LABORATORY CEL62: Cryptography and System Security Winter 2021

Experiment 3:	Crypto Encryption

Name: Jash Jain
Roll No: 2019130021
TE Comps

Note: Students are advised to read through this lab sheet before doing an 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 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

```
You should read the INSTALL file first:

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

openssl-1.0.1. To configure and install openssl libraries, run the following commands.

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

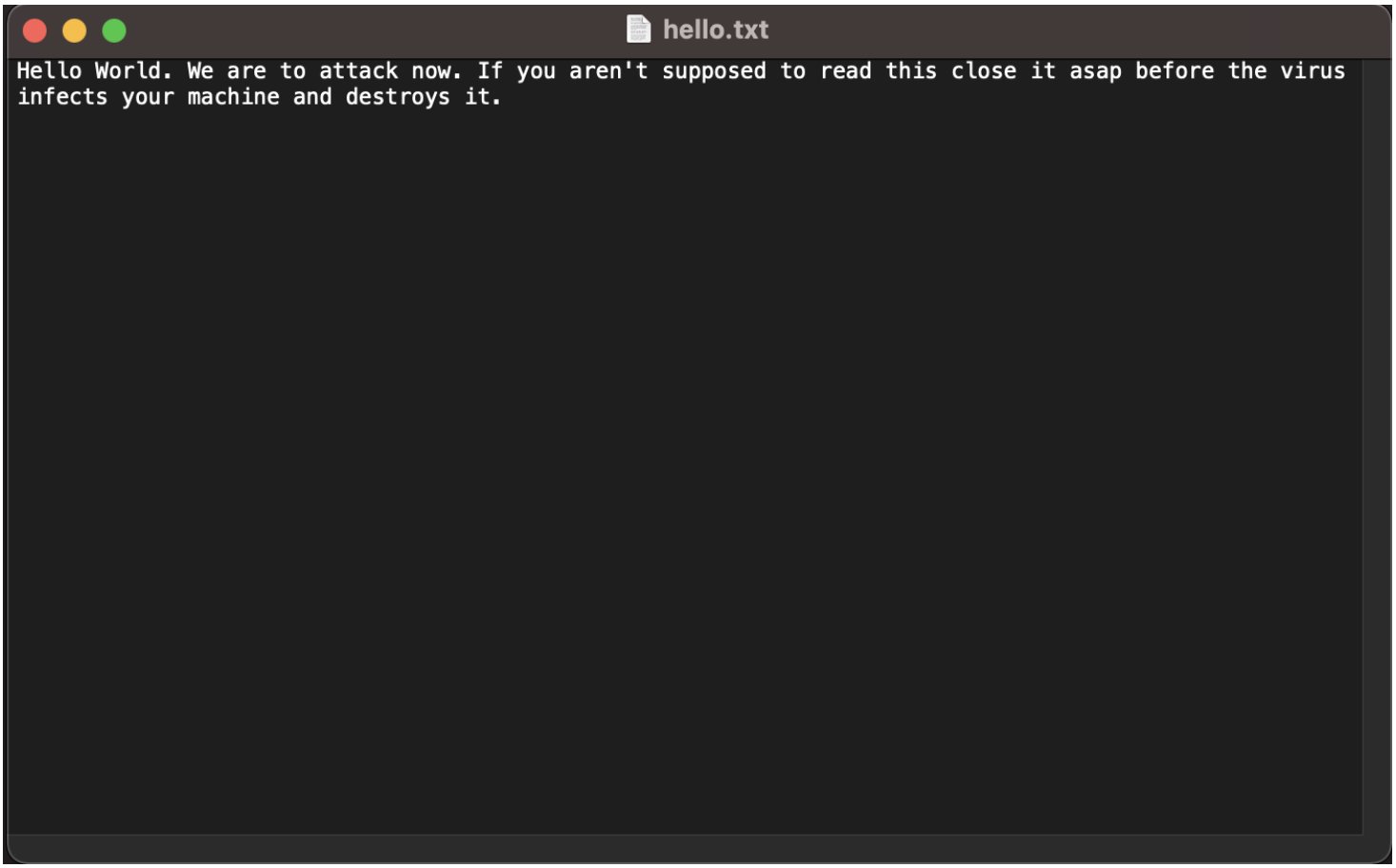
```
% openssl enc ciphertype -e -in plain.txt -out
                                cipher.bin \ -K
        00112233445566778889aabbccddeeff \
-iv 0102030405060708
```

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:

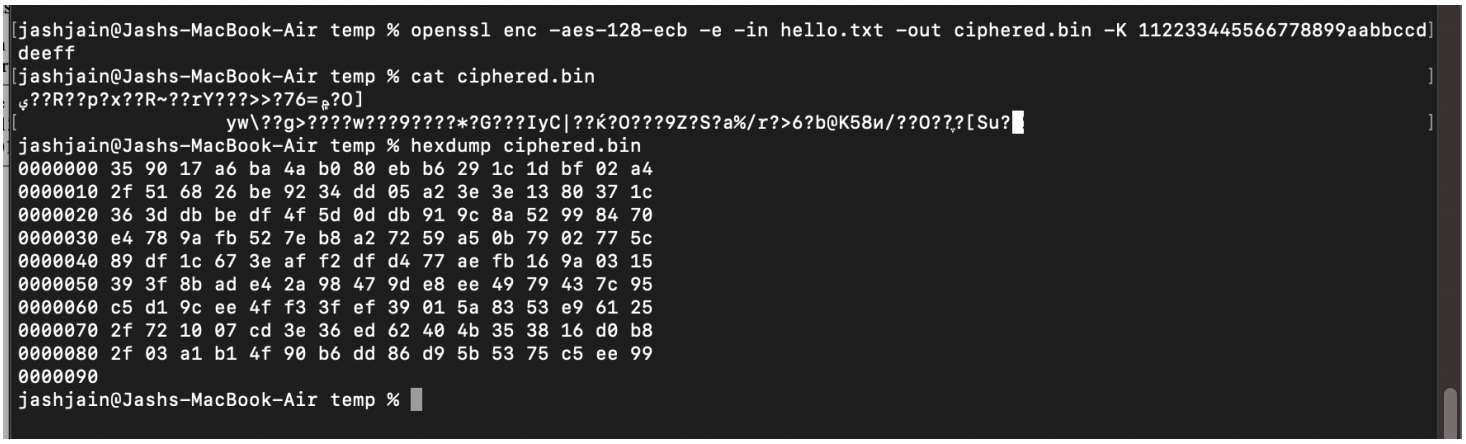
-in <file>	input file
-out <file>	output file
-e	encrypt
-d	decrypt
-K/-iv	key/iv in hex is the next argument
-[pP]	print the iv/key (then exit if -P)

Encryption of plain text using different Ciphers. The plain text is as follows:



1. Encryption using aes-128-ecb

The Encrypted text and The hexdump for the above cipher text is:



2. Encryption using aes-128-cbc

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

3. Encryption using aes-128-cfb

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

4. Encryption using aes-192-ecb

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

```
jashjain@Jashs-MacBook-Air temp % openssl enc -aes-192-ecb -e -in hello.txt -out ciphered.bin -K 112233445566778899aabbccddeeff
jashjain@Jashs-MacBook-Air temp % cat ciphered.bin
m3?*H2?????!jv??::?f?
?g?vr?|?<F??6 ~?       3??S??{?H?g?s???|?Po?5rP6[i6]???zA'u??~b?v$??j?q(?"?E?[IH9B?*vrGse'??|%
jashjain@Jashs-MacBook-Air temp % hexdump ciphered.bin
0000000 99 b7 0d 6d 33 9b 2a 48 32 97 ab f8 17 fb 21 05
0000010 6a 76 ed d4 3a 18 3a dd c8 93 ec aa 0a a7 0e 0f
0000020 67 e6 03 76 72 93 7c fb 3c 46 18 8b 02 84 de ab
0000030 08 c0 36 ed 89 8d 7e f1 ac 1e 19 09 d5 85 d9 d7
0000040 ae c8 53 d3 fb 7b b6 48 a5 67 d8 73 c0 ca ef 7c
0000050 ab 50 6f 9c 35 72 50 36 5b 69 36 c3 08 5d b2 a6
0000060 d2 15 7a 41 d5 9a 06 75 c3 d4 cb 98 62 12 05 ce
0000070 76 24 8a d6 6a 85 71 28 ce 22 9c 45 fd 5b 49 48
0000080 39 42 f2 2a 76 72 47 12 73 65 27 a1 17 81 7c 05
0000090
jashjain@Jashs-MacBook-Air temp %
```

5. Encryption using aes-192-cbc

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

```
jashjain@Jashs-MacBook-Air temp % openssl enc -aes-192-cbc -e -in hello.txt -out ciphered.bin -K 112233445566778899aabbccddeeff -iv 01020304050607080
jashjain@Jashs-MacBook-Air temp % cat ciphered.bin
???j??f??I
?4        ?a?(v?Ur??P????K??h&?w?Tyy$"?"a?U3v??????>?P
  r?8P1??63wd~???WI?o+n?jy????.yU????2e??????F??
                                          ??\?q???`T?.?P&?*%
jashjain@Jashs-MacBook-Air temp % hexdump ciphered.bin
0000000 a1 f1 d3 6a d3 ea 66 04 bb 7f 9d 49 0b 02 8a 61
0000010 cf 28 76 cd 55 72 11 1a df c9 50 fe 14 81 b9 91
0000020 4b c8 d4 68 26 bb 12 77 11 9e 54 79 79 24 22 b1
0000030 22 16 61 11 7f f0 aa 55 33 76 a9 bc a1 93 8e 89
0000040 3e 01 a3 50 0d c1 34 0b 72 ab cf a8 50 1d 31 a0
0000050 ab d7 a9 36 33 64 7e e6 03 9f f1 57 49 a8 6f 2b
0000060 6e ce 6a 79 a4 cd ce dd 2e 79 55 90 cd 16 c4 32
0000070 9b 65 07 ed c9 15 a3 a4 a7 ee 46 a0 02 cc 0c ab
0000080 87 5c dd 71 88 ad f4 60 54 d5 2e 84 50 26 fc 2a
0000090
jashjain@Jashs-MacBook-Air temp %
```

6. Encryption using aes-192-cfb

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

```
jashjain@Jashs-MacBook-Air temp % openssl enc -aes-192-cfb -e -in hello.txt -out ciphered.bin -K 112233445566778899aabbccddeeff -iv 01020304050607080
jashjain@Jashs-MacBook-Air temp % cat ciphered.bin
'??????9.P;B?$(p?0??)??E?T2\
?eG9?X?u???X??              ?@V:w???t?
ʰIC?_3i?Dv?~?$^D"63@??+=?hd??a?d?C?g/F??I?#Y??o`?z?ӕf??a??@%
jashjain@Jashs-MacBook-Air temp % hexdump ciphered.bin
0000000 27 c0 a4 b1 c9 11 86 18 81 39 15 2e 50 11 3b 42
0000010 b1 24 28 0e 70 f0 30 96 c5 29 b0 f8 45 ee 54 32
0000020 5c 0c 9d 40 56 3a 77 b3 fa d6 74 ee 0d a4 65 ce
0000030 08 47 39 dc 58 e6 c6 b0 b9 81 ca 58 fa b6 0b dc
0000040 c3 a4 7c bd 88 b2 0d 11 df a4 49 43 ea 5f 33 69
0000050 a4 44 76 98 7e 89 24 5e 44 22 36 33 40 ef 93 d7
0000060 2b 3d b7 68 64 97 e1 16 61 8e 64 fc 43 8b 67 2f
0000070 0f 46 c6 e9 ba 49 ef 23 59 d3 10 f1 6f 60 cf 7a
0000080 01 8b c7 a3 66 96 d3 61 9f bc 40
000008b
jashjain@Jashs-MacBook-Air temp %
```

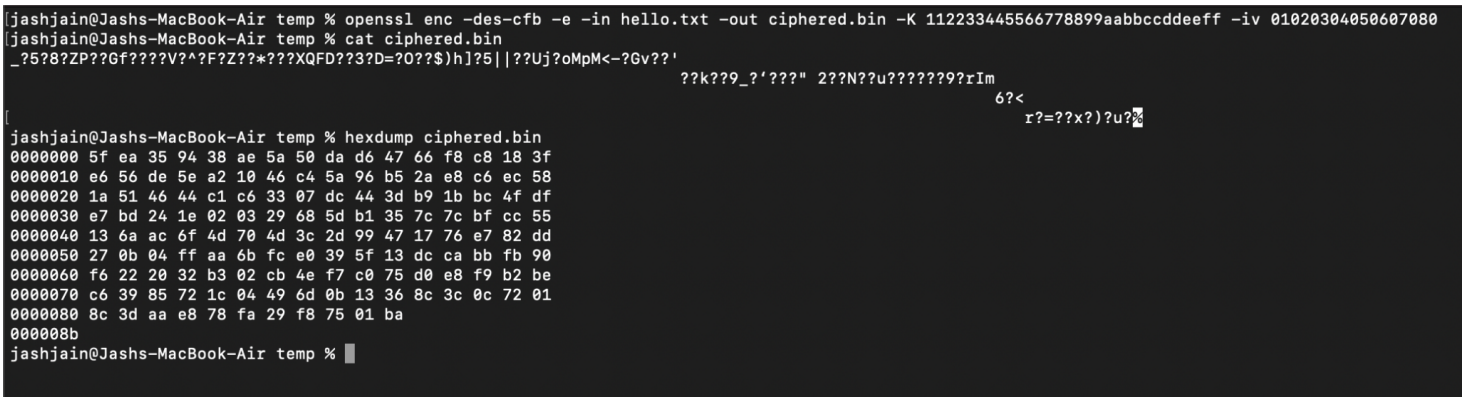
7. Encryption using des-ecb

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

```
jashjain@Jashs-MacBook-Air temp % openssl enc -des-ecb -e -in hello.txt -out ciphered.bin -K 112233445566778899aabbccddeeff
jashjain@Jashs-MacBook-Air temp % cat ciphered.bin
??????à??F?i???dpm)9????Z??}???9???q?}??\9&$¿?.??
                                          ?l{^B?l??p?Q6Px????u?ı??^?k?????3?;S}????7HPc*
                                                                                    ö?t?l?Ÿ?[+c\?tXM??Iép??%
jashjain@Jashs-MacBook-Air temp % hexdump ciphered.bin
0000000 8b 1c 1b 9a 9c db f0 bc f7 f0 c3 a0 d1 c0 46 0f
0000010 ac 69 85 eb a1 07 fe 64 70 6d 29 39 8d 07 fb ab
0000020 a5 5a b5 9c 7d 1d 9f 05 8c 39 f7 84 b8 71 e7 7d
0000030 ae c0 5c 39 26 24 da a8 a1 1e 2e 9b f1 0b 82 6c
0000040 7b 5e 42 f6 6c e8 d6 70 bf 51 d0 b1 50 78 89 b1
0000050 97 be 75 fe da 90 94 e4 5e 93 6b 15 b3 de e0 9e
0000060 d2 33 ef 3b 53 7d fe 00 b2 91 fe 37 48 1b 8b 50
0000070 63 2a 0c c3 b6 ab 74 fe 6c b7 d3 b2 93 5b 2b 63
0000080 5c 1b 85 f8 74 58 4d c1 db 49 ce ad 70 04 b4 de
0000090
jashjain@Jashs-MacBook-Air temp %
```

8. Encryption using des-cfb

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



9. Encryption using des-cbc

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

Observations:

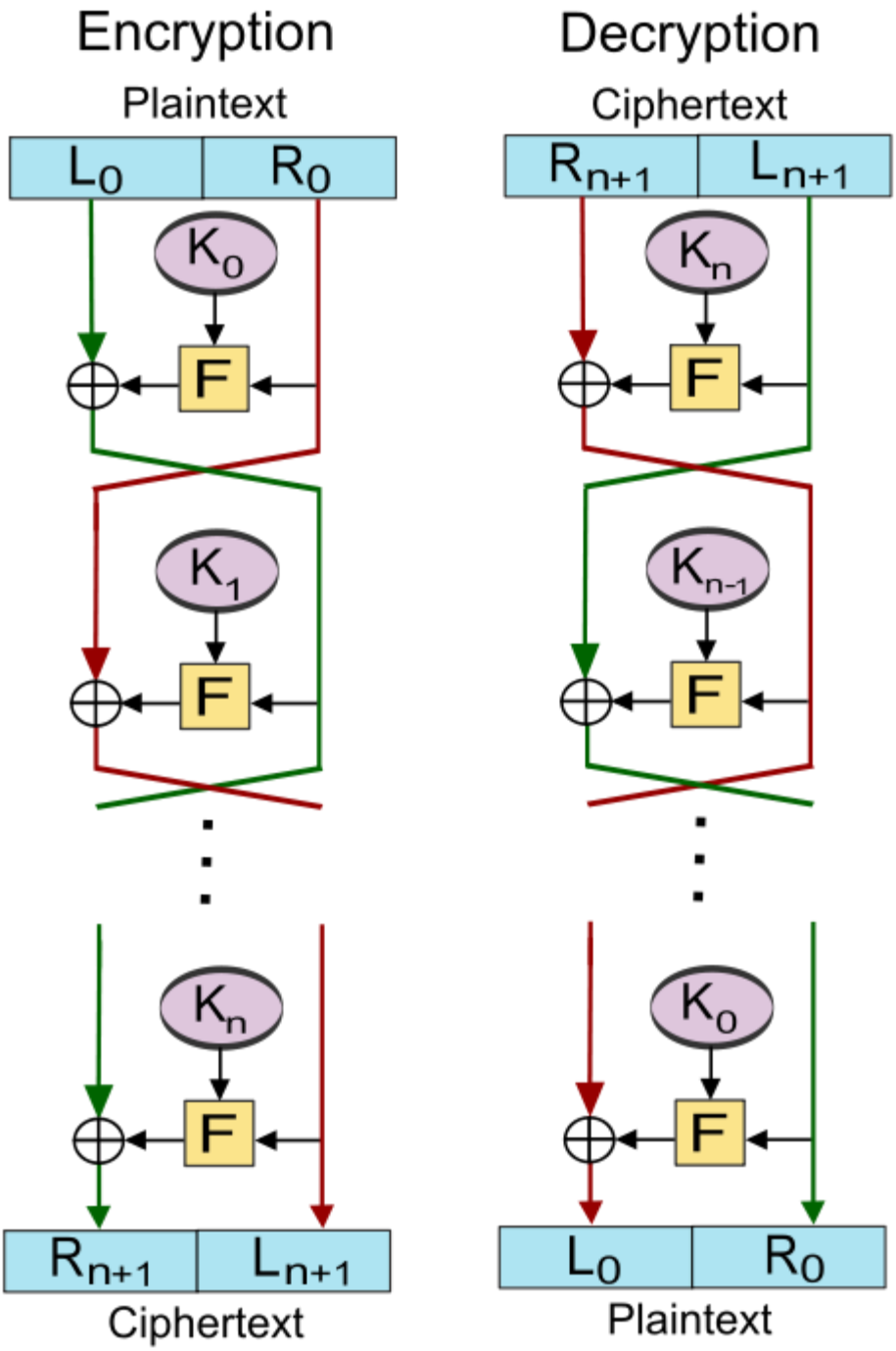
1. AES comprises a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations). AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys.

2.AES adds more redundancy as compared to DES as length of key in AES has to be at least 128 and can go upto 256 bits However DES uses a 64-bit key 8 × 8 including 1 bit for parity, so the actual key is 56 bits

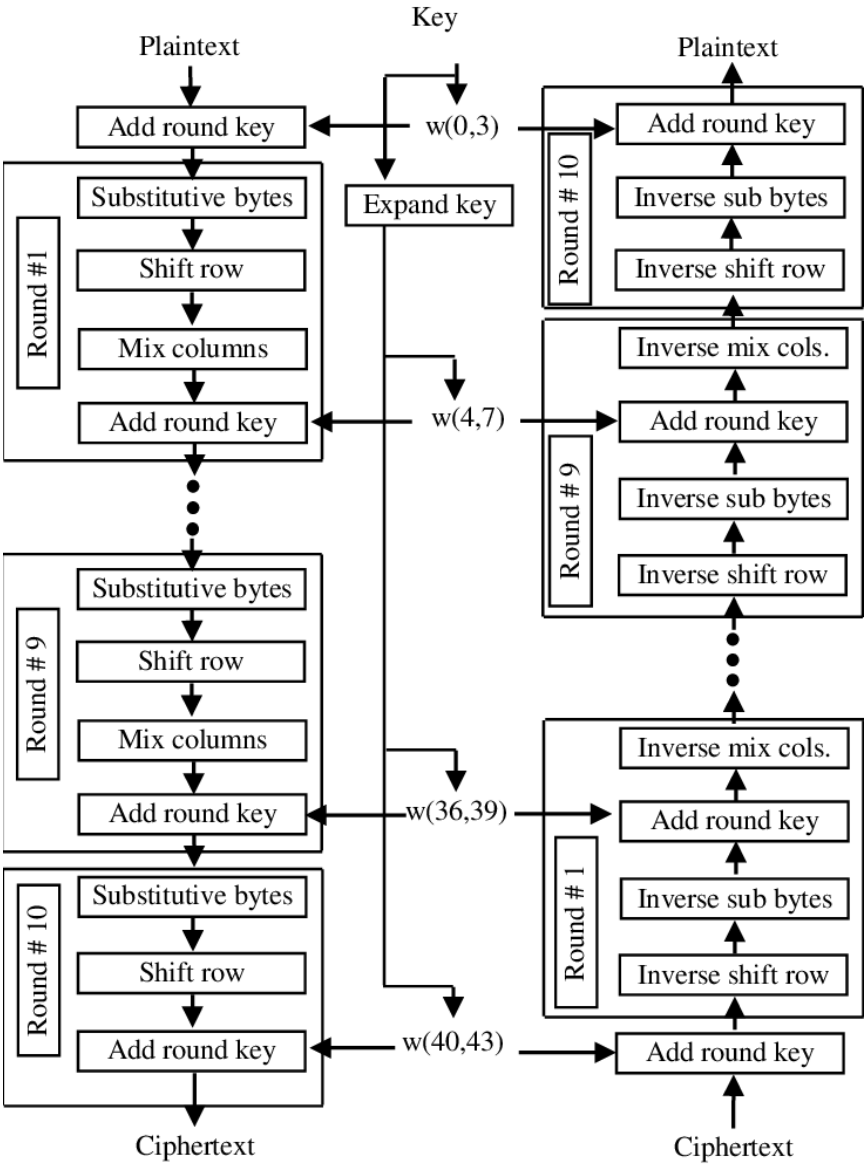
3.In terms of structure, DES uses the Feistel network(Fig 1.0) which divides the block into two halves before going through the encryption steps. AES(Fig 1.1) on the other hand, uses

permutation-substitution, which involves a series of substitution and permutation steps to create the encrypted block

Fiestel Network(Fig 1.0)



AES(Fig 1.1)



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.

Original Picture(Fig 1.2)

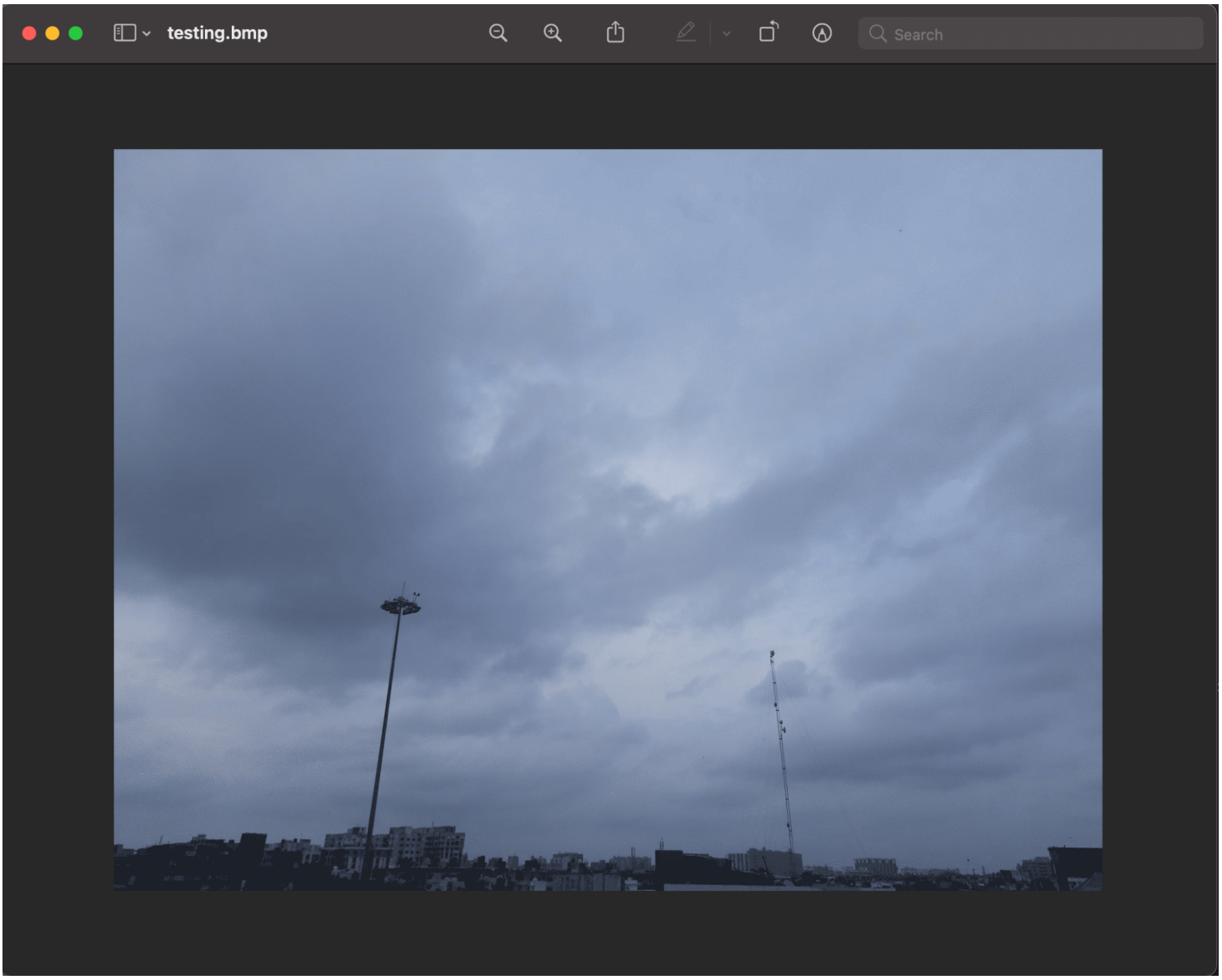


Image Encrypted using ecb(Fig 1.3)

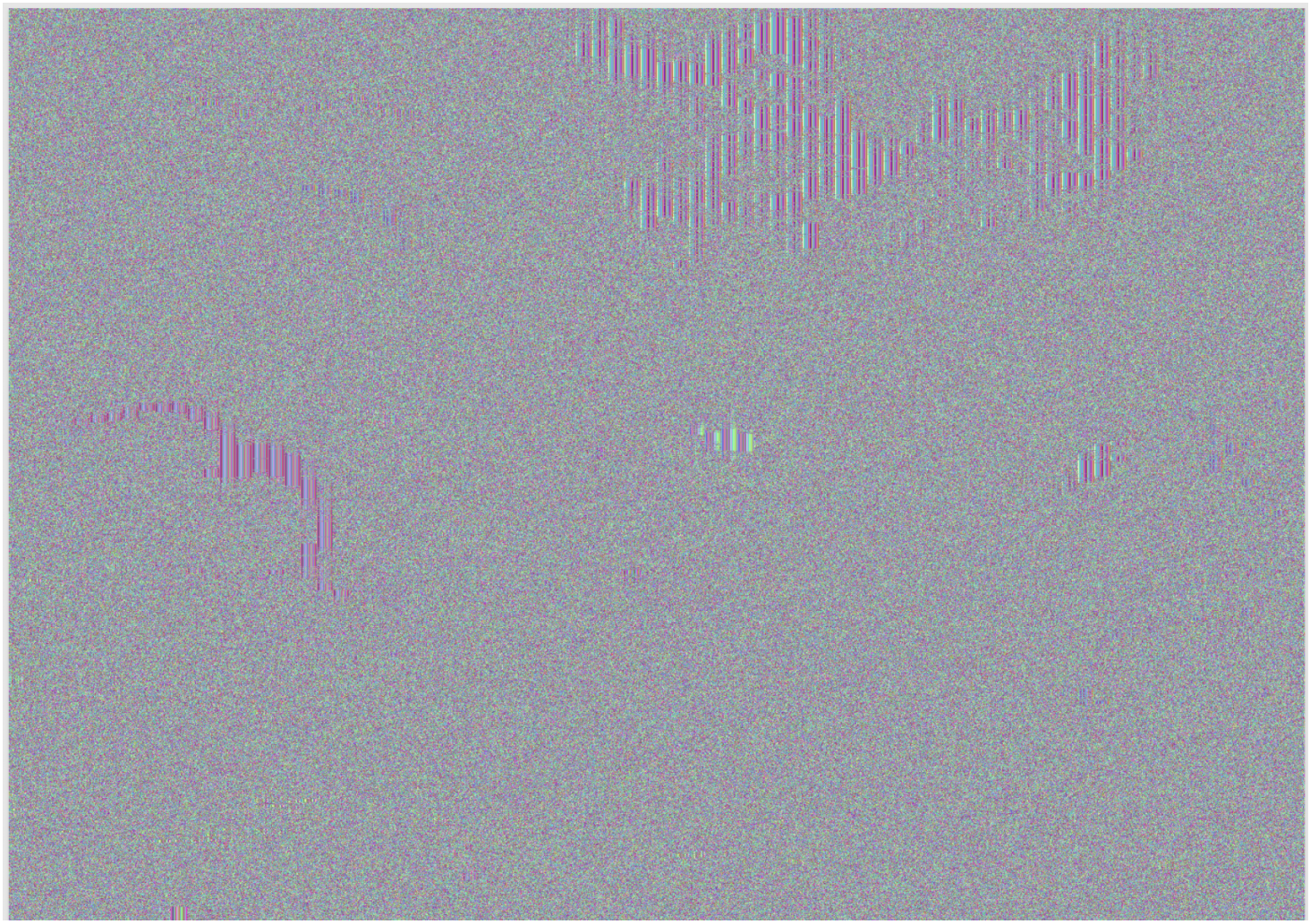
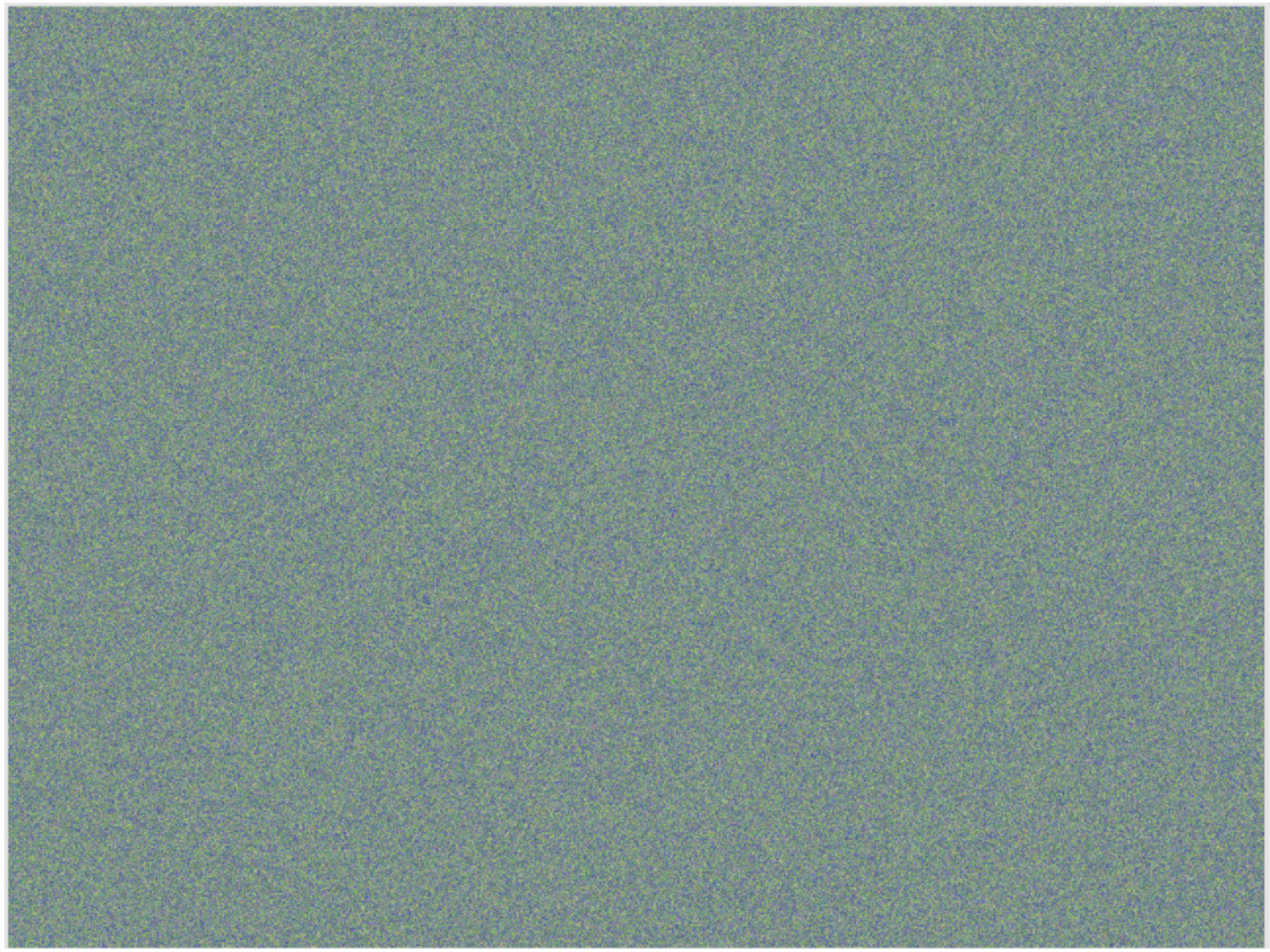


Image encryption using cbc (Fig 1.4)

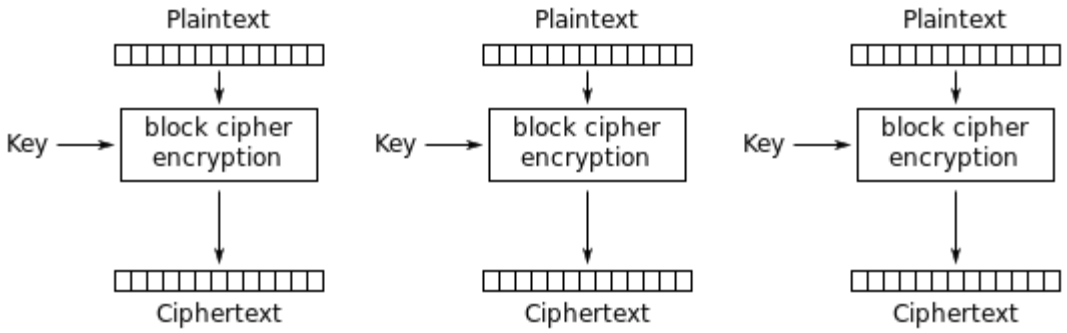


Observations:

1) In the image encrypted using ECB(Fig 1.3), 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.

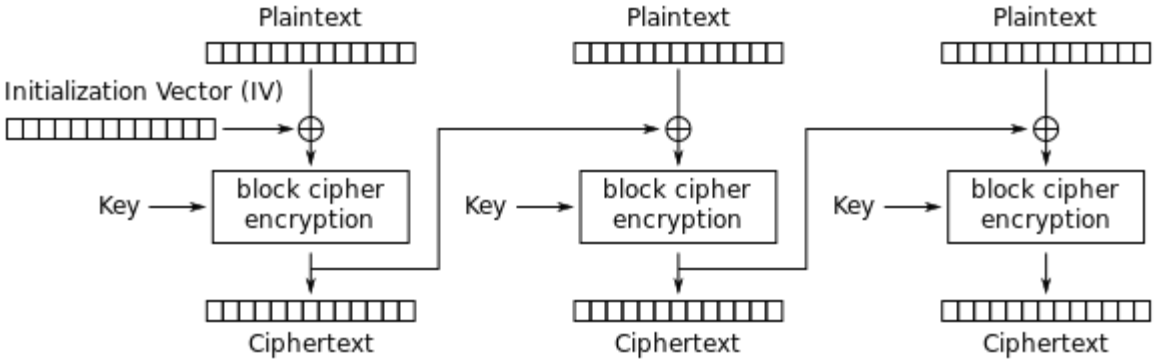
2) In the image encryption using CBC(Fig 1.4) the entire image is distorted and there is no way by which we can try to figure out the original picture.

3) The reason behind the visibility of the overall structure in the image encrypted using ECB is because we use the same key on each and every block and get the output for each of them.



Electronic Codebook (ECB) mode encryption

4) In CBC, the overall structure of the original image is not transferred to the encrypted image. This means that it is impossible to guess the original image based on the encrypted image. This mainly happens because in CBC, we XOR every block with the last encrypted block and then encrypt the current block, this helps in breaking patterns in the data and makes it more secure.



Cipher Block Chaining (CBC) mode encryption

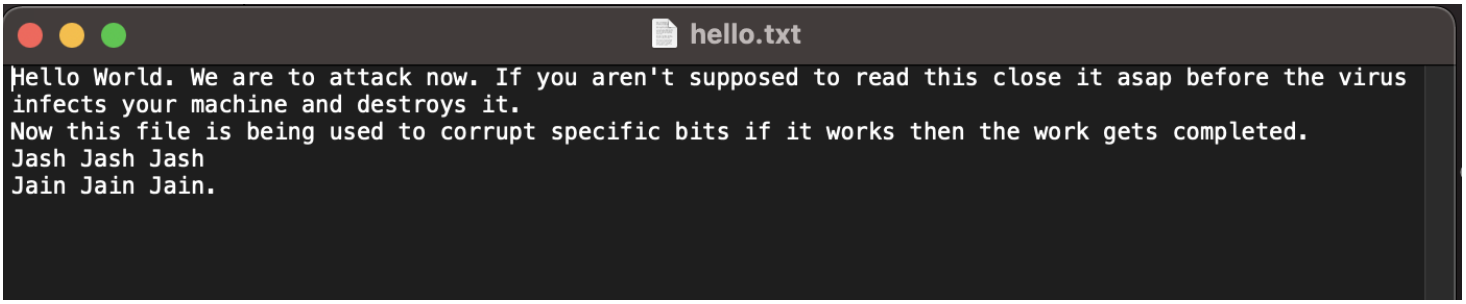
3.3 Task 3: Encryption Mode – Corrupted Cipher Text

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.

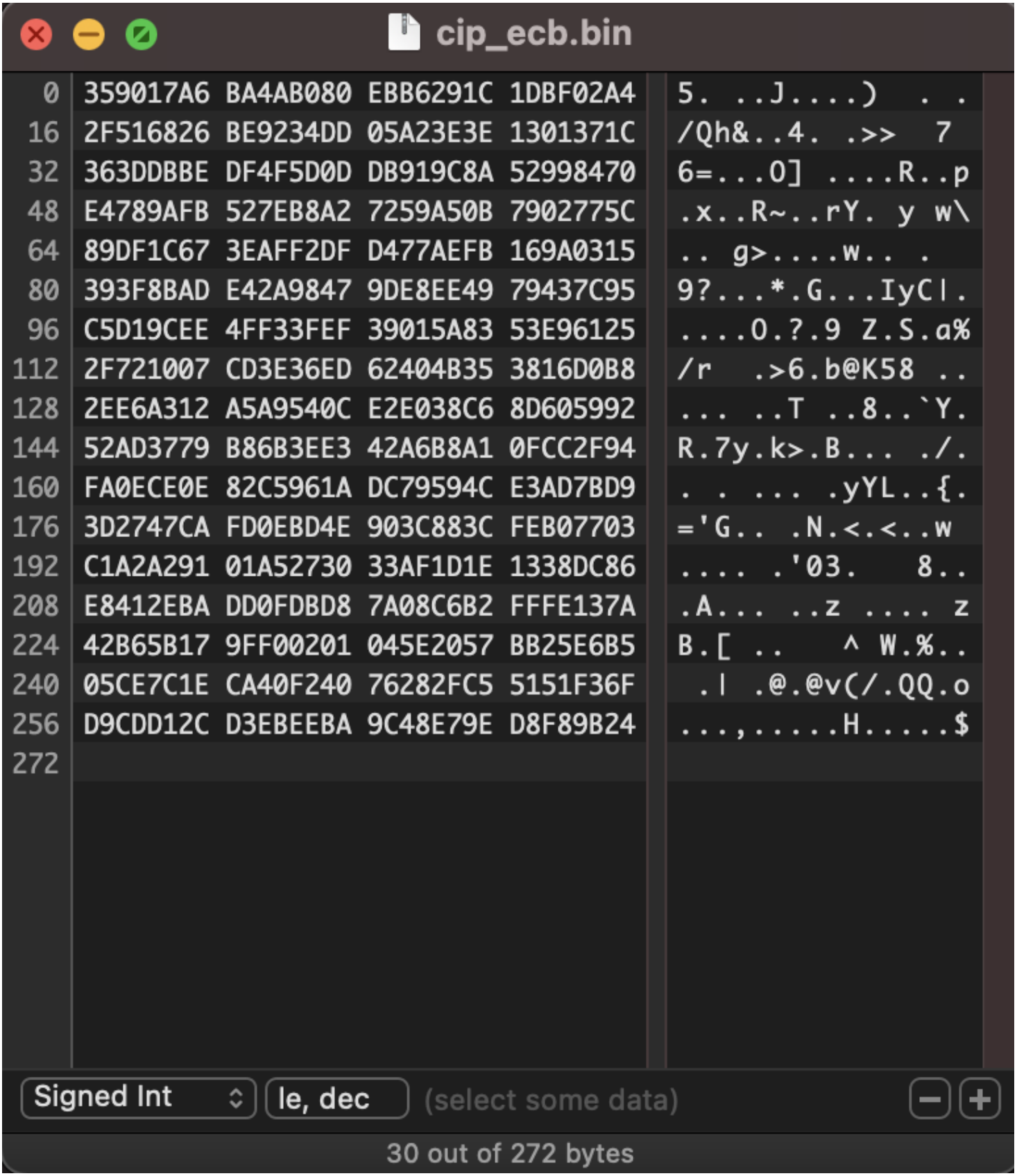
Please answer the following questions: (1) How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, CFB, or OFB, respectively? Please answer this question before you conduct this task, and then find out whether your answer is correct or wrong after you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

The original plain text is as follows:

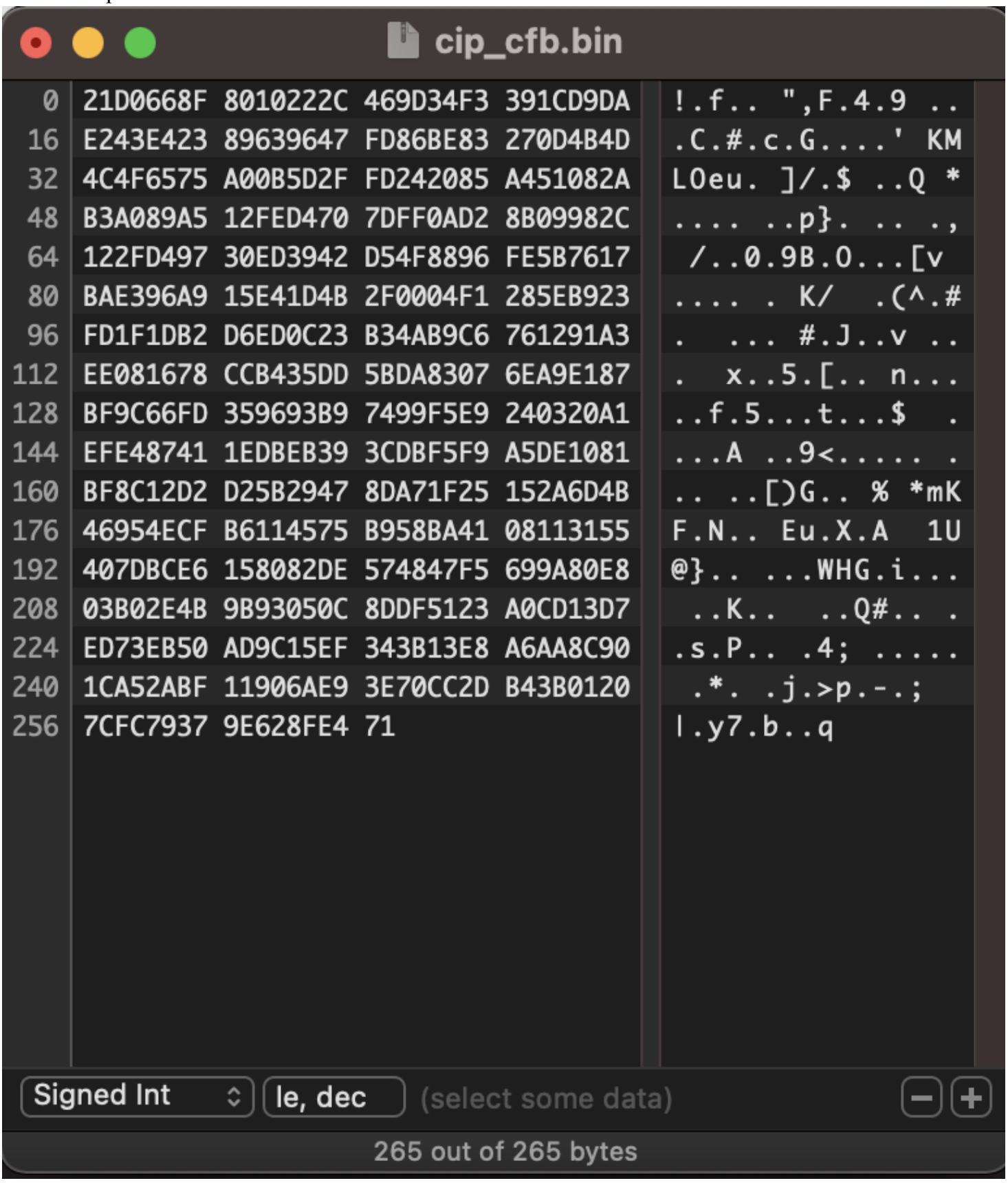


The above text is encrypted using aes-128 in 4 different modes: cbc,cfb,ofb and ecb. After corrupting 30th byte of each cbc, cfb, ofb and ecb :

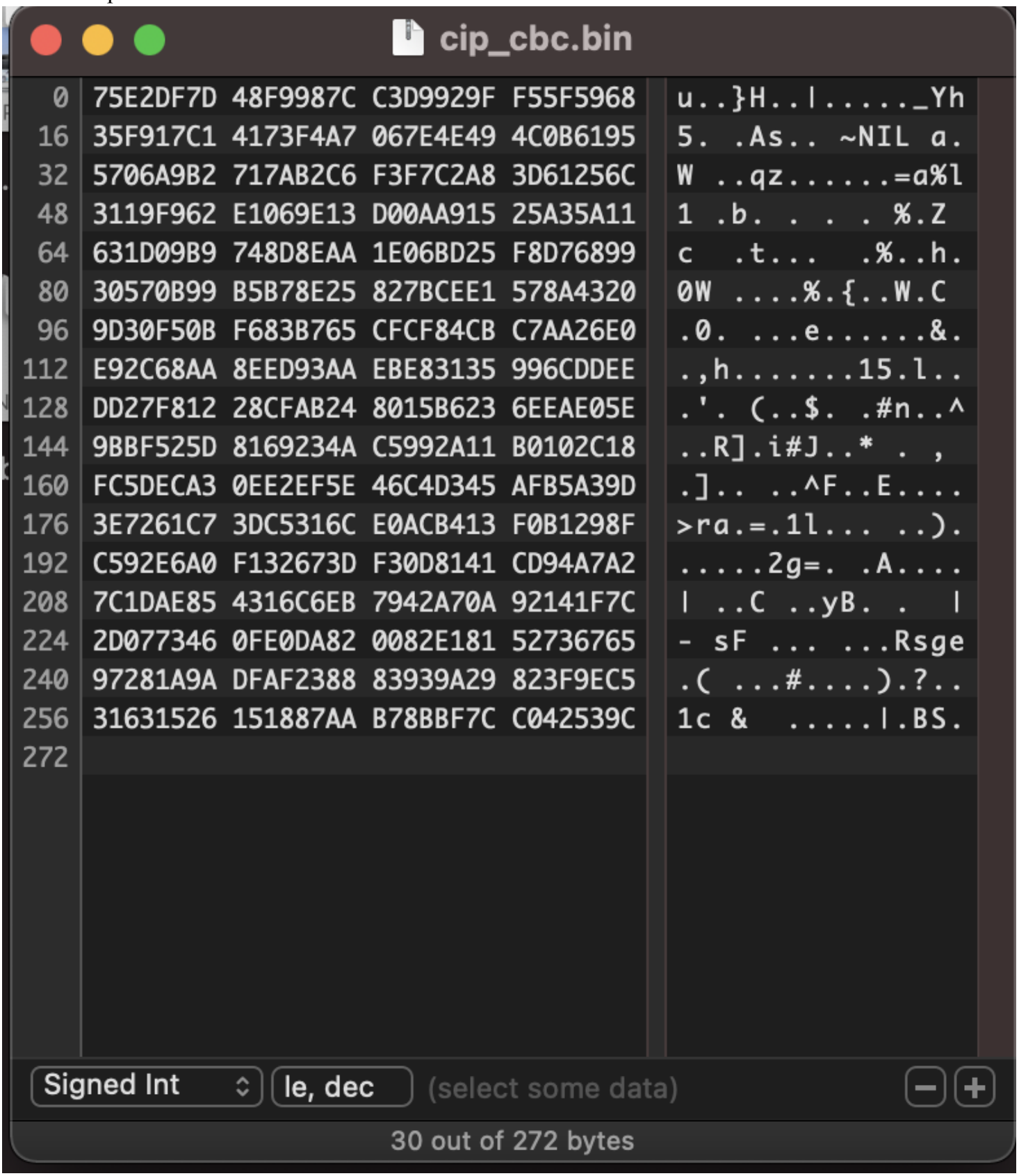
ECB Corruption of file



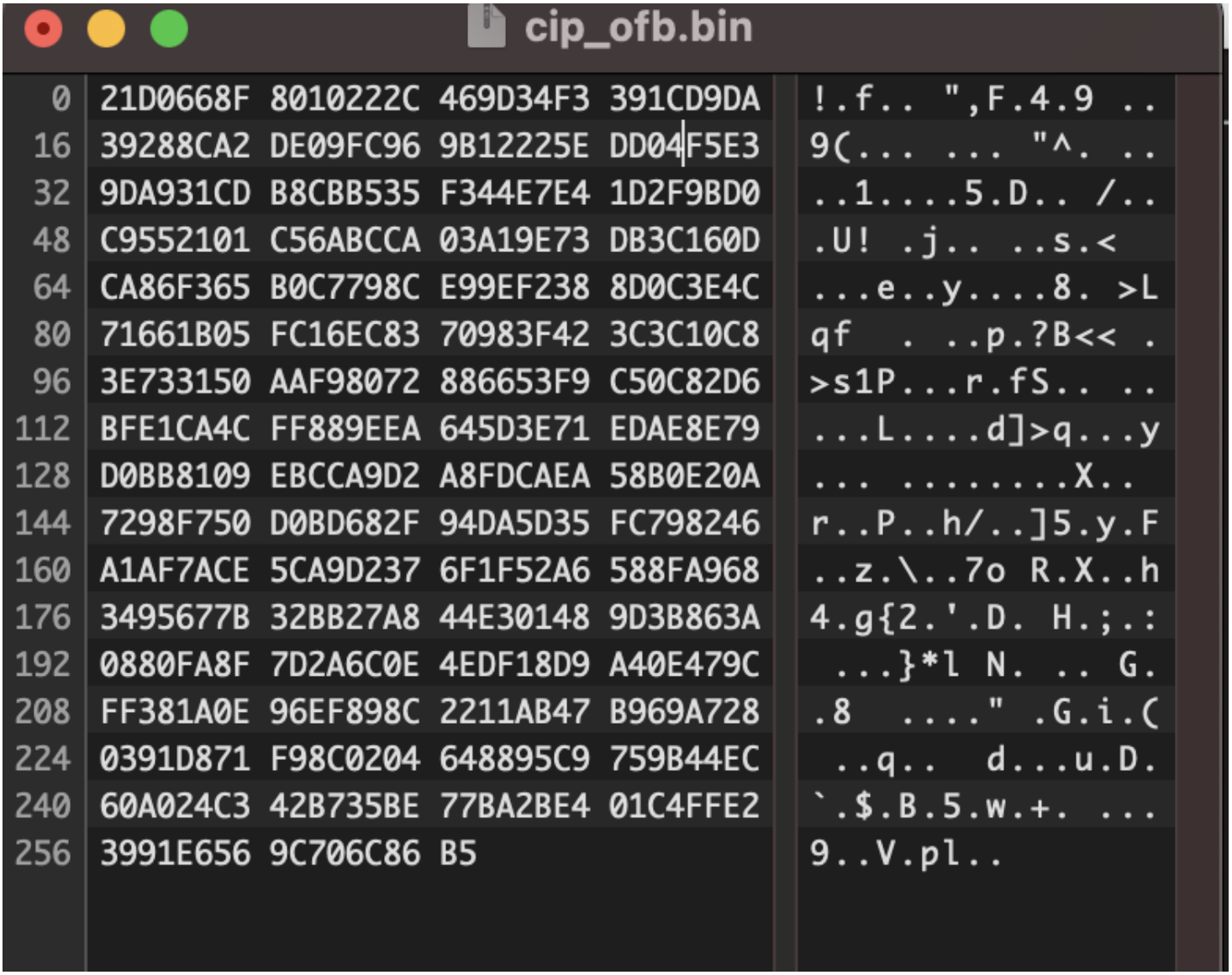
CFB Corruption of file



CFB Corruption of File



OFB Encryption of File

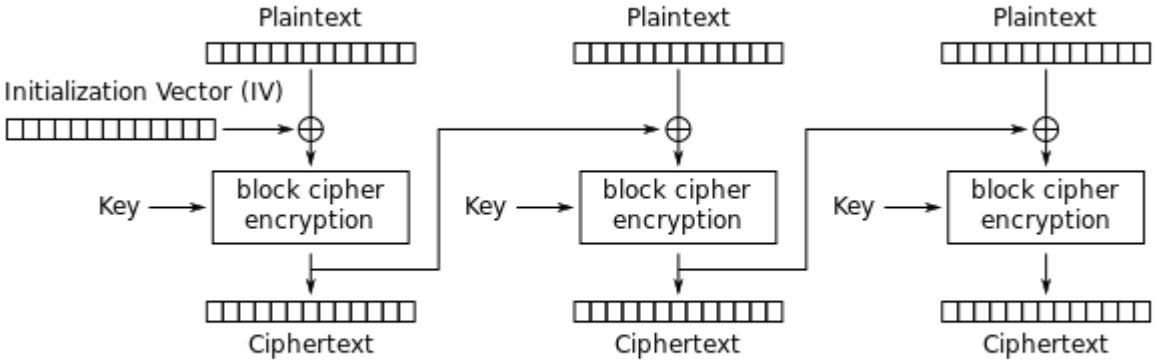


The decrypted text for the same is as follows:

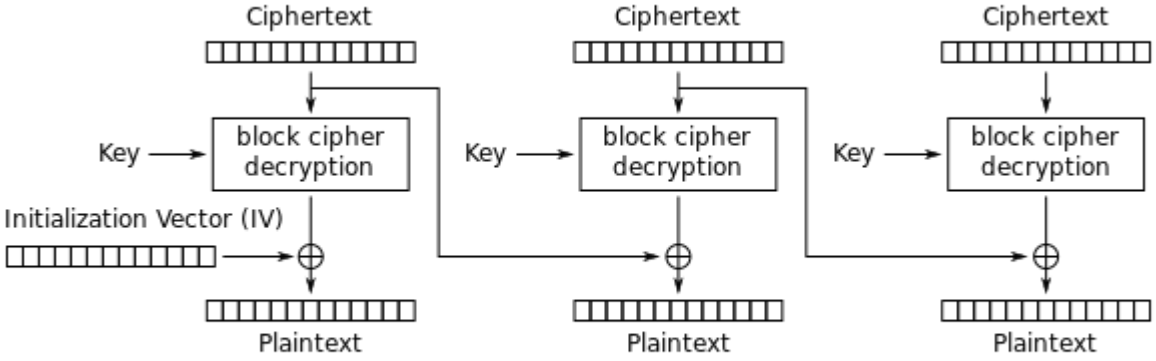
CBC

```
(base) jashjain@Jashs-MacBook-Air temp % openssl enc -aes-128-cbc -d -in cip_cbc.bin -out cbc.txt -K 112233445566778899aabbccddeeff -iv 01020304050607080
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
(base) jashjain@Jashs-MacBook-Air temp % cat cbc.txt
Hello World. We r??!???vr??w. If you are/'t supposed to read this close it asap before the virus infects your machine and destroys it.
Now this file is being used to corrupt specific bits if it works then the work gets completed.
Jash Jash Jash
Jain Jain Jain.%
(base) jashjain@Jashs-MacBook-Air temp %
```

In CBC every encrypted block cipher depends on the previous encrypted block cipher. However while decrypting, we perform XOR on the decrypted block by using the previous cipher text. Hence a one-bit change to the ciphertext causes complete corruption of the corresponding block of plaintext, and inverts the corresponding bit in the following block of plaintext, but the rest of the blocks remain intact.



Cipher Block Chaining (CBC) mode encryption

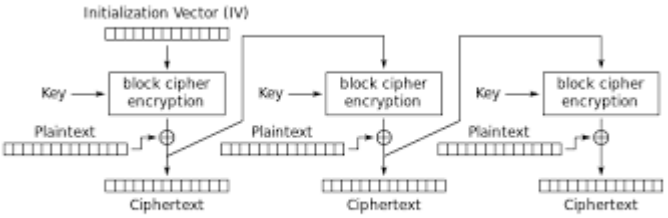


Cipher Block Chaining (CBC) mode decryption

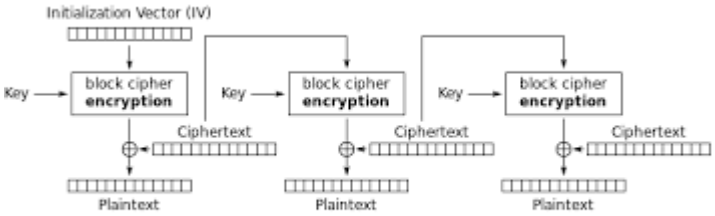
CFB

```
(base) jashjain@Jashs-MacBook-Air temp % openssl enc -aes-128-cfb -d -in cip_cfb.bin -out cfb.txt -K 112233445566778899aabbccddeeff -iv 01020304050607080
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
(base) jashjain@Jashs-MacBook-Air temp % cat cfb.txt
Hello World. We are to attack?no|?u:H????Py?? supposed to read this close it asap before the virus infects your machine and destroys it.
Now this file is being used to corrupt specific bits if it works then the work gets completed.
Jash Jash Jash
Jain Jain Jain.%
(base) jashjain@Jashs-MacBook-Air temp %
```

Loss in CFB is much less as compared to ECB and CBC. This is because the same encryption algorithm is used for decryption and the corrupted ciphertext for a block is used for two corresponding blocks hence limiting the damage to two blocks of cipher text.



Cipher Feedback (CFB) mode encryption

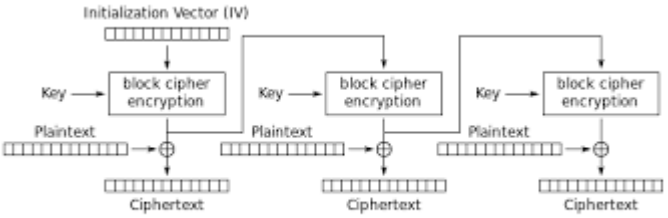


Cipher Feedback (CFB) mode decryption

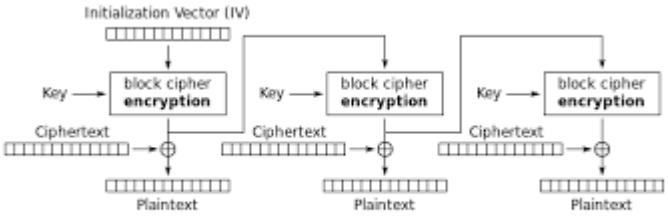
OFB

```
jashjain@Jashs-MacBook-Air temp % openssl enc -aes-128-ofb -e -in hello.txt -out cip_ofb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
jashjain@Jashs-MacBook-Air temp % openssl enc -aes-128-ofb -d -in cip_ofb.bin -out ofb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
jashjain@Jashs-MacBook-Air temp % cat ofb.txt
Hello World. We are to attackhnow. If you aren't supposed to read this close it asap before the virus infects your machine and destroys it.
Now this file is being used to corrupt specific bits if it works then the work gets completed.
Jash Jash Jash
Jain Jain Jain.%
jashjain@Jashs-MacBook-Air temp %
```

The data lost in OFB is the least as compared to other block cipher modes. This is mainly because flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location. This property allows many error-correcting codes to function normally even when applied before encryption.



Output Feedback (OFB) mode encryption



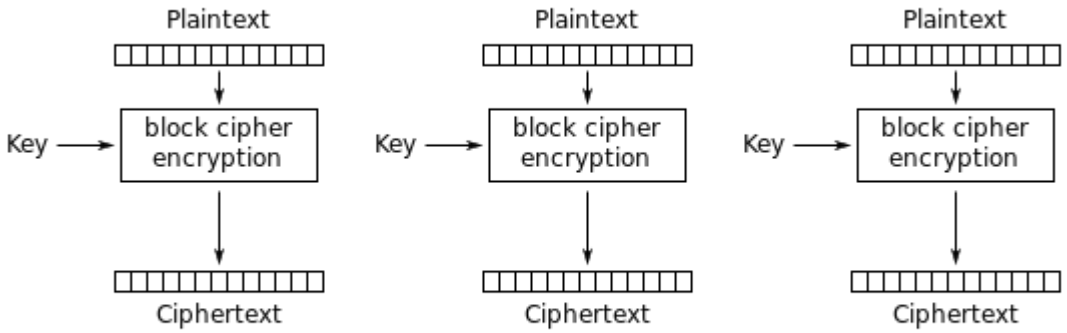
Output Feedback (OFB) mode decryption

ECB

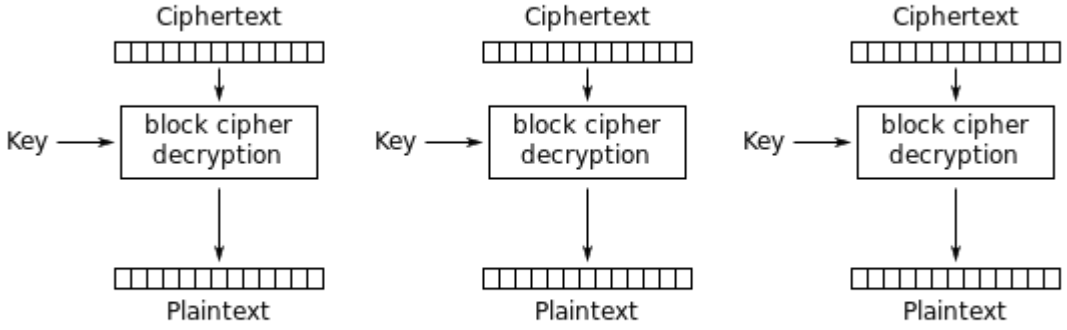
```
(base) jashjain@Jashs-MacBook-Air temp % openssl enc -aes-128-ecb -d -in cip_ecb.bin -out ecb.txt -K 112233445566778899aabbccddeeff
hex string is too short, padding with zero bytes to length
(base) jashjain@Jashs-MacBook-Air temp % cat ecb.txt
Hello World. We )'?@??????槭Rw. If you aren't supposed to read this close it asap before the virus infects your machine and destroys it.
Now this file is being used to corrupt specific bits if it works then the work gets completed.
Jash Jash Jash
Jain Jain Jain.%
(base) jashjain@Jashs-MacBook-Air temp %
```

From the results above I can conclude that:

In ECB mode, since every block cipher is encrypted separately using the key, when one byte is corrupted and then decrypted, it only affects the corresponding block of data while the other blocks of data are unaffected.



Electronic Codebook (ECB) mode encryption



Electronic Codebook (ECB) mode decryption

3.4 Task4 : Padding

For block ciphers, when the size of the plain text 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:

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.

1. 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.

```
(base) jashjain@Jashs-MacBook-Air temp % cd padding
(base) jashjain@Jashs-MacBook-Air padding % ls -l
total 16
-rw-r--r--@ 1 jashjain  staff  32 Nov  8 07:23 large.txt
-rw-r--r--@ 1 jashjain  staff  20 Nov  8 07:24 small.txt
(base) jashjain@Jashs-MacBook-Air 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

Fig(1.4)

```
(base) jashjain@Jashs-MacBook-Air padding % ls -l
total 80
-rw-r--r--  1 jashjain  staff  32 Nov  8 07:26 cbc_20.txt
-rw-r--r--  1 jashjain  staff  48 Nov  8 07:27 cbc_32.txt
-rw-r--r--  1 jashjain  staff  20 Nov  8 07:26 cfb_20.txt
-rw-r--r--  1 jashjain  staff  32 Nov  8 07:26 cfb_32.txt
-rw-r--r--  1 jashjain  staff  32 Nov  8 07:28 ecb_20.txt
-rw-r--r--  1 jashjain  staff  48 Nov  8 07:28 ecb_32.txt
-rw-r--r--@ 1 jashjain  staff  32 Nov  8 07:23 large.txt
-rw-r--r--  1 jashjain  staff  20 Nov  8 07:27 ofb_20.txt
-rw-r--r--  1 jashjain  staff  32 Nov  8 07:27 ofb_32.txt
-rw-r--r--@ 1 jashjain  staff  20 Nov  8 07:24 small.txt
(base) jashjain@Jashs-MacBook-Air padding %
```

1. The cipher modes which require encryption are CBC and ECB as we can see that for small as well as large text files, it is adding padding and hence increasing the size of the file to be an exact multiple of the block size, which is 16 Bytes for AES.

2. We can confirm that the padding standard is PKCS5 because, whenever a mode which requires padding is used, PKCS5 adds at least 1 Byte of padding and at most n Bytes of padding where n is the block size. In short, it adds padding to the file so that its size becomes the next closest multiple of block size.

3. We can observe this in figure 1.4, as the size of large.txt is increased to 48 Bytes (next multiple of 16 after 32) and 32 Bytes (next multiple of 16 after 20) for small.txt for CBC and ECB

4.We can observe this in figure 1.4, as the size of large.txt and small.txt for CFB and OFB remains the same as default(32 bytes and 20 bytes respectively) since they are stream ciphers and do not require the size of file to be an exact multiple of block size.

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

Finally we can see the decrypted and encrypted files as follows:
Fig(1.5)

```
jashjain@Jashs-MacBook-Air padding % openssl enc -aes-128-cbc -d -nopad -in cbc_20.txt -out cbc_20_dec.txt -K 00112233445566778899aabbccddeeff -iv 01020304050607080
jashjain@Jashs-MacBook-Air padding % openssl enc -aes-128-cbc -d -nopad -in cbc_32.txt -out cbc_32_dec.txt -K 00112233445566778899aabbccddeeff -iv 01020304050607080
jashjain@Jashs-MacBook-Air padding % openssl enc -aes-128-ofb -d -nopad -in ofb_32.txt -out ofb_32_dec.txt -K 00112233445566778899aabbccddeeff -iv 01020304050607080
jashjain@Jashs-MacBook-Air padding % openssl enc -aes-128-ofb -d -nopad -in ofb_20.txt -out ofb_20_dec.txt -K 00112233445566778899aabbccddeeff -iv 01020304050607080
jashjain@Jashs-MacBook-Air padding % openssl enc -aes-128-cfb -d -nopad -in cfb_20.txt -out cfb_20_dec.txt -K 00112233445566778899aabbccddeeff -iv 01020304050607080
jashjain@Jashs-MacBook-Air padding % openssl enc -aes-128-cfb -d -nopad -in cfb_32.txt -out cfb_32_dec.txt -K 00112233445566778899aabbccddeeff -iv 01020304050607080
jashjain@Jashs-MacBook-Air padding % openssl enc -aes-128-ecb -d -nopad -in ecb_20.txt -out ecb_20_dec.txt -K 00112233445566778899aabbccddeeff
jashjain@Jashs-MacBook-Air padding % openssl enc -aes-128-ecb -d -nopad -in ecb_32.txt -out ecb_32_dec.txt -K 00112233445566778899aabbccddeeff
jashjain@Jashs-MacBook-Air padding % ls -l
total 144
-rw-r--r--  1 jashjain  staff  32 Nov  8 10:45 cbc_20.txt
-rw-r--r--  1 jashjain  staff  32 Nov  8 10:48 cbc_20_dec.txt
-rw-r--r--  1 jashjain  staff  48 Nov  8 10:48 cbc_32.txt
-rw-r--r--  1 jashjain  staff  48 Nov  8 10:48 cbc_32_dec.txt
-rw-r--r--  1 jashjain  staff  20 Nov  8 10:44 cfb_20.txt
-rw-r--r--  1 jashjain  staff  20 Nov  8 10:49 cfb_20_dec.txt
-rw-r--r--  1 jashjain  staff  32 Nov  8 10:44 cfb_32.txt
-rw-r--r--  1 jashjain  staff  32 Nov  8 10:49 cfb_32_dec.txt
-rw-r--r--  1 jashjain  staff  32 Nov  8 10:46 ecb_20.txt
-rw-r--r--  1 jashjain  staff  32 Nov  8 10:50 ecb_20_dec.txt
-rw-r--r--  1 jashjain  staff  48 Nov  8 10:46 ecb_32.txt
-rw-r--r--  1 jashjain  staff  48 Nov  8 10:50 ecb_32_dec.txt
-rw-r--r--@ 1 jashjain  staff  32 Nov  8 07:23 large.txt
-rw-r--r--  1 jashjain  staff  20 Nov  8 10:46 ofb_20.txt
-rw-r--r--  1 jashjain  staff  20 Nov  8 10:49 ofb_20_dec.txt
-rw-r--r--  1 jashjain  staff  32 Nov  8 10:46 ofb_32.txt
-rw-r--r--  1 jashjain  staff  32 Nov  8 10:49 ofb_32_dec.txt
-rw-r--r--@ 1 jashjain  staff  20 Nov  8 07:24 small.txt
jashjain@Jashs-MacBook-Air 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.

On decryption, by default PKCS5 padding is expected to be present and is removed; if the padding is not valid an error occurs.

-nopad means no attempt is made to remove padding; if padding is present, in your ciphertext, it is left in the data.
If encrypt was done (successfully) with -nopad you need -nopad on decrypt so you don't try to remove padding that isn't there and fail.

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/descrypt 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 pro- vides 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 an English word list from the Internet. We have also linked one on the web

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

page of this lab. The plaintext and ciphertext is in the following:

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:

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

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

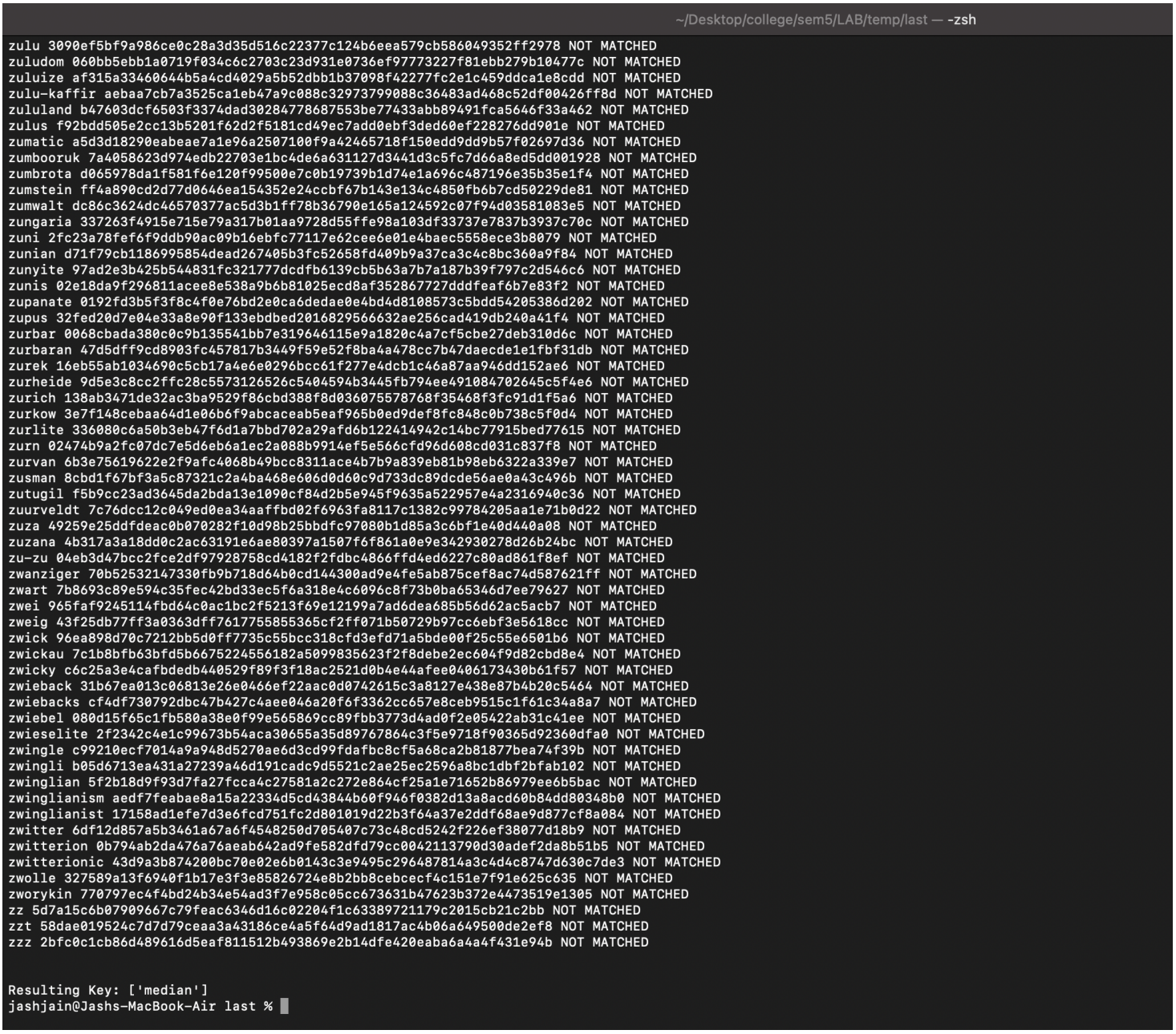
Code:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad

plain_text = b"This is a top secret."
cipher_hex = "8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9"

result_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)
```

Fig 1.6



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. If it is used to encrypt an image, an outline of the main contents of the image is still visible in the encrypted image.

3) ECB and CBC use padding while encryption while the other two(OFB and CFB) don't. This proves that ECB and CBC are block ciphers while CFB and OFB are stream ciphers. I could also verify that openssl uses PKCS5 standard for padding as it adds padding till the file size becomes

the next closest multiple of the block size.

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 we know the plaintext, ciphertext and iv, I can easily find the key using brute force method. In this case, we tried out each word with less than 16 characters from the dictionary and found the key to be “median” as seen in figure 1.6.