTEIN AT IT
S 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 WHETH
ER 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 MARC
H TO
AVOID CONFLICTING WITH THE CLOSING CEREMONY OF THE WINTER OLYMPICS TH
ANKS
```

This has given us the fully deciphered text.

Timothy Yip

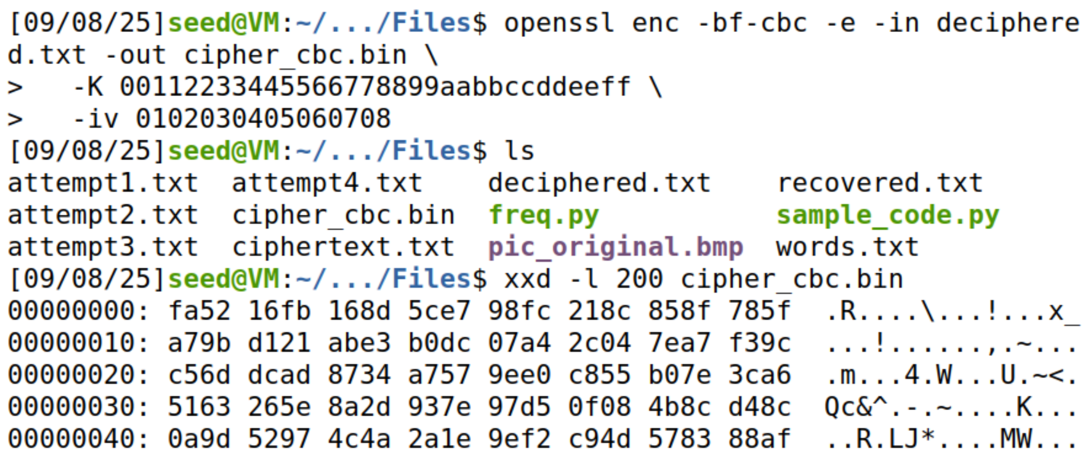
Task 2: Encryption using Different Ciphers and Modes

aes-128-cbc



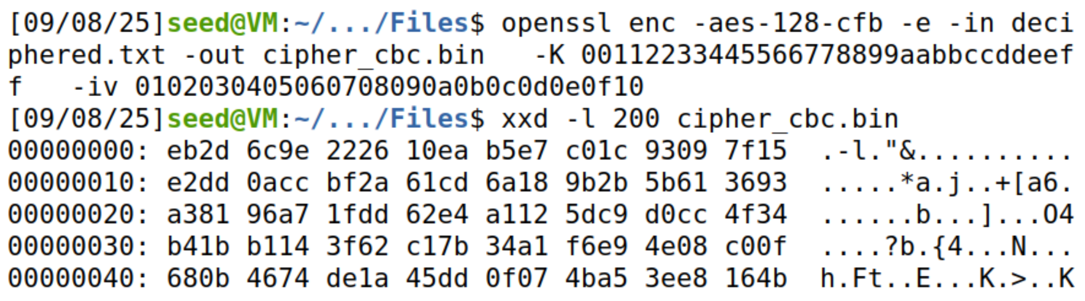
```
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in deci
phered.txt -out cipher_cbc.bin \
>   -K 00112233445566778899aabbccddeeff \
>   -iv 0102030405060708090a0b0c0d0e0f10
[09/08/25]seed@VM:~/.../Files$ ls
attempt1.txt  attempt4.txt    deciphered.txt    recovered.txt
attempt2.txt  cipher_cbc.bin  freq.py           sample_code.py
attempt3.txt  ciphertext.txt  pic_original.bmp  words.txt
[09/08/25]seed@VM:~/.../Files$ xxd -l 200 cipher_cbc.bin
00000000: 457d 80dc c9b5 f8fc e5db 99e5 8e22 0f7d  E}.........".}
00000010: 1273 b53e 8e09 549e 717a 6c52 8f3f 5ded  .s.>..T.qzlR.?].
00000020: a984 685a fca4 83ef ce6b a7fd 90a6 ed51  ..hZ.....k.....Q
00000030: e66f a92f 0200 0c3e 210c 11a6 b118 8fcb  .o./...>!.......
00000040: ba97 84c7 ed1d 7740 588b ee98 8247 17cc  ......w@X....G..
```

bf-cbc



```
[09/08/25]seed@VM:~/.../Files$ openssl enc -bf-cbc -e -in deciphere
d.txt -out cipher_cbc.bin \
>   -K 00112233445566778899aabbccddeeff \
>   -iv 0102030405060708
[09/08/25]seed@VM:~/.../Files$ ls
attempt1.txt  attempt4.txt    deciphered.txt    recovered.txt
attempt2.txt  cipher_cbc.bin  freq.py           sample_code.py
attempt3.txt  ciphertext.txt  pic_original.bmp  words.txt
[09/08/25]seed@VM:~/.../Files$ xxd -l 200 cipher_cbc.bin
00000000: fa52 16fb 168d 5ce7 98fc 218c 858f 785f  .R....\...!...x_
00000010: a79b d121 abe3 b0dc 07a4 2c04 7ea7 f39c  ...!......,.~...
00000020: c56d dcad 8734 a757 9ee0 c855 b07e 3ca6  .m...4.W...U.~<.
00000030: 5163 265e 8a2d 937e 97d5 0f08 4b8c d48c  Qc&^.-.~....K...
00000040: 0a9d 5297 4c4a 2a1e 9ef2 c94d 5783 88af  ..R.LJ*....MW...
```

aes-128-cfb



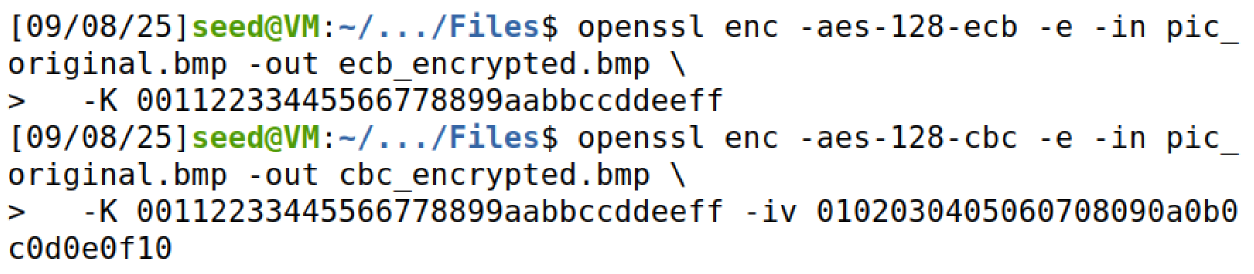
```
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in deci
phered.txt -out cipher_cbc.bin   -K 00112233445566778899aabbccddeef
f   -iv 0102030405060708090a0b0c0d0e0f10
[09/08/25]seed@VM:~/.../Files$ xxd -l 200 cipher_cbc.bin
00000000: eb2d 6c9e 2226 10ea b5e7 c01c 9309 7f15  .-l."&..........
00000010: e2dd 0acc bf2a 61cd 6a18 9b2b 5b61 3693  .....*a.j..+[a6.
00000020: a381 96a7 1fdd 62e4 a112 5dc9 d0cc 4f34  ......b...]...O4
00000030: b41b b114 3f62 c17b 34a1 f6e9 4e08 c00f  ....?b.{4...N...
00000040: 680b 4674 de1a 45dd 0f07 4ba5 3ee8 164b  h.Ft..E...K.>..K
```

From this we can see that each of the different cipher types resulted different files, as we can see from the first 40 bytes of each encrypted file

Timothy Yip

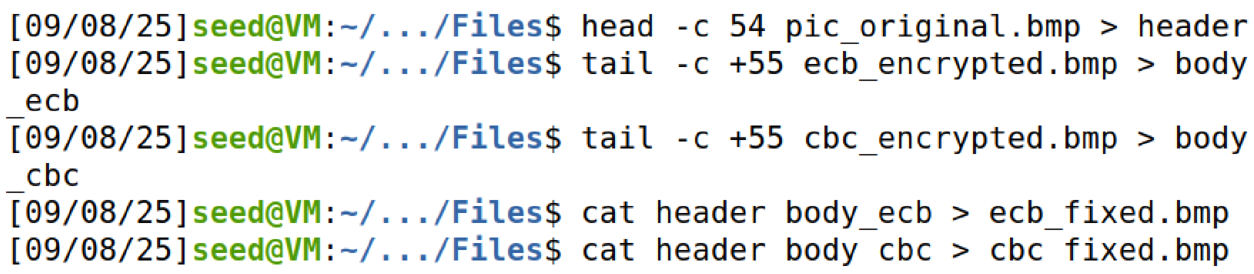
Task 3: Encryption Mode - ECB vs. CBC

Encrypt file using both encryption modes



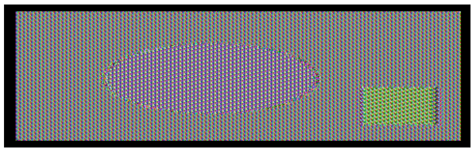
```
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in pic_
original.bmp -out ecb_encrypted.bmp \
>   -K 00112233445566778899aabbccddeeff
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in pic_
original.bmp -out cbc_encrypted.bmp \
>   -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0
c0d0e0f10
```

Fix the header of the encrypted files

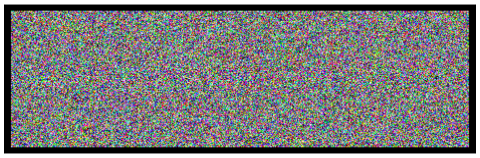


```
[09/08/25]seed@VM:~/.../Files$ head -c 54 pic_original.bmp > header
[09/08/25]seed@VM:~/.../Files$ tail -c +55 ecb_encrypted.bmp > body
_ecb
[09/08/25]seed@VM:~/.../Files$ tail -c +55 cbc_encrypted.bmp > body
_cbc
[09/08/25]seed@VM:~/.../Files$ cat header body_ecb > ecb_fixed.bmp
[09/08/25]seed@VM:~/.../Files$ cat header body_cbc > cbc_fixed.bmp
```

ecb_fixed.bmp



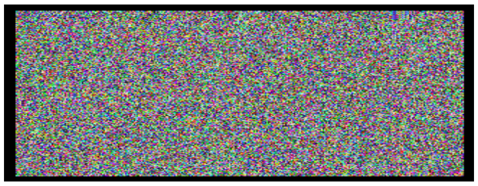
cbc_fixed.bmp



From this we can see that the ECB image will still be recognizable. The overall shapes, outlines, and structure of the original are shown. On the other hand, the CBC image looks like total noise, no structure is visible.

My Own Example:

ecb_fixed.bmp



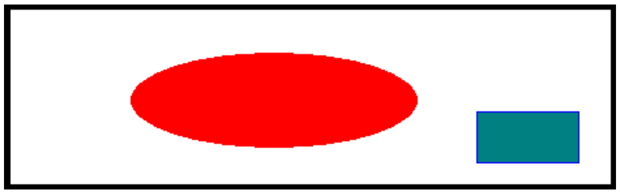
cbc_fixed.bmp

In my own example, both ended with total noise. I believe this is the case because my personal image was much more complex in terms of color and shapes

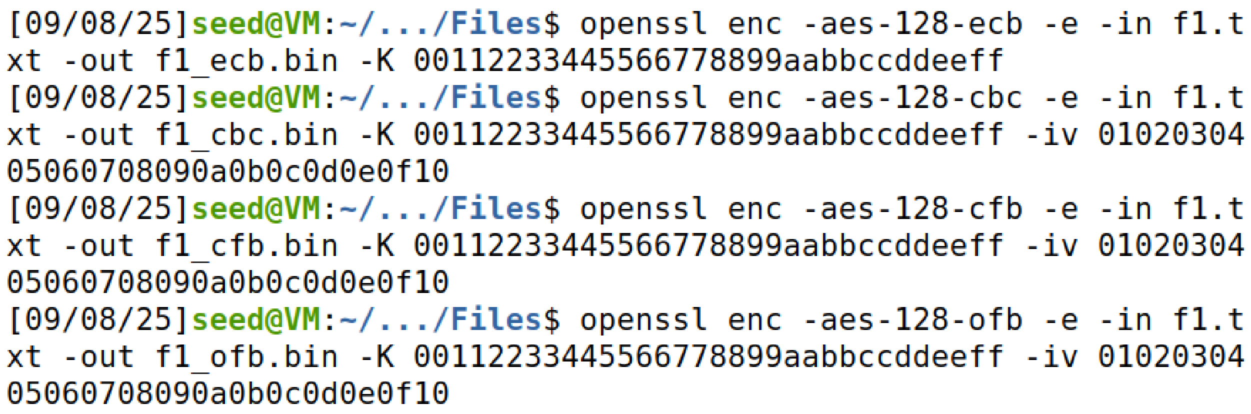
Timothy Yip

my picture

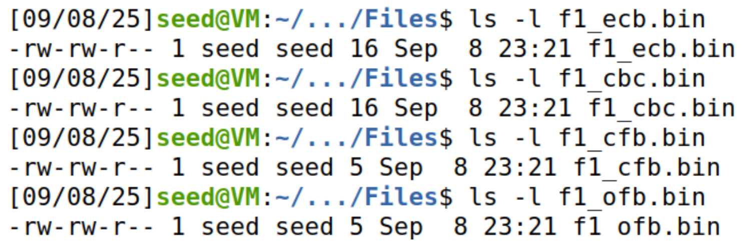
provided picture



Task 4: Padding



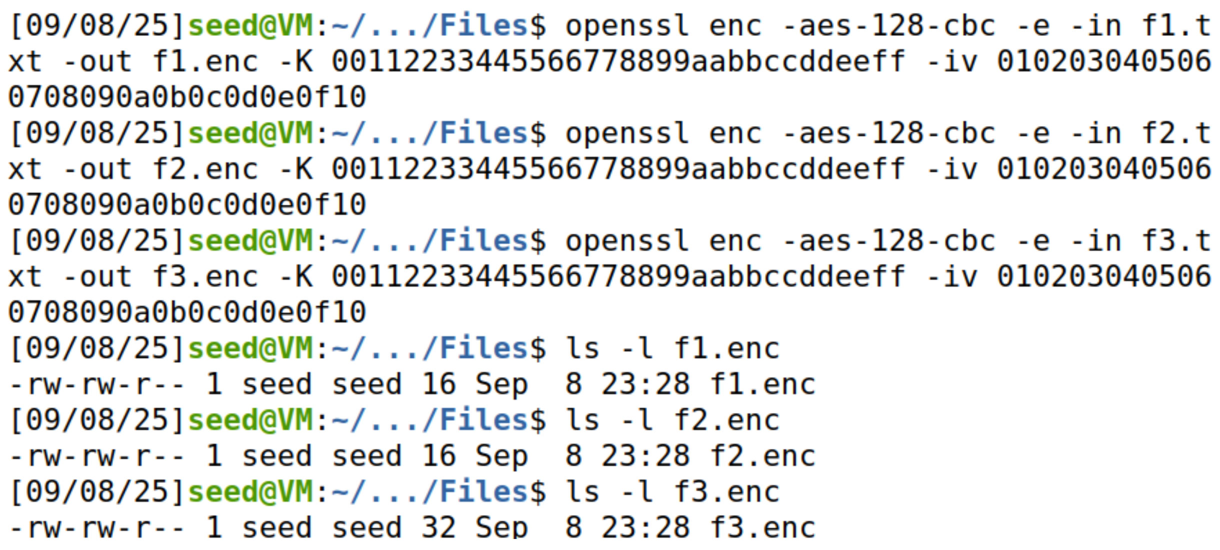
```
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in f1.t
xt -out f1_ecb.bin -K 00112233445566778899aabbccddeeff
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f1.t
xt -out f1_cbc.bin -K 00112233445566778899aabbccddeeff -iv 01020304
05060708090a0b0c0d0e0f10
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in f1.t
xt -out f1_cfb.bin -K 00112233445566778899aabbccddeeff -iv 01020304
05060708090a0b0c0d0e0f10
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in f1.t
xt -out f1_ofb.bin -K 00112233445566778899aabbccddeeff -iv 01020304
05060708090a0b0c0d0e0f10
```



```
[09/08/25]seed@VM:~/.../Files$ ls -l f1_ecb.bin
-rw-rw-r-- 1 seed seed 16 Sep  8 23:21 f1_ecb.bin
[09/08/25]seed@VM:~/.../Files$ ls -l f1_cbc.bin
-rw-rw-r-- 1 seed seed 16 Sep  8 23:21 f1_cbc.bin
[09/08/25]seed@VM:~/.../Files$ ls -l f1_cfb.bin
-rw-rw-r-- 1 seed seed 5 Sep  8 23:21 f1_cfb.bin
[09/08/25]seed@VM:~/.../Files$ ls -l f1_ofb.bin
-rw-rw-r-- 1 seed seed 5 Sep  8 23:21 f1_ofb.bin
```

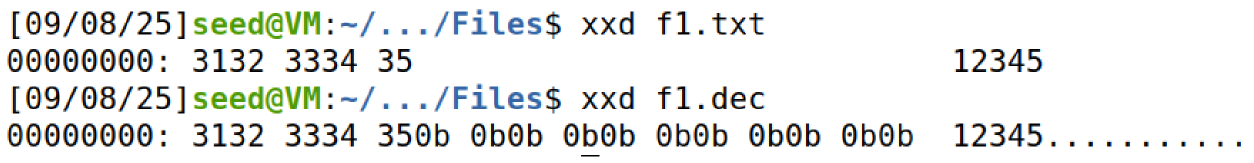
Since both ECB and CBC have outputs that are multiples of 16, we need to add 11 (16-5) bytes of padding. On the other hand, CFB and OFB are stream-like (output length = input length), so no padding is needed.

Timothy Yip

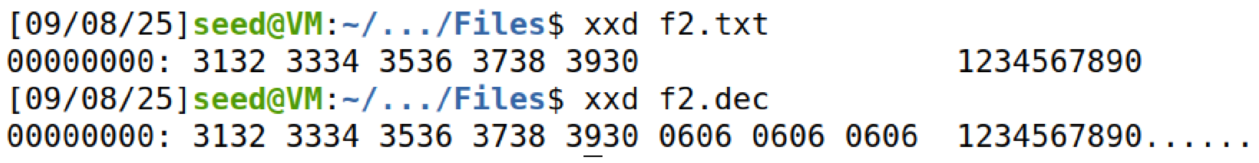


```
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f1.t
xt -out f1.enc -K 00112233445566778899aabbccddeeff -iv 010203040506
0708090a0b0c0d0e0f10
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f2.t
xt -out f2.enc -K 00112233445566778899aabbccddeeff -iv 010203040506
0708090a0b0c0d0e0f10
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f3.t
xt -out f3.enc -K 00112233445566778899aabbccddeeff -iv 010203040506
0708090a0b0c0d0e0f10
[09/08/25]seed@VM:~/.../Files$ ls -l f1.enc
-rw-rw-r-- 1 seed seed 16 Sep  8 23:28 f1.enc
[09/08/25]seed@VM:~/.../Files$ ls -l f2.enc
-rw-rw-r-- 1 seed seed 16 Sep  8 23:28 f2.enc
[09/08/25]seed@VM:~/.../Files$ ls -l f3.enc
-rw-rw-r-- 1 seed seed 32 Sep  8 23:28 f3.enc
```

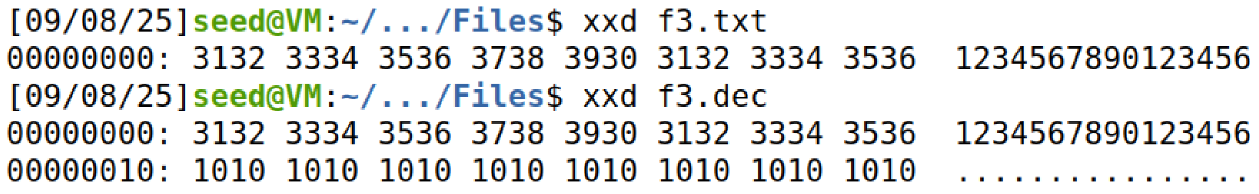
From this we can see that the files with 5 and 10 bytes are both padded to 16 bytes (a whole block) while the 16 byte file was padded up to 32 bytes even though it is the same size as a whole block.



```
[09/08/25]seed@VM:~/.../Files$ xxd f1.txt
00000000: 3132 3334 35                             12345
[09/08/25]seed@VM:~/.../Files$ xxd f1.dec
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345...........
```



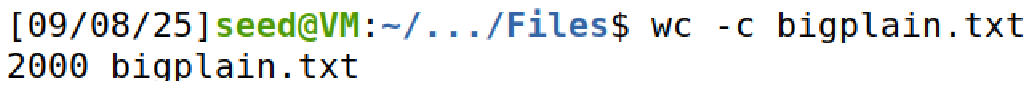
```
[09/08/25]seed@VM:~/.../Files$ xxd f2.txt
00000000: 3132 3334 3536 3738 3930                 1234567890
[09/08/25]seed@VM:~/.../Files$ xxd f2.dec
00000000: 3132 3334 3536 3738 3930 0606 0606 0606  1234567890......
```



```
[09/08/25]seed@VM:~/.../Files$ xxd f3.txt
00000000: 3132 3334 3536 3738 3930 3132 3334 3536  1234567890123456
[09/08/25]seed@VM:~/.../Files$ xxd f3.dec
00000000: 3132 3334 3536 3738 3930 3132 3334 3536  1234567890123456
00000010: 1010 1010 1010 1010 1010 1010 1010 1010  ................
```

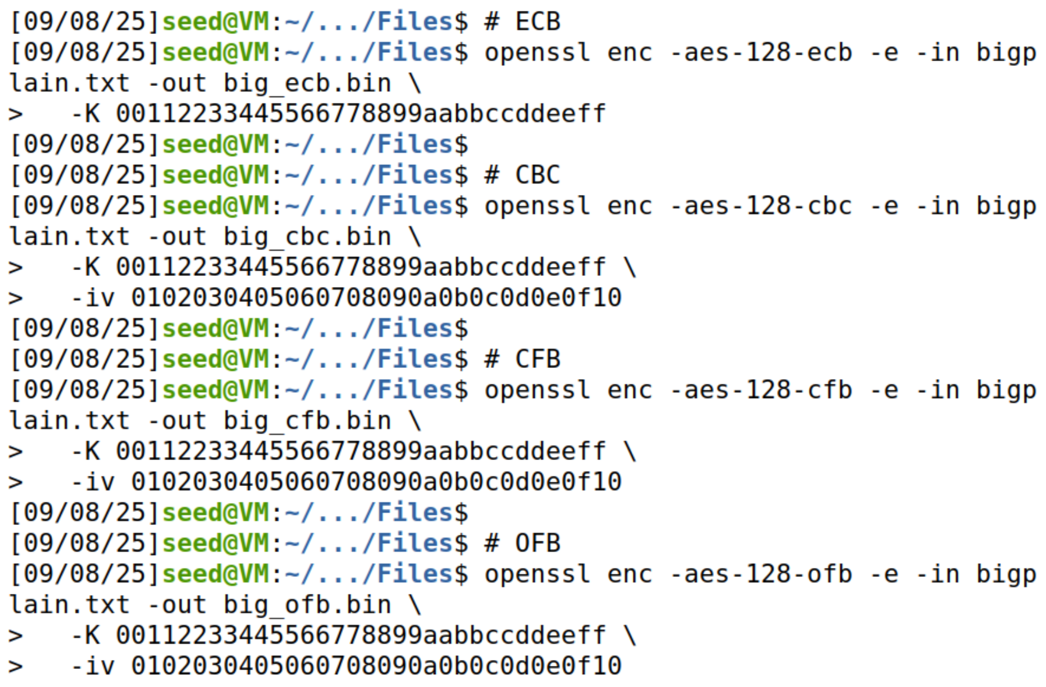
From each one of these screenshots, we can clearly see what the difference is, which is what is padded by the encryption.

TASK 5: Error Propagation – Corrupted Cipher Text



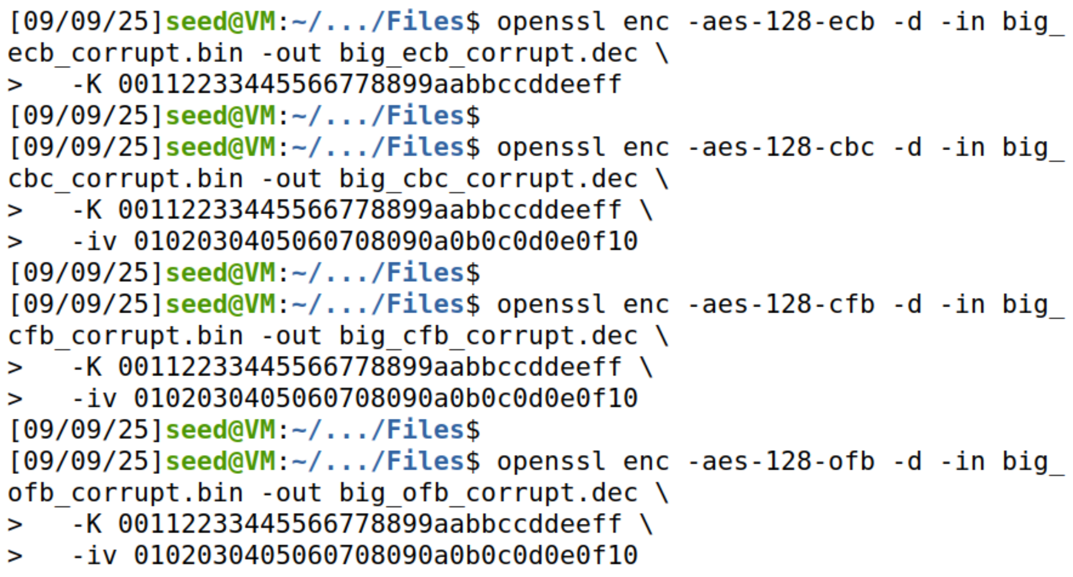
```
[09/08/25]seed@VM:~/.../Files$ wc -c bigplain.txt
2000 bigplain.txt
```

Timothy Yip



```
[09/08/25]seed@VM:~/.../Files$ # ECB
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in bigp
lain.txt -out big_ecb.bin \
>   -K 00112233445566778899aabbccddeeff
[09/08/25]seed@VM:~/.../Files$
[09/08/25]seed@VM:~/.../Files$ # CBC
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in bigp
lain.txt -out big_cbc.bin \
>   -K 00112233445566778899aabbccddeeff \
>   -iv 0102030405060708090a0b0c0d0e0f10
[09/08/25]seed@VM:~/.../Files$
[09/08/25]seed@VM:~/.../Files$ # CFB
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in bigp
lain.txt -out big_cfb.bin \
>   -K 00112233445566778899aabbccddeeff \
>   -iv 0102030405060708090a0b0c0d0e0f10
[09/08/25]seed@VM:~/.../Files$
[09/08/25]seed@VM:~/.../Files$ # OFB
[09/08/25]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in bigp
lain.txt -out big_ofb.bin \
>   -K 00112233445566778899aabbccddeeff \
>   -iv 0102030405060708090a0b0c0d0e0f10
```

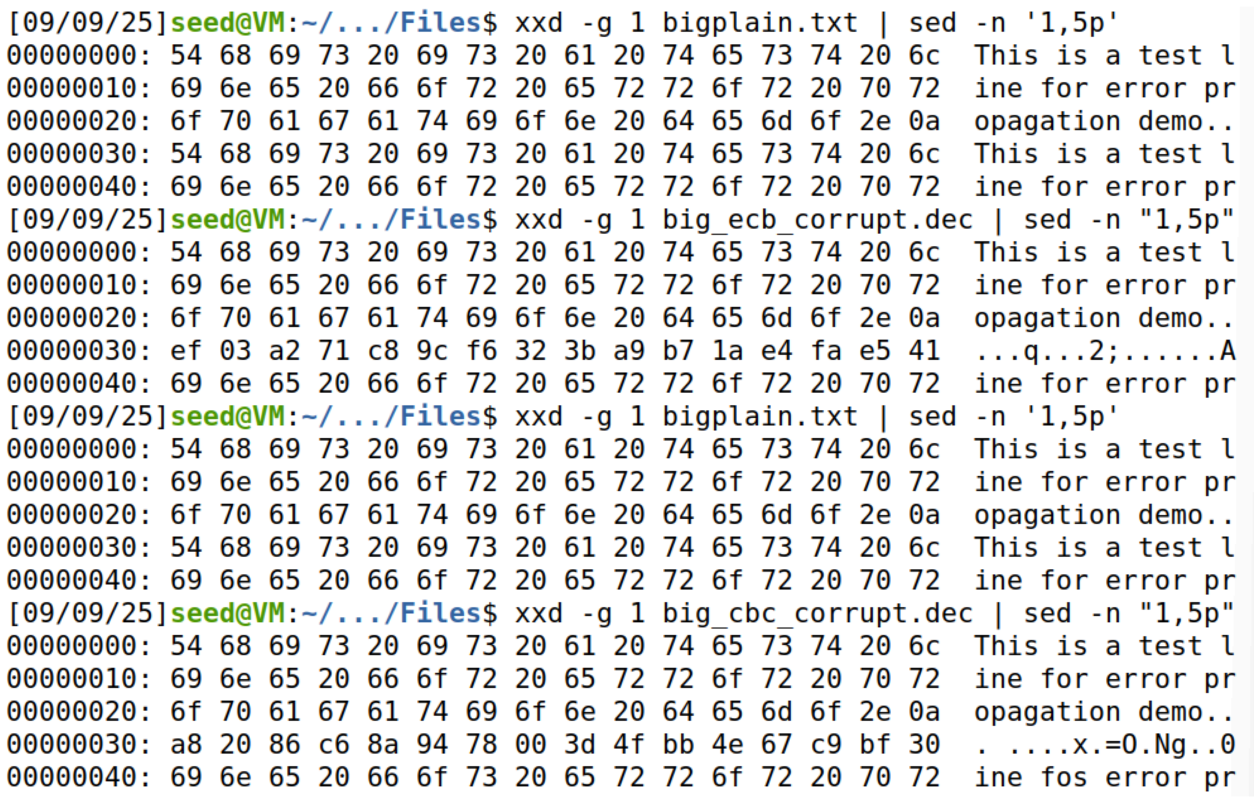
After modifying each file to create a “corrupted” file by changing the 55 byte, we decipher each one.



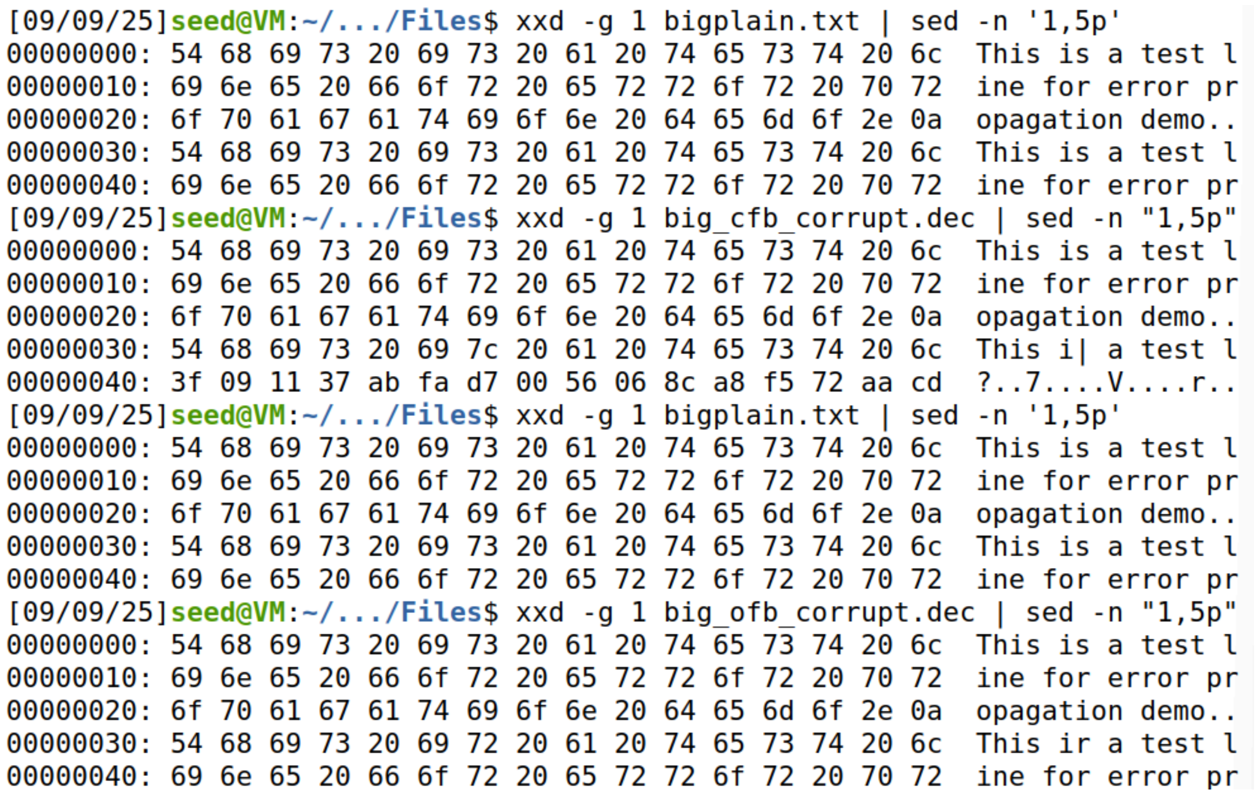
```
[09/09/25]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -d -in big_
ecb_corrupt.bin -out big_ecb_corrupt.dec \
>   -K 00112233445566778899aabbccddeeff
[09/09/25]seed@VM:~/.../Files$
[09/09/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in big_
cbc_corrupt.bin -out big_cbc_corrupt.dec \
>   -K 00112233445566778899aabbccddeeff \
>   -iv 0102030405060708090a0b0c0d0e0f10
[09/09/25]seed@VM:~/.../Files$
[09/09/25]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in big_
cfb_corrupt.bin -out big_cfb_corrupt.dec \
>   -K 00112233445566778899aabbccddeeff \
>   -iv 0102030405060708090a0b0c0d0e0f10
[09/09/25]seed@VM:~/.../Files$
[09/09/25]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -d -in big_
ofb_corrupt.bin -out big_ofb_corrupt.dec \
>   -K 00112233445566778899aabbccddeeff \
>   -iv 0102030405060708090a0b0c0d0e0f10
```

Now we compare the differences:

Timothy Yip



```
[09/09/25]seed@VM:~/.../Files$ xxd -g 1 bigplain.txt | sed -n '1,5p'
00000000: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6c  This is a test l
00000010: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
00000020: 6f 70 61 67 61 74 69 6f 6e 20 64 65 6d 6f 2e 0a  opagation demo..
00000030: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6c  This is a test l
00000040: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
[09/09/25]seed@VM:~/.../Files$ xxd -g 1 big_ecb_corrupt.dec | sed -n "1,5p"
00000000: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6c  This is a test l
00000010: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
00000020: 6f 70 61 67 61 74 69 6f 6e 20 64 65 6d 6f 2e 0a  opagation demo..
00000030: ef 03 a2 71 c8 9c f6 32 3b a9 b7 1a e4 fa e5 41  ...q...2;......A
00000040: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
[09/09/25]seed@VM:~/.../Files$ xxd -g 1 bigplain.txt | sed -n '1,5p'
00000000: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6c  This is a test l
00000010: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
00000020: 6f 70 61 67 61 74 69 6f 6e 20 64 65 6d 6f 2e 0a  opagation demo..
00000030: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6c  This is a test l
00000040: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
[09/09/25]seed@VM:~/.../Files$ xxd -g 1 big_cbc_corrupt.dec | sed -n "1,5p"
00000000: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6c  This is a test l
00000010: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
00000020: 6f 70 61 67 61 74 69 6f 6e 20 64 65 6d 6f 2e 0a  opagation demo..
00000030: a8 20 86 c6 8a 94 78 00 3d 4f bb 4e 67 c9 bf 30  . ....x.=O.Ng..0
00000040: 69 6e 65 20 66 6f 73 20 65 72 72 6f 72 20 70 72  ine fos error pr
```



```
[09/09/25]seed@VM:~/.../Files$ xxd -g 1 bigplain.txt | sed -n '1,5p'
00000000: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6c  This is a test l
00000010: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
00000020: 6f 70 61 67 61 74 69 6f 6e 20 64 65 6d 6f 2e 0a  opagation demo..
00000030: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6c  This is a test l
00000040: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
[09/09/25]seed@VM:~/.../Files$ xxd -g 1 big_cfb_corrupt.dec | sed -n "1,5p"
00000000: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6c  This is a test l
00000010: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
00000020: 6f 70 61 67 61 74 69 6f 6e 20 64 65 6d 6f 2e 0a  opagation demo..
00000030: 54 68 69 73 20 69 7c 20 61 20 74 65 73 74 20 6c  This i| a test l
00000040: 3f 09 11 37 ab fa d7 00 56 06 8c a8 f5 72 aa cd  ?..7....V....r..
[09/09/25]seed@VM:~/.../Files$ xxd -g 1 bigplain.txt | sed -n '1,5p'
00000000: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6c  This is a test l
00000010: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
00000020: 6f 70 61 67 61 74 69 6f 6e 20 64 65 6d 6f 2e 0a  opagation demo..
00000030: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6c  This is a test l
00000040: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
[09/09/25]seed@VM:~/.../Files$ xxd -g 1 big_ofb_corrupt.dec | sed -n "1,5p"
00000000: 54 68 69 73 20 69 73 20 61 20 74 65 73 74 20 6c  This is a test l
00000010: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
00000020: 6f 70 61 67 61 74 69 6f 6e 20 64 65 6d 6f 2e 0a  opagation demo..
00000030: 54 68 69 73 20 69 72 20 61 20 74 65 73 74 20 6c  This ir a test l
00000040: 69 6e 65 20 66 6f 72 20 65 72 72 6f 72 20 70 72  ine for error pr
```

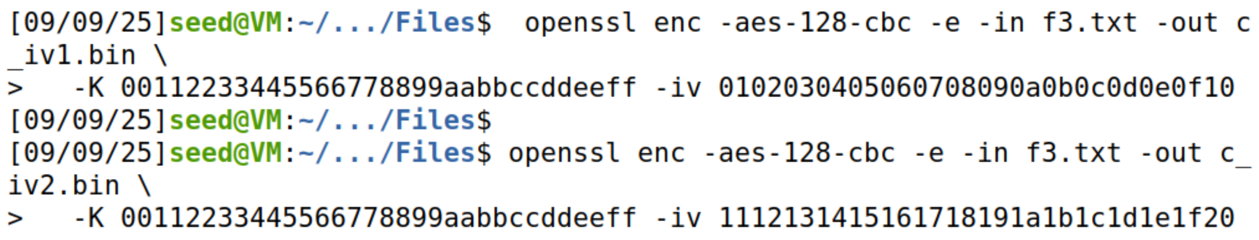
Timothy Yip

From these screenshots, we can see that from ECB, one corrupted bit resulted in an entire block of 16 bytes being lost, but later blocks are fine. For CBC, one corrupted bit resulted in an entire block of 16 bytes lost plus 1 bit error in the next block. For CFB, one corrupted bit resulted in that bit flipping in the current block, but the next block becomes garbage. Finally for OFB, one corrupted bit resulted in only that bit flipping, no propagation.

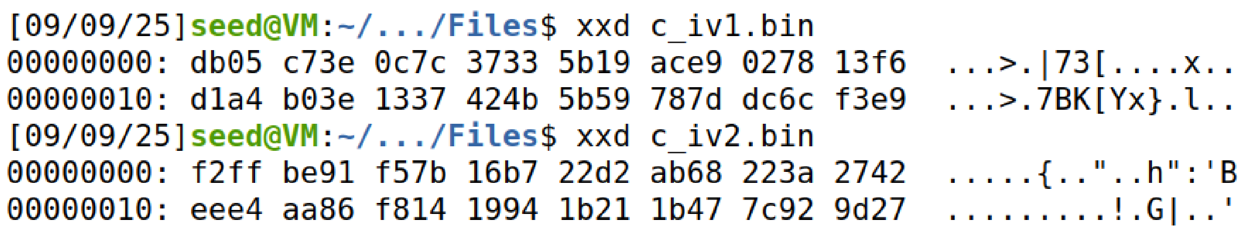
Task 6: Initial Vector (IV) and Common Mistakes

Task 6.1

Encrypting with different IV values



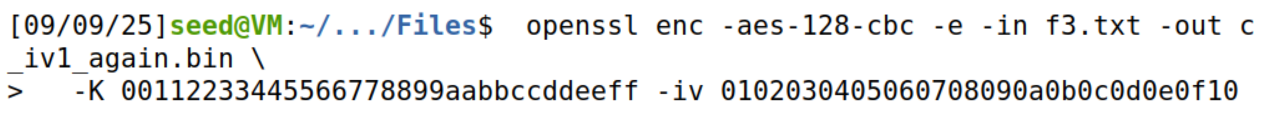
```
[09/09/25]seed@VM:~/.../Files$  openssl enc -aes-128-cbc -e -in f3.txt -out c
_iv1.bin \
>   -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
[09/09/25]seed@VM:~/.../Files$
[09/09/25]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f3.txt -out c_
iv2.bin \
>   -K 00112233445566778899aabbccddeeff -iv 1112131415161718191a1b1c1d1e1f20
```



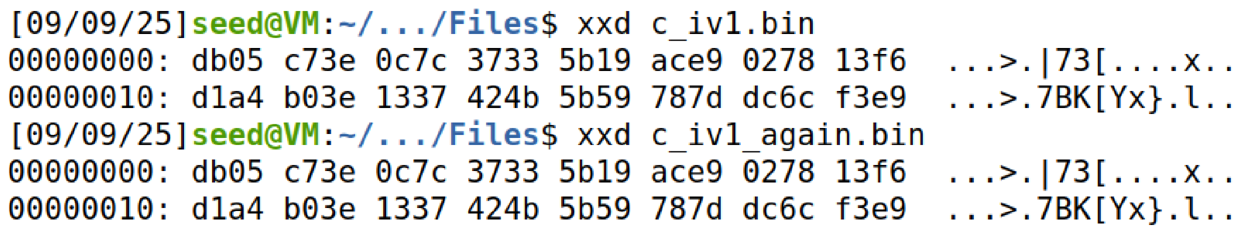
```
[09/09/25]seed@VM:~/.../Files$ xxd c_iv1.bin
00000000: db05 c73e 0c7c 3733 5b19 ace9 0278 13f6  ...>.|73[....x..
00000010: d1a4 b03e 1337 424b 5b59 787d dc6c f3e9  ...>.7BK[Yx}.l..
[09/09/25]seed@VM:~/.../Files$ xxd c_iv2.bin
00000000: f2ff be91 f57b 16b7 22d2 ab68 223a 2742  .....{..."..h":'B
00000010: eee4 aa86 f814 1994 1b21 1b47 7c92 9d27  .........!.G|..'
```

We can see that using different IV values resulted in creating different encrypted files. The bytes are completely different from each other.

Encrypting with the same IV



```
[09/09/25]seed@VM:~/.../Files$  openssl enc -aes-128-cbc -e -in f3.txt -out c
_iv1_again.bin \
>   -K 00112233445566778899aabbccddeeff -iv 0102030405060708090a0b0c0d0e0f10
```



```
[09/09/25]seed@VM:~/.../Files$ xxd c_iv1.bin
00000000: db05 c73e 0c7c 3733 5b19 ace9 0278 13f6  ...>.|73[....x..
00000010: d1a4 b03e 1337 424b 5b59 787d dc6c f3e9  ...>.7BK[Yx}.l..
[09/09/25]seed@VM:~/.../Files$ xxd c_iv1_again.bin
00000000: db05 c73e 0c7c 3733 5b19 ace9 0278 13f6  ...>.|73[....x..
00000010: d1a4 b03e 1337 424b 5b59 787d dc6c f3e9  ...>.7BK[Yx}.l..
```

From this, we can see that using the same IV value will result with the encryption producing the same encrypted file. The bytes are completely the same as each other.

Timothy Yip

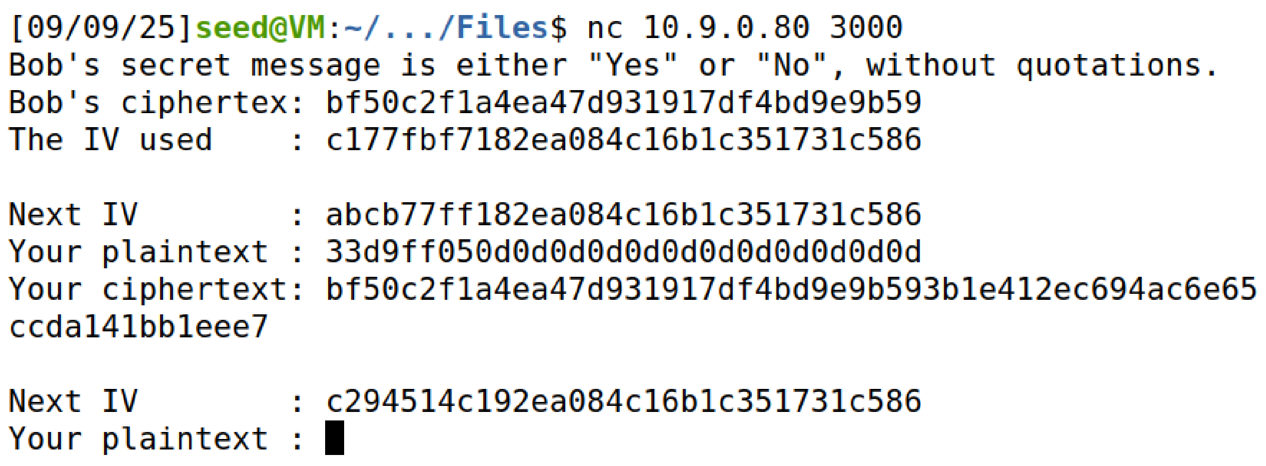
Task 6.2

Assuming that we are using the same IV value, we would be able to recover all of P2. CFB produces a K independent of plaintext, and ciphertext is C = P XOR K. If the same IV (K) is reused and an attacker knows a (P1, C1) pair, they can compute K = C1 XOR P1 and then recover any other plaintext encrypted with the same K by P2 = C2 XOR K. That recovers the full P2.

Task 6.3

Using the IVs given by the command, these are the submit hexes that we computed for yes/no

Yes: 33d9ff050d0d0d0d0d0d0d0d0d0d0d0d
No: 24d382060e0e0e0e0e0e0e0e0e0e0e0e

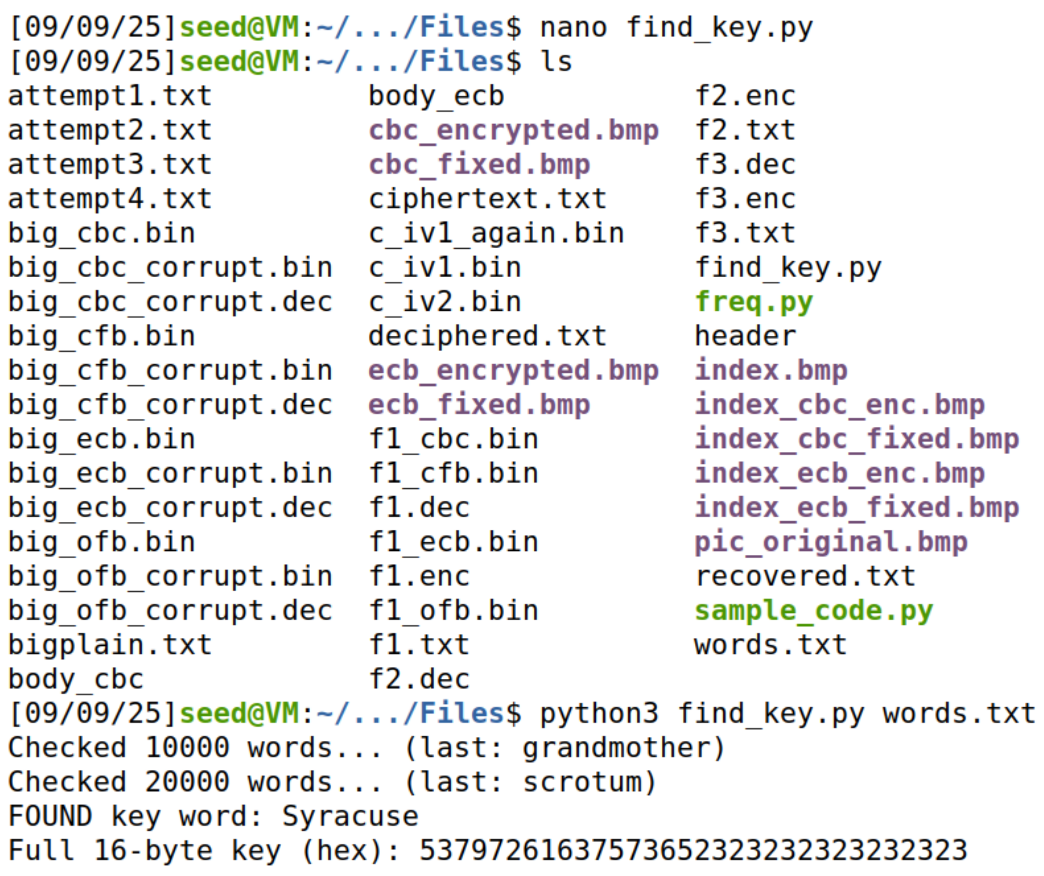


```
[09/09/25]seed@VM:~/.../Files$ nc 10.9.0.80 3000
Bob's secret message is either "Yes" or "No", without quotations.
Bob's ciphertex: bf50c2f1a4ea47d931917df4bd9e9b59
The IV used    : c177fbf7182ea084c16b1c351731c586

Next IV        : abcb77ff182ea084c16b1c351731c586
Your plaintext : 33d9ff050d0d0d0d0d0d0d0d0d0d0d0d
Your ciphertext: bf50c2f1a4ea47d931917df4bd9e9b593b1e412ec694ac6e65
ccda141bb1eee7

Next IV        : c294514c192ea084c16b1c351731c586
Your plaintext : █
```

Once I entered the hex for yes, I got the exact same ciphertext as Bob’s ciphertext which means Bob’s secret message was “Yes”.

Task 7: Programming using the Crypto Library

Timothy Yip



```
[09/09/25]seed@VM:~/.../Files$ nano find_key.py
[09/09/25]seed@VM:~/.../Files$ ls
attempt1.txt          body_ecb            f2.enc
attempt2.txt          cbc_encrypted.bmp   f2.txt
attempt3.txt          cbc_fixed.bmp       f3.dec
attempt4.txt          ciphertext.txt      f3.enc
big_cbc.bin           c_iv1_again.bin     f3.txt
big_cbc_corrupt.bin   c_iv1.bin           find_key.py
big_cbc_corrupt.dec   c_iv2.bin           freq.py
big_cfb.bin           deciphered.txt      header
big_cfb_corrupt.bin   ecb_encrypted.bmp   index.bmp
big_cfb_corrupt.dec   ecb_fixed.bmp       index_cbc_enc.bmp
big_ecb.bin           f1_cbc.bin          index_cbc_fixed.bmp
big_ecb_corrupt.bin   f1_cfb.bin          index_ecb_enc.bmp
big_ecb_corrupt.dec   f1.dec              index_ecb_fixed.bmp
big_ofb.bin           f1_ecb.bin          pic_original.bmp
big_ofb_corrupt.bin   f1.enc              recovered.txt
big_ofb_corrupt.dec   f1_ofb.bin          sample_code.py
bigplain.txt          f1.txt              words.txt
body_cbc              f2.dec
[09/09/25]seed@VM:~/.../Files$ python3 find_key.py words.txt
Checked 10000 words... (last: grandmother)
Checked 20000 words... (last: scrotum)
FOUND key word: Syracuse
Full 16-byte key (hex): 53797261637573652323232323232323
```

After creating, compiling and running a C code, we have found the key “Syracuse” from the words.txt file provided.