Lab Report: Secret Key Encryption Lab

Robert D. Hernandez rherna70@uic.edu

Task 1: Frequency Analysis

Given some trial and error first substituted the longest n-grams with the highest ngrams "the" and "and", then moved onto a few longer 2-n grams testing "in" and "er" and from there I was able to make out some smaller words and was able to character-by-character read and translate the doc. The resulting key was "ytnvupmuqxifahlbrcdzgsokejw" and the plantext is viewable with:

```
tr 'ytnvupmuqxifahlbrcdzgsokejw' 'THEANDINSOLVCRWFGMYUBKJXPQZ' <
ciphertext.txt > out.txt
```

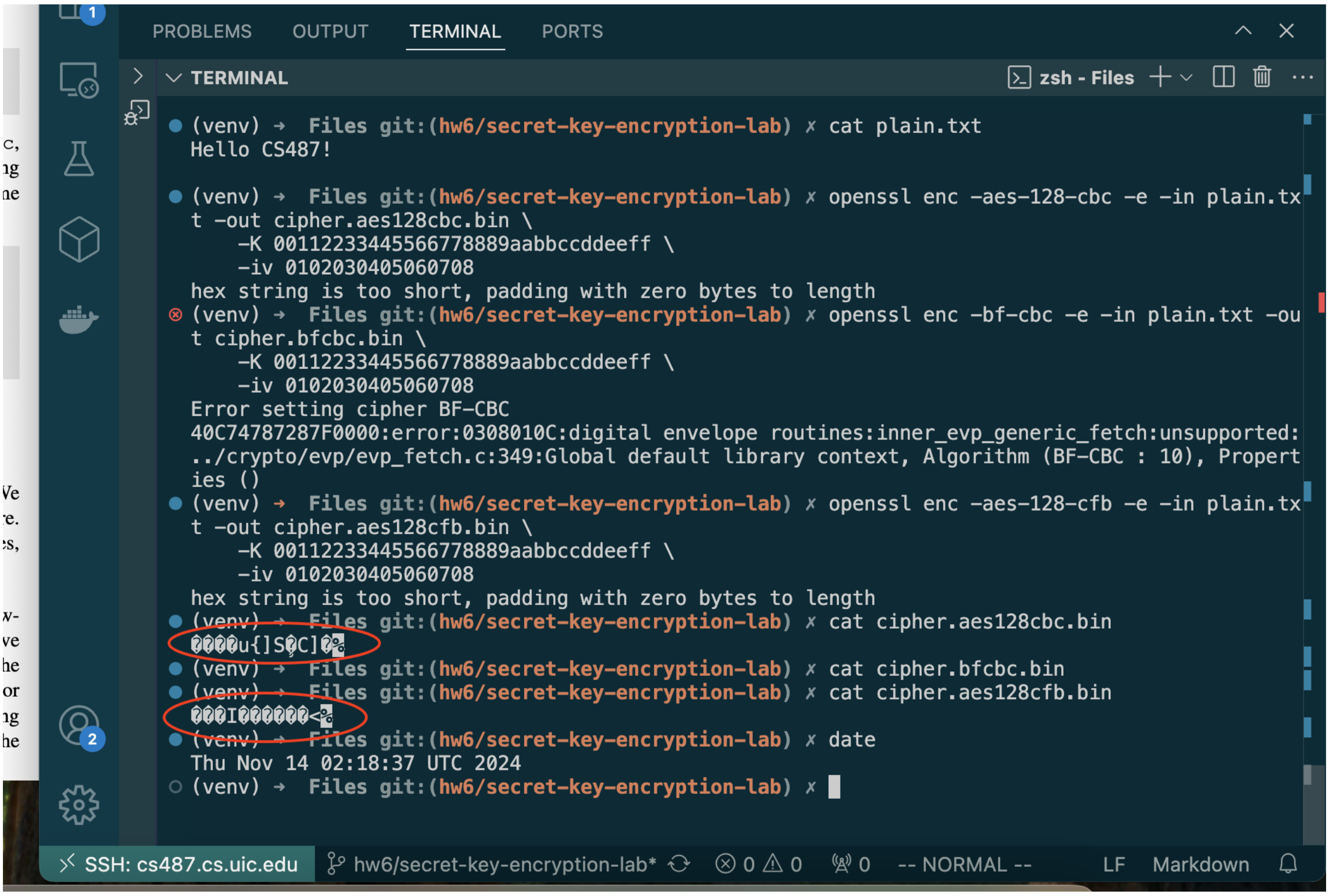
Task 2:

Below you can see the encryption commands I used, I tried all three cipher modes described, it seems like my vm did not support `-bf-cbc`

```
openssl enc -aes-128-cbc -e -in plain.txt -out cipher.bin \
    -K 00112233445566778889aabbccddeeff \
    -iv 0102030405060708

openssl enc -bf-cbc -e -in plain.txt -out cipher.bin \
    -K 00112233445566778889aabbccddeeff \
    -iv 0102030405060708

openssl enc -aes-128-cfb -e -in plain.txt -out cipher.bin \
    -K 00112233445566778889aabbccddeeff \
    -iv 0102030405060708
```

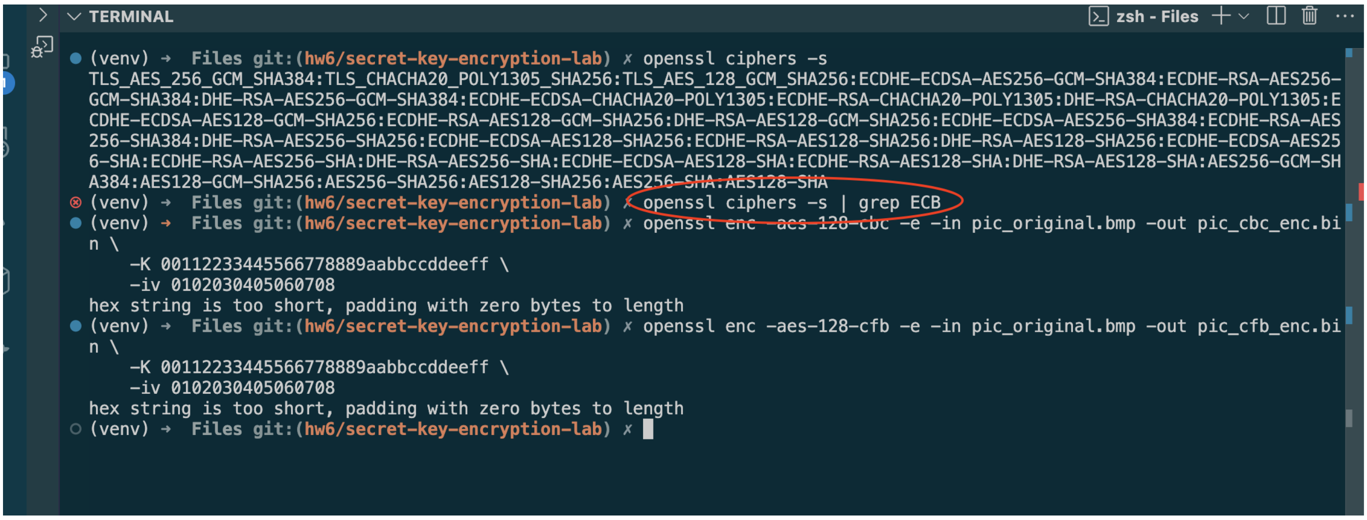


```
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ cat plain.txt
  Hello CS487!

● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ openssl enc -aes-128-cbc -e -in plain.tx
  t -out cipher.aes128cbc.bin \
      -K 00112233445566778889aabbccddeeff \
      -iv 0102030405060708
  hex string is too short, padding with zero bytes to length
⊗ (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ openssl enc -bf-cbc -e -in plain.txt -ou
  t cipher.bfcbc.bin \
      -K 00112233445566778889aabbccddeeff \
      -iv 0102030405060708
  Error setting cipher BF-CBC
  40C74787287F0000:error:0308010C:digital envelope routines:inner_evp_generic_fetch:unsupported:
  ../crypto/evp/evp_fetch.c:349:Global default library context, Algorithm (BF-CBC : 10), Propert
  ies ()
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ openssl enc -aes-128-cfb -e -in plain.tx
  t -out cipher.aes128cfb.bin \
      -K 00112233445566778889aabbccddeeff \
      -iv 0102030405060708
  hex string is too short, padding with zero bytes to length
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ cat cipher.aes128cbc.bin
  ����u{]S�C]�%
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ cat cipher.bfcbc.bin
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ cat cipher.aes128cfb.bin
  ���I�������<%
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ date
  Thu Nov 14 02:18:37 UTC 2024
○ (venv) → Files git:(hw6/secret-key-encryption-lab) ✗
```

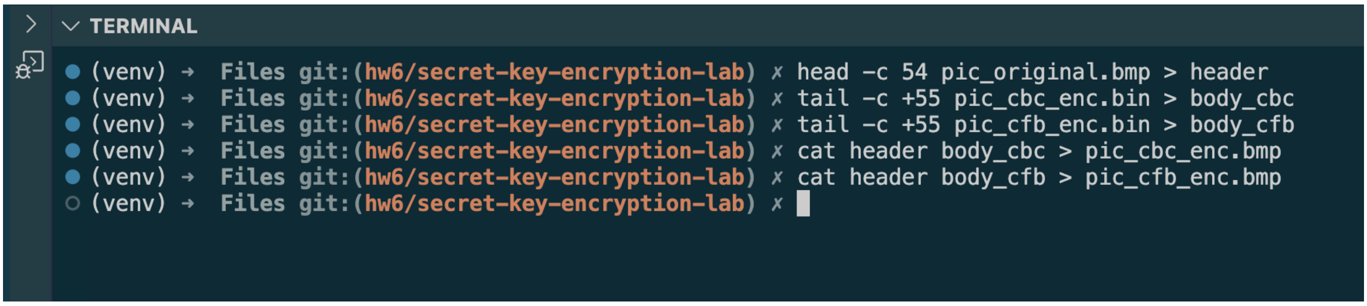
Task 3:

I was able to encrypt the picture using CBC mode, which was straightforward, although my machine did not support ECB mode so I will also encrypt and compare with CFB mode



```
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ openssl ciphers -s
  TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
  GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-CHACHA20-POLY1305:E
  CDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES
  256-SHA384:DHE-RSA-AES256-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES25
  6-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES128-SHA:DHE-RSA-AES128-SHA:AES256-GCM-SH
  A384:AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-SHA
⊗ (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ openssl ciphers -s | grep ECB
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ openssl enc -aes-128-cbc -e -in pic_original.bmp -out pic_cbc_enc.bi
  n \
      -K 00112233445566778889aabbccddeeff \
      -iv 0102030405060708
  hex string is too short, padding with zero bytes to length
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ openssl enc -aes-128-cfb -e -in pic_original.bmp -out pic_cfb_enc.bi
  n \
      -K 00112233445566778889aabbccddeeff \
      -iv 0102030405060708
  hex string is too short, padding with zero bytes to length
○ (venv) → Files git:(hw6/secret-key-encryption-lab) ✗
```

Commands used for re-writing the header hex values so we can render the image:



```
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ head -c 54 pic_original.bmp > header
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ tail -c +55 pic_cbc_enc.bin > body_cbc
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ tail -c +55 pic_cfb_enc.bin > body_cfb
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ cat header body_cbc > pic_cbc_enc.bmp
● (venv) → Files git:(hw6/secret-key-encryption-lab) ✗ cat header body_cfb > pic_cfb_enc.bmp
○ (venv) → Files git:(hw6/secret-key-encryption-lab) ✗
```

And the image encrypted with CBC mode here:

And the image encrypted with CFB mode here:

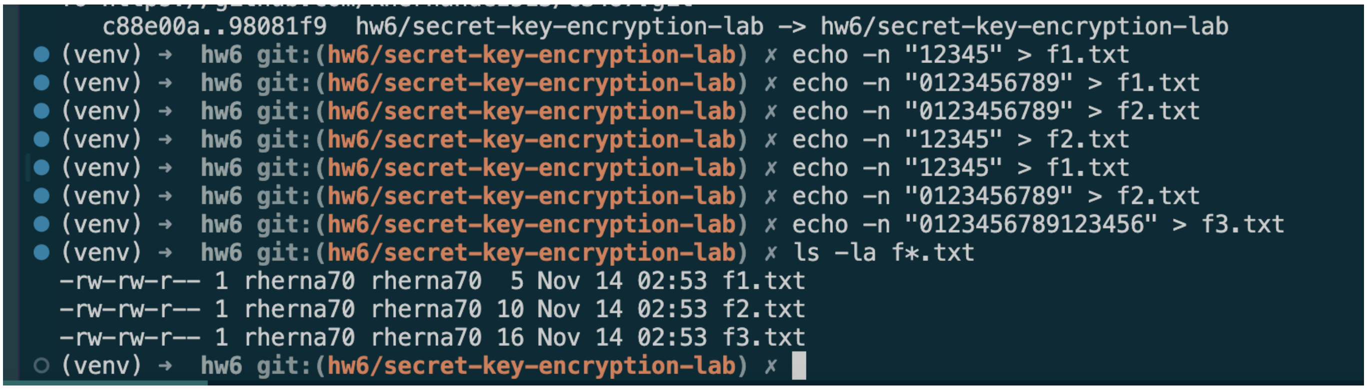
We do not see any useful information from the encrypted picture

Task 4:

Requiring padding for an given cipher algorithm is really a function of whether we use block cipher or not. If we use a block cipher, the plaintext must be the same size as the block cipher encryption which we can right-size using padding. Converting a block cipher to a stream cipher elimintates this need.

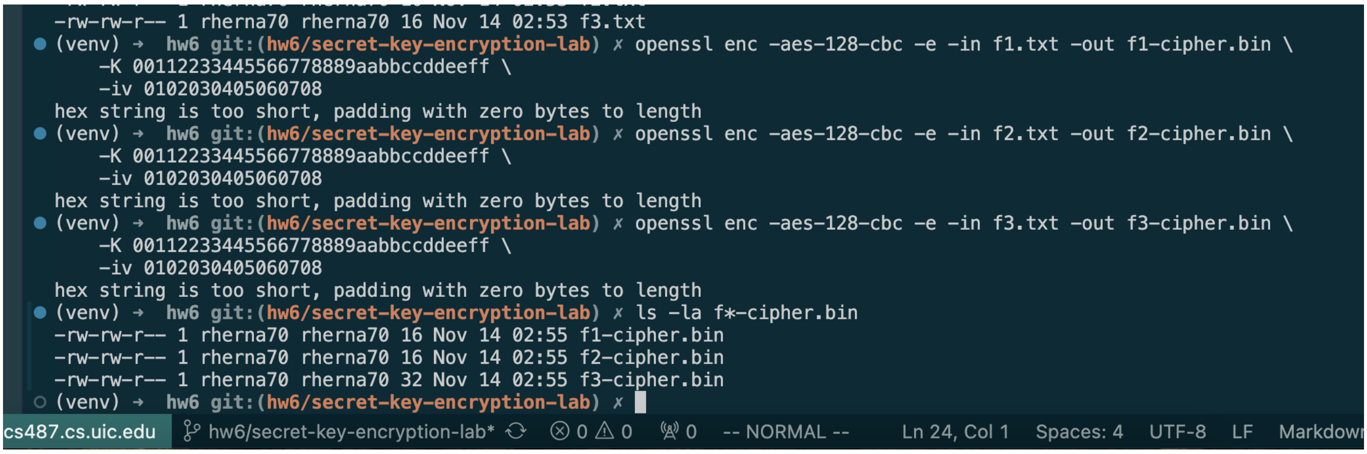
4.2

First we create the files:



```
   c88e00a..98081f9  hw6/secret-key-encryption-lab -> hw6/secret-key-encryption-lab
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ echo -n "12345" > f1.txt
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ echo -n "0123456789" > f1.txt
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ echo -n "0123456789" > f2.txt
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ echo -n "12345" > f2.txt
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ echo -n "12345" > f1.txt
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ echo -n "0123456789" > f2.txt
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ echo -n "0123456789123456" > f3.txt
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ ls -la f*.txt
  -rw-rw-r-- 1 rherna70 rherna70  5 Nov 14 02:53 f1.txt
  -rw-rw-r-- 1 rherna70 rherna70 10 Nov 14 02:53 f2.txt
  -rw-rw-r-- 1 rherna70 rherna70 16 Nov 14 02:53 f3.txt
○ (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗
```

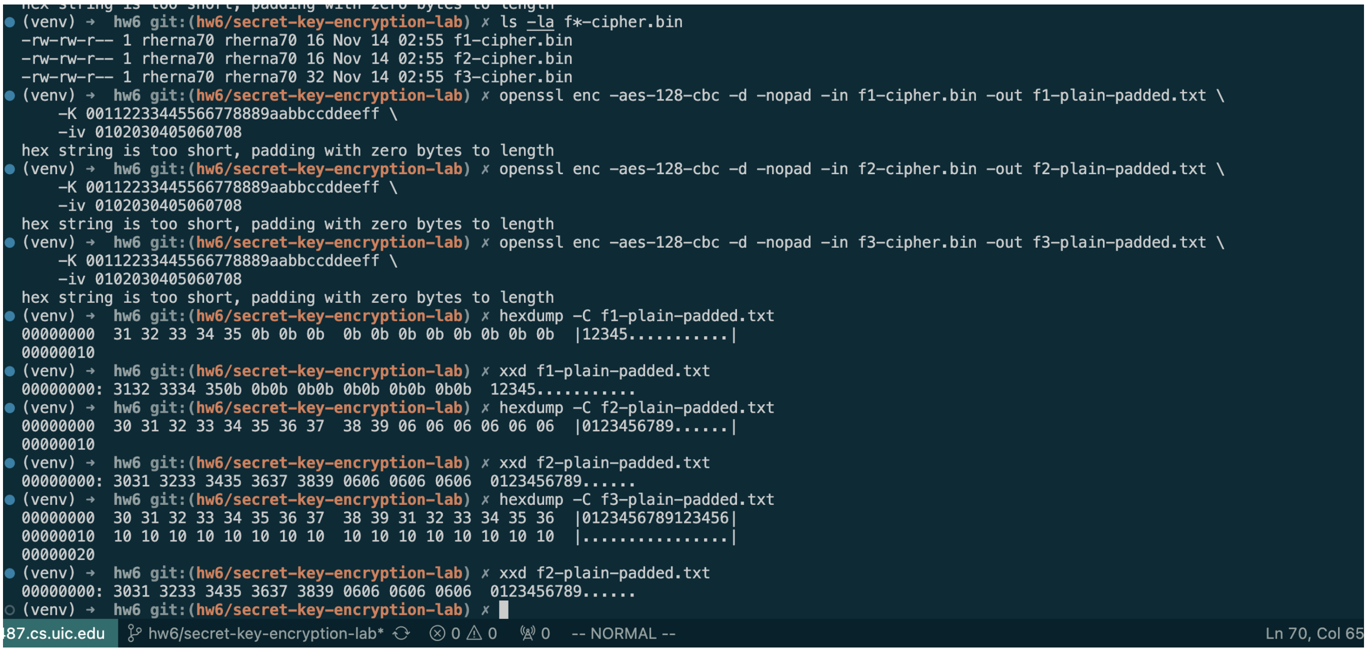
Then we create encrypted versions using `-aes-128-cbc`



```
-rw-rw-r-- 1 rherna70 rherna70 16 Nov 14 02:53 f3.txt
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ openssl enc -aes-128-cbc -e -in f1.txt -out f1-cipher.bin \
    -K 00112233445566778889aabbccddeeff \
    -iv 0102030405060708
hex string is too short, padding with zero bytes to length
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ openssl enc -aes-128-cbc -e -in f2.txt -out f2-cipher.bin \
    -K 00112233445566778889aabbccddeeff \
    -iv 0102030405060708
hex string is too short, padding with zero bytes to length
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ openssl enc -aes-128-cbc -e -in f3.txt -out f3-cipher.bin \
    -K 00112233445566778889aabbccddeeff \
    -iv 0102030405060708
hex string is too short, padding with zero bytes to length
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ ls -la f*-cipher.bin
-rw-rw-r-- 1 rherna70 rherna70 16 Nov 14 02:55 f1-cipher.bin
-rw-rw-r-- 1 rherna70 rherna70 16 Nov 14 02:55 f2-cipher.bin
-rw-rw-r-- 1 rherna70 rherna70 32 Nov 14 02:55 f3-cipher.bin
○ (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗
```

We notice that the first two files that were less than 16 bytes were padded to 16 bytes and we see that curious enough the one that was 16 bytes was padded to 32 bytes.

Finally we decrypt and view the padded bytes using a hexdump tool:



```
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ ls -la f*-cipher.bin
-rw-rw-r-- 1 rherna70 rherna70 16 Nov 14 02:55 f1-cipher.bin
-rw-rw-r-- 1 rherna70 rherna70 16 Nov 14 02:55 f2-cipher.bin
-rw-rw-r-- 1 rherna70 rherna70 32 Nov 14 02:55 f3-cipher.bin
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ openssl enc -aes-128-cbc -d -nopad -in f1-cipher.bin -out f1-plain-padded.txt \
    -K 00112233445566778889aabbccddeeff \
    -iv 0102030405060708
hex string is too short, padding with zero bytes to length
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ openssl enc -aes-128-cbc -d -nopad -in f2-cipher.bin -out f2-plain-padded.txt \
    -K 00112233445566778889aabbccddeeff \
    -iv 0102030405060708
hex string is too short, padding with zero bytes to length
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ openssl enc -aes-128-cbc -d -nopad -in f3-cipher.bin -out f3-plain-padded.txt \
    -K 00112233445566778889aabbccddeeff \
    -iv 0102030405060708
hex string is too short, padding with zero bytes to length
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ hexdump -C f1-plain-padded.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345...........|
00000010
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ xxd f1-plain-padded.txt
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345...........
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ hexdump -C f2-plain-padded.txt
00000000  30 31 32 33 34 35 36 37  38 39 06 06 06 06 06 06  |0123456789......|
00000010
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ xxd f2-plain-padded.txt
00000000: 3031 3233 3435 3637 3839 0606 0606 0606  0123456789......
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ hexdump -C f3-plain-padded.txt
00000000  30 31 32 33 34 35 36 37  38 39 31 32 33 34 35 36  |0123456789123456|
00000010  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  |................|
00000020
● (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗ xxd f2-plain-padded.txt
00000000: 3031 3233 3435 3637 3839 0606 0606 0606  0123456789......
○ (venv) → hw6 git:(hw6/secret-key-encryption-lab) ✗
```

Task 5:

5.1: Create a text file that is at least 1000 bytes long.

```
dd if=/dev/urandom of=output_file.bin bs=1 count=1000
```

5.2 Encrypt the file using the AES-128 cipher

```
openssl enc -aes-128-cbc -e -in output_file.bin -out
output_file_cipher.bin \
    -K 00112233445566778889aabbccddeeff \
    -iv 0102030405060708
```

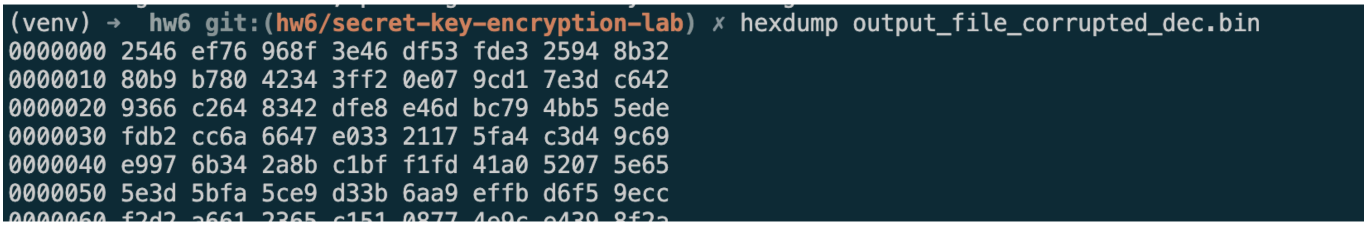
5.3 Oh no! One byte got corrupted

```
xxd -r -s 54 -l 4 -p <(echo deadbeef) output_file_cipher.bin
```

5.4 Decrypt and hexdump corrupted version and compare with original

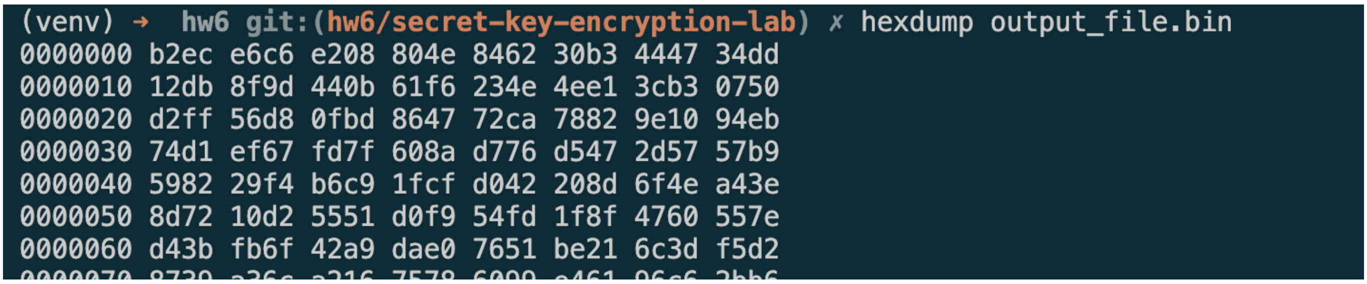
We can clearly demonstrate that changing one byte of an encrypted file stops it from being decrypted correctly entirely

Corrupted version



```
(venv) ➜  hw6 git:(hw6/secret-key-encryption-lab) ✗ hexdump output_file_corrupted_dec.bin
0000000 2546 ef76 968f 3e46 df53 fde3 2594 8b32
0000010 80b9 b780 4234 3ff2 0e07 9cd1 7e3d c642
0000020 9366 c264 8342 dfe8 e46d bc79 4bb5 5ede
0000030 fdb2 cc6a 6647 e033 2117 5fa4 c3d4 9c69
0000040 e997 6b34 2a8b c1bf f1fd 41a0 5207 5e65
0000050 5e3d 5bfa 5ce9 d33b 6aa9 effb d6f5 9ecc
0000060 f2d2 a661 2365 c151 0877 4e9c e439 8f2a
```

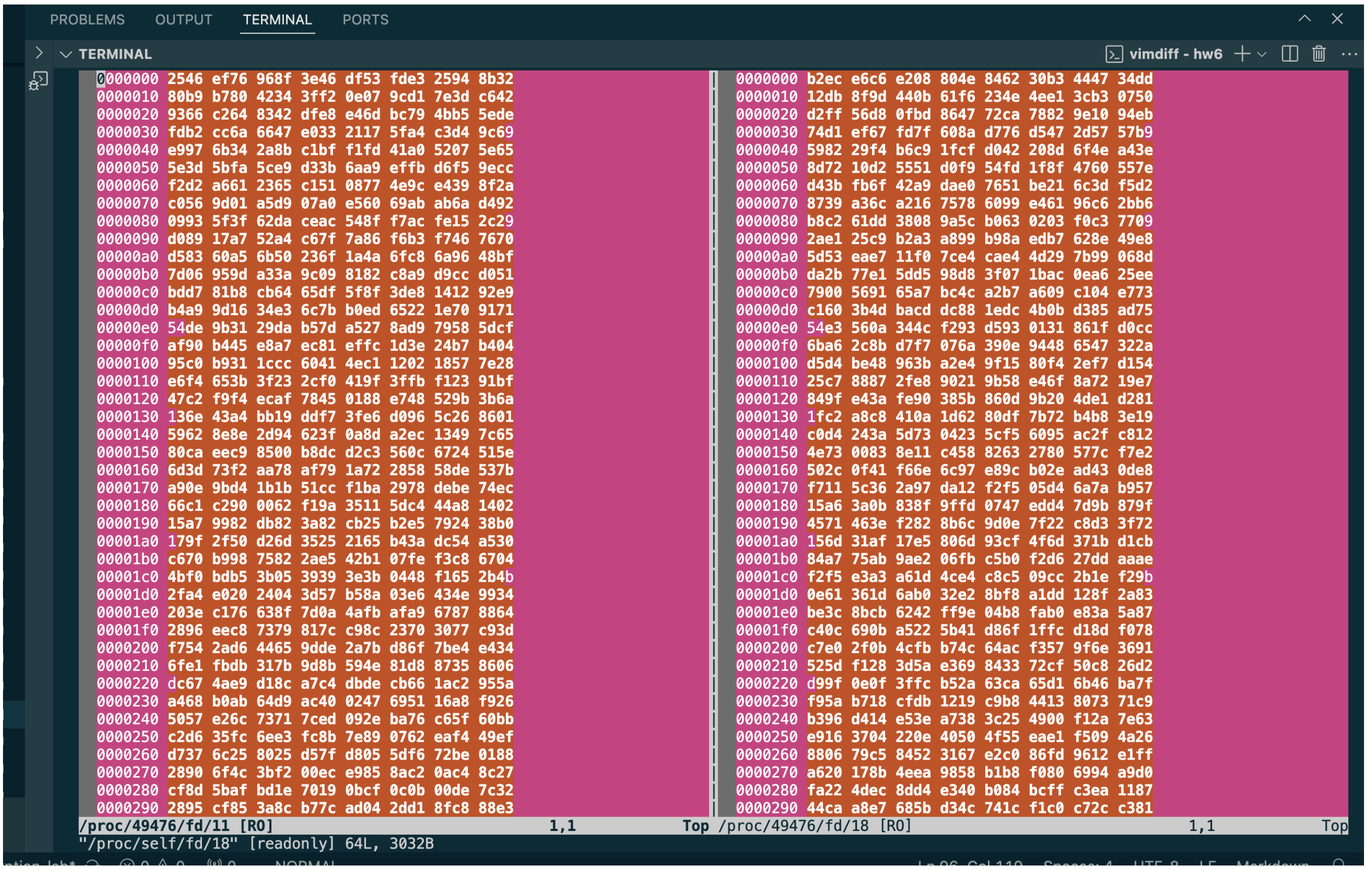
Original version



```
(venv) ➜  hw6 git:(hw6/secret-key-encryption-lab) ✗ hexdump output_file.bin
0000000 b2ec e6c6 e208 804e 8462 30b3 4447 34dd
0000010 12db 8f9d 440b 61f6 234e 4ee1 3cb3 0750
0000020 d2ff 56d8 0fbd 8647 72ca 7882 9e10 94eb
0000030 74d1 ef67 fd7f 608a d776 d547 2d57 57b9
0000040 5982 29f4 b6c9 1fcf d042 208d 6f4e a43e
0000050 8d72 10d2 5551 d0f9 54fd 1f8f 4760 557e
0000060 d43b fb6f 42a9 dae0 7651 be21 6c3d f5d2
0000070 8739 a36c a216 7578 6099 e461 96c6 2bb6
```

Comparing using vimdiff for good measure

```
vimdiff <(hexdump output_file_corrupted_dec.bin) <(hexdump
output_file.bin)
```



```
0000000 2546 ef76 968f 3e46 df53 fde3 2594 8b32 | 0000000 b2ec e6c6 e208 804e 8462 30b3 4447 34dd
0000010 80b9 b780 4234 3ff2 0e07 9cd1 7e3d c642 | 0000010 12db 8f9d 440b 61f6 234e 4ee1 3cb3 0750
0000020 9366 c264 8342 dfe8 e46d bc79 4bb5 5ede | 0000020 d2ff 56d8 0fbd 8647 72ca 7882 9e10 94eb
0000030 fdb2 cc6a 6647 e033 2117 5fa4 c3d4 9c69 | 0000030 74d1 ef67 fd7f 608a d776 d547 2d57 57b9
0000040 e997 6b34 2a8b c1bf f1fd 41a0 5207 5e65 | 0000040 5982 29f4 b6c9 1fcf d042 208d 6f4e a43e
0000050 5e3d 5bfa 5ce9 d33b 6aa9 effb d6f5 9ecc | 0000050 8d72 10d2 5551 d0f9 54fd 1f8f 4760 557e
0000060 f2d2 a661 2365 c151 0877 4e9c e439 8f2a | 0000060 d43b fb6f 42a9 dae0 7651 be21 6c3d f5d2
0000070 c056 9d01 a5d9 07a0 e560 69ab ab6a d492 | 0000070 8739 a36c a216 7578 6099 e461 96c6 2bb6
0000080 0993 5f3f 62da ceac 548f f7ac fe15 2c29 | 0000080 b8c2 61dd 3808 9a5c b063 0203 f0c3 7709
0000090 d089 17a7 52a4 c67f 7a86 f6b3 f746 7670 | 0000090 2ae1 25c9 b2a3 a899 b98a edb7 628e 49e8
00000a0 d583 60a5 6b50 236f 1a4a 6fc8 6a96 48bf | 00000a0 5d53 eae7 11f0 7ce4 cae4 4d29 7b99 068d
00000b0 7d06 959d a33a 9c09 8182 c8a9 d9cc d051 | 00000b0 da2b 77e1 5dd5 98d8 3f07 1bac 0ea6 25ee
00000c0 bdd7 81b8 cb64 65df 5f8f 3de8 1412 92e9 | 00000c0 7900 5691 65a7 bc4c a2b7 a609 c104 e773
00000d0 b4a9 9d16 34e3 6c7b b0ed 6522 1e70 9171 | 00000d0 c160 3b4d bacd dc88 1edc 4b0b d385 ad75
00000e0 54de 9b31 29da b57d a527 8ad9 7958 5dcf | 00000e0 54e3 560a 344c f293 d593 0131 861f d0cc
00000f0 af90 b445 e8a7 ec81 effc 1d3e 24b7 b404 | 00000f0 6ba6 2c8b d7f7 076a 390e 9448 6547 322a
0000100 95c0 b931 1ccc 6041 4ec1 1202 1857 7e28 | 0000100 d5d4 be48 963b a2e4 9f15 80f4 2ef7 d154
0000110 e6f4 653b 3f23 2cf0 419f 3ffb f123 91bf | 0000110 25c7 8887 2fe8 9021 9b58 e46f 8a72 19e7
0000120 47c2 f9f4 ecaf 7845 0188 e748 529b 3b6a | 0000120 849f e43a fe90 385b 860d 9b20 4de1 d281
0000130 136e 43a4 bb19 ddf7 3fe6 d096 5c26 8601 | 0000130 1fc2 a8c8 410a 1d62 80df 7b72 b4b8 3e19
0000140 5962 8e8e 2d94 623f 0a8d a2ec 1349 7c65 | 0000140 c0d4 243a 5d73 0423 5cf5 6095 ac2f c812
0000150 80ca eec9 8500 b8dc d2c3 560c 6724 515e | 0000150 4e73 0083 8e11 c458 8263 2780 577c f7e2
0000160 6d3d 73f2 aa78 af79 1a72 2858 58de 537b | 0000160 502c 0f41 f66e 6c97 e89c b02e ad43 0de8
0000170 a90e 9bd4 1b1b 51cc f1ba 2978 debe 74ec | 0000170 f711 5c36 2a97 da12 f2f5 05d4 6a7a b957
0000180 66c1 c290 0062 f19a 3511 5dc4 44a8 1402 | 0000180 15a6 3a0b 838f 9ffd 0747 edd4 7d9b 879f
0000190 15a7 9982 db82 3a82 cb25 b2e5 7924 38b0 | 0000190 4571 463e f282 8b6c 9d0e 7f22 c8d3 3f72
00001a0 179f 2f50 d26d 3525 2165 b43a dc54 a530 | 00001a0 156d 31af 17e5 806d 93cf 4f6d 371b d1cb
00001b0 c670 b998 7582 2ae5 42b1 07fe f3c8 6704 | 00001b0 84a7 75ab 9ae2 06fb c5b0 f2d6 27dd aaae
00001c0 4bf0 bdb5 3b05 3939 3e3b 0448 f165 2b4b | 00001c0 f2f5 e3a3 a61d 4ce4 c8c5 09cc 2b1e f29b
00001d0 2fa4 e020 2404 3d57 b58a 03e6 434e 9934 | 00001d0 0e61 361d 6ab0 32e2 8bf8 a1dd 128f 2a83
00001e0 203e c176 638f 7d0a 4afb afa9 6787 8864 | 00001e0 be3c 8bcb 6242 ff9e 04b8 fab0 e83a 5a87
00001f0 2896 eec8 7379 817c c98c 2370 3077 c93d | 00001f0 c40c 690b a522 5b41 d86f 1ffc d18d f078
0000200 f754 2ad6 4465 9dde 2a7b d86f 7be4 e434 | 0000200 c7e0 2f0b 4cfb b74c 64ac f357 9f6e 3691
0000210 6fe1 fbdb 317b 9d8b 594e 81d8 8735 8606 | 0000210 525d f128 3d5a e369 8433 72cf 50c8 26d2
0000220 dc67 4ae9 d18c a7c4 dbde cb66 1ac2 955a | 0000220 d99f 0e0f 3ffc b52a 63ca 65d1 6b46 ba7f
0000230 a468 b0ab 64d9 ac40 0247 6951 16a8 f926 | 0000230 f95a b718 cfdb 1219 c9b8 4413 8073 71c9
0000240 5057 e26c 7371 7ced 092e ba76 c65f 60bb | 0000240 b396 d414 e53e a738 3c25 4900 f12a 7e63
0000250 c2d6 35fc 6ee3 fc8b 7e89 0762 eaf4 49ef | 0000250 e916 3704 220e 4050 4f55 eae1 f509 4a26
0000260 d737 6c25 8025 d57f d805 5df6 72be 0188 | 0000260 8806 79c5 8452 3167 e2c0 86fd 9612 e1ff
0000270 2890 6f4c 3bf2 00ec e985 8ac2 0ac4 8c27 | 0000270 a620 178b 4eea 9858 b1b8 f080 6994 a9d0
0000280 cf8d 5baf bd1e 7019 0bcf 0c0b 00de 7c32 | 0000280 fa22 4dec 8dd4 e340 b084 bcff c3ea 1187
0000290 2895 cf85 3a8c b77c ad04 2dd1 8fc8 88e3 | 0000290 44ca a8e7 685b d34c 741c f1c0 c72c c381
/proc/49476/fd/11 [RO]                1,1           Top /proc/49476/fd/18 [RO]                1,1           Top
"/proc/self/fd/18" [readonly] 64L, 3032B
```

This doesn't entire hold with the theory of ECB, CBC, CFB, and OFB which where we should expect only a small amount of data to be undecryptable, where the theory would purport that we would only loose a single 16 byte block for ECB and CFB mode, 2 16-byte blocks for CBC, and only the single corrupted byte for OFB. Perhaps I made a mistake decrypting.

Task 6:

Task 7: