

# LABORATORY CEL62: Cryptography and System Security Winter 2021

<b>Experiment 3:</b>	<b>Crypto Encryption</b>
----------------------	--------------------------

Name: Divya Shah  
Roll No: 2019130057  
TE Comps

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.

# 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 installed 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

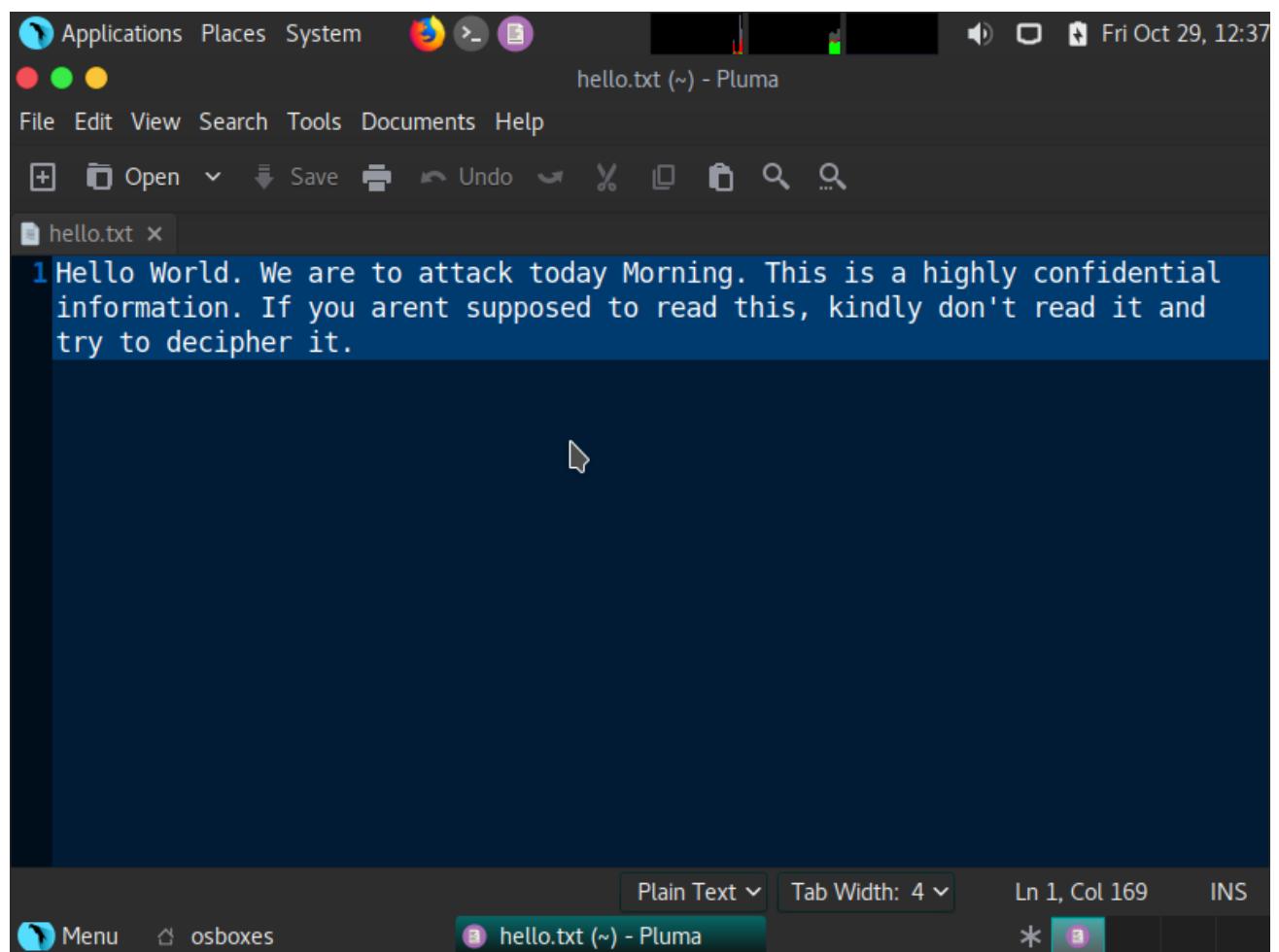
```
% openssl enc ciphertype -e -in plain.txt -out  
cipher.bin \ -K  
00112233445566778899aabbccddeeff\  
-iv 0102030405060708  
enc.
```

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 plain text using different Ciphers.

The plain text is as follows:

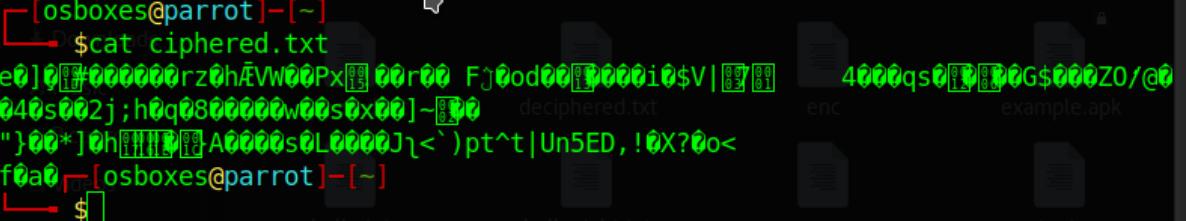


## 1. Encryption using aes-128-ecb



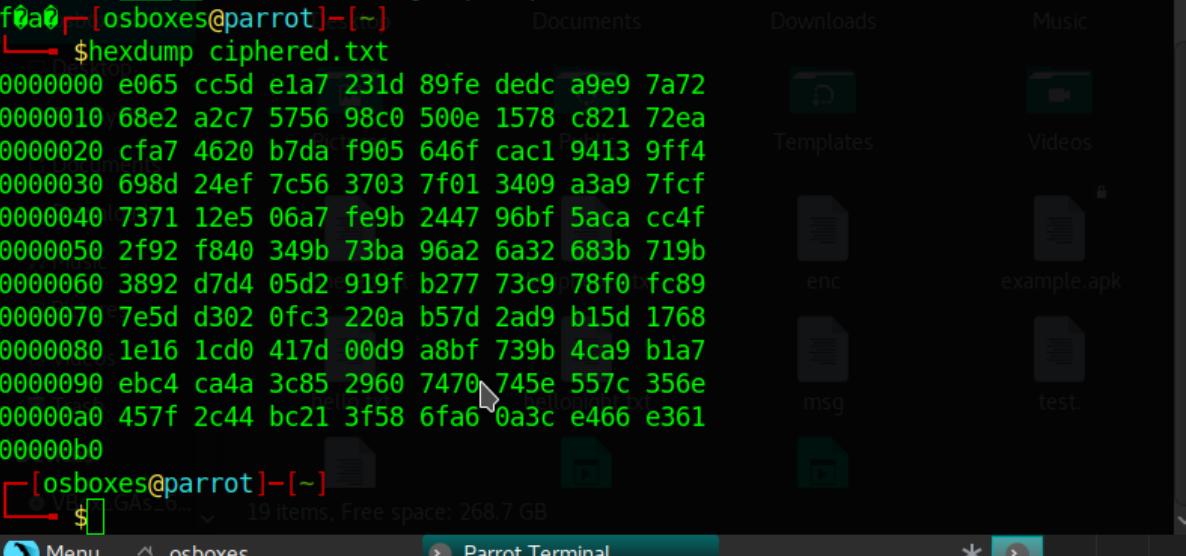
```
[osboxes@parrot] -[~]
$ openssl enc -aes-128-ecb -e -in hello.txt -out ciphered.txt -K 0011223345566778899aabbcdddeeff
```

The Encrypted text is :



```
[osboxes@parrot] -[~]
$ cat ciphered.txt
e@]@#000000rz@h@W@Px@00r@ Fj0od00@000i@$V|@#
@40s@2j ;h@q@8@0000w@00s@x@]~@# deciphered.txt
"}@*]@h@#A@000s@L@000J@<` )pt^t|Un5ED , !@X?@o<
f@o@-[osboxes@parrot] -[~]
$
```

The hexdump for the above cipher text is:



```
[osboxes@parrot] -[~]
$ hexdump ciphered.txt
00000000 e065 cc5d e1a7 231d 89fe dedc a9e9 7a72
00000010 68e2 a2c7 5756 98c0 500e 1578 c821 72ea
00000020 cfa7 4620 b7da f905 646f cac1 9413 9ff4
00000030 698d 24ef 7c56 3703 7f01 3409 a3a9 7fcf
00000040 7371 12e5 06a7 fe9b 2447 96bf 5aca cc4f
00000050 2f92 f840 349b 73ba 96a2 6a32 683b 719b
00000060 3892 d7d4 05d2 919f b277 73c9 78f0 fc89
00000070 7e5d d302 0fc3 220a b57d 2ad9 b15d 1768
00000080 1e16 1cd0 417d 00d9 a8bf 739b 4ca9 b1a7
00000090 ebc4 ca4a 3c85 2960 7470 745e 557c 356e
00000a0 457f 2c44 bc21 3f58 6fa6 0a3c e466 e361
00000b0
```

## 2. Encryption using aes-128-cbc

The cipher text and the hexdump of the cipher text is as follows:

The screenshot shows a Parrot OS desktop environment. In the foreground, a terminal window titled "Parrot Terminal" is open, displaying the following command and its output:

```
[osboxes@parrot]~$ openssl enc -aes-128-cbc -e -in hello.txt -out ciphered.txt -K 0011223345566778899aabbccddeeff -iv 0102030405060708
[osboxes@parrot]~$ cat ciphered.txt
```

The terminal then displays the hexdump of the encrypted file:

```
00+0005q0xG0000009 [osboxes@parrot]~$ hexdump ciphered.txt
00000000 b80c 3254 40e4 22e8 203e 52b0 0204 2770
00000010 d8cc cfcc 09bb 638d 87bc 9b56 97fe c1f1
00000020 48b5 e3fd c104 47c3 88e0 7e11 85e8 6d7c
00000030 d94d 321d bd11 2173 aed0 2b94 e09f 27a7
00000040 34ea 44f0 c13e e256 09a4 138c 4e4a b623
00000050 c28f e228 abb6 a4f2 7b87 7ec2 39e1 9474
00000060 04a1 15df e5c8 394f dc23 0024 53c7 d767
00000070 2a55 dc79 2f78 eb02 43c6 8147 60c3 532e
00000080 e22f 7e1a 0c16 526d e49b 793a d34d e914
00000090 70ac e8fd e50f 81e0 9a81 7db4 dfd7 dde3
000000a0 4e2b e78f 7135 7895 8547 fdaf bcb0 39fd
```

### 3. Encryption using aes-128-cfb

The cipher text and the hexdump for the same is as follows:

The screenshot shows a Parrot OS desktop environment. The terminal window displays the following session:

```
[osboxes@parrot]~$ openssl enc -aes-128-cfb -e -in hello.txt -out ciphered.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[osboxes@parrot]~$ cat ciphered.txt
[osboxes@parrot]~$ hexdump ciphered.txt
```

The terminal output shows the encrypted content of `hello.txt` as raw hex data.

#### 4. Encryption using aes-192-ecb

The cipher text and hexdump of the same is as follows:

The screenshot shows a Parrot OS desktop environment. In the top-left, a terminal window displays the command `hexdump ciphered.txt` followed by a large block of hex dump output. In the top-right, a file manager window shows a desktop folder containing several files: Documents, Downloads, Music, Templates, Videos, enc, example.apk, msg, and test. A cursor is hovering over the `test.` file. At the bottom, a dock bar includes icons for Menu, osboxes, and Parrot Terminal.

```
[osboxes@parrot]~$ hexdump ciphered.txt
00000000 c8aa c34c 5603 9f2f 2190 e752 44fd 2845
00000010 ebde d0f3 14f9 1bb6 4103 37b2 400d aaf5
00000020 ced2 fdc6 686b 070b 7cc0 846b d6ff 7e87
00000030 2b22 5afe bbca 64f6 1826 9e1e b84e a0e3
00000040 92c0 668b 9f1a c612 f72c 3383 f8f6 1a4b
00000050 716e 3b61 e1dc 1fe7 5622 350f e506 b6cf
00000060 8bc6 ef24 0988 a301 0fb9 8648 3917 88a0
00000070 9251 af8a 0464 de2e 3d3b cc77 8caf 5532
00000080 5a6d 7d0b a243 7285 5bf8 0aa7 d4b1 0f91
00000090 4eb9 1067 d7d4 2a3b b0e4 9b54 d4d2 adde
000000a0 62ac 1ffd 3f6f 9ef1 1e50 069b 5672 0a42
000000b0
[osboxes@parrot]~$
```

## 5. Encryption using aes-192-cbc

The cipher text and hexdump of the same is as follows:

The screenshot shows a Parrot OS desktop environment. In the top-left, a terminal window displays the command `openssl enc -aes-192-cbc -e -in hello.txt -out ciphered.txt -K 00112233445566778899aabbccddeefff -iv 0102030405060708` followed by the command `$ cat ciphered.txt`. In the top-right, a file manager window shows a desktop folder containing several files: Computer, Desktop, Documents, Downloads, Music, Templates, Videos, enc, example.apk, msg, and test. A cursor is hovering over the `test.` file. At the bottom, a dock bar includes icons for Menu, osboxes, and Parrot Terminal.

```
[osboxes@parrot]~$ openssl enc -aes-192-cbc -e -in hello.txt -out ciphered.txt -K 00112233445566778899aabbccddeefff -iv 0102030405060708
[osboxes@parrot]~$ cat ciphered.txt
M-[osboxes@parrot]~$ $ hexdump ciphered.txt
00000000 0a12 7dbf 3b60 a4d1 fe09 a00e a53a 0484
00000010 209f 98d5 9ca3 2117 9ac9 9302 9827 714f
00000020 1b00 9d59 95fb ffbe b6e6 8d8d 0e73 7eee
00000030 874c 81fd 8ac9 48e6 6db8 8d51 f380 a946
00000040 2cf7 cee0 8a27 c468 3136 8b6e 91ca 6d21
00000050 1e59 3aa1 d5c5 ae9a 9b3c 1e80 6c72 2dae
00000060 f4ff ea8a d35c 28f7 c429 932b c389 c3ef
00000070 3ddc a6cd 42ce ee4b 5ba1 f09c 4382 d8de
00000080 f74c b0d7 926d c4be 84a7 be08 5576 c2b6
00000090 6fa8 1b1c c1d8 99e2 93d0 bc8d 97b6 e2b7
000000a0 2093 0c38 6f5f 457d 2ae3 97a9 4254 4d7e
000000b0
```

## 6. Encryption using aes-192-cfb

The cipher text and hexdump of the same is as follows:

The screenshot shows a terminal window titled "Parrot Terminal" running on a Parrot OS desktop environment. The terminal session starts with the command \$openssl enc -aes-192-cfb -e -in hello.txt -out ciphered.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708, followed by the output of the encrypted file (ciphered.txt) which appears as a series of non-printable characters. Then, the command \$cat ciphered.txt is run to show the raw binary data. Finally, the command \$hexdump ciphered.txt is run to display the hex dump of the encrypted file.

```
$openssl enc -aes-192-cfb -e -in hello.txt -out ciphered.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
$cat ciphered.txt
$hexdump ciphered.txt
```

Address	Value	Character
00000000	21ef	
00000010	9f40	a77b
00000020	9540	1a45
00000030	7acc	df14
00000040	5be6	f08e
00000050	df9f	86fa
00000060	fa69	4c0d
00000070	aa9f	86b2
00000080	942f	b22d
00000090	ae10	2866
000000a0	3e1e	c158
000000b0	c158	ae82
000000c0	4266	0d70
000000d0	ae82	8225
000000e0	677b	e4eb
000000f0	67a1	d9fd
00000100	c2ed	677b
00000110	391c	e872
00000120	391c	a4ab
00000130	756d	9929
00000140	756d	7ed2
00000150	c4aa	acaf
00000160	285f	391c
00000170	2a6f	756d
00000180	ff0e	c4aa
00000190	2f4c	285f
000001a0	3648	2a6f
000001b0	1c6c	ff0e
000001c0	0f42	2f4c
000001d0	89c5	3648
000001e0	9d41	1c6c
000001f0	c7dd	0f42
00000200	a8ec	89c5
00000210	00d2	9d41
00000220		c7dd
00000230		a8ec
00000240		00d2
00000250		

## 7. Encryption using des-ecb

The cipher text and hexdump of the same is as follows:

## 8. Encryption using des-cfb

The cipher text and hexdump of the same is as follows:

The screenshot shows a Parrot OS desktop environment. The terminal window in the foreground displays the following session:

```
[x]-[osboxes@parrot]-[~]
$ openssl enc -des-cfb -e -in hello.txt -out ciphered.txt -K 0123456789abc
def -iv 0102030405060708
[osboxes@parrot]-[~]
$ cat ciphered.txt
00000000  eaae 7715 36c4 8983 65f0 99f0 8275 b272
00000010  de25 36d7 32b6 0b0f a1dc 9f17 d65c 74c3
00000020  21d7 2ec2 6f56 738e 3703 2fa0 7570 d4db
00000030  3b46 413a f555 be47 433e c5a7 148d b7bc
00000040  17e0 a449 d5c7 2f0e 9322 de29 4a38 6931
00000050  1454 08de 2b95 c82d 04f5 3368 6aac 3bb1
00000060  9519 0510 0888 5ce1 95cd 0718 2a6b b1eb
00000070  f109 3837 0d37 34cb a29d ba88 8a54 821a
00000080  f5cd bbbc 6f09 aa8e d17f 4c3a b458 cbcc
00000090  c4b3 1dd9 af91 29b3 b665 3b55 4dbb cbb3
000000a0  7370 1254 35df c7d0 004e
000000a9  GAs_6...
[osboxes@parrot]-[~]
```

The file manager window in the background shows a directory structure with files like `ciphered.txt`, `deciphered.txt`, `enc`, `msg`, `test`, and `example.apk`.

## 9. Encryption using des-cbc

The cipher text and hexdump of the same is as follows:

The screenshot shows a Parrot OS desktop environment. At the top is a dark grey header bar with icons for Applications, Places, System, a browser, and system status. The date and time 'Thu Oct 28, 07:47' are also visible. Below the header is a window titled 'Parrot Terminal'. The terminal window has a menu bar with File, Edit, View, Search, Terminal, Help. It contains several command-line sessions:

```

[osboxes@parrot]~
└─$ openssl enc -des-cbc -e -in hello.txt -out ciphered.txt -K 0123456789abc
def -iv 0102030405060708 osboxes>
[osboxes@parrot]~
└─$ cat ciphered.txt
00n-b50000IS0:oU[Ds0U[K0I0I0%000#<00I00*!000zbm00^0<cU[K0hv0000[K000
vr0J0-x[B0rG000tYz000[V0#000HJ0090x/000{m00^0?d[r0]800
0-0010[B30(00[H000
0000'B0K-[osboxes@parrot]~
└─$ hexdump ciphered.txt
00000000 dac2 6e30 622d a335 e1dd 498e c453 6f3a
00000010 1855 73e7 55ce 5813 168c 0313 25d8 b09e
00000020 c7bd 3c23 d70f 49d7 30fb 2a9e 9721 cf1f
00000030 dde6 62bc e26d 5ed1 3c9a 635f 1d55 d04b
00000040 cce1 7fa5 6808 a976 f69d 91fd 1e09 a34b
00000050 12c5 7637 c872 ca4a 2d60 0378 e438 4772
00000060 ab9d a8f4 5974 e57a c7b7 511f c456 a723
00000070 138d fa88 0f48 0e4a a6aa cb39 780e 942f
00000080 97a0 6d7b 849e ae5e c93f 7296 16c1 38b6
00000090 fdcf dd0b fe7e 319b 11db 3342 288c c799
000000a0 1fac bc03 4fa4 9af5 c2c7 ddd4 4227 4ba1
000000b0

```

The terminal window also shows the path '/home/osboxes/Desktop' and the message '19 items, Free space: 268.7 GB'. To the right of the terminal is a file browser window showing files like 'Templates', 'Videos', 'enc', 'example.apk', 'msg', and 'test.'. At the bottom of the screen is a dock with icons for 'Menu', 'osboxes', 'Parrot Terminal', and other system icons.

### 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. However, 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 information about the original picture from the encrypted picture? Please explain your observations.

Original Picture



Image Encrypted using ecb

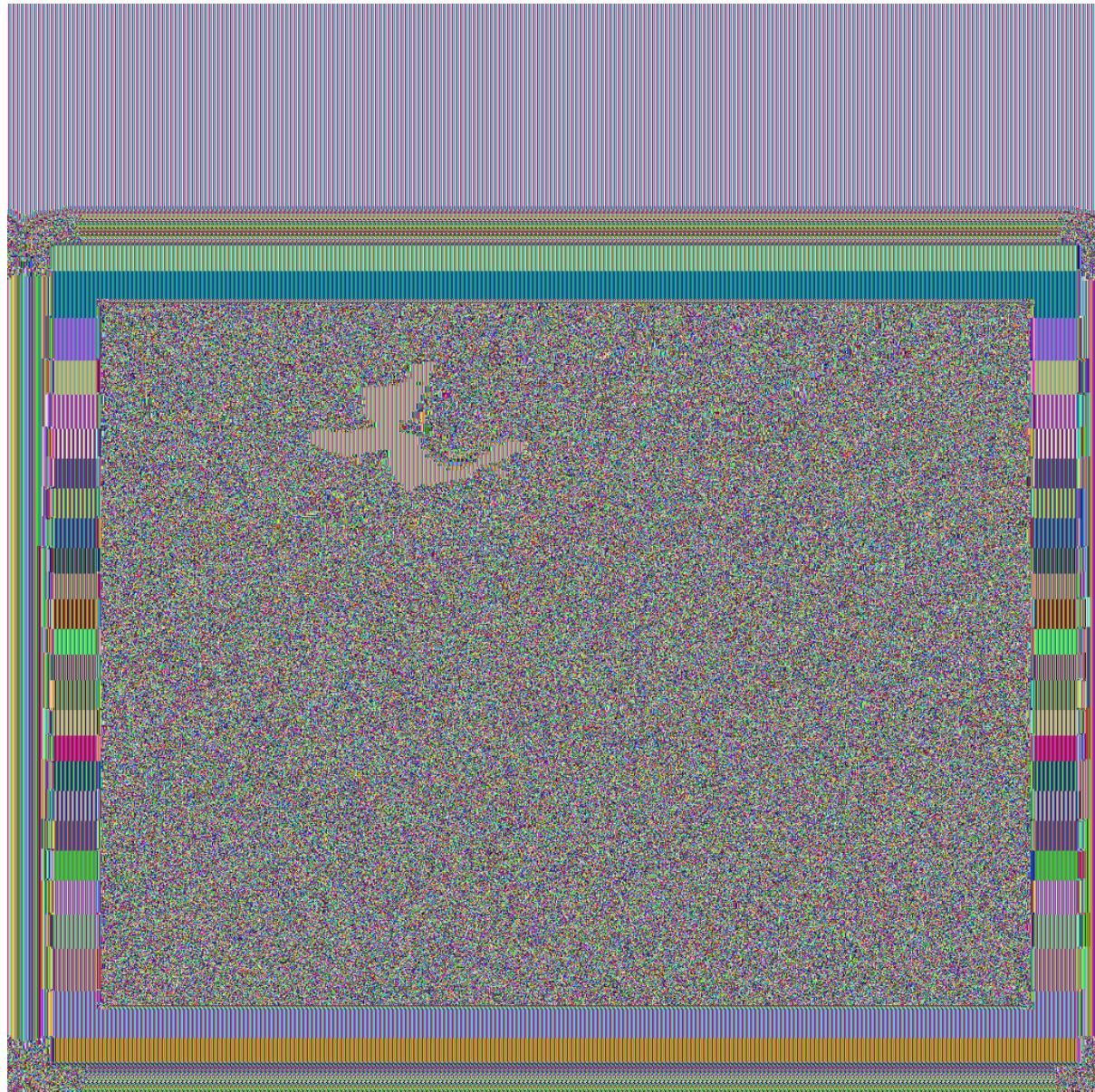
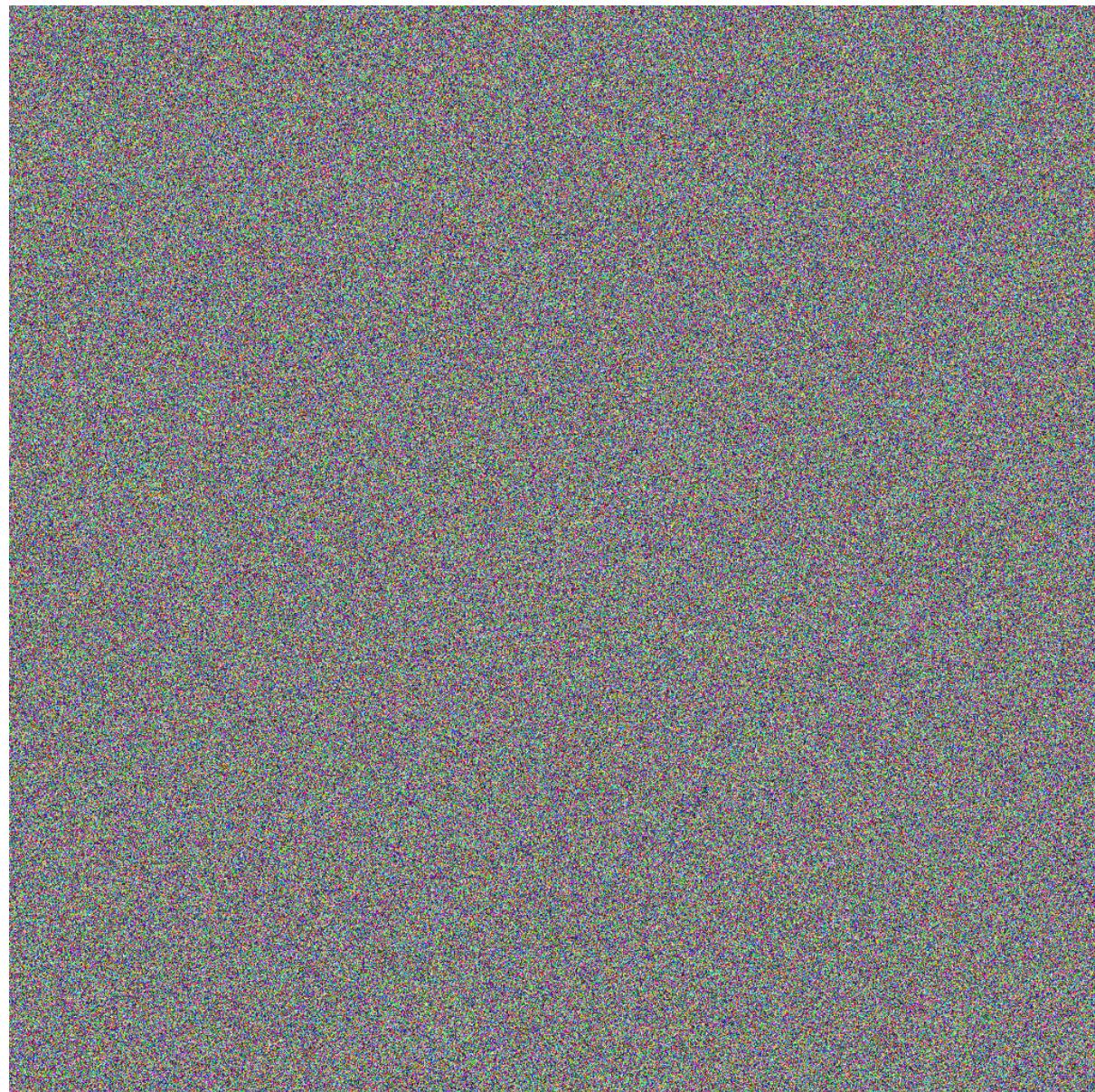


Image encrypted using cbc



### Observations:

- 1) In the image encrypted using ECB, we can still see a set of certain colors and shapes from which we can try to figure out what the original picture could be.
  - 2) In the image encrypted using CBC the entire images is distorted and there is no way by which we can try to figure out the original picture.

### 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?

The original plain text is as follows:

The screenshot shows the Atom code editor interface. The title bar displays "test.txt — ~ — Atom". The menu bar includes "File", "Edit", "View", "Selection", "Find", "Packages", and "Help". A sidebar on the left shows "Welcome Guide". The main editor area contains the following text:

```
1 hello world.  
2 This is a text file to check if corrupted encrypted file decrypts properly.  
3 qwertyuiopasdfghjklzxcvbnm12345678999223bbbdd cmcmcmd  
4 WCWCWCCWCWCWCCCCWCWCCCCWCWCWCWC  
5 eeeeeeeeeeeeeeeeeeeeeeeee  
6
```

The text from line 3 onwards is heavily corrupted, with many characters being replaced by underscores and other symbols. The status bar at the bottom shows "test.txt 6:1", "LF", "UTF-8", "Plain Text", and "0 files".

The above text is encrypted using aes-128 in 4 different modes: cbc,cfb,ofb and ecb.

After corrupting 30<sup>th</sup> byte of each cbc, cfb, ofb and ecb :

```
File: testenc.txt          ASCII Offset: 0x0000001D / 0x000000CF (%14) M^
00000000 A1 C8 C8 4D DB A2 66 FC 20 EE 7A 45 29 C9 6E 4E ...M..f. .zE).nN
00000010 FC 98 05 D5 59 C7 D3 05 1B D8 6F CE 58 7D 80 C2 ....Y.....o.X}..
00000020 11 03 A3 D1 E4 1D 68 08 0A 03 34 58 12 FD 68 1F .....h...4X..h.
00000030 AD 61 87 AB 71 01 A1 B7 12 94 17 25 F8 56 06 63 .a..q.....%.V.c
00000040 C8 41 9E 1C 7F 5A B2 9D E0 08 3E BE 00 23 1B 07 .A...Z....>.#..
00000050 9F 1F B6 C1 95 8A 4A AF 51 DA C2 6A AD CC B1 1A .....J.Q..j....
00000060 4F 10 50 30 AF 57 04 6A 79 E7 26 88 DA 15 82 5C 0.P0.W.jy.&....\
00000070 91 C7 66 BA 35 E0 8D 21 66 0F B9 29 30 DE 62 0B ..f.5..!f..)0.b.
00000080 FD DB F0 BA 91 18 93 FF 3E 56 36 32 A8 43 91 91 .....>V62.C..
00000090 4D 5A 2E 33 5A 01 0E 64 C8 33 50 AE 48 74 C9 FB MZ.3Z..d.3P.Ht..
000000A0 BD 95 3B 2B 76 40 AE 12 0A 6B 0C 21 80 7C CA 98 ..;+v@....k.!..|..
000000B0 CC 0D 3A 86 88 BB 20 BB BA D5 54 A0 72 AE 4F AC ..:....T.r.O.
000000C0 8A 70 7F 6F 60 DC 3E EB 7D A3 71 D4 C5 4D 22 C1 .p.o`.>.).q..M".
```

Applications Places System Parrot Terminal Thu Oct 28, 09:52

File Edit View Search Terminal Help

```
File: testenc.txt          ASCII Offset: 0x0000001D / 0x000000C3 (%15) M^
00000000 C9 1E EA 51 6F E6 7A 3B 17 5B 9D 7D 3C 6F 2B B2 ...Qo.z;.[.]<o+.
00000010 76 26 01 A6 23 C5 EA 28 C0 8C BE 83 83 83 3C 3A v&..#...(. ....<:
00000020 40 D7 DF DA AD 5B 8D 14 E1 C8 25 0F 0E 09 70 21 @....[....%....p!
00000030 37 63 43 EE 14 13 8C E9 96 C8 71 AB 87 A2 78 34 7cC.....q...x4
00000040 A1 EF 5C 61 11 74 F5 4B 4B 70 F4 DD 47 29 80 A2 ..\a.t.KKp..G)..Y
00000050 D7 BA B8 99 9F 44 41 A8 79 30 9E B4 43 92 83 59 .....DA.y0..C..Y
00000060 CF 16 0E 27 98 2A AD 84 4F 5D 37 7C F1 17 70 63 ...'*...0]7|..pc
00000070 65 FF E9 1E 30 EF 0E EF 2B EA 60 44 8E 6B E1 69 e...0...+.`D.k.i
00000080 19 43 E4 1D F4 78 67 AE 18 8D 72 F7 CD 61 F5 5D .C...xg...r..a.]
00000090 58 E5 5E 04 03 A3 41 BF 5E 8A A3 88 38 46 C5 44 X.^....A.^...8F.D
000000A0 12 C8 E3 6F C0 39 F9 81 C9 47 D4 59 11 B2 64 0C ...o.9...G.Y..d.
000000B0 09 F1 89 73 19 3F 85 C8 62 4C 3F 6F 63 01 07 3D ...s.?..bl?oc..=
```

Applications Places System Parrot Terminal Thu Oct 28, 09:58

File Edit View Search Terminal Help

```
File: testenc.txt          ASCII Offset: 0x0000001D / 0x000000C3 (%15) M^
00000000 C9 1E EA 51 6F E6 7A 3B 17 5B 9D 7D 3C 6F 2B B2 ...Qo.z;.[.]<o+.
00000010 07 1D 90 45 C7 C0 4C 83 81 27 E3 A5 46 83 83 A5 C0 ...E..L..'.F...
00000020 E4 19 87 8D 71 51 CE 96 AD EA 9B EA 32 83 83 80 41 ....qQ.....2..A
00000030 90 3A B2 55 F9 14 4D BD 04 AB BA C8 A2 9B 48 5D ...U..M.....H]
00000040 60 B5 54 1D 87 8F FC B6 CE FD 60 E1 7B F1 7F 93 `..T.....`.{...
00000050 C7 B5 CF D6 A9 F8 41 CD 63 65 DC 32 20 AE 85 D4 .....A.ce.2 ...
00000060 10 5C 85 E4 FB E2 BB 9B 33 30 55 E7 2E 3B 24 3B .\.....30U..;$;
00000070 1B 23 B9 A3 9D 4E 45 99 5E 9C 88 4C 9F 75 DF 16 .#.NE.^..L.u..
00000080 F8 26 8F A4 AD FC D9 19 27 70 43 92 40 40 68 7E .&.....'pC.@@h-
00000090 54 23 1E AB B2 89 49 4C 32 7A A6 FB B6 FD C5 2A T#....IL2z.....*
000000A0 D0 E1 98 2A 48 3C 33 4B 9F C9 8A C9 A3 BF A9 07 ...*H<3K.....
000000B0 6B CC 9F C4 26 C2 8F 6F 5C 7B 28 CC 2B F7 DA 9C k...&..o\{(.....
000000C0 EB 30 83 2F .....0./
```

^G Help ^C Exit (No Save) ^T goTo Offset ^X Exit and Save ^W Search

Menu osboxes Parrot Terminal Save Screenshot

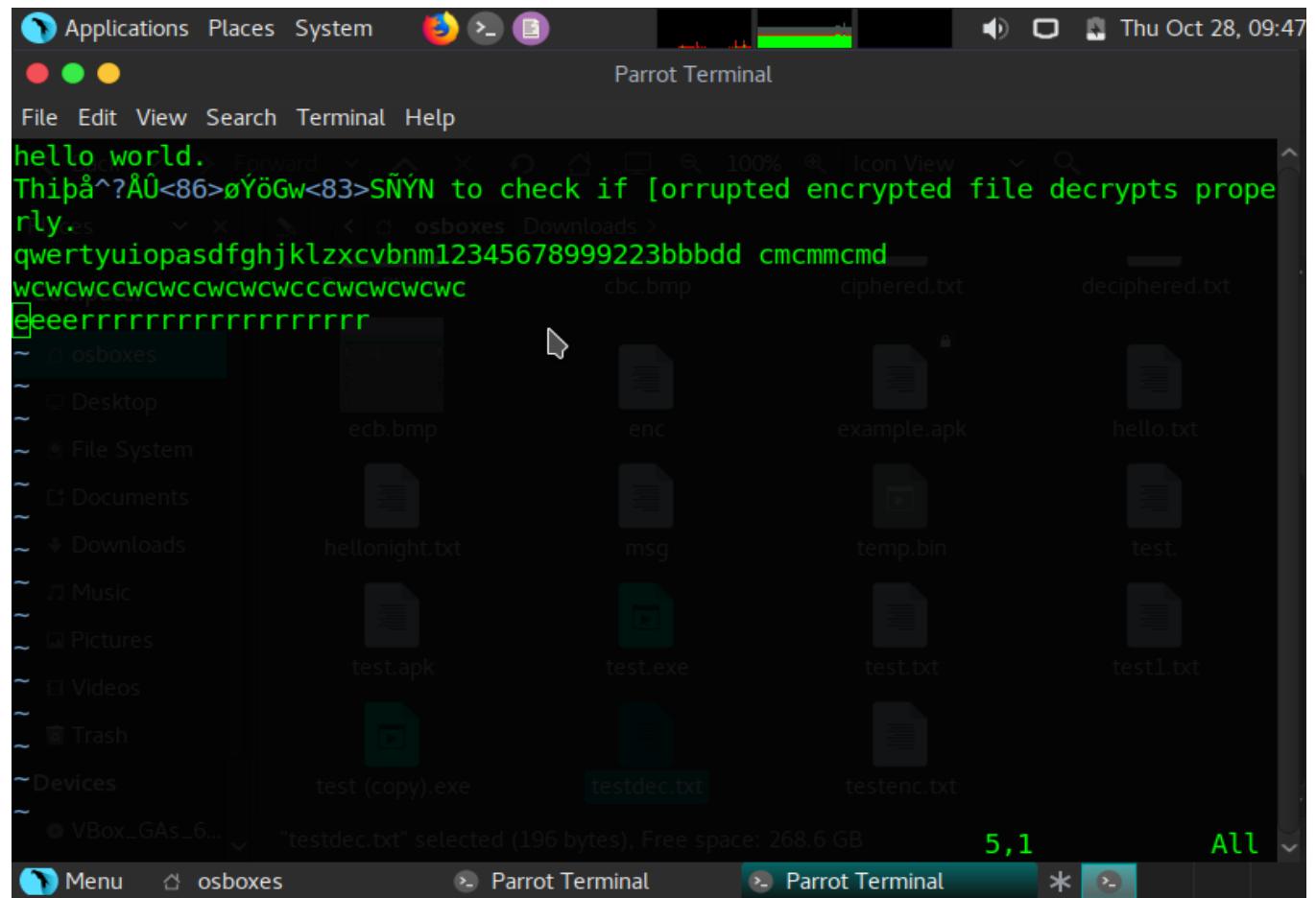
Applications Places System Parrot Terminal Thu Oct 28, 10:04

File Edit View Search Terminal Help

```
File: testenc.txt          ASCII Offset: 0x0000001E / 0x000000CF (%14) M^
00000000 D4 F5 40 D7 DC 65 38 73 8D 48 4C E1 FD 5E C8 A5 ..@..e8s.HL..^..
00000010 87 18 62 D5 6C 91 C7 19 7D F4 51 A6 89 1A BC C8 ..b.l...}.Q.....
00000020 0A 01 87 BA 4C 2F 3E 8A 1F 09 F6 43 1C CF 31 68 ....L/>....C..1h
00000030 A7 33 FF DF C0 04 50 CA 8D 0B 7A 7F DE 0B BF 6B .3....P....z....k
00000040 F4 A3 D2 46 9C C2 31 06 45 4F 03 AB 76 88 9F 8E ...F..1.E0..v...
00000050 B6 0B 0F 13 F2 93 87 45 AD 86 05 4F 4E 81 AA 2A .....E...ON...*
00000060 BC F5 C2 5A 4A 86 A9 95 24 31 03 48 10 E2 49 71 ...ZJ...$1.H..Iq
00000070 F4 1A 40 B8 78 65 2B 3A 5E 5D 8E F0 97 BE A1 86 ..@.xe+:^].....
```

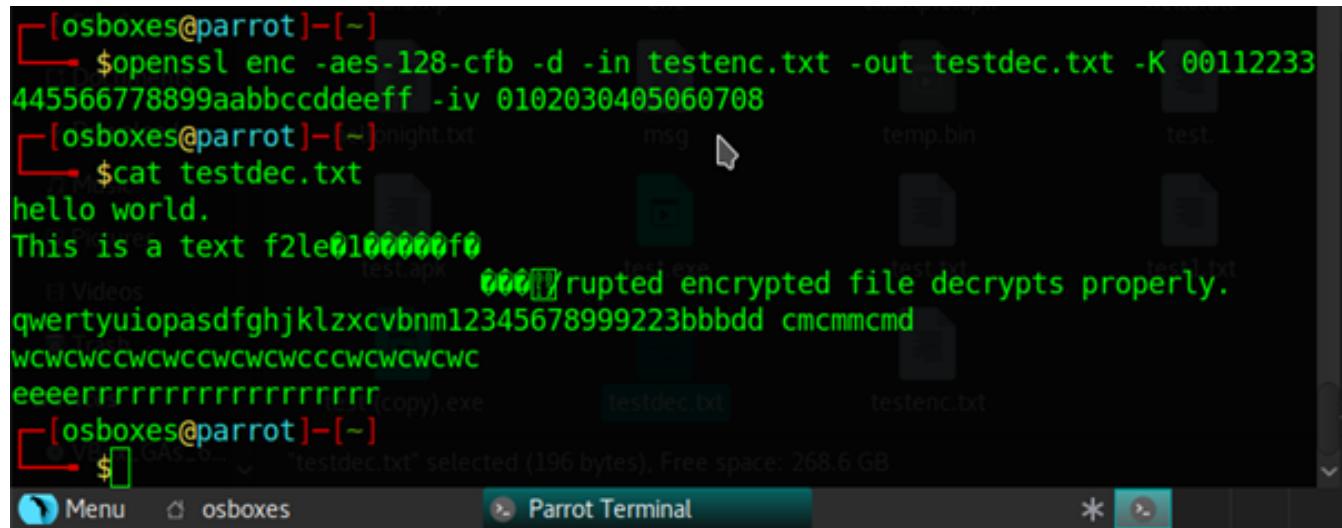
The decrypted text for the same is as follows:

CBC



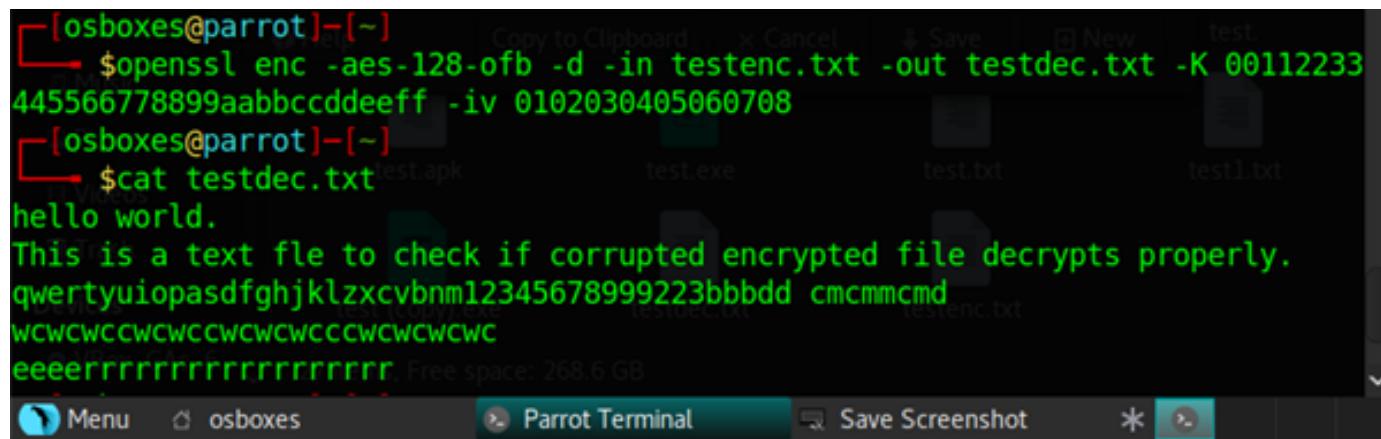
## CFB

```
[osboxes@parrot]~$ openssl enc -aes-128-cfb -d -in testenc.txt -out testdec.txt -K 00112233  
445566778899aabccddeff -iv 0102030405060708  
[osboxes@parrot]~$ cat testdec.txt  
hello world.  
This is a text file to check if corrupted encrypted file decrypts properly.  
qwertyuiopasdfghjklzxcvbnm12345678999223bbbdd cmcmmcmd  
WCWCWCCWCWCWCWCWCCCCWCWCWC  
eeeeeeeeeeeeeeeeeeee  
[osboxes@parrot]~$  
$ [osboxes@parrot]~$ testdec.txt selected (196 bytes), Free space: 268.6 GB
```



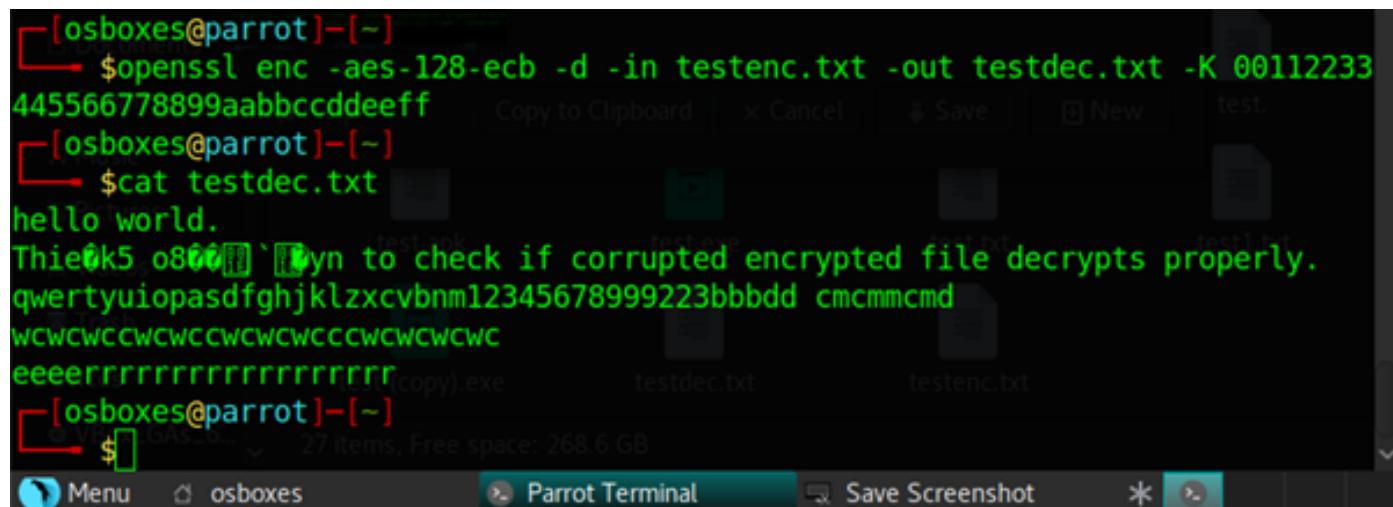
## OFB

```
[osboxes@parrot]~$ openssl enc -aes-128-ofb -d -in testenc.txt -out testdec.txt -K 00112233  
445566778899aabccddeff -iv 0102030405060708  
[osboxes@parrot]~$ cat testdec.txt  
hello world.  
This is a text file to check if corrupted encrypted file decrypts properly.  
qwertyuiopasdfghjklzxcvbnm12345678999223bbbdd cmcmmcmd  
WCWCWCCWCWCWCWCWCCCCWCWC  
eeeeeeeeeeeeeeeeeeee  
[osboxes@parrot]~$ [osboxes@parrot]~$ testdec.txt selected (196 bytes), Free space: 268.6 GB
```



## ECB

```
[osboxes@parrot]~$ openssl enc -aes-128-ecb -d -in testenc.txt -out testdec.txt -K 00112233  
445566778899aabccddeff  
[osboxes@parrot]~$ cat testdec.txt  
hello world.  
This is a text file to check if corrupted encrypted file decrypts properly.  
qwertyuiopasdfghjklzxcvbnm12345678999223bbbdd cmcmmcmd  
WCWCWCCWCWCWCWCWCCCCWCWC  
eeeeeeeeeeeeeeeeeeee  
[osboxes@parrot]~$  
$ [osboxes@parrot]~$ testdec.txt selected (196 bytes), Free space: 268.6 GB
```



From the results above I can conclude that:

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 8 bytes might spread to all n bits in plaintext block 8 bytes since we do the decryption one block at a time.

In CBC mode, there was an effect in two blocks.

In CFB mode, there is a problem in  $n / r$  number of blocks.

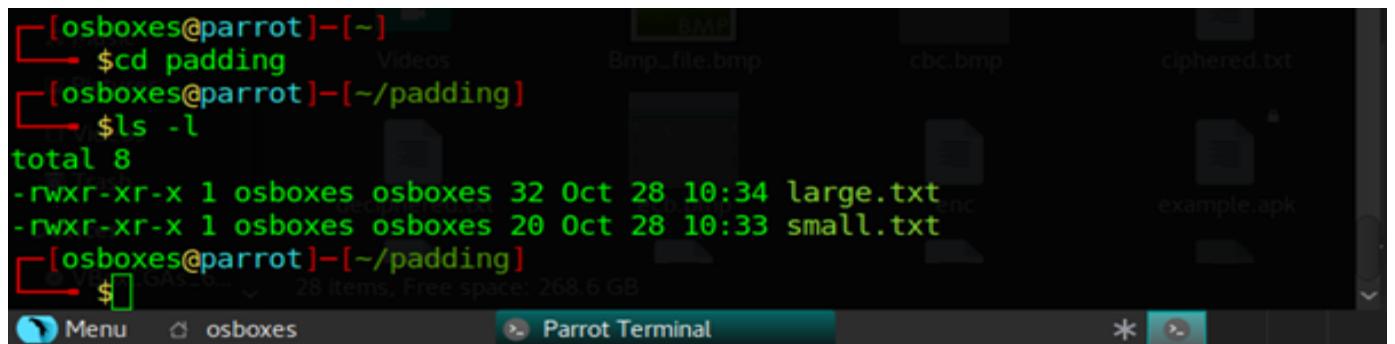
$r$  = number of bits taken at a time to XOR.

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.

### 3.4 Task4 : Padding

For block ciphers, when the size of the plaintex 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.



```
[osboxes@parrot]~
└─$ cd padding
[osboxes@parrot]~/padding
└─$ ls -l
total 8
-rwxr-xr-x 1 osboxes osboxes 32 Oct 28 10:34 large.txt
-rwxr-xr-x 1 osboxes osboxes 20 Oct 28 10:33 small.txt
[osboxes@parrot]~/padding
└─$
```

Two files namely large.txt and small.txt are made with each having 32 and 20 bytes respectively.

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

The screenshot shows a Parrot OS desktop environment with a terminal window open. The terminal window title is "Parrot Terminal". The terminal content displays a series of OpenSSL encryption commands. It starts by listing two files: "large.txt" and "small.txt". Then, it runs four commands to encrypt "small.txt" using ECB mode with different key sizes (128 and 32 bits) and different padding schemes (padding and no padding). Finally, it runs three commands to encrypt "large.txt" using CBC mode with different key sizes (128 and 32 bits) and different padding schemes (padding and no padding), each time specifying an IV. The terminal window has a dark background with light-colored text. The desktop interface includes a menu bar at the top with icons for Applications, Places, System, and a browser icon. The desktop itself is visible in the background, showing a few icons and a progress bar.

```
-rwxr-xr-x 1 osboxes osboxes 32 Oct 28 10:34 large.txt
-rwxr-xr-x 1 osboxes osboxes 20 Oct 28 10:33 small.txt
[osboxes@parrot]~/padding]
$ openssl enc -aes-128-ecb -e -in small.txt -out small_ecb_20.txt -K 00112
233445566778899aabcccddeeff
[osboxes@parrot]~/padding]
$ openssl enc -aes-128-ecb -e -in large.txt -out small_ecb_32.txt -K 00112
233445566778899aabcccddeeff
[osboxes@parrot]~/padding]
$ openssl enc -aes-128-cbc -e -in small.txt -out small_cbc_20.txt -K 00112
233445566778899aabcccddeeff
$ openssl enc -aes-128-cbc -e -in small.txt -out small_cbc_20.txt -K 00112
233445566778899aabcccddeeff -iv 0102030405060708
[osboxes@parrot]~/padding]
$ openssl enc -aes-128-cbc -e -in large.txt -out large_cbc_20.txt -K 00112
233445566778899aabcccddeeff -iv 0102030405060708
[osboxes@parrot]~/padding]
$ openssl enc -aes-128-cfb -e -in small.txt -out small_cfb_20.txt -K 00112
233445566778899aabcccddeeff -iv 0102030405060708
[osboxes@parrot]~/padding]
```

The screenshot shows a Parrot OS desktop environment with a terminal window open. The terminal window title is "Parrot Terminal". The terminal content displays a series of OpenSSL encryption commands being run on two input files: "small.txt" and "large.txt". The commands involve different cipher modes (cbc, cfb, ofb) and keys (K 00112). The terminal also shows the resulting output files like "small\_cbc\_20.txt", "large\_cbc\_20.txt", etc.

```
[osboxes@parrot]~/padding]$ openssl enc -aes-128-cbc -e -in small.txt -out small_cbc_20.txt -K 00112  
233445566778899aabbccddeeff  
iv undefined  
[x]~[osboxes@parrot]~/padding]$ openssl enc -aes-128-cbc -e -in small.txt -out small_cbc_20.txt -K 00112  
233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~/padding]$ openssl enc -aes-128-cbc -e -in large.txt -out large_cbc_20.txt -K 00112  
233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~/padding]$ openssl enc -aes-128-cfb -e -in small.txt -out small_cfb_20.txt -K 00112  
233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~/padding]$ openssl enc -aes-128-cfb -e -in large.txt -out large_cfb_32.txt -K 00112  
233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~/padding]$ openssl enc -aes-128-ofb -e -in small.txt -out small_ofb_20.txt -K 00112  
233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~/padding]$ openssl enc -aes-128-ofb -e -in large.txt -out large_ofb_32.txt -K 00112  
233445566778899aabbccddeeff -iv 0102030405060708
```

The screenshot shows a Parrot OS desktop environment with a terminal window open. The terminal window title is "Parrot Terminal". The terminal content displays a series of OpenSSL encryption commands being run on two input files: "small.txt" and "large.txt". The commands involve different cipher modes (aes-128-ofb) and keys (K 00112). The terminal also shows the resulting output files like "small\_ofb\_20.txt", "large\_ofb\_32.txt", etc.

```
233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~/padding]$ openssl enc -aes-128-ofb -e -in small.txt -out small_ofb_20.txt -K 00112  
233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~/padding]$ openssl enc -aes-128-ofb -e -in large.txt -out large_ofb_32.txt -K 00112  
233445566778899aabbccddeeff -iv 0102030405060708
```

```
[osboxes@parrot]~/padding]
$ ls -l
total 40
-rw-r--r-- 1 osboxes osboxes 48 Oct 28 10:40 large_cbc_32.txt
-rw-r--r-- 1 osboxes osboxes 32 Oct 28 10:42 large_cfb_32.txt
-rw-r--r-- 1 osboxes osboxes 48 Oct 28 10:38 large_ecb_32.txt
-rw-r--r-- 1 osboxes osboxes 32 Oct 28 10:44 large_ofb_32.txt
-rwxr-xr-x 1 osboxes osboxes 32 Oct 28 10:34 large.txt
-rw-r--r-- 1 osboxes osboxes 32 Oct 28 10:40 small_cbc_20.txt
-rw-r--r-- 1 osboxes osboxes 20 Oct 28 10:41 small_cfb_20.txt
-rw-r--r-- 1 osboxes osboxes 32 Oct 28 10:38 small_ecb_20.txt
-rw-r--r-- 1 osboxes osboxes 20 Oct 28 10:43 small_ofb_20.txt
-rwxr-xr-x 1 osboxes osboxes 20 Oct 28 10:33 small.txt
[osboxes@parrot]~/padding]
$
```

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, so if I use the nopad option, I can see the padding in the decrypted file.

The screenshot shows a terminal window titled "Parrot Terminal" running on a Parrot OS desktop environment. The terminal displays a series of openssl encryption commands for AES-128 modes. The user has run several commands to encrypt files using ECB, CBC, OFB, and CFB modes with a specific key and IV. The terminal window includes a menu bar with File, Edit, View, Search, Terminal, and Help options. The status bar at the bottom shows the current working directory as "/padding" and the title "Parrot Terminal".

```
[x]~[osboxes@parrot]~/padding$ openssl enc -aes-128-ecb -d -nopad -in large_ecb_32.txt -out large_ecb_dec.txt -K 00112233445566778899aabbccddeeff
[x]~[osboxes@parrot]~/padding$ openssl enc -aes-128-ecb -d -nopad -in small_ecb_20.txt -out small_ecb_dec.txt -K 00112233445566778899aabbccddeeff
[x]~[osboxes@parrot]~/padding$ openssl enc -aes-128-cbc -d -nopad -in large_cbc_32.txt -out large_cbc_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[x]~[osboxes@parrot]~/padding$ openssl enc -aes-128-cbc -d -nopad -in small_cbc_20.txt -out small_cbc_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[x]~[osboxes@parrot]~/padding$ openssl enc -aes-128-ofb -d -nopad -in small_ofb_20.txt -out small_ofb_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[x]~[osboxes@parrot]~/padding$ openssl enc -aes-128-cfb -d -nopad -in large_cfb_32.txt -out large_cfb_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
[x]~[osboxes@parrot]~/padding$
```

The screenshot shows a terminal window titled "Parrot Terminal" running on Parrot OS. The terminal displays a series of OpenSSL commands used for encrypting and decrypting files using AES-128 in various modes (ECB, CBC, OFB, CFB) with different padding options (nopad, PKCS7). The user has decrypted several files, including "small\_ecb\_20.txt", "large\_cbc\_32.txt", "small\_cbc\_20.txt", "large\_ofb\_32.txt", and "large\_cfb\_32.txt", resulting in files like "small\_ecb\_dec.txt", "large\_cbc\_dec.txt", and "small\_cfb\_dec.txt". The terminal also shows the command \$ls -l at the end.

```
$ openssl enc -aes-128-ecb -d -nopad -in small_ecb_20.txt -out small_ecb_dec.txt -K 00112233445566778899aabbccddeeff  
[osboxes@parrot]~[~/padding]  
$ openssl enc -aes-128-cbc -d -nopad -in large_cbc_32.txt -out large_cbc_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~[~/padding]  
$ openssl enc -aes-128-cbc -d -nopad -in small_cbc_20.txt -out small_cbc_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~[~/padding]  
$ openssl enc -aes-128-ofb -d -nopad -in small_ofb_20.txt -out small_ofb_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~[~/padding]  
$ openssl enc -aes-128-ofb -d -nopad -in large_ofb_32.txt -out large_ofb_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~[~/padding]  
$ openssl enc -aes-128-cfb -d -nopad -in large_cfb_32.txt -out large_cfb_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~[~/padding]  
$ openssl enc -aes-128-cfb -d -nopad -in small_cfb_20.txt -out small_cfb_dec.txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708  
[osboxes@parrot]~[~/padding]  
$ ls -l
```

Finally we can see the decrypted and encrypted files as follows:

The screenshot shows a terminal window titled "Parrot Terminal" running on Parrot OS. The command \$ls -l is run, displaying a list of files in the current directory. The terminal output shows 72 files, mostly named "large\_ecb\_32.txt", "large\_cfb\_32.txt", "large\_ofb\_32.txt", and "small\_ecb\_32.txt". The terminal window has a dark theme with green text. Below the terminal is a dock with icons for "Menu", "padding", and "Parrot Terminal". To the right of the terminal is a file browser window showing the same file names as the terminal output.

```
$ ls -l
total 72
-rw-r--r-- 1 osboxes osboxes 48 Oct 28 10:40 large_cbc_32.txt
-rw-r--r-- 1 osboxes osboxes 48 Oct 28 10:54 large_cbc_dec.txt
-rw-r--r-- 1 osboxes osboxes 32 Oct 28 10:42 large_cfb_32.txt
-rw-r--r-- 1 osboxes osboxes 32 Oct 28 10:58 large_cfb_dec.txt
-rw-r--r-- 1 osboxes osboxes 48 Oct 28 10:38 large_ecb_32.txt
-rw-r--r-- 1 osboxes osboxes 48 Oct 28 10:50 large_ecb_dec.txt
-rw-r--r-- 1 osboxes osboxes 32 Oct 28 10:44 large_ofb_32.txt
-rw-r--r-- 1 osboxes osboxes 32 Oct 28 10:57 large_ofb_dec.txt
-rwxr-xr-x 1 osboxes osboxes 32 Oct 28 10:34 large.txt
-rw-r--r-- 1 osboxes osboxes 32 Oct 28 10:40 small_cbc_20.txt
-rw-r--r-- 1 osboxes osboxes 32 Oct 28 10:55 small_cbc_dec.txt
-rw-r--r-- 1 osboxes osboxes 20 Oct 28 10:41 small_cfb_20.txt
-rw-r--r-- 1 osboxes osboxes 20 Oct 28 10:58 small_cfb_dec.txt
-rw-r--r-- 1 osboxes osboxes 32 Oct 28 10:38 small_ecb_20.txt
-rw-r--r-- 1 osboxes osboxes 32 Oct 28 10:53 small_ecb_dec.txt
-rw-r--r-- 1 osboxes osboxes 20 Oct 28 10:43 small_ofb_20.txt
-rw-r--r-- 1 osboxes osboxes 20 Oct 28 10:57 small_ofb_dec.txt
-rwxr-xr-x 1 osboxes osboxes 20 Oct 28 10:33 small.txt
[osboxes@parrot]~[~/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/descript 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](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  
13e10d1df4a2ef2ad4540fae1ca0aa9

Note 1: If you choose to store the plaintex 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:

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

```
all:
gcc -IS(INC) -LS(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_k = []
file = open('words.txt', 'r')
lines = file.readlines()
words = [str.strip(line) for line in lines]
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))
    is_matched = "NOT MATCHED"
    if bytes.hex(ciphertext) == cipher_hex:
        is_matched = "MATCHED"
        result_k.append(word)
print(word, bytes.hex(ciphertext), is_matched)
print("\n\nResulting Key:", result_k)
```

```

200177e0c77c0d949e0e0d0d4d1011002169031d011/01302c777/04205dd1e/100022 NOT MATCHED
zuza 49259e25ddfdeac0b078282f10d98b25bbdfc97888b1d85a3c6bf1e40d448a08 NOT MATCHED
zuzana 4b317a3a18dd0c2ac63191le6ae08397a1587ff6f861a8e9e3242930278d26b24bc NOT MATCHED
zu-zu 04eb3d47bbc2fce2df97928758cd1802f2fdbe866ffd4ed6227c80ad61fbef NOT MATCHED
zwanziger 70b52532147330fb9b718d64b0cd144390ad94fe5ab875cef8ac74d587621ff NOT MATCHED
zwart 7b8693c89e594c35fee42bd3e5cf6a318e4c6969c8f73b0ba65346d7ee79e227 NOT MATCHED
zwei 965faf9245114fb6d4c8a1bc2f5213f69e12199a7ad6dea685b56d62ac5acb7 NOT MATCHED
zweig 43f250b77ff3a0363dff761775585365cf2ff071b5872997cc6ebf3e5618cc NOT MATCHED
wick 96ea898d78c7212bb5d0ff7735c55bc318cfcd3ef7d1a5bed0f25c56e6501b6 NOT MATCHED
wickau 7c1b8fb63fbfd5b667524556182a5899835623f2f8fdebe2ec604f9082cbd8e4 NOT MATCHED
wicky c6c3a3e4cafcded440829f89f318ac2521d0b4e44afee0406173430b61f57 NOT MATCHED
wrieback 31b67ea013c06813e26e0466ef22aac8d0742615c3a8127e438e87b4d28c5464 NOT MATCHED
wriebacks cf4df738792dbc47b427c4ae046a28ff6f3362cc657e8ce951c1f61c34aa8a7 NOT MATCHED
wriebel 088015f65c1fb588a38e0f99e565869cc89fb5773d4ad0f2e05422ab1c41ee NOT MATCHED
wrieselite 2f2342c4e1c99673b54aca30655a35d89767864c3f5e9718f90365d92360dfa0 NOT MATCHED
wingle c9218eccf7014a9e948d5278aedd3cd999fdaafbc8cf5a68ca2b81877bea74f39b NOT MATCHED
wingli_b05d6f13ea431a27239a46d191cad9d521c2a2e25e2c59a6b8c1db2fbab102 NOT MATCHED
winglian 5f2b18d9f93d7fa27fc4a4c27581a2c272e864cf25a1e71652b86979ee6b5bac NOT MATCHED
winglianian ae0f7feabae6a15a22334d5cd43844b60f946f0382d13a8acd6b84dd80348b0 NOT MATCHED
winglianist 7158ad1fe7d3e6fcfd751fc2d881819d22b3f64a37e2ddfe8ae9d877cf8a08a NOT MATCHED
witter 6df120857a5b3461a67af4548258d705407c73c48cd524f226e7f38077d18b9 NOT MATCHED
witterion 0b794ab2da76a6a42ad9ff65820fd79cc0042113790a30adeff2daab51b5 NOT MATCHED
witterionic 43d9a3b874200bc78e02eb6b143c3e9495c296487814a3c4d4e8747d630c7de3 NOT MATCHED
wolle 327589a13f6940f1b17e3f3e8582672ae8bb8cebeccef4c151e7f91625c635 NOT MATCHED
zwyorkin 770797e6c4f4bd24b34e54ad3f7e958c85cc73d31b47623b372e4473519e1305 NOT MATCHED
zz 5d7a15c6b0879896467c9feac6346d16c02284f1c63389721179c2015cb21c2bb NOT MATCHED
ztt 58dae019524c7d7d79ceaaa34318cce4a5d49d9ad1817ac4b86a649500de2ef8 NOT MATCHED
zzz 2bfc0c1cb6d489616d5seaf811512b493869e2b14dfe420eaba64a4fe431e94b NOT MATCHED

```

Resulting Key: ['median']

Process finished with exit code 0

The key used to encrypt is median.

Conclusion:

- 1) AES, DES 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 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.
- 4) 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.
- 5) I could conclude from this experiment that , If I have the plaintext, ciphertext and iv know , I can easily find the key using brute force method.

Github Link: <https://github.com/Divya-127/CSS/>