LABORATORY CEL62: Cryptography and System Security Winter 2021

Experiment 3:	**Crypto Encryption**

Note: Students are advised to read through this lab sheet before doing experiment. On-the-spot evaluation may be carried out during or at the end of the experiment. Your performance, teamwork effort, and learning attitude will count towards the marks.

Name: Ameya Jangam

Branch: Computer Engineering

UID: 2019130025

Experiment 4: Crypto Lab – Secret-Key Encryption

1 OBJECTIVE

The learning objective of this lab is for students to get familiar with the concepts in the secret-key encryption. After finishing the lab, students should be able to gain a first-hand experience on encryption algorithms, encryption modes, paddings, and initial vector (IV). Moreover, students will be able to use tools and write programs to encrypt/decrypt messages.

2 LAB ENVIRONMENT

Installing OpenSSL. In this lab, we will use openssl commands and libraries. We have already in-stalled openssl binaries in our VM. It should be noted that if you want to use openssl libraries in your programs, you need to install several other things for the programming environment, including the header files, libraries, manuals, etc. We have already downloaded the necessary files under the directory openssl-1.0.1. To configure and install openssl libraries, run the following commands.

You should read the INSTALL file first:

```
% ./config
% make
% make test
% sudo make install
```

Installing GHex. In this lab, we need to be able to view and modify files of binary format. We have installed in our VM GHex a hex editor for GNOME. It allows the user to load data from any file, view and edit it in either hex or ascii.

3 LAB TASKS

3.1 Task 1: Encryption using different ciphers and modes

In this task, we will play with various encryption algorithms and modes. You can use the following openssl enc command to encrypt/decrypt a file. To see the manuals, you can type man openssl and man enc.

```
% openssl enc ciphertype -e -in plain.txt -out cipher.bin \ -K
                        00112233445566778889aabbccddeeff \
                        -iv 0102030405060708
```

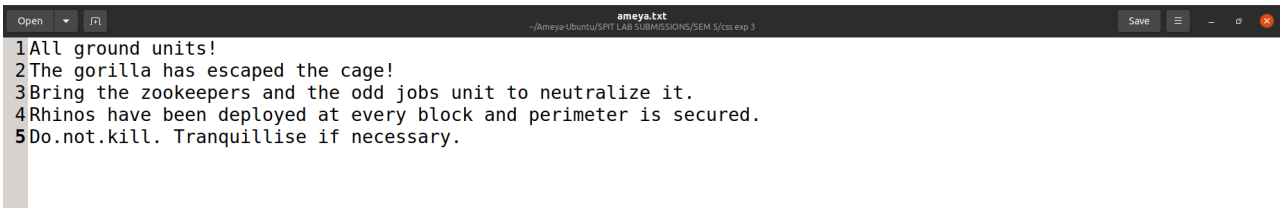
Crypto Encryption/PV

Please replace the ciphertype with a specific cipher type, such as -aes-128-cbc, -aes-128-cfb, -bf-cbc, etc. In this task, you should try at least 3 different ciphers and three different modes. You can find the meaning of the command-line options and all the supported cipher types by typing "man enc". We include some common options for the openssl enc command in the following:

-in <file> input file
-out <file> output file
-e encrypt
-d decrypt
-K/-iv key/iv in hex is the next argument
-[pP] print the iv/key (then exit if -P)

Encryption of Text using Different Ciphers:

The plaintext which will be used in this task:



1. Encryption using AES-128-CBC

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ openssl enc -aes-128-cbc -e -in ameya.txt -out ciphered.txt -K 00112233445566778899aabbccdd
eeff -iv 0102030405060708
```

Ciphered text

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ cat ciphered.txt
[illegible]
[illegible]
[illegible]
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/c
```

Hexdump

```
[illegible]ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/c
ss exhexdump ciphered.txt
0000000 a494 e88b fb90 4d7c 60fd 214c 551f 2043
0000010 9d6d c9ef 92e8 50c6 f4cc 0914 29d6 037a
0000020 adf5 688b 8f81 59f4 263c df71 8bf7 c590
0000030 394a 6a68 16e4 ad5e a90c 16ea 8517 3f7f
0000040 4a6c 84f8 711d 2fee cc4c 871e 0330 7faf
0000050 0564 6c10 1819 847b fe78 4daa f33d 239e
0000060 0169 7483 6bfe 4b95 ef16 c4f3 9fdf 1ae4
0000070 5157 cee8 c98c 064a cd17 e3c3 0aa2 9b2a
0000080 6829 6d70 d262 89b5 b24d d688 9f83 f3a3
0000090 5883 30c1 84dc 5095 ef87 6f5d fcd1 4ae2
00000a0 93b7 3727 5f13 e7cd 93dc 3e2b 8cc7 fad0
00000b0 3faa 1d92 f249 bb70 dd25 288d a5bb 8344
00000c0 07cb 182e 8d99 ffc7 f750 3f7f 74e7 6cdf
00000d0 a509 9646 86fe 7643 7b0e d60d a455 812f
00000e0
```

2. Encryption using AES-128-CFB

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ openssl enc -aes-128-cfb -e -in ameya.txt -out ciphered.txt -K 00112233445566778899aabbccdd
eeff -iv 0102030405060708
```

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ cat ciphered.txt
[illegible]
gd\"[illegible]
[illegible]ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUB
```

```
MISSIhexdump ciphered.txt
0000000 17e0 1dea b467 2162 530b 26d9 5258 a837
0000010 39f6 ef76 2f39 e3e2 3a41 c3e3 d8d7 6b35
0000020 166d e5b2 0936 0e08 23c8 b7ce 27c2 8719
0000030 5c5b dfd2 d3ff 3039 b389 c779 d8ec 81da
0000040 0b85 b4bc 1613 a31b 2ffd ff41 58ae 5e73
0000050 2f69 4af0 87f8 6e8f 919a cc20 b356 3551
0000060 3683 adcf 022a b108 c90f a7f4 48a6 6467
0000070 225c 7ed6 7c7d 36cc 1832 824b e7a5 573f
0000080 6c4a 1535 6353 255b 04f7 547a bf64 2b46
0000090 1260 ce4e 352c f163 8496 bead 2596 7c2d
00000a0 292b deb5 2c2b 8677 eb56 14a4 7af9 ce4f
00000b0 500b a64d 26b2 441a 91d7 9cab cc55 d865
00000c0 6b5a 400d 588d be2f 71fd 5bab 4c06 d06b
00000d0 ed8c 1928 1e2b 32a7 8363 2373
00000dc
```

3. Encryption using AES-128-ECB

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ openssl enc -aes-128-ecb -e -in ameya.txt -out ciphered.txt -K 00112233445566778899aabbccdd
eeff
```

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ cat ciphered.txt
[illegible]
[illegible]ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ hexdump ciphered.txt
0000000 cc8b f672 8361 8cd5 27db b776 b233 8f99
0000010 ea77 24cb a883 e226 aedd d8d1 4474 9c13
0000020 0996 f462 488c d53e 6425 9f67 adbc 07f1
0000030 8872 2c61 a45b 951c 7880 56bf c37d 0200
0000040 419b a325 8546 e24d 78f3 18e0 b954 3b2d
0000050 4397 48aa d97d 7d34 ff8e 34d5 376c 2343
0000060 aed8 2a78 8c7d 100b 08b1 0e67 1759 8256
0000070 7467 d5db f54e ec0e c9e9 22cb b6ac 027a
0000080 a70a 5488 fe31 5d8b 5b62 e963 1d5b 026c
0000090 f7b2 a9c0 ae08 e3b1 7595 18cb 68f2 7364
00000a0 bdaf da39 3731 4a1b 6cc9 bfb6 669b a669
00000b0 5453 859c 68b3 3af8 b144 90c6 cd72 73b2
00000c0 3d2e 0682 7d50 047b 6cf3 66df 6a18 6658
00000d0 ecb0 7810 fa2e 6ffa 48e5 a442 b617 4c88
00000e0
```

4. Encryption using AES-256-CBC

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ openssl enc -aes-256-cbc -e -in ameya.txt -out ciphered.txt -K 00112233445566778899aabbccdd
eeff -iv 0102030405060708
```

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ cat ciphered.txt
��p�.n�F�j�'jP���
                  Uy�fC�D���M�����Q1�m����r�Db[��h9��Y�B�v@@@@@@@c@Wk@<sx@t@@@A1W@@@p@A@4s@@R@,,@
d@@`@@@@@@@(i@@@@9⍁N@@H@@@s@o/?@t@@Kta(@n@z@@OV@Xse2@Vdi@@@@l@CG@d@Y'sF@@@v@<@@@$@4@ameya@ameya-
```

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ hexdump ciphered.txt
0000000 b9b2 9b70 012e e66e 460f 6ad7 27e3 6a10
0000010 c050 e9be 550c e579 4366 44d0 a3f7 c2b5
0000020 4d92 1614 fed0 a3e6 c514 51c1 1e31 1186
0000030 d36d eccd 72c4 9cf0 1eb5 6244 af5b 19e3
0000040 3968 157f 59cc 42b1 76a2 fd0e bc40 b303
0000050 1b94 858f eaaf 631d 579d 8c6b 733c cf78
0000060 bd74 dbe9 3141 ed57 083d edf9 0570 4181
0000070 1e40 c834 1908 1873 daed f852 2c2c 648b
0000080 c1b4 9060 afb0 afc1 df9d 6928 f08e b240
0000090 abda 4e39 e496 d848 f7f0 e773 6f99 3f2f
00000a0 74d7 9aa4 744b 2861 6ecd 1e92 107a 10e7
00000b0 bef2 074f c956 7358 1765 c432 6456 a269
00000c0 81be 6cf1 43fd fe47 08dc d964 2759 4673
00000d0 9cea c2f9 0376 3cb5 1da1 9e86 9124 d334
00000e0
```

5. Encryption using AES-256-CFB

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ openssl enc -aes-256-cfb -e -in ameya.txt -out ciphered.txt -K 00112233445566778899aabbccdd
eeff -iv 0102030405060708
```

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ cat ciphered.txt
:@z@@@@ĉ@@⍰@/5@@=.ABj@W]$@.@@@@/zG
@bJ@F@1@        @@@_,@i"@'1@⁄@ə8@M@n@G$@@(M@@@@F@@LV@Y5@t@Yp/@@@n@R:NBvc(@n1t@@@@@d@;g@@yw@@V`H@
@ @&@@⤫"L*d@@B@_@@@w@<@8@@@mV@@6@@N@uj-GQŸ@@rS@ₑₛ@@@ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0
```

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css ex
p 3$ hexdump ciphered.txt
0000000 a014 2a18 3a0d 7a8e cedf deff 89c4 b8b0
0000010 84c7 2fdb 93cd e735 1aa7 2e3d 4241 cd6a
0000020 5d57 a324 1e2e 83a6 b31c 7fd2 062f 477a
0000030 0a02 62ba 9e4a f146 ea31 8c09 e6c8 5f82
0000040 b42c 6900 af22 3127 d2c7 cb82 8fc6 fb38
0000050 a64d ef6e 47a1 e524 28f2 d84d d6e3 a7f0
0000060 18ad c746 4cf3 e856 3559 84ed e974 7059
0000070 fb2f a983 a66e 3a52 4e0f 7642 2863 6ebf
0000080 7431 85b9 9e99 c41e f164 673b ac84 7779
0000090 80ba 6056 8e48 20e5 26b8 d4b7 e205 aea4
00000a0 2210 2a4c aa64 19ee 0e00 8442 0f5f 8905
00000b0 9c86 af77 933c a338 02fa 6dc4 9956 36ca
00000c0 01d0 d596 f888 1175 d66a 478a ce51 a1ab
00000d0 72cf 0253 ea9f a3a7 da3b a5fe
00000dc
```

6. Encryption using AES-256-ECB

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3$ openssl enc -
aes-256-ecb -e -in ameya.txt -out ciphered.txt -K 00112233445566778899aabbccddeeff
hex string is too short, padding with zero bytes to length
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3$ cat ciphered.
txt
h3w_  *g U FR r
e  wJNK' c c  B f E    =s !s   `dRgG  =a Q 5(  ^js 5  _ `c,_g yF Y  E1 d22Y
                                                                                &&  y  8` |t    9>Bge  j
,G   }  E@<  1  [>  "8   V   v     |  m H3 x z
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3$ hexdump ciphe
red.txt
0000000 68a2 7733 a0dd 9414 2abc 67f4 55b5 46f3
0000010 5213 72d1 ba86 e3a9 cceb f70a 1875 4635
0000020 c3d0 8602 03a1 fd50 7f14 0d30 c40d d61a
0000030 8665 1584 77c1 86ca c64e 2798 63a3 63f1
0000040 debe aa91 42b6 669b de92 c745 c8f6 e100
0000050 3d02 73a1 21d3 73fa a3fc 60dd 5264 4767
0000060 d5c0 dbb2 df3d c9a5 ff51 2835 dbea 1e5e
0000070 6a00 dd73 f535 5f1a 60aa 2c63 675f 798f
0000080 fb46 9d59 45a1 fb31 3264 5932 0c00 2698
0000090 dba0 7988 8eb6 6038 7ca1 fc74 98c1 39b4
00000a0 3e11 6742 f865 6a8b 1ad9 b304 efe6 9896
00000b0 472c d4b8 7ddd cfa8 4045 d33c a731 5bcb
00000c0 9d3e 3822 99b4 5681 b78c 76b1 7c09 a1ac
00000d0 ee6d a8ac 3348 78ad 07a9 0c7a 2dde 3291
00000e0
```

7. Encryption using SEED-CBC

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3$ openssl enc -
SEED-cbc -e -in ameya.txt -out ciphered.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3$ cat ciphered.
txt
$]*@-GN
v Vo 4= r F f{ <      t^ *A O   )K I~m. ]'C@k
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3$ hexdump ciphe
red.txt
0000000 6901 83d7 2d30 092f 4aaf 4457 de82 e2e9
0000010 4e47 b0fc dc8f 0dad 5d24 2ae8 0fb8 b201
0000020 f8e7 40c5 1f80 df2d 3b0b 9259 a6bb d895
0000030 760d 1b97 87f5 6f56 d8c0 3d34 18c7 9172
0000040 46d5 9c1d 7b66 3ce1 0ecb a49a adb3 01cf
0000050 5e74 f991 ad2a a641 c04f f48b 4b29 498f
0000060 6d7e ac2e c2f7 275d 4317 6b40 ee0a d21e
0000070 85ee 1e2e c928 6ab8 723e af10 afda 5faa
0000080 762f d099 2988 b105 ab2f f4f3 b28b c222
0000090 76d8 9291 1955 4466 f1a8 f500 ad2c 4a81
00000a0 ea8e 2803 83f8 70e3 cdb2 9711 4ead a551
00000b0 b190 68b0 8f15 98b7 3bf9 4d90 4453 22dd
00000c0 7520 4fca 5221 6227 ca63 888a b6d9 f2b1
00000d0 47b0 9556 45a5 0965 18fb 40eb d2ee 8347
00000e0
```

8. Encryption using SEED-CFB

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3$ openssl enc -
SEED-cfb -e -in ameya.txt -out ciphered.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3$ cat ciphered.
txt
|[illegible]RO[illegible]x[illegible]+[illegible]\[illegible],[illegible]F[illegible]y[illegible]cG[illegible]][illegible]E[illegible]
P[illegible]t*[illegible]+^[illegible]0r[illegible]/[illegible]s[illegible]bLeQ[illegible]/[illegible]VA[illegible]>[illegible]0[illegible]DØC\[illegible]=[illegible]A[illegible]5[illegible]0ic;X\[illegible]ë[illegible]1[illegible]B[illegible]cM[illegible]D[illegible]x[illegible]z[illegible]t[illegible]j[illegible]h[illegible],C2[illegible]@U[illegible]d[illegible]oa[illegible]1[illegible]_[illegible]a}
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3$ hexdump ciphe
red.txt
0000000 997c abd1 5218 fa4f e0f2 ec83 a108 8cf4
0000010 9e83 78ec a8ea e42b 5ca6 c4bf e22c f1a3
0000020 e246 e279 4763 d0a2 845d 458e 19a2 b4bf
0000030 0cfb b8a2 fb29 3677 5be9 b090 f202 0dcb
0000040 b350 2a74 c78a 5e2b 85a3 1b30 7427 f972
0000050 192f 738a 19bf d1a0 dfb0 4c95 5165 8da7
0000060 a22f 4156 c0f2 3ea5 308e e0dc 44b5 98c3
0000070 87d5 f25c d23d 9e41 9cb7 a335 6930 3b63
0000080 0458 a15c f282 02af d184 f491 ab1a a431
0000090 da42 8ae7 0063 b04d a99c e2ba 0ee2 10c9
00000a0 8a44 7814 7a9e da07 b774 b46a 88da 17b8
00000b0 9068 192c d743 32aa ea94 97d6 5540 fc1e
00000c0 64ec 6fcb 7f61 31fc 1fcd 5ffa 618e 697d
00000d0 b4cf 785a cd06 280f b247 1cbd
00000dc
```

9. Encryption using SEED-ECB

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3$ openssl enc -
SEED-ecb -e -in ameya.txt -out ciphered.txt -K 00112233445566778899aabbccddeeff
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3$ cat ciphered.
txt
[illegible]YP. [illegible]qZ[illegible]Cq[illegible]y[illegible]W[illegible]{[illegible]]![illegible]#& jS[illegible]*2Ca[illegible]Bk̄_[illegible]ţ[illegible]F[illegible]R[illegible]
                                                                a[illegible]*[illegible]|b[illegible]u[illegible]E┼[illegible]P[illegible]s9f[illegible]ēm[illegible]    >[illegible]+][illegible]5C*[illegible]s[illegible]![illegible]f
N[illegible]B[illegible]&[illegible]s[illegible]=z[illegible]X[illegible]f%Z[illegible]ÁI[illegible]L[illegible][\_[illegible]\ '     [illegible]A*[illegible]P[illegible]1[illegible]L<_'[illegible]mH[illegible]'
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3$ hexdump ciphe
red.txt
0000000 c599 59ca 0150 202e fd05 0671 8b5a 8eaf
0000010 cf98 17be e371 79eb 1de8 f1af ae57 937b
0000020 a89c 215d c7a9 23df 2026 536a 15a4 2a8e
0000030 3216 6143 4285 136b 84cc 915f a17f d2b7
0000040 d7ad c0fe ee46 9252 0b84 a461 c82a f9d2
0000050 a089 627c eae6 fa1d 75d0 e5f2 db8a 457f
0000060 aecf 5088 c4f8 73cd 6639 9be2 93c4 016d
0000070 09a1 803e ccf5 19f0 5d2b 1191 4335 2a01
0000080 7387 08ae f39b 0421 6697 df4e a342 dc26
0000090 d273 7a3d f3e1 928d a358 f666 2508 7f1d
00000a0 bd5a a786 81c3 9d49 4ca5 fa5b 5f5c cb9b
00000b0 5cdf 2720 ba09 2a41 cb91 9850 05d3 18df
00000c0 31c5 4cd0 5f3c b9ca 1fd5 6dda a748 27fd
00000d0 420a eb0d 3169 855f 19e3 84ed 296e 893e
00000e0
```

I researched about the above algorithms and what I got to know from them :

AES: AES is a symmetric block cipher which uses 128 bit data block with many variations like 128 bits/256 bits key variations and was found to be a replacement for DES since the key in DES was too small and also was bruteforce-able.

SEED Block Cipher: SEED is a block cipher developed by the Korea Internet & Security Agency (KISA) as the 40 bit encryption used earlier was not considered strong enough. SEED is a 128-bit symmetric key block cipher and is robust against known attacks like Differential Cryptanalysis, Linear Cryptanalysis and other key attacks the DES is susceptible to.

Also, I researched a bit and found out that AES was more performant than SEED in terms of speed and security (when we consider the 256 bit key version).

3.2 Task 2: Encryption Mode – ECB vs. CBC

The file pic original.bmp contains a simple picture. We would like to encrypt this picture, so people without the encryption keys cannot know what is in the picture. Please encrypt the file using the ECB (Electronic Code Book) and CBC (Cipher Block Chaining) modes, and then do the following:

1. Let us treat the encrypted picture as a picture, and use a picture viewing software to display it. How ever, For the .bmp file, the first 54 bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate .bmp file. We will replace the header of the encrypted picture with that of the original picture. You can use the ghex tool to directly modify binary files.
2. Display the encrypted picture using any picture viewing software. Can you derive any useful infor-mation about the original picture from the encrypted picture? Please explain your observations.

Image used:

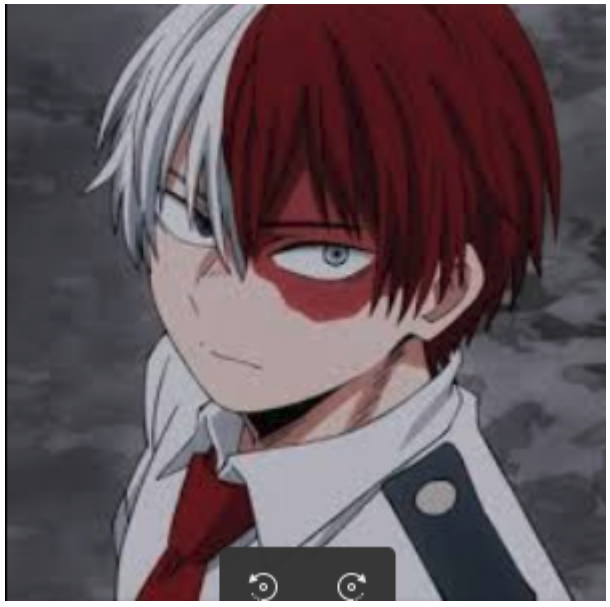


Image size

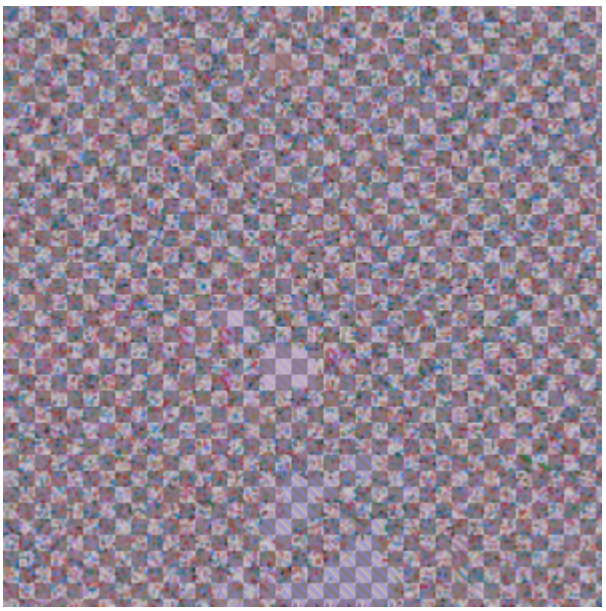
```
-rw-rw-r-- 1 ameya ameya  202638 Nov  8 14:24 images.bmp
```

How to encrypt the image with ECB mode

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISS
IONS/SEM 5/css exp 3$ head -c 54 images.bmp > ecb.bmp
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISS
IONS/SEM 5/css exp 3$ openssl enc -aes-128-ecb -e -in images.bmp -out temp.bmp
-K 00112233445566778899aabbccddeeff
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISS
IONS/SEM 5/css exp 3$ tail -c 202584 temp.bmp>>ecb.bmp
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISS
IONS/SEM 5/css exp 3$
```

- The head command will grab the first 54 bytes (which contain the bitmap image header which is required to recognise the output as a legitimate bmp file) to ecb.bmp
- Now we will run our encryption with the help of openssl on the image and store the result in temp.bmp
- Now, since the header of temp.bmp is also encrypted, it will not be recognised as a bmp file anymore
- So we will look at the size of the images.bmp (original image) and grab all the bytes from the end of the file of temp.bmp (the tail command helps us to achieve that) except the first 54 bytes and transfer that to ecb.bmp file.

ecb.bmp

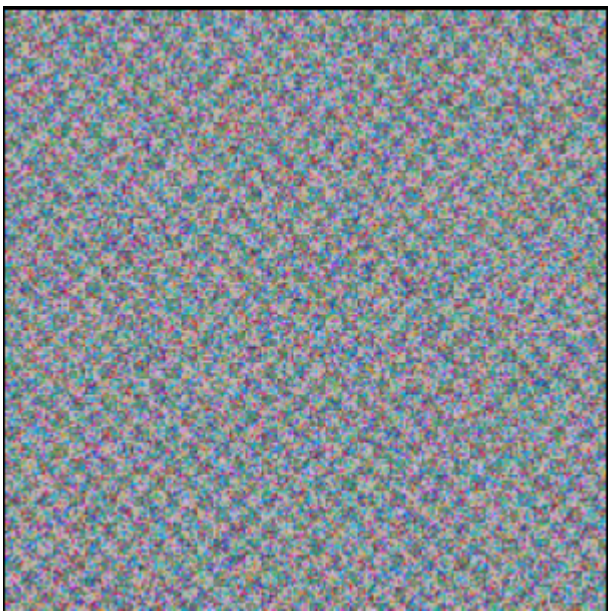


How to encrypt the image with CBC mode

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx: ~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISS
IONS/SEM 5/css exp 3$ head -c 54 images.bmp > cbc.bmp
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISS
IONS/SEM 5/css exp 3$ openssl enc -aes-128-cbc -e -in images.bmp -out temp.bmp
-K 00112233445566778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISS
IONS/SEM 5/css exp 3$ tail -c 202584 temp.bmp>>cbc.bmp
```

- The head command will grab the first 54 bytes (which contain the bitmap image header which is required to recognise the output as a legitimate bmp file) to cbc.bmp
- Now we will run our encryption with the help of openssl on the image and store the result in temp.bmp
- Now, since the header of temp.bmp is also encrypted, it will not be recognised as a bmp file anymore
- So we will look at the size of the images.bmp (original image) and grab all the bytes from the end of the file of temp.bmp (the tail command helps us to achieve that) except the first 54 bytes and transfer that to the cbc.bmp file.

cbc.bmp



Observations:

1. In ECB, we can actually see some of the outlines of the character in the image , but they are too faint and also it has a reddish tint on the image

2. In CBC, none of the elements of the image are visible, and we cannot figure out what the image was in the first place.
3. If a comparison has to be made between the two, then CBC seems to be better as ECB retains some properties of the image (like the outlines of the image in this case) whereas in CBC, we cannot make any guess of what the image will be at all.
4. I tried to understand why this is the case, and I found out that The problem with ECB is that it is basically the same as splitting up the plaintext into blocks and encrypting them completely independent of each other. When the block cipher is considered secure then the resulting ciphertext of a single block is of course also secure - when analyzed independently. However when multiple messages or or messages are encrypted then ECB is not secure because identical plaintext blocks will result in identical ciphertext. This means that an attacker can easily detect repetition.
5. CBC solves this by introducing a vector, which alters the first plaintext block before it is encrypted. The effect is that the entire ciphertext is randomized, as the ciphertext created by that block is used as a vector for the next block (and so forth). In other words: it ensures that each block is encrypted differently even if the block contains the same information.

3.3 Task 3: Encryption Mode – Corrupted Cipher Text

To understand the properties of various encryption modes, we would like to do the following

exercise: 1. Create a text file that is at least 64 bytes long.

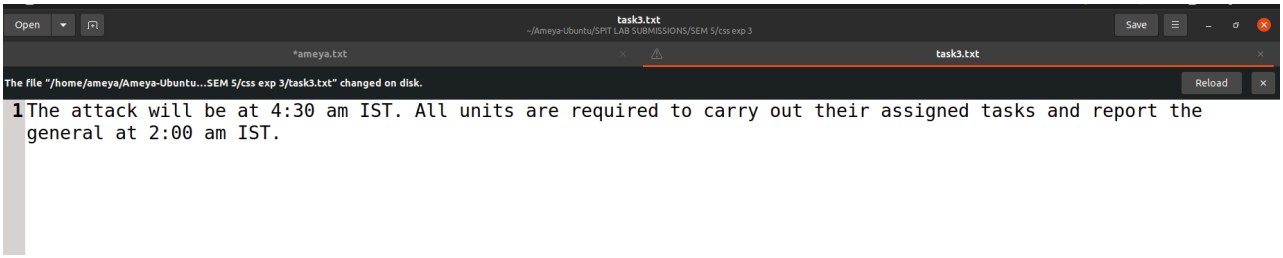
2. Encrypt the file using the AES-128 cipher.

3. Unfortunately, a single bit of the 30th byte in the encrypted file got corrupted. You can achieve this corruption using ghex.

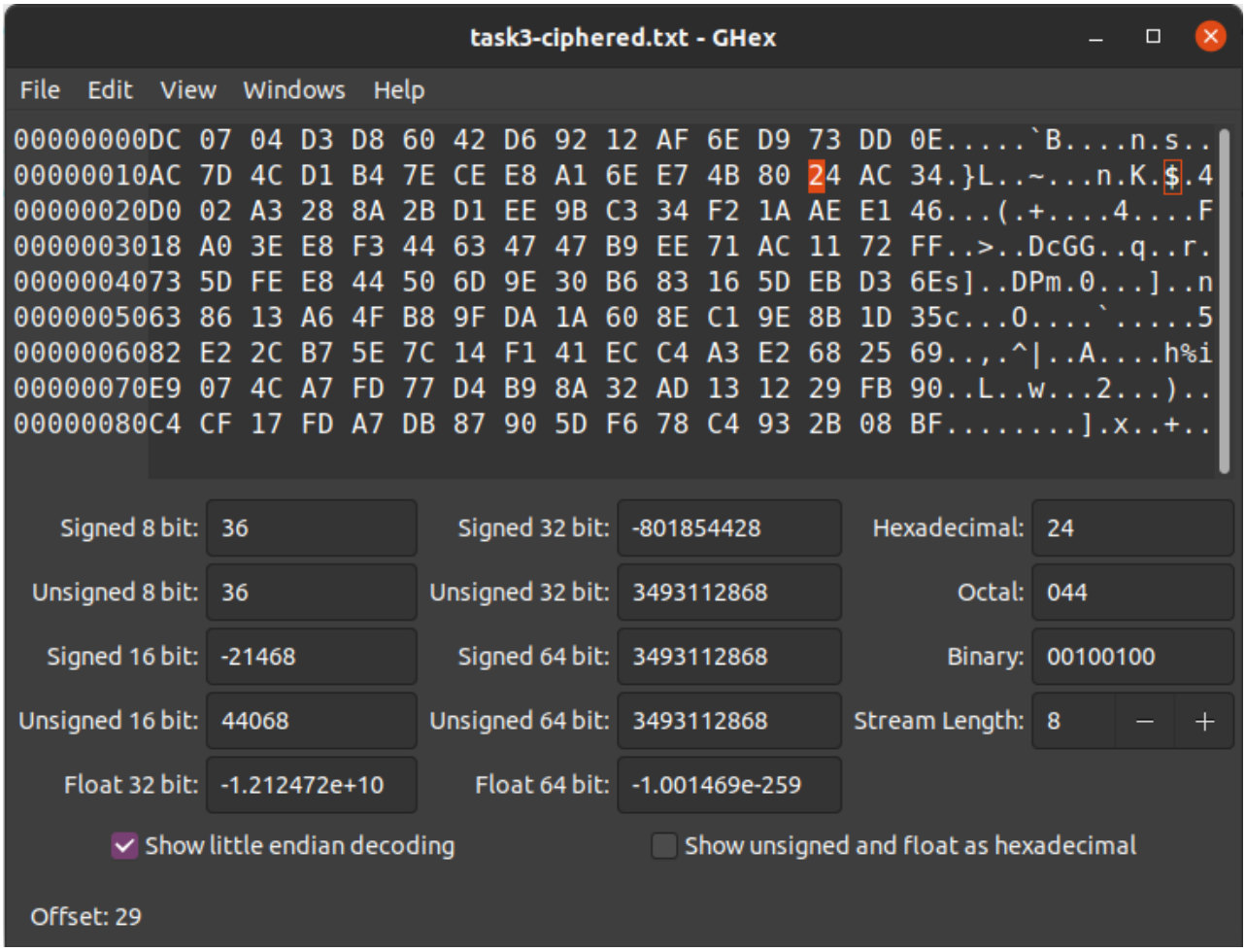
4. Decrypt the corrupted file (encrypted) using the correct key and IV.

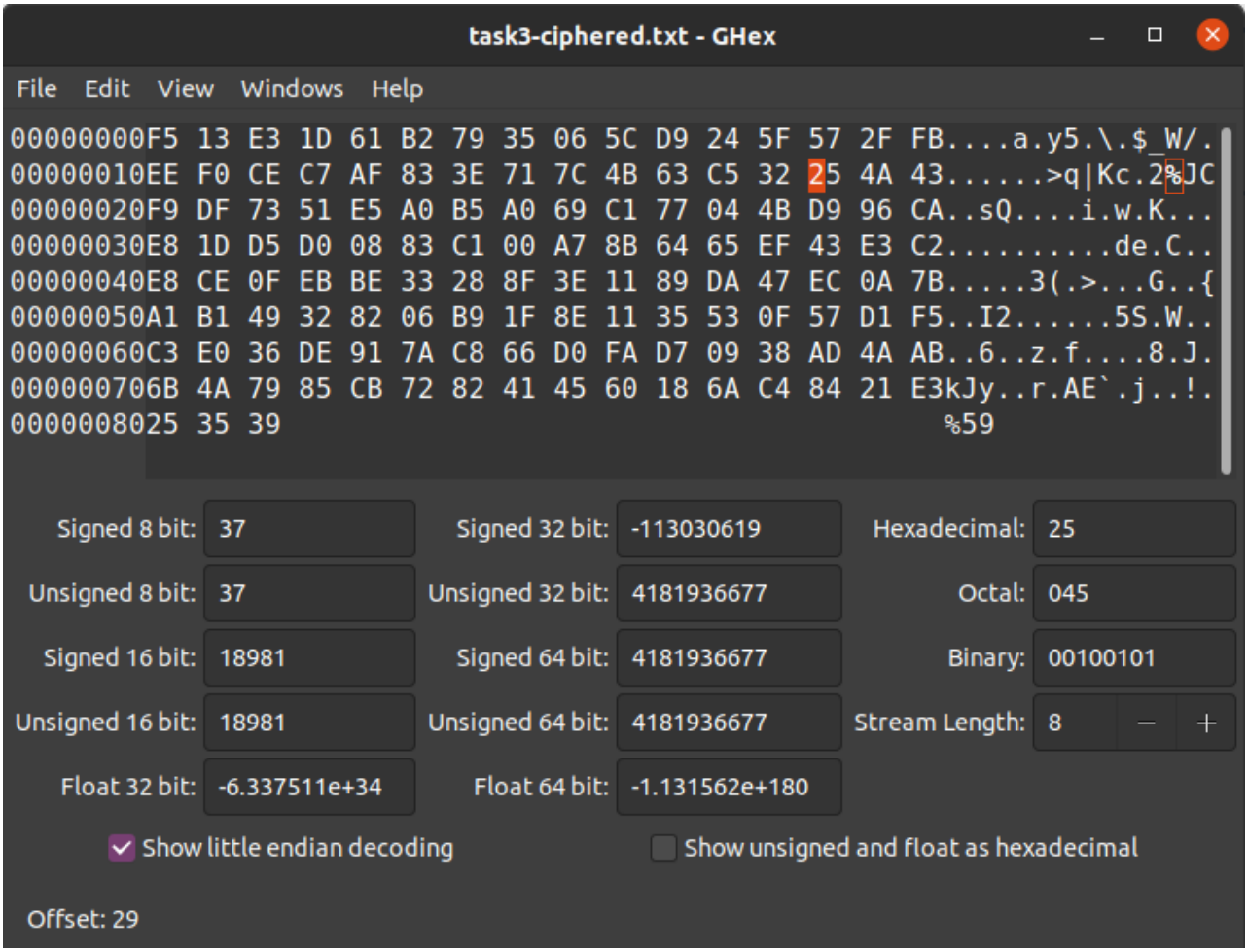
Please answer the following questions: (1) How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you finish this task. (2) Please explain why. (3) What are the implication of these differences?

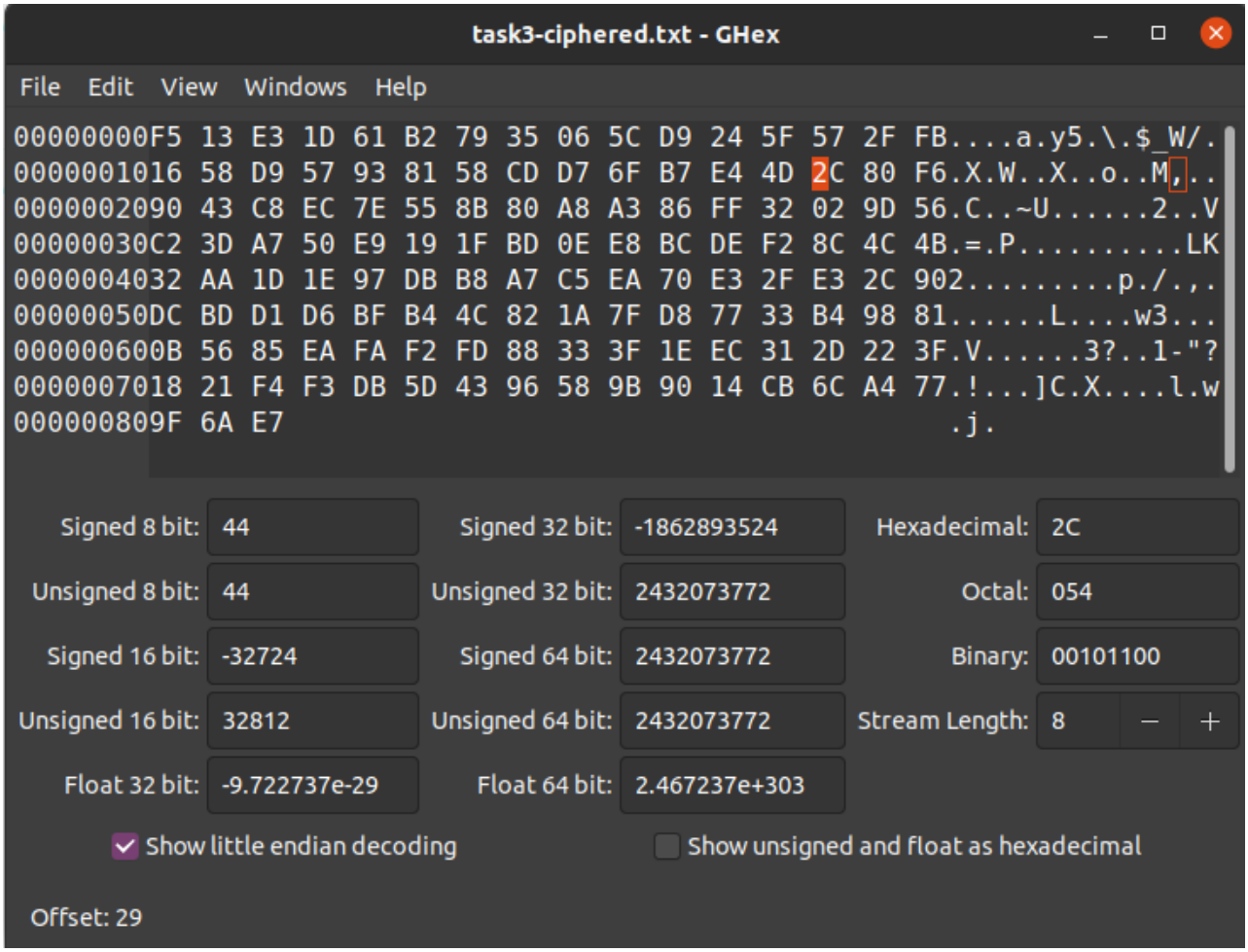
This is the plaintext we will be using:

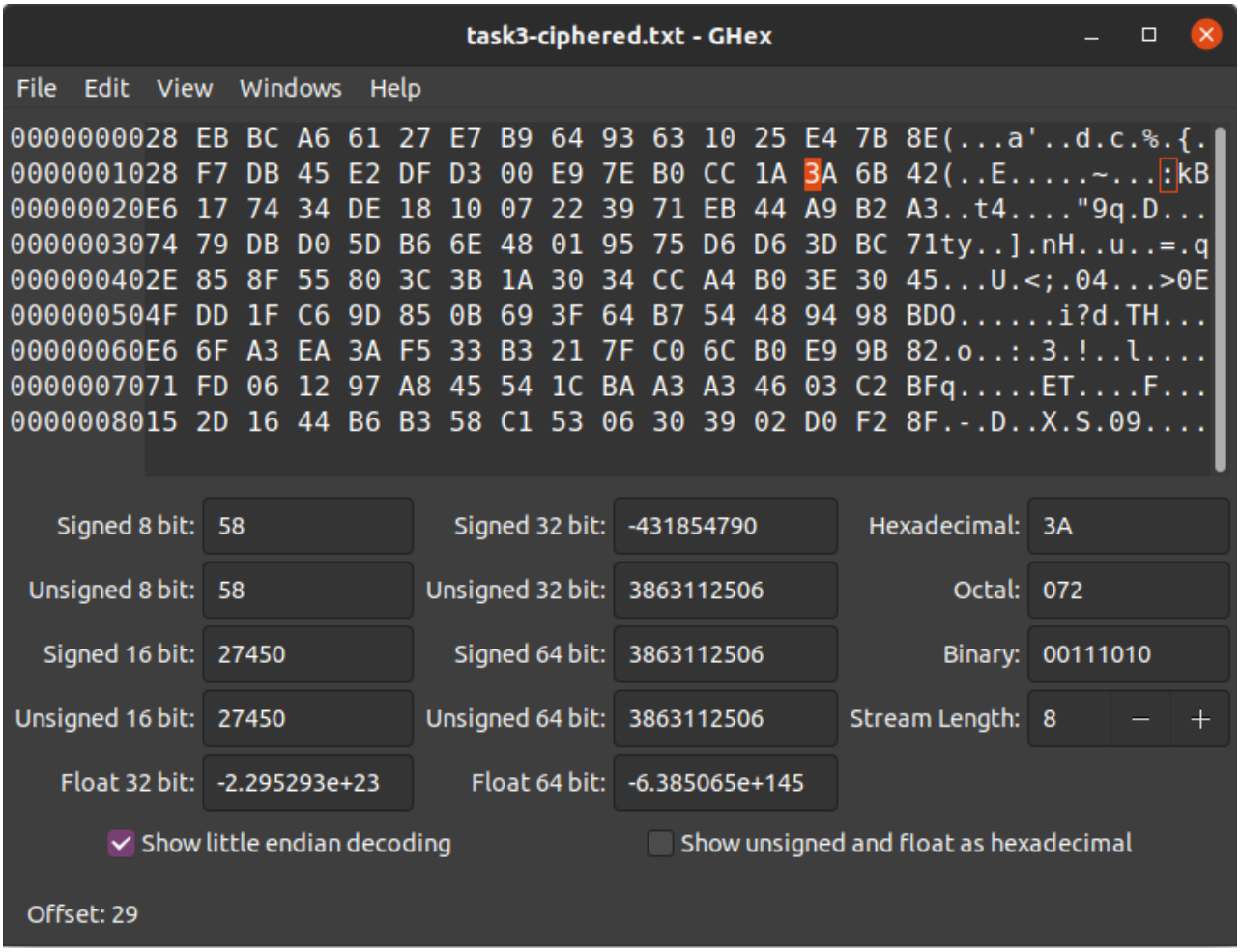


The above text is encrypted using aes-128 in 4 different modes: cbc,cfb,ofb and ecb. After corrupting 30 th byte of each cbc, cfb, ofb and ecb:



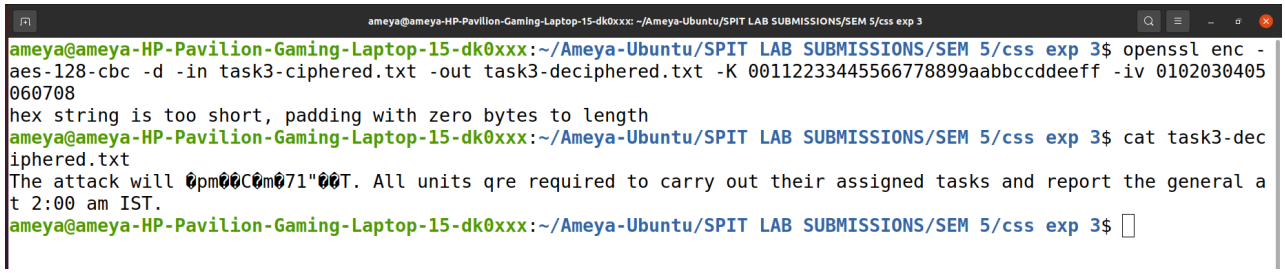

task3-ciphered.txt - GHex
File Edit View Windows Help
00000000F5 13 E3 1D 61 B2 79 35 06 5C D9 24 5F 57 2F FB....a.y5.\.$_W/.
00000010EE F0 CE C7 AF 83 3E 71 7C 4B 63 C5 32 25 4A 43......>q|Kc.2%JC
00000020F9 DF 73 51 E5 A0 B5 A0 69 C1 77 04 4B D9 96 CA..sQ....i.w.K...
00000030E8 1D D5 D0 08 83 C1 00 A7 8B 64 65 EF 43 E3 C2..........de.C..
00000040E8 CE 0F EB BE 33 28 8F 3E 11 89 DA 47 EC 0A 7B.....3(.>...G..{
00000050A1 B1 49 32 82 06 B9 1F 8E 11 35 53 0F 57 D1 F5..I2......5S.W..
00000060C3 E0 36 DE 91 7A C8 66 D0 FA D7 09 38 AD 4A AB..6..z.f....8.J.
000000706B 4A 79 85 CB 72 82 41 45 60 18 6A C4 84 21 E3kJy..r.AE`.j..!.
0000008025 35 39 %59
Signed 8 bit: 37
Signed 32 bit: -113030619
Hexadecimal: 25
Unsigned 8 bit: 37
Unsigned 32 bit: 4181936677
Octal: 045
Signed 16 bit: 18981
Signed 64 bit: 4181936677
Binary: 00100101
Unsigned 16 bit: 18981
Unsigned 64 bit: 4181936677
Stream Length: 8
Float 32 bit: -6.337511e+34
Float 64 bit: -1.131562e+180
Show little endian decoding
Show unsigned and float as hexadecimal
Offset: 29


task3-ciphered.txt - GHex
File Edit View Windows Help
00000000F5 13 E3 1D 61 B2 79 35 06 5C D9 24 5F 57 2F FB....a.y5.\.$_W/.
0000001016 58 D9 57 93 81 58 CD D7 6F B7 E4 4D 2C 80 F6.X.W..X..o..M,..
0000002090 43 C8 EC 7E 55 8B 80 A8 A3 86 FF 32 02 9D 56.C..~U......2..V
00000030C2 3D A7 50 E9 19 1F BD 0E E8 BC DE F2 8C 4C 4B.=.P.........LK
0000004032 AA 1D 1E 97 DB B8 A7 C5 EA 70 E3 2F E3 2C 902.........p./.,.
00000050DC BD D1 D6 BF B4 4C 82 1A 7F D8 77 33 B4 98 81......L....w3...
000000600B 56 85 EA FA F2 FD 88 33 3F 1E EC 31 2D 22 3F.V......3?..1-"?
0000007018 21 F4 F3 DB 5D 43 96 58 9B 90 14 CB 6C A4 77.!...]C.X....l.w
000000809F 6A E7 .j.
Signed 8 bit: 44
Signed 32 bit: -1862893524
Hexadecimal: 2C
Unsigned 8 bit: 44
Unsigned 32 bit: 2432073772
Octal: 054
Signed 16 bit: -32724
Signed 64 bit: 2432073772
Binary: 00101100
Unsigned 16 bit: 32812
Unsigned 64 bit: 2432073772
Stream Length: 8
Float 32 bit: -9.722737e-29
Float 64 bit: 2.467237e+303
Show little endian decoding
Show unsigned and float as hexadecimal
Offset: 29

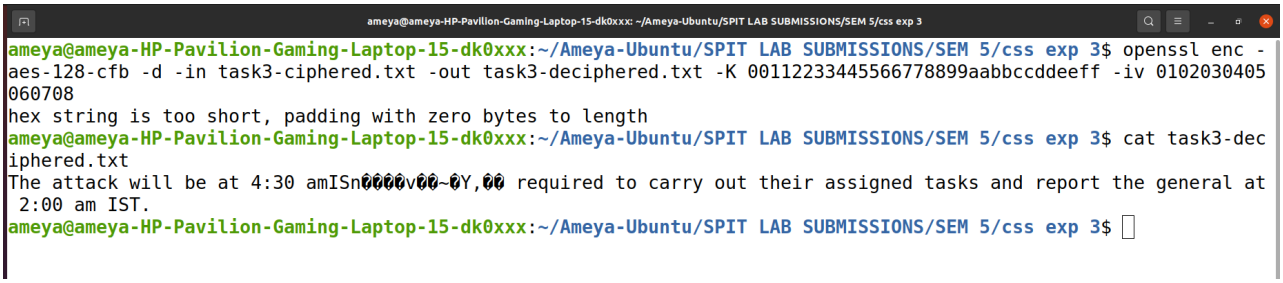


Decrypted Texts:

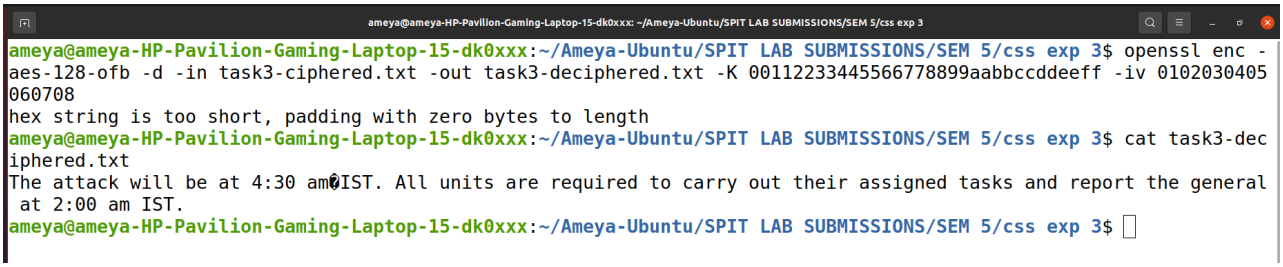
CBC:



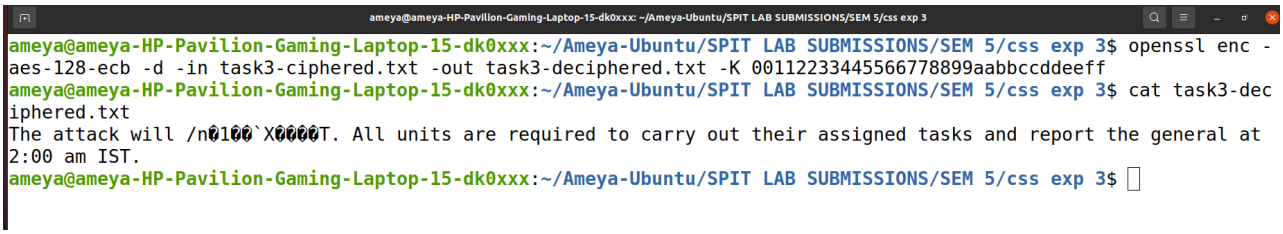
CFB:



OFB:



ECB:



From the above results, I made the following observations:

In **CBC mode**, there was an effect in two blocks,since the results of the decryption depend on the results of XOR of the previous cipher blocks.

In **CFB mode**, there is a problem in n / r number of blocks where r = number of bits taken at a time to XOR.

In **OFB mode**, the single digit of the 30th byte is corrupted, then in plain text only that byte or character is corrupted. Thus, only OFB mode shows the most promising result and almost all the texts are recovered.

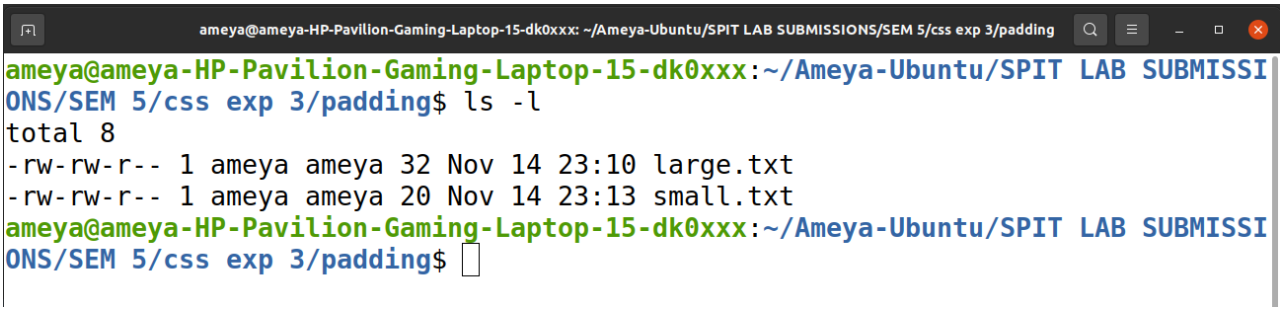
In **ECB mode**, only one block is affected when any problem in a ciphertext happens.Each block is decrypted independently. However, the corrupted bit of the 30th byte in ciphertext block might spread to all n bits in the plaintext block since we do the decryption one block at a time.

3.4 Task4 : Padding

For block ciphers, when the size of the plaintext is not the multiple of the block size, padding may be required. In this task, we will study the padding schemes. Please do the following exercises:

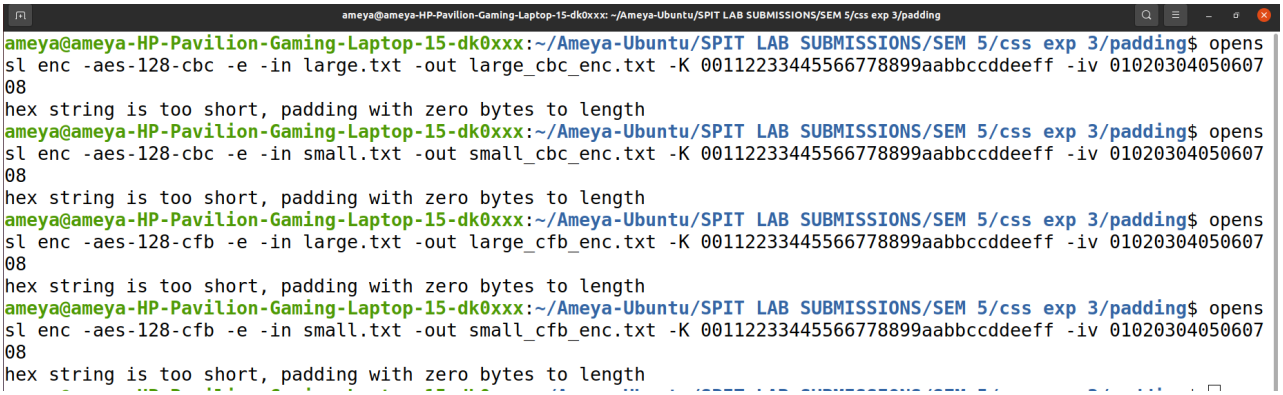
1. The openssl manual says that openssl uses PKCS5 standard for its padding. Please design an experiment to verify this. In particular, use your experiment to figure out the paddings in the AES encryption when the length of the plaintext is 20 octets and 32 octets.

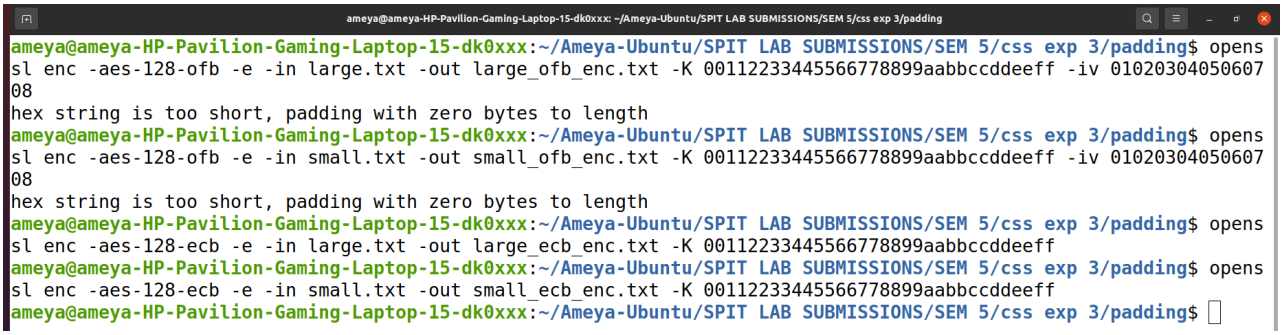
2. Please use ECB, CBC, CFB, and OFB modes to encrypt a file (you can pick any cipher). Please report which modes have paddings and which ones do not. For those that do not need paddings, please explain why.



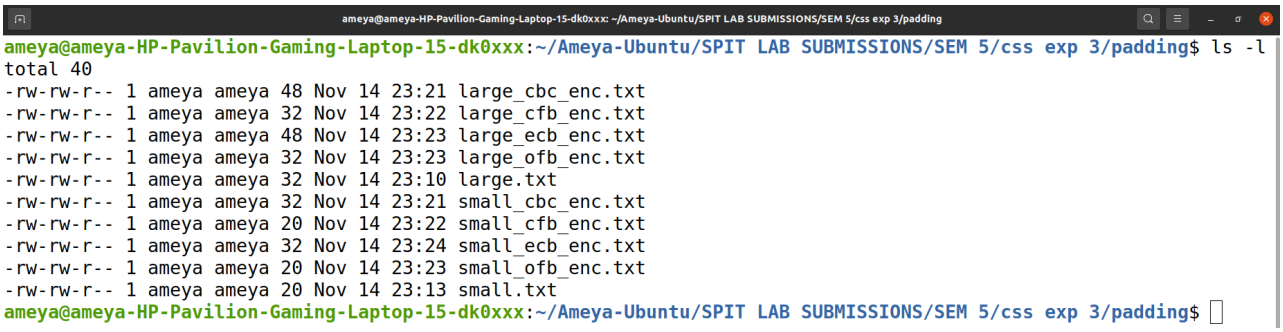
I have created two files namely large.txt and small.txt with 32 bytes and 20 bytes respectively.

The files are encrypted using aes-128 in 4 modes namely: cbc, cfb, ofb and ecb

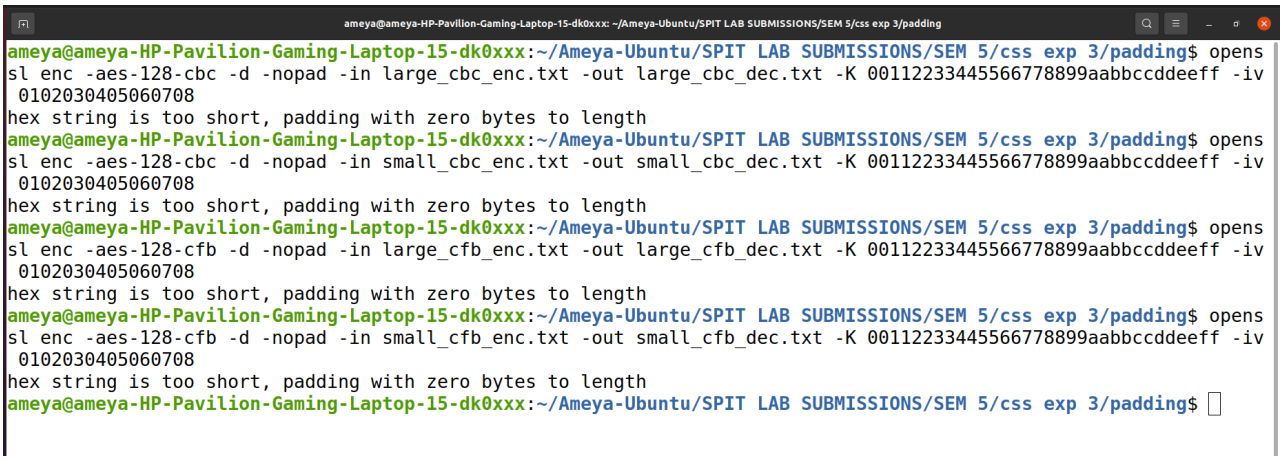


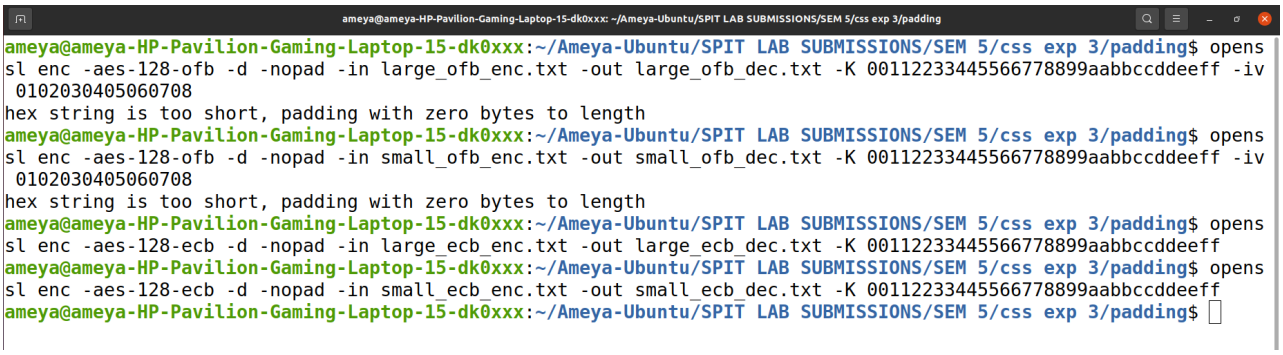


Let's see the file sizes of all the generated files and the original files together.



To confirm that openssl uses PKCS5 padding, decrypt the encrypted file with option–nopad. This option turns off the standard block padding. Normally, the padding is included by default during encryption.





Let's look at the file sizes of decrypted,encrypted and original files together

```
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3/padding$ ls -l
total 72
-rw-rw-r-- 1 ameya ameya 48 Nov 14 23:30 large_cbc_dec.txt
-rw-rw-r-- 1 ameya ameya 48 Nov 14 23:21 large_cbc_enc.txt
-rw-rw-r-- 1 ameya ameya 32 Nov 14 23:31 large_cfb_dec.txt
-rw-rw-r-- 1 ameya ameya 32 Nov 14 23:22 large_cfb_enc.txt
-rw-rw-r-- 1 ameya ameya 48 Nov 14 23:33 large_ecb_dec.txt
-rw-rw-r-- 1 ameya ameya 48 Nov 14 23:23 large_ecb_enc.txt
-rw-rw-r-- 1 ameya ameya 32 Nov 14 23:32 large_ofb_dec.txt
-rw-rw-r-- 1 ameya ameya 32 Nov 14 23:23 large_ofb_enc.txt
-rw-rw-r-- 1 ameya ameya 32 Nov 14 23:10 large.txt
-rw-rw-r-- 1 ameya ameya 32 Nov 14 23:30 small_cbc_dec.txt
-rw-rw-r-- 1 ameya ameya 32 Nov 14 23:21 small_cbc_enc.txt
-rw-rw-r-- 1 ameya ameya 20 Nov 14 23:31 small_cfb_dec.txt
-rw-rw-r-- 1 ameya ameya 20 Nov 14 23:22 small_cfb_enc.txt
-rw-rw-r-- 1 ameya ameya 32 Nov 14 23:33 small_ecb_dec.txt
-rw-rw-r-- 1 ameya ameya 32 Nov 14 23:24 small_ecb_enc.txt
-rw-rw-r-- 1 ameya ameya 20 Nov 14 23:32 small_ofb_dec.txt
-rw-rw-r-- 1 ameya ameya 20 Nov 14 23:23 small_ofb_enc.txt
-rw-rw-r-- 1 ameya ameya 20 Nov 14 23:13 small.txt
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3/padding$
```

The screenshot above shows that the size of CBC and ECB encrypted files with the nopad option is 12 bytes more for the 20 bytes file and 16 bytes larger for the 32 bytes file, however the size of OFB and CFB decrypted files is the same.

Result:

1. The experiment shows that padding is needed for ECB and CBC encryption modes. This can be because ECB and CBC are block ciphers and for a block cipher length of input must be an exact multiple of block length. If this is not the case then padding must be added to make it so. This padding is removed after decrypting.

2. In OFB and CFB, the padding is not required because they are stream ciphers and the ciphertext is always the same length as plain text.

3.5 Task 5: Programming using the Crypto Library

So far, we have learned how to use the tools provided by openssl to encrypt and decrypt messages. In this task, we will learn how to use openssl's crypto library to encrypt/decrypt messages in programs.

OpenSSL provides an API called EVP, which is a high-level interface to cryptographic functions. Although OpenSSL also has direct interfaces for each individual encryption algorithm, the EVP library provides a common interface for various encryption algorithms. To ask EVP to use a specific algorithm, we simply need to pass our choice to the EVP interface. A sample code is given in http://www.openssl.org/docs/crypto/EVP_EncryptInit.html. Please get yourself familiar with this program, and then do the following exercise.

You are given a plaintext and a ciphertext, and you know that aes-128-cbc is used to generate the ciphertext

from the plaintext, and you also know that the numbers in the IV are all zeros (not the ASCII character '0'). Another clue that you have learned is that the key used to encrypt this plaintext is an English word shorter than 16 characters; the word that can be found from a typical English dictionary. Since the word has less than 16 characters (i.e. 128 bits), space characters (hexadecimal value 0x20) are appended to the end of the word to form a key of 128 bits. Your goal is to write a program to find out this key. You can download a English word list from the Internet. We have also linked one on the web page of this lab. The plaintext and ciphertext is in the following:

Plaintext (total 21 characters): This is a top secret. Ciphertext (in hex format):
8d20e5056a8d24d0462ce74e4904c1b5
13e10d1df4a2ef2ad4540fae1ca0aaf9

Note 1: If you choose to store the plaintext message in a file, and feed the file to your program, you need to check whether the file length is 21. Some editors may add a special character to the end of the file. If that happens, you can use the ghex tool to remove the special character.

Note 2: In this task, you are supposed to write your own program to invoke the crypto library. No credit will be given if you simply use the openssl commands to do this task.

Note 3: To compile your code, you may need to include the header files in openssl, and link to openssl libraries. To do that, you need to tell your compiler where those files are. In your Makefile, you may want to specify the following:

Crypto Encryption/PV

```
INC=/usr/local/ssl/include/
LIB=/usr/local/ssl/lib/

all:
        gcc -I$(INC) -L$(LIB) -o enc yourcode.c -lcrypto -ldl
```

Code:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad
plain_text = b"This is a top secret."
cipher_hex = "8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9"
result_key = []
with open('words.txt', 'r') as file:
   words = list(map(str.strip, file.readlines()))
for word in words:
   if len(word) >= 16:
       continue
   word = word.lower()
   key = word.encode() + b' '*(16-len(word))
   cipher = AES.new(key, AES.MODE_CBC, iv=bytes.fromhex('0'*32))
   ciphertext = cipher.encrypt(pad(plain_text, AES.block_size))
```

```
    match_result = "NOT MATCHED"
    if bytes.hex(ciphertext) == cipher_hex:
        match_result = "MATCHED"
        result_key.append(word)
    print(word, bytes.hex(ciphertext), match_result)
print("\n\nResulting Key:",result_key)
```

Output:

```
zuurveldt 7c76dcc12c049ed0ea34aaffbd02f6963fa8117c1382c99784205aa1e71b0d22 NOT MATCHED
zuza 49259e25ddfdeac0b070282f10d98b25bbdfc97080b1d85a3c6bf1e40d440a08 NOT MATCHED
zuzana 4b317a3a18dd0c2ac63191e6ae80397a1507f6f861a0e9e342930278d26b24bc NOT MATCHED
zu-zu 04eb3d47bcc2fce2df97928758cd4182f2fdbc4866ffd4ed6227c80ad861f8ef NOT MATCHED
zwanziger 70b52532147330fb9b718d64b0cd144300ad9e4fe5ab875cef8ac74d587621ff NOT MATCHED
zwart 7b8693c89e594c35fec42bd33ec5f6a318e4c6096c8f73b0ba65346d7ee79627 NOT MATCHED
zwei 965faf9245114fbd64c0ac1bc2f5213f69e12199a7ad6dea685b56d62ac5acb7 NOT MATCHED
zweig 43f25db77ff3a0363dff7617755855365cf2ff071b50729b97cc6ebf3e5618cc NOT MATCHED
zwick 96ea898d70c7212bb5d0ff7735c55bcc318cfd3efd71a5bde00f25c55e6501b6 NOT MATCHED
zwickau 7c1b8bfb63bfd5b6675224556182a5099835623f2f8debe2ec604f9d82cbd8e4 NOT MATCHED
zwicky c6c25a3e4cafbdedb440529f89f3f18ac2521d0b4e44afee0406173430b61f57 NOT MATCHED
zwieback 31b67ea013c06813e26e0466ef22aac0d0742615c3a8127e438e87b4b20c5464 NOT MATCHED
zwiebacks cf4df730792dbc47b427c4aee046a20f6f3362cc657e8ceb9515c1f61c34a8a7 NOT MATCHED
zwiebel 080d15f65c1fb580a38e0f99e565869cc89fbb3773d4ad0f2e05422ab31c41ee NOT MATCHED
zwieselite 2f2342c4e1c99673b54aca30655a35d89767864c3f5e9718f90365d92360dfa0 NOT MATCHED
zwingle c99210ecf7014a9a948d5270ae6d3cd99fdafbc8cf5a68ca2b81877bea74f39b NOT MATCHED
zwingli b05d6713ea431a27239a46d191cadc9d5521c2ae25ec2596a8bc1dbf2bfab102 NOT MATCHED
zwinglian 5f2b18d9f93d7fa27fcca4c27581a2c272e864cf25a1e71652b86979ee6b5bac NOT MATCHED
zwinglianism aedf7feabae8a15a22334d5cd43844b60f946f0382d13a8acd60b84dd80348b0 NOT MATCHED
zwinglianist 17158ad1efe7d3e6fcd751fc2d801019d22b3f64a37e2ddf68ae9d877cf8a084 NOT MATCHED
zwitter 6df12d857a5b3461a67a6f4548250d705407c73c48cd5242f226ef38077d18b9 NOT MATCHED
zwitterion 0b794ab2da476a76aeab642ad9fe582dfd79cc0042113790d30adef2da8b51b5 NOT MATCHED
zwitterionic 43d9a3b874200bc70e02e6b0143c3e9495c296487814a3c4d4c8747d630c7de3 NOT MATCHED
zwolle 327589a13f6940f1b17e3f3e85826724e8b2bb8cebcecf4c151e7f91e625c635 NOT MATCHED
zworykin 770797ec4f4bd24b34e54ad3f7e958c05cc673631b47623b372e4473519e1305 NOT MATCHED
zz 5d7a15c6b07909667c79feac6346d16c02204f1c63389721179c2015cb21c2bb NOT MATCHED
zzt 58dae019524c7d7d79ceaa3a43186ce4a5f64d9ad1817ac4b06a649500de2ef8 NOT MATCHED
zzz 2bfc0c1cb86d489616d5eaf811512b493869e2b14dfe420eaba6a4a4f431e94b NOT MATCHED


Resulting Key: ['median']
ameya@ameya-HP-Pavilion-Gaming-Laptop-15-dk0xxx:~/Ameya-Ubuntu/SPIT LAB SUBMISSIONS/SEM 5/css exp 3/task-5$
```

Final Conclusion from all Tasks:

1. AES, SEED are symmetric key algorithms using the same keys to encrypt and decrypt the data.
2. ECB mode of encryption is the weakest form of encryption in comparison to CBC, CFB and OFB.
3. I learned how different modes react to a corrupted bit of a cipher text. The best decryption in such a case is provided by OFB where only the corrupted bit of cipher text is affected while encrypting.
4. I could conclude from the experiment that ECB and CBC use padding while encryption while the other two don't. This proves that ECB and CBC are block ciphers while CFB and OFB are stream ciphers.
5. I could conclude from this experiment that , If I have the plaintext, ciphertext and the initialisation vector, I can easily find the key using brute force method