SAN JOSÉ STATE UNIVERSITY

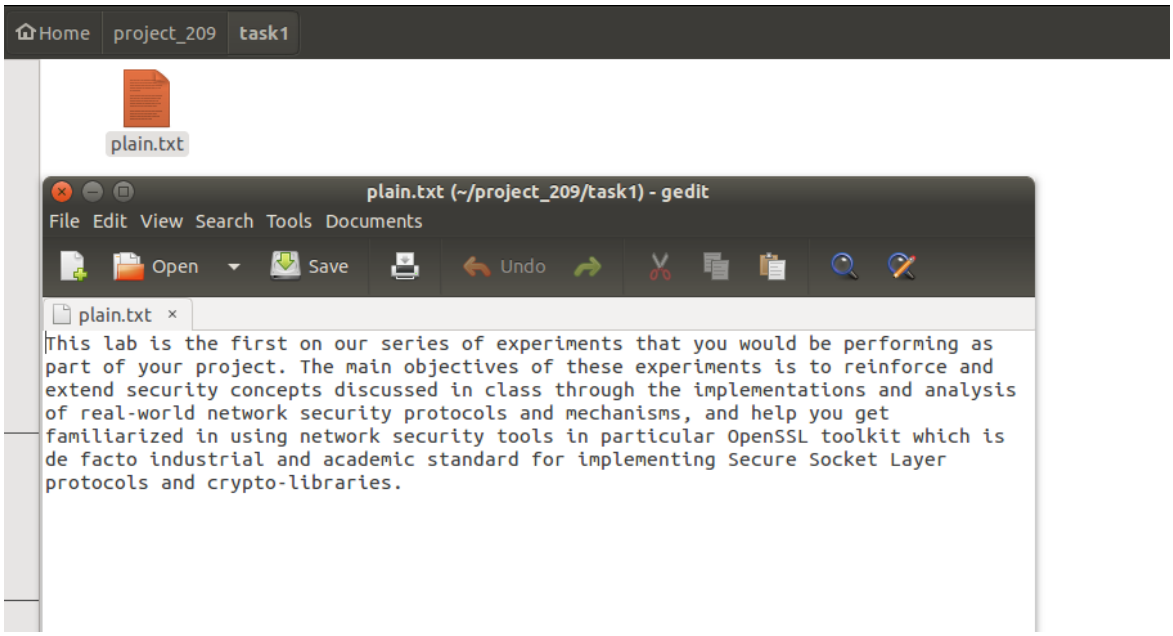
NETWORK SECURITY

Crypto Lab 1 – Secret-Key Encryption

IFUNANYA NNOKA – 010754159
RAHUL DILEEP JADHAV - 010687638

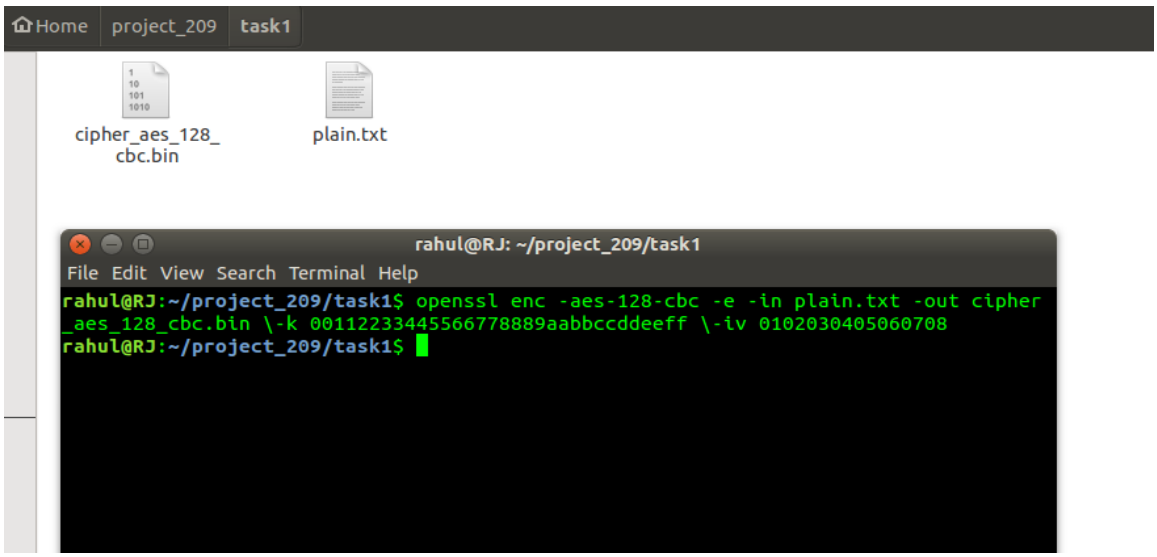
Task 1: Encryption using different ciphers and modes

The objective of the task is to encrypt a text file using at least 2 different ciphers and 3 different modes and observe the results. To implement this, we have taken a plain text as shown below.

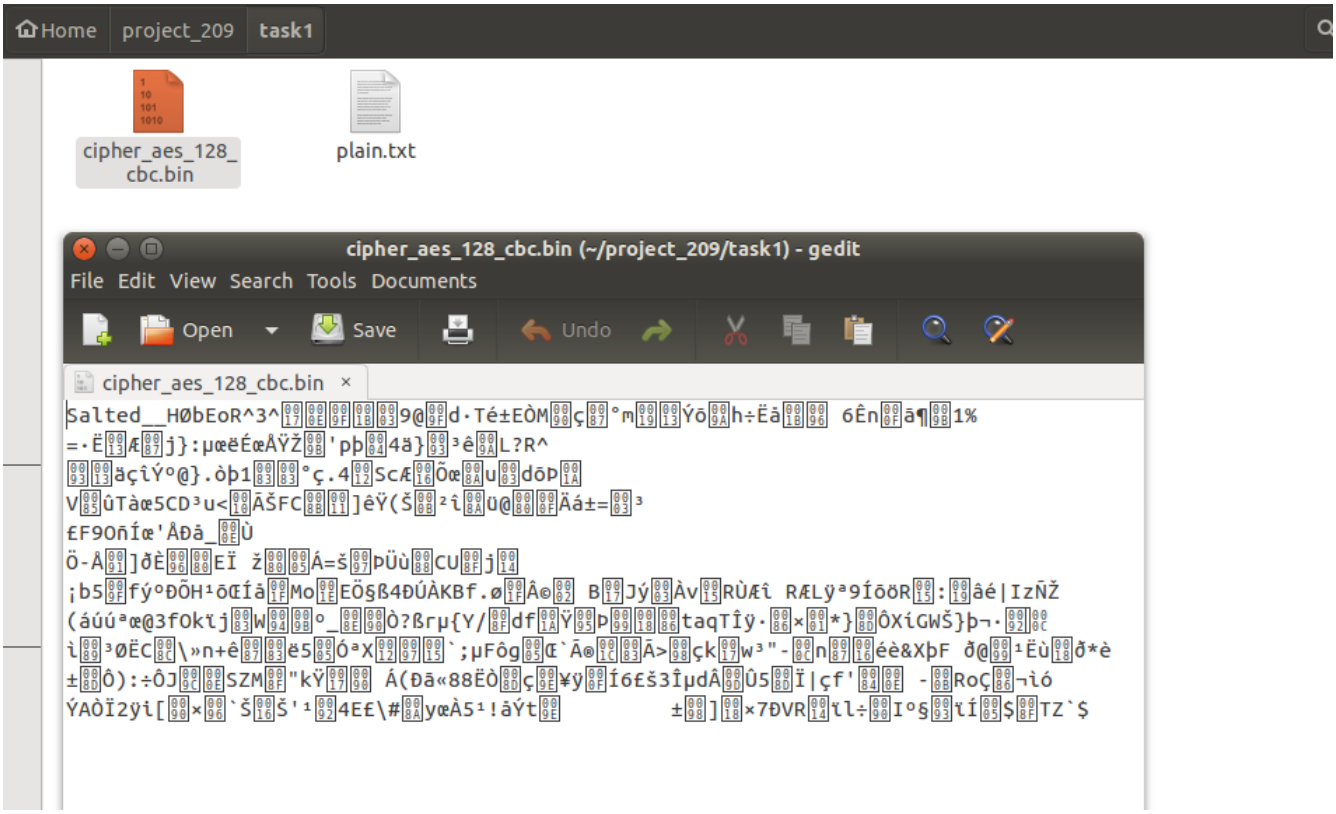


- **For cipher and mode: -aes-128-cbc**

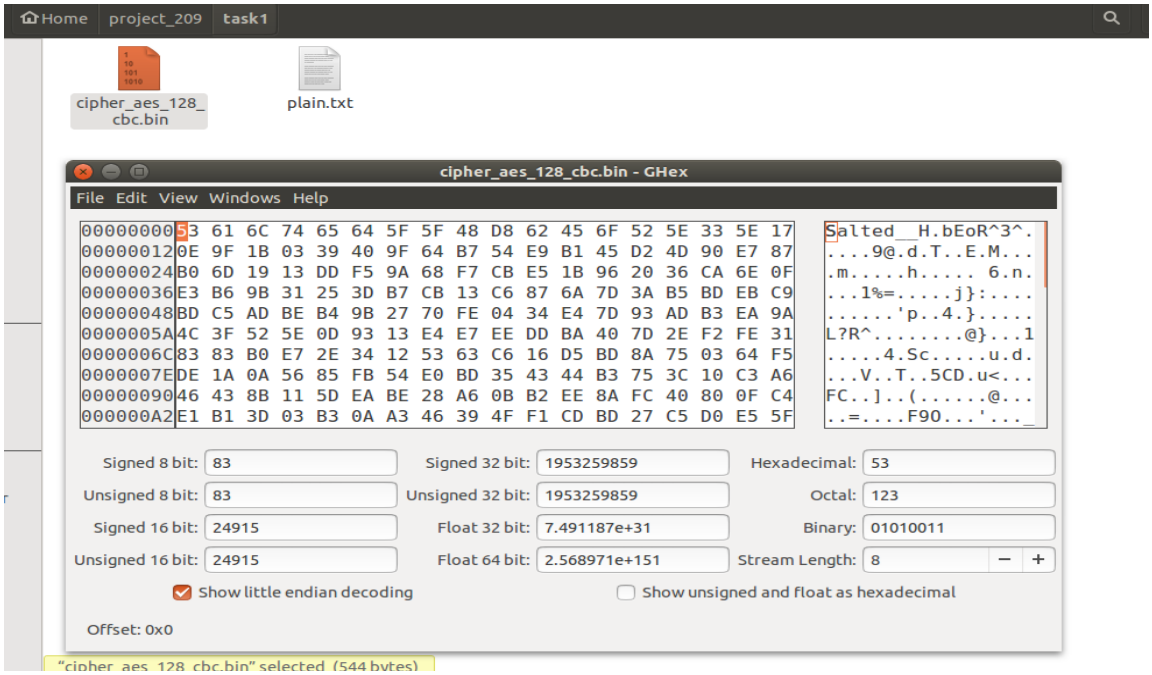
Encryption:



The plain text has been encrypted by the OpenSSL. But when we try to open the encrypted file, we get a file with undecipherable data as shown below

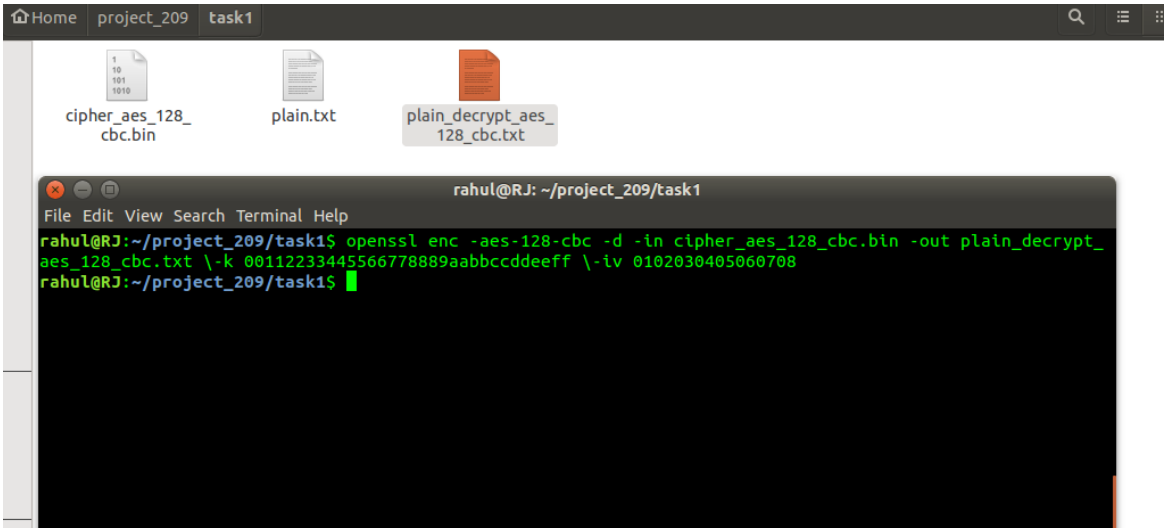


The encrypted file contains hex values so we used ghex editor to open the file.

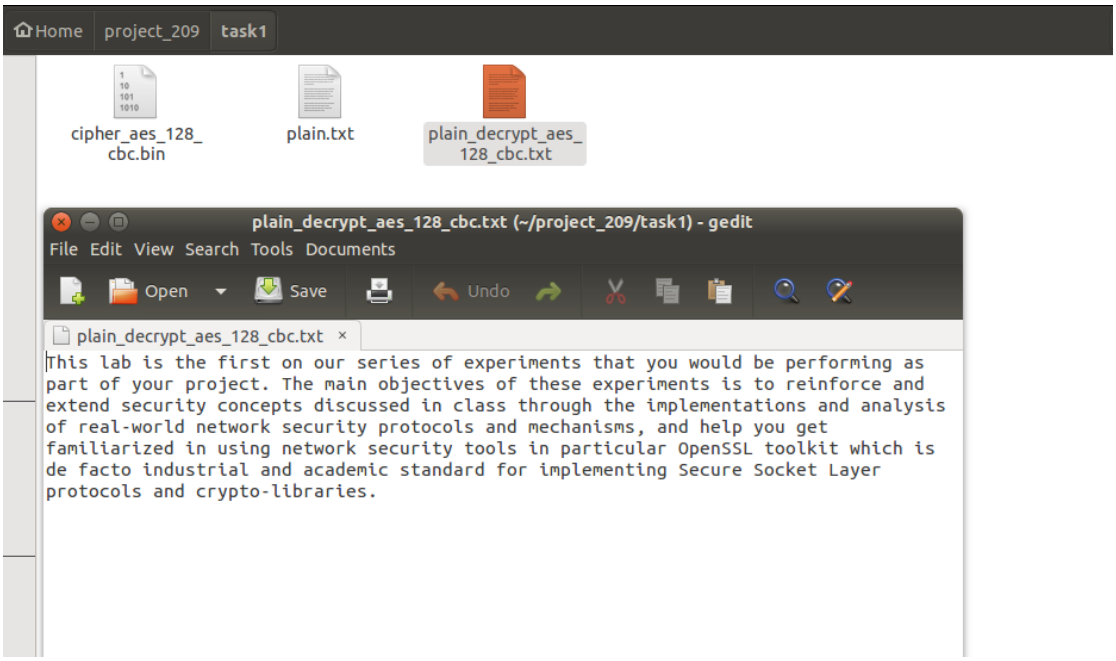


Decryption:

The encrypted file is decrypted to obtain the original plain text.

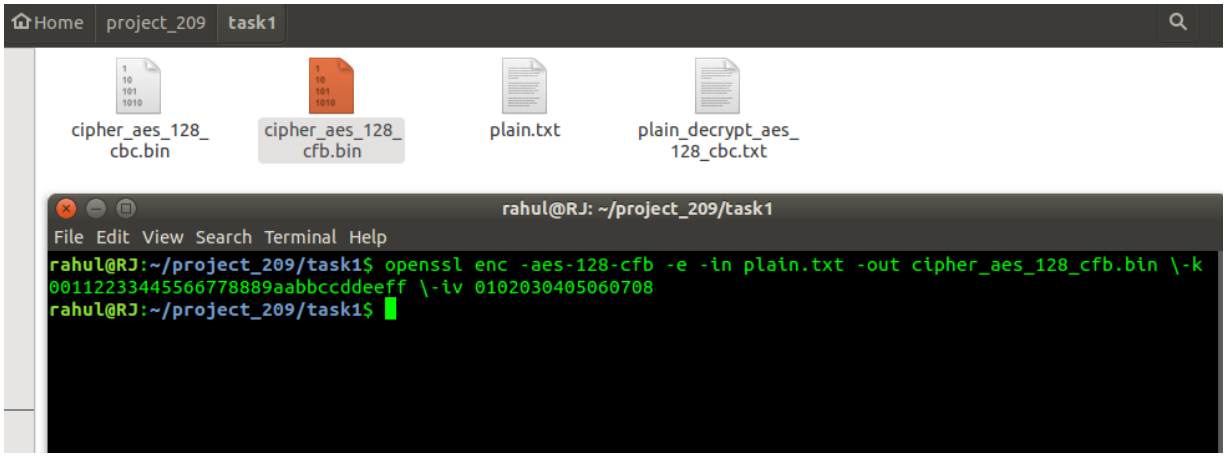


Plain Text:

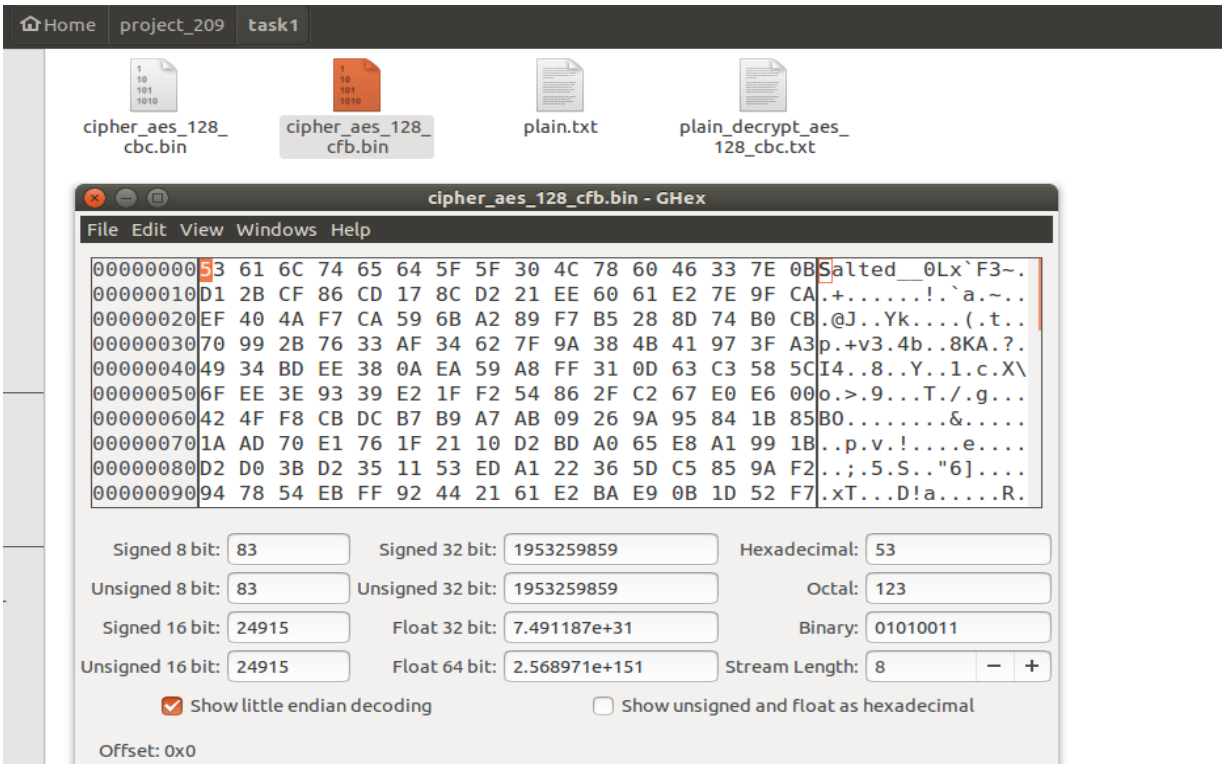


- **For cipher and mode: -aes-128-cfb**

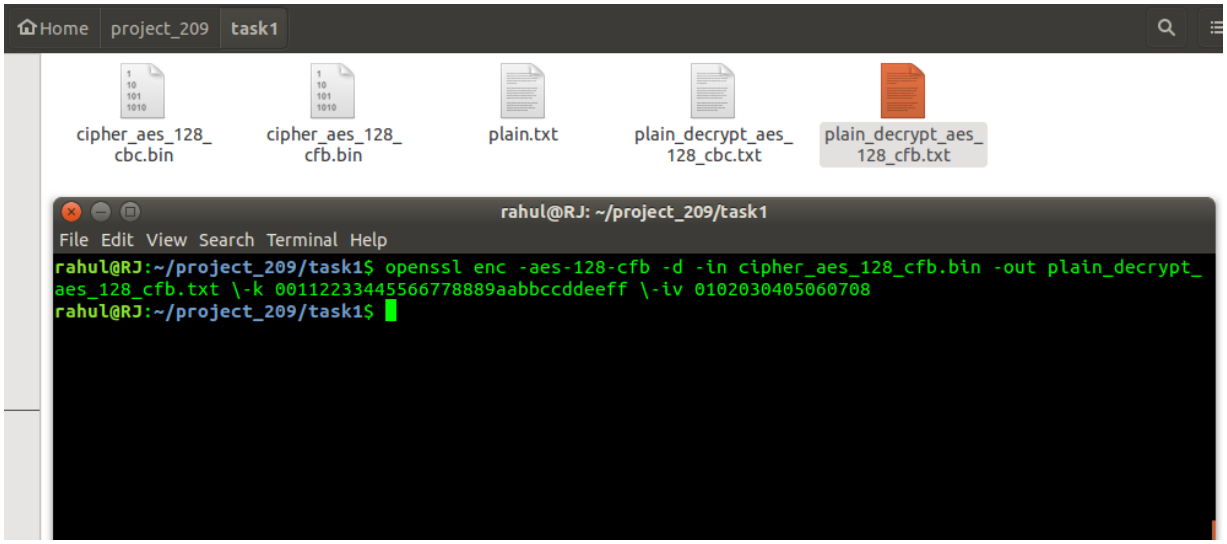
Encryption:



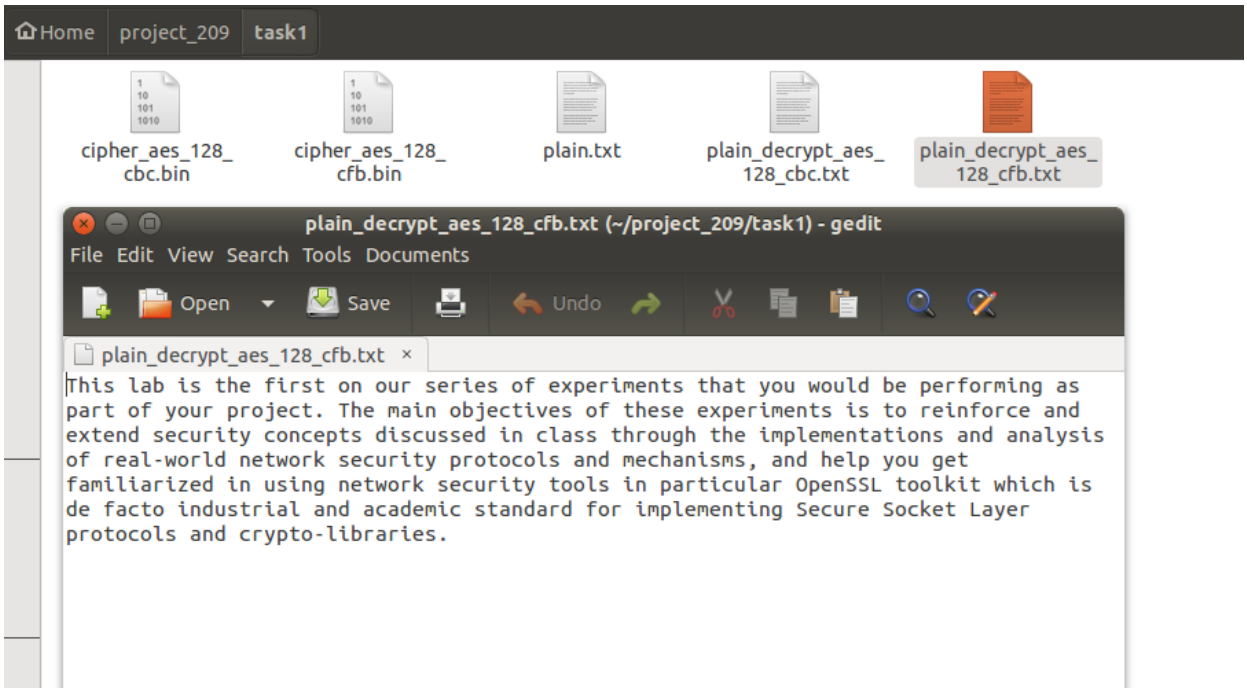
The encrypted file contains hex values so we used ghex editor to open the file.



Decryption:

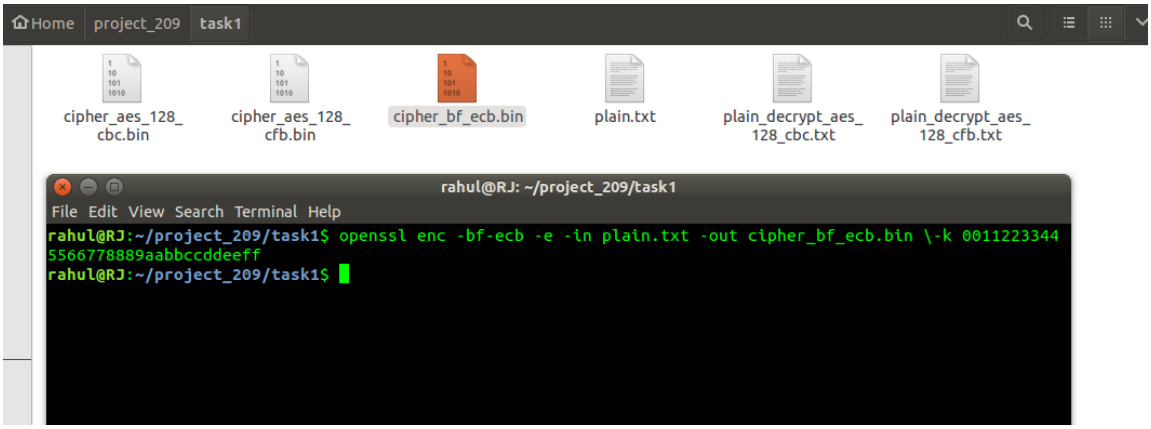


Plain Text:

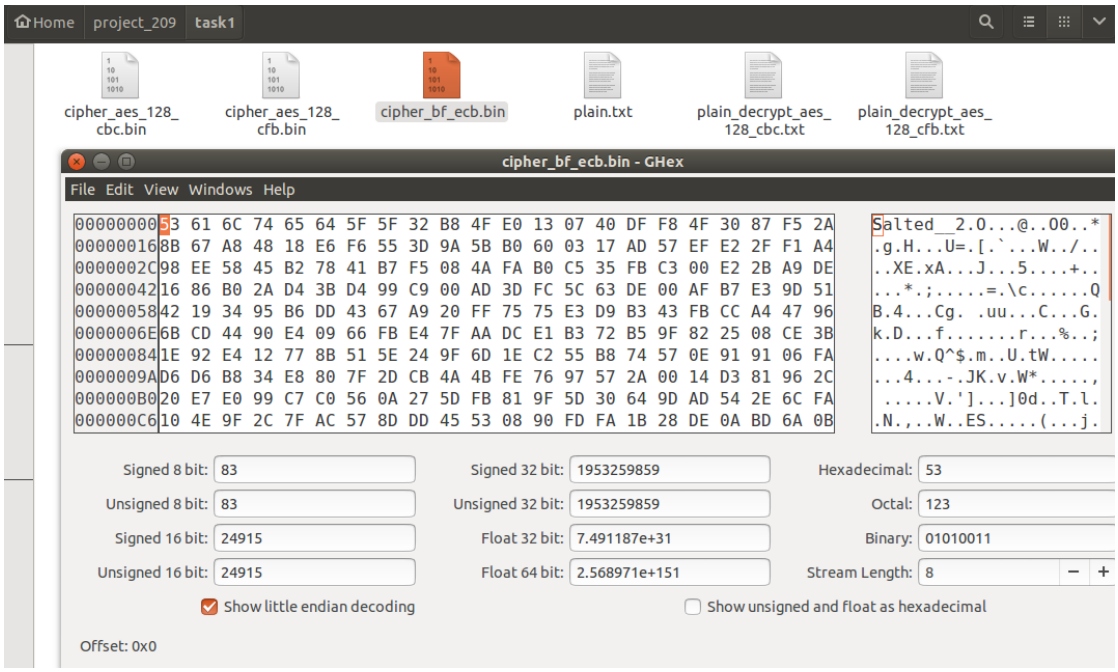


- **For cipher and mode: -bf-ecb**

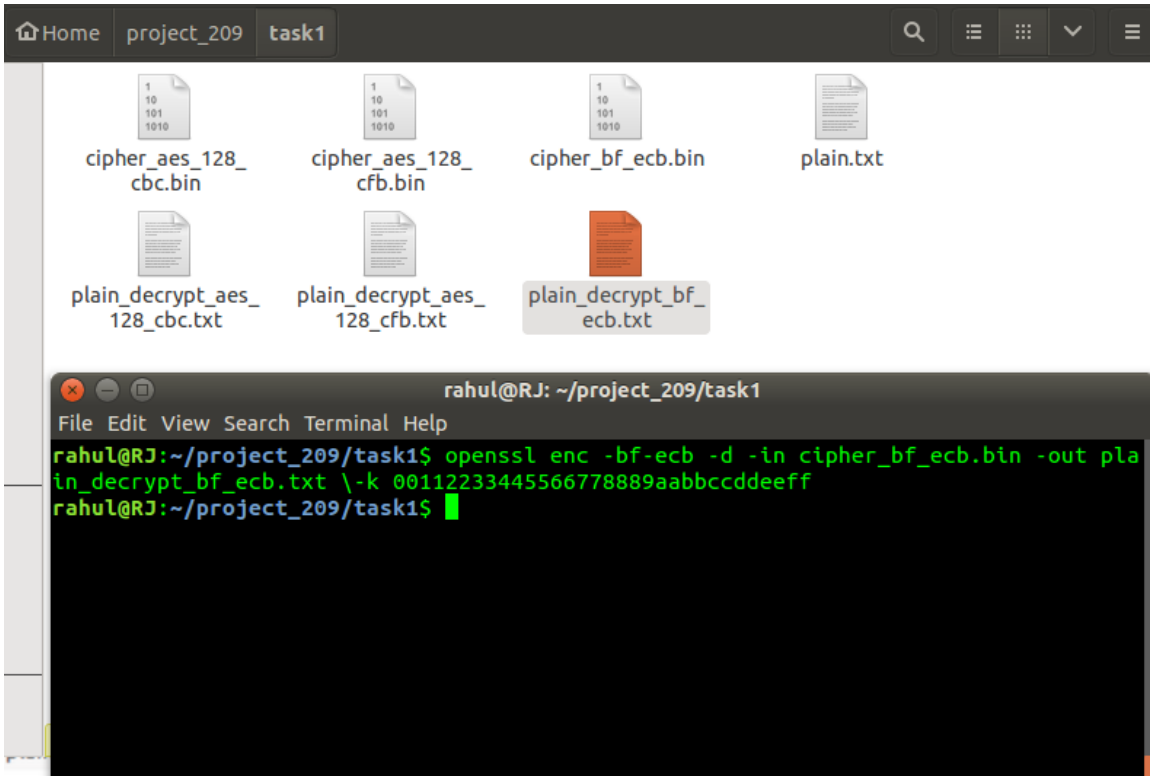
Encryption:



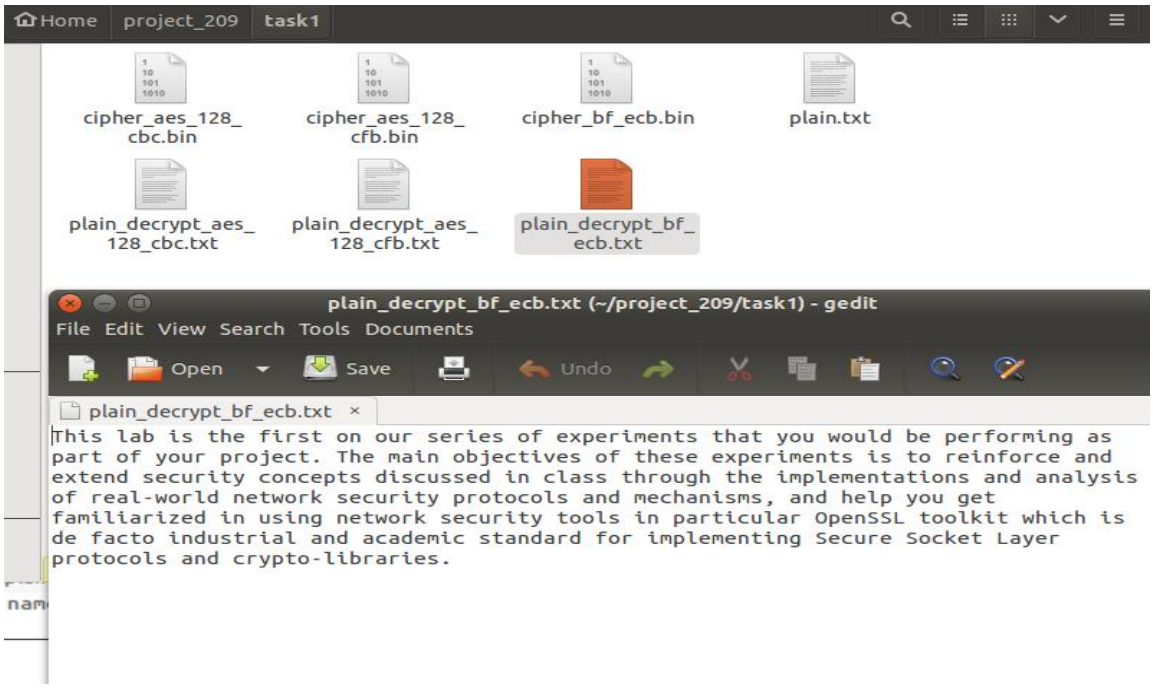
The encrypted file contains hex values so we used ghex editor to open the file.



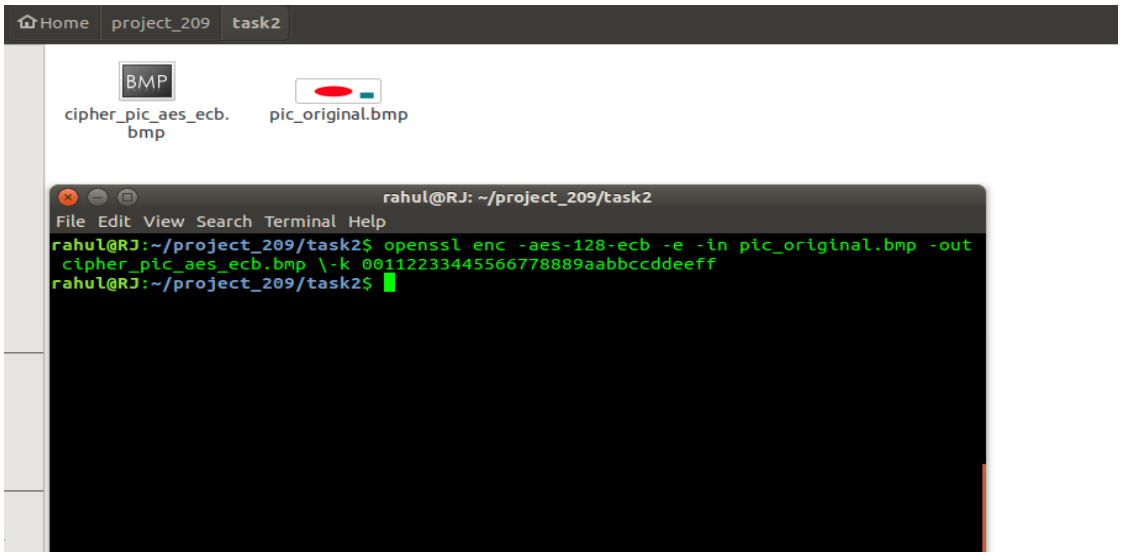
Decryption:



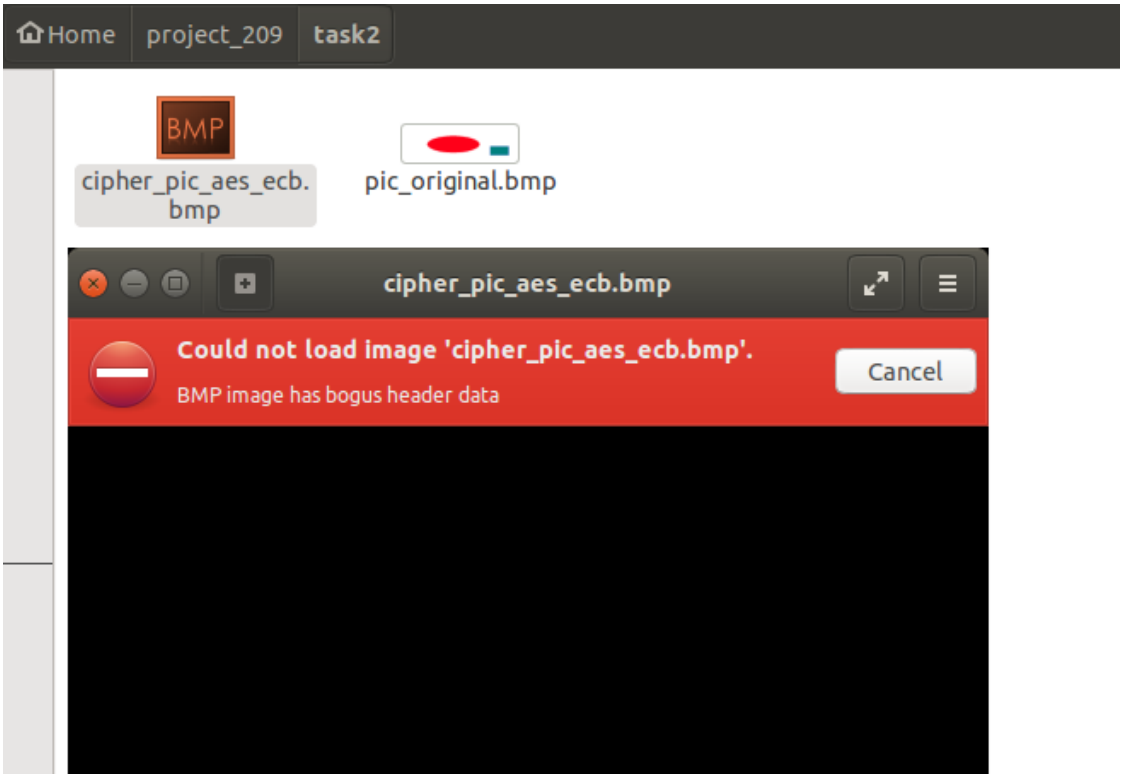
Plain Text:



Task 2: Encryption Mode – ECB vs. CBC

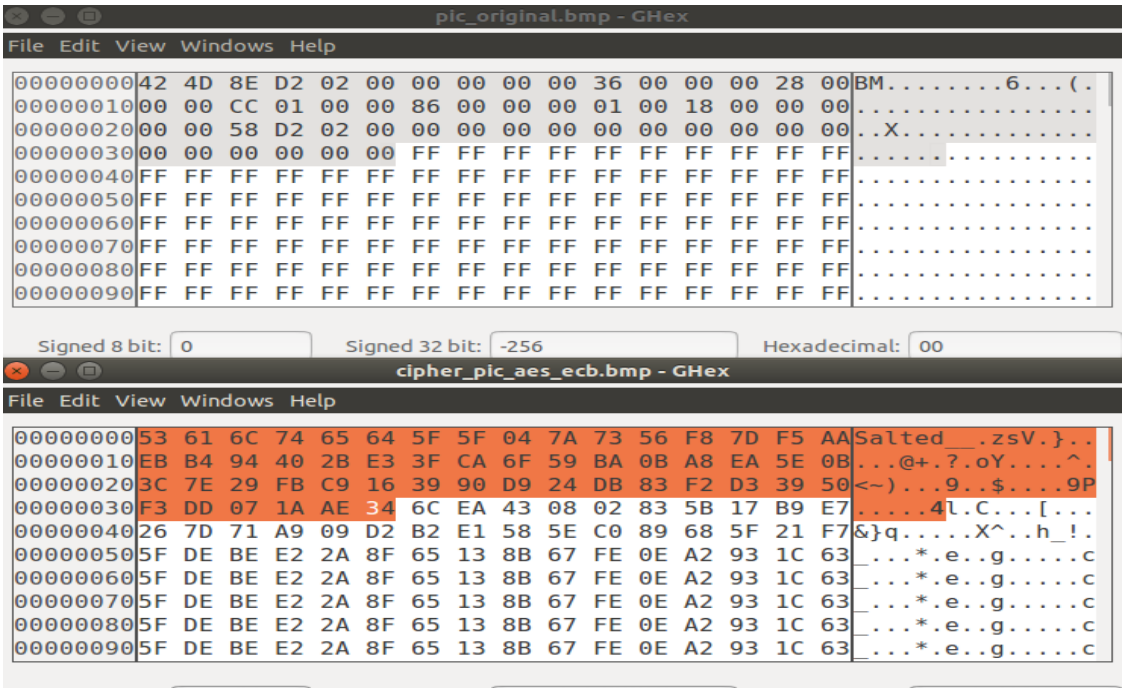


The given image “pic_original.bmp” is encrypted using electronic codebook encryption mode. But when we try to open the encrypted image it throws an error and it does not open the image.

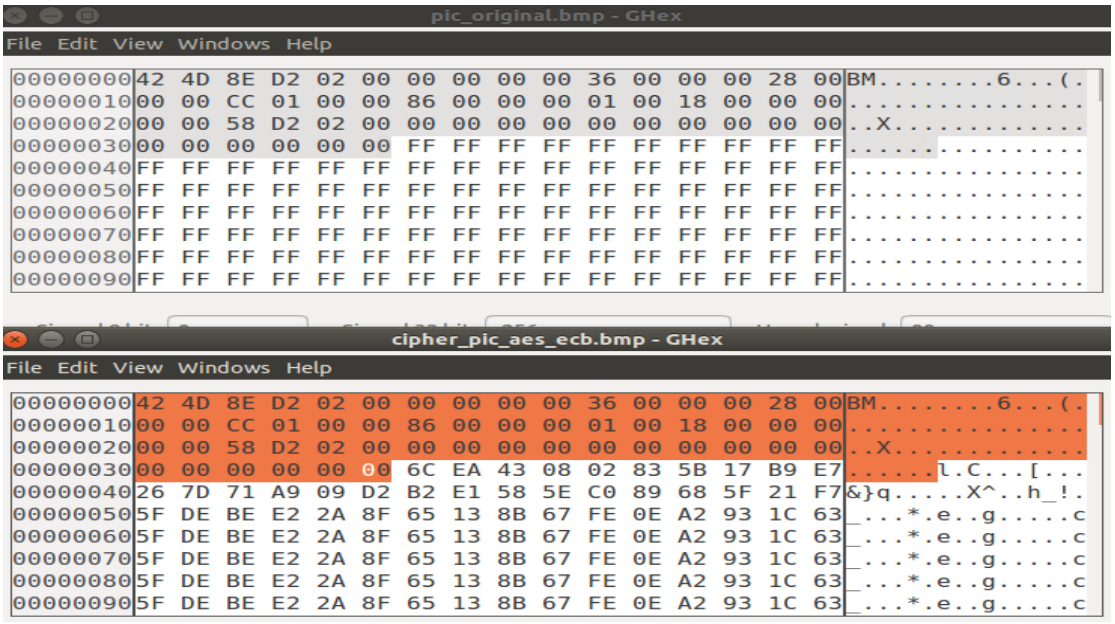


This occurred because the header of the file is not set correctly. In order to set the header, we modified the header of the encrypted file to match the header of the original file.

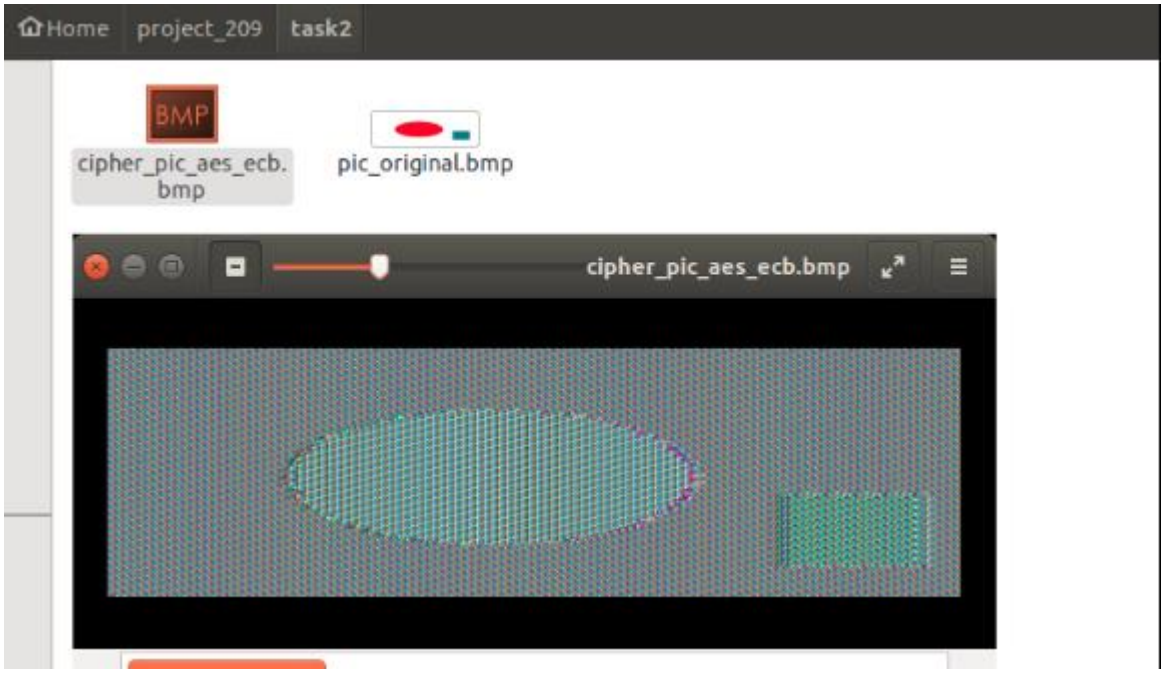
Before Changing the header (54 bytes)



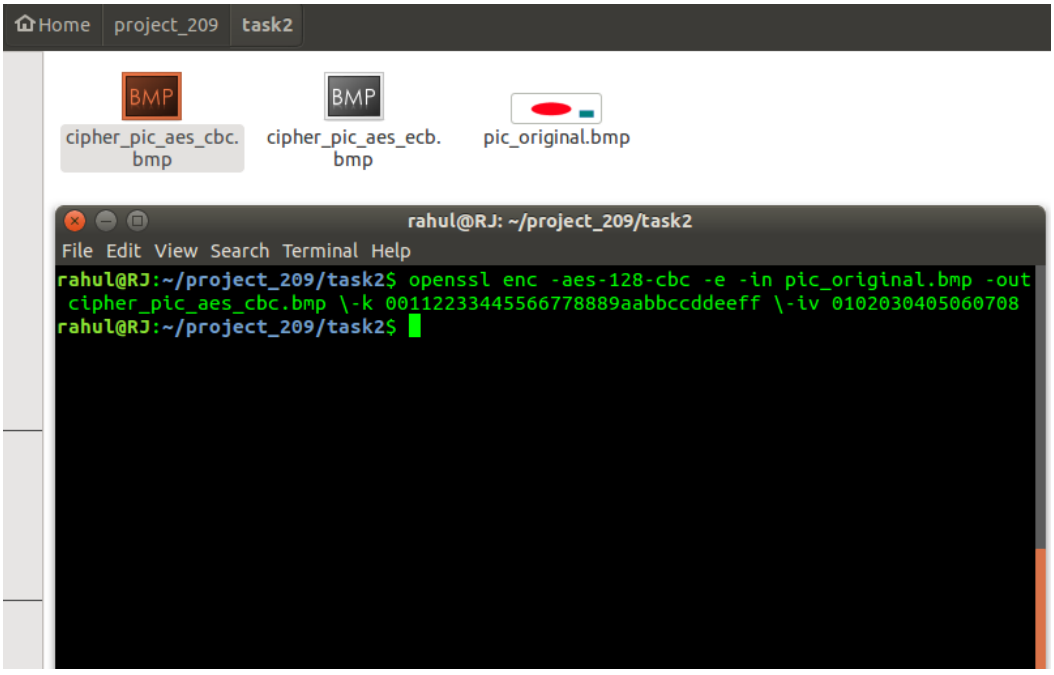
After Changing the header (54 bytes):



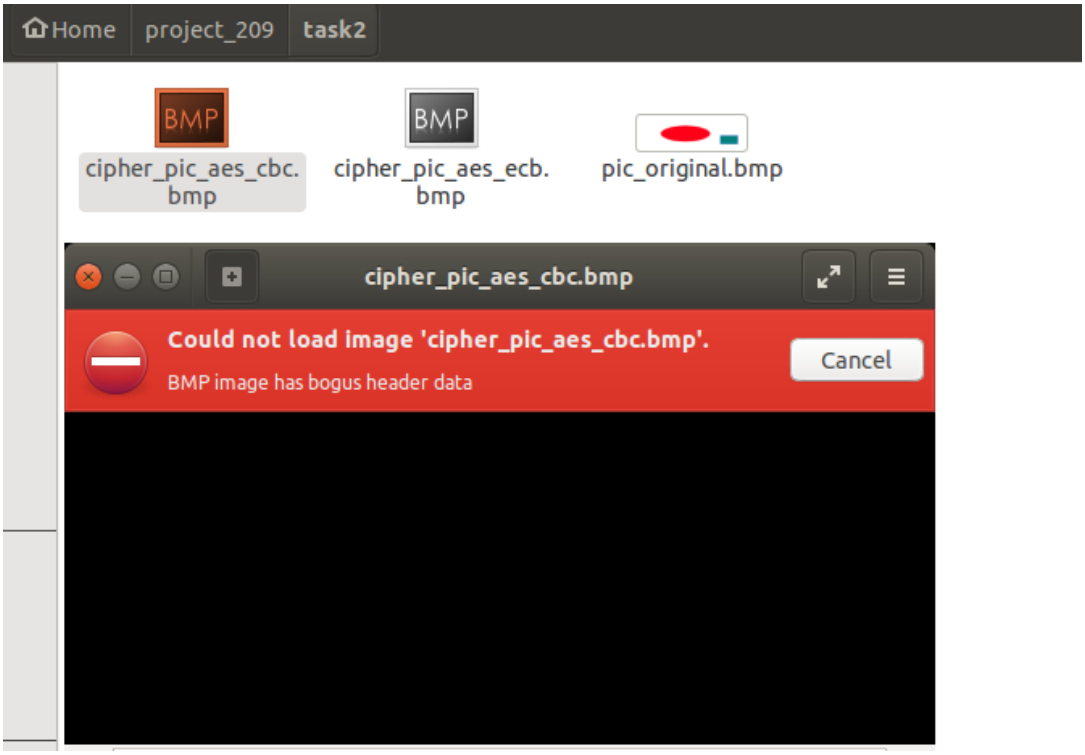
After changing the header of the encrypted image it can be viewed using any image viewing software



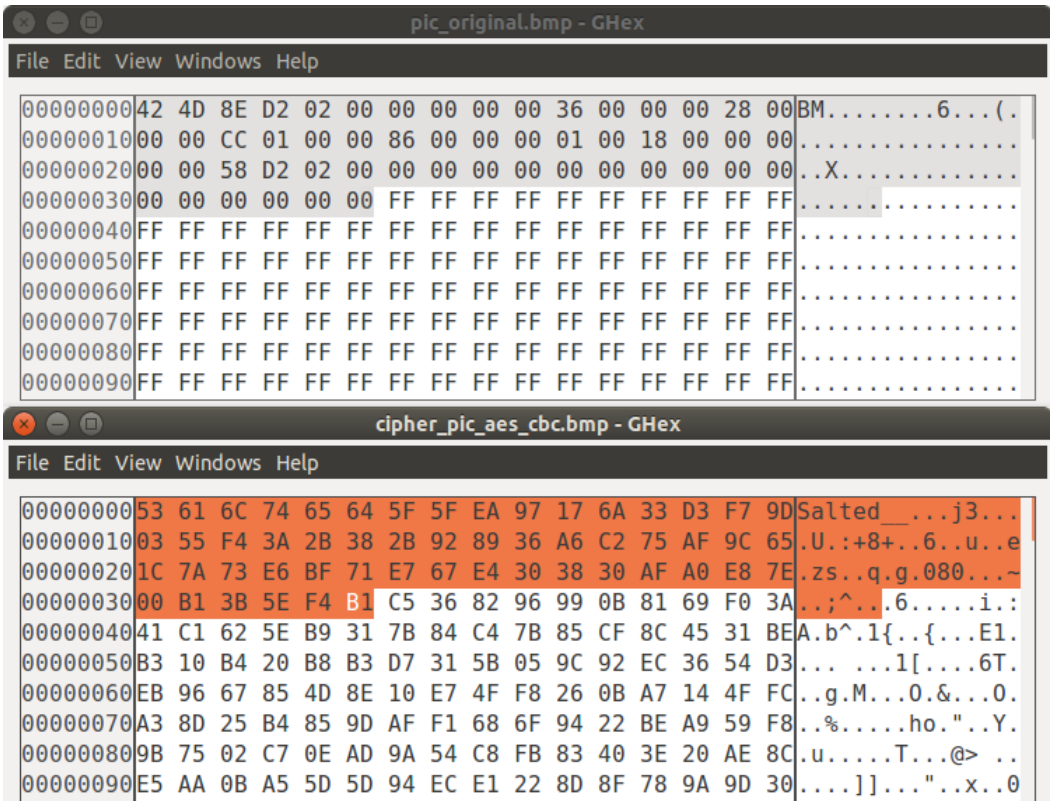
Encryption – CBC:



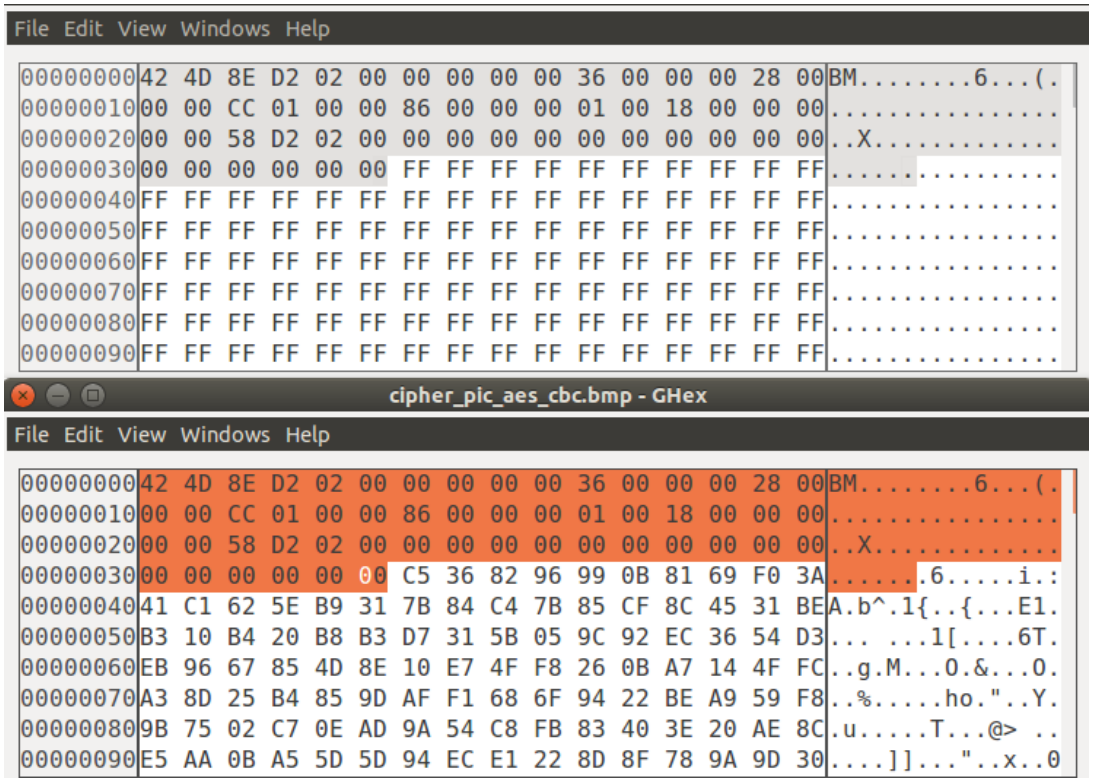
We encrypted the image using CBC mode. In order to open the encrypted image, we changed 54 bytes of the header of the image.



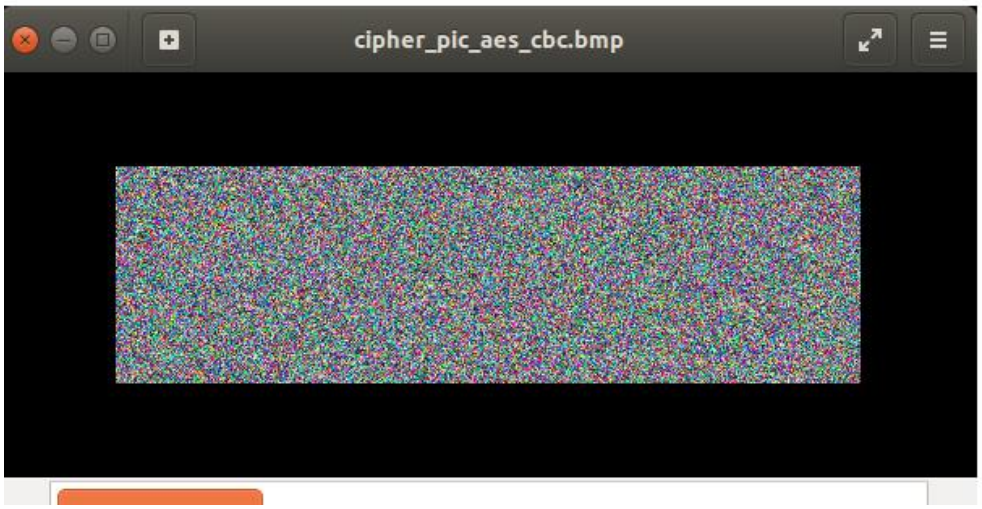
Before Changing the header (54 bytes):



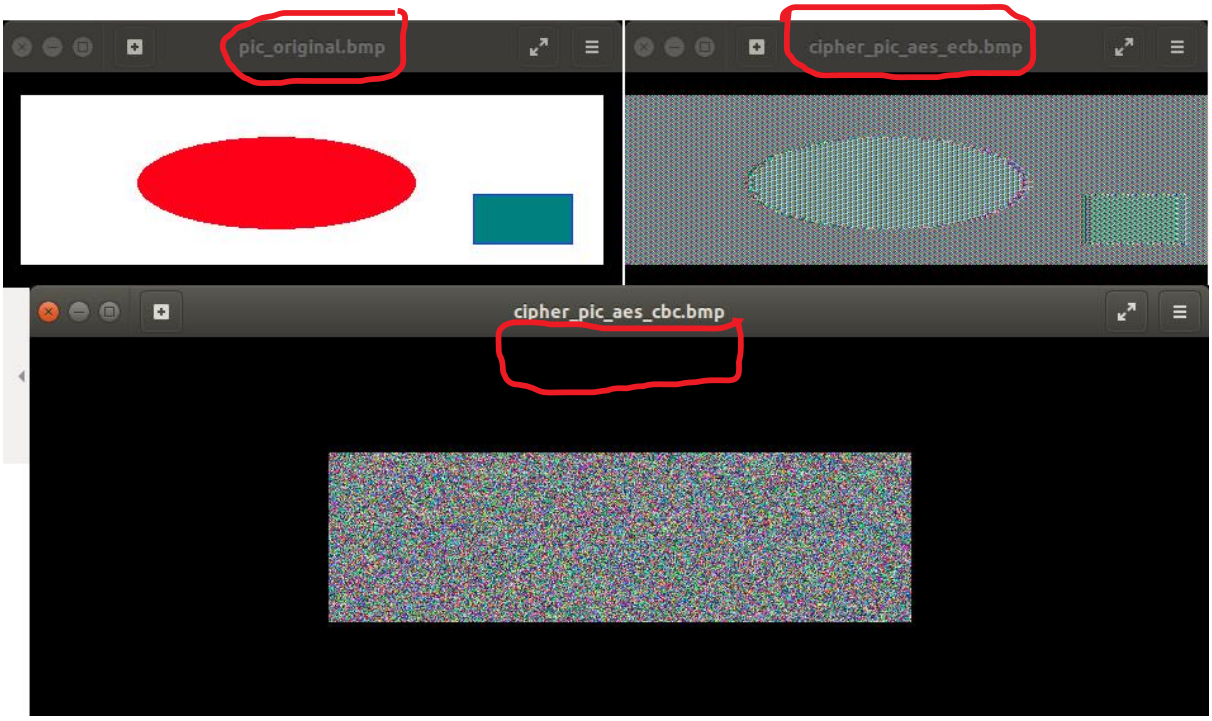
After Changing the header (54 bytes):



Encrypted Image:



Observation:



- In ECB mode the data patters are clearly visible because the identical plaintext blocks are encrypted into identical cipher texts. This is the disadvantage of ECB; therefore, it is not secure.
- In CBC mode, the data patterns are not at all visible and it provides more secure mechanism. Similar plain text does not generate similar cipher text because of the use of XOR operation before encryption of each cipher block.

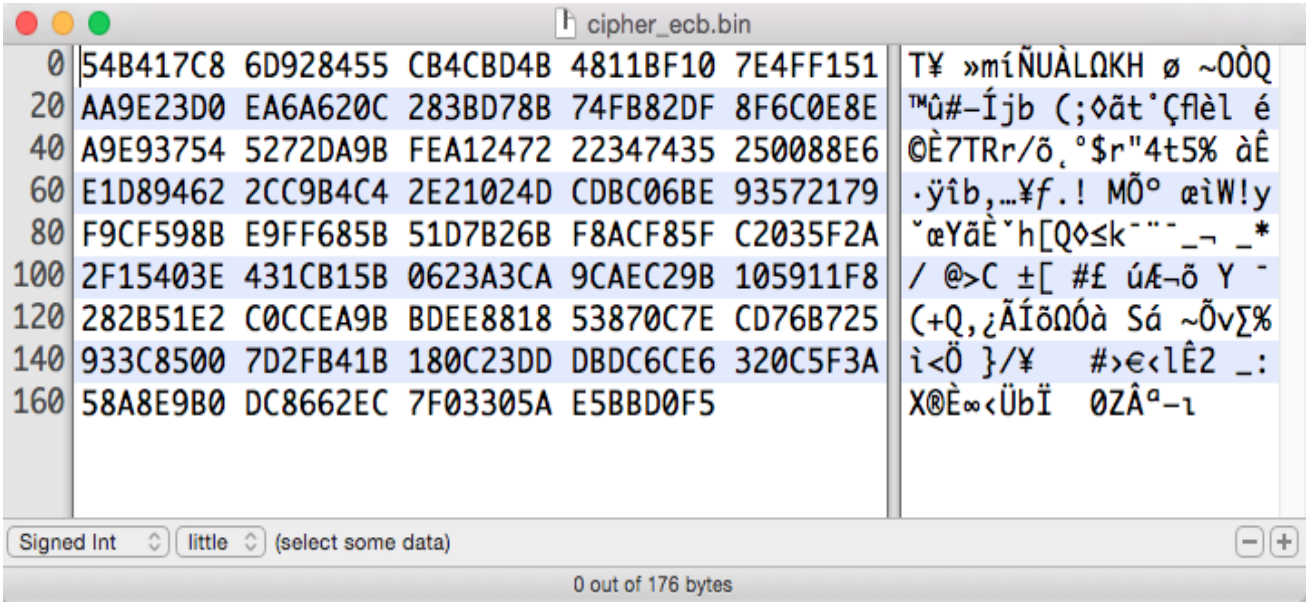
Task 3: Encryption Mode – Corrupted Cipher Text:

Plain text (input.txt): “This is a text file used as input for Crypto_lab_1. Extending the text file to at least 64 bytes in order to solve question number 3 task 3. Is this up to 64bytes> I hope it is.”

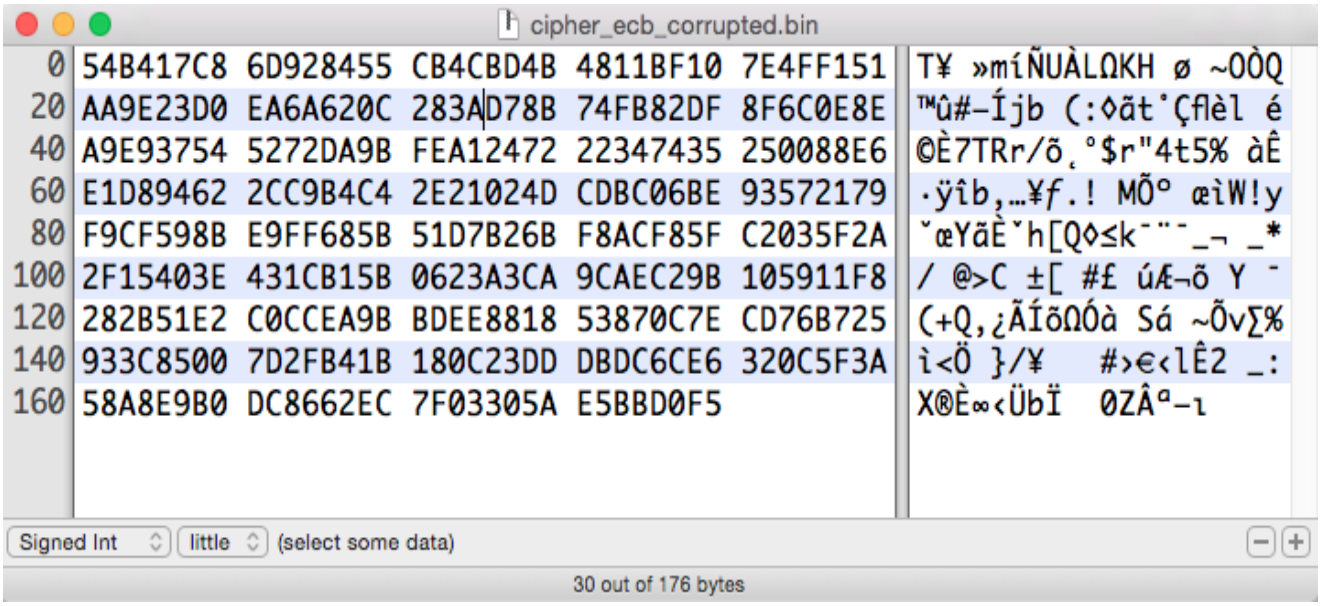
Encryption/Decryption Algorithm ECB:

```
Ifunanyas-MacBook-Pro:Desktop Funa$ openssl enc -aes-128-ecb -e -in input.txt -out cipher_ecb.bin \-K 00112233445566
778889aabbccddeeff
Ifunanyas-MacBook-Pro:Desktop Funa$ cat cipher_ecb.bin
T??m??U?L?KH?~O?Q??#??jb
?k???_?_*/@>C?[#?H?Y?(+Q???▯S"r"4t5%???Б,N?.!Mc??W!y??Y???h[Q$??„l???7TRr▯?t▯)
                                                                                ~?v?%?<?}/
?
 ??l?2
      _:X??▯?0Z???Ifunanyas-MacBook-Pro:Desktop Funa$
Ifunanyas-MacBook-Pro:Desktop Funa$ openssl enc -aes-128-ecb -d -in cipher_ecb_corrupted.bin -out decryted_corrupted
_ecb.txt \-K 00112233445566778889aabbccddeeff
Ifunanyas-MacBook-Pro:Desktop Funa$ cat decryted_corrupted_ecb.txt
This is a text fO?#FW?B?9?„?t for Crypto_lab_1. Exending the text file to atleast 64 bytes in order to solve questio
n number 3 task 3.Is this up to 64bytes> I hope it is.
Ifunanyas-MacBook-Pro:Desktop Funa$
```

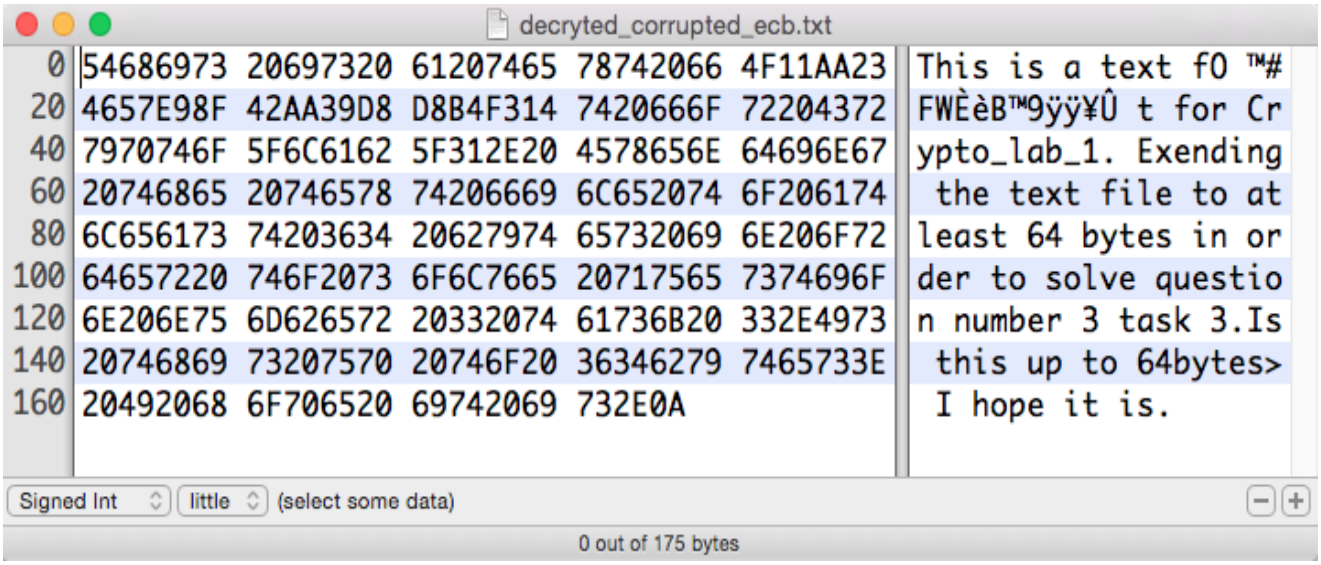
Uncorrupted ECB Cipher:



Corrupted ECB Cipher:



Decrypted Corrupted ECB Text file:



Original Text File:

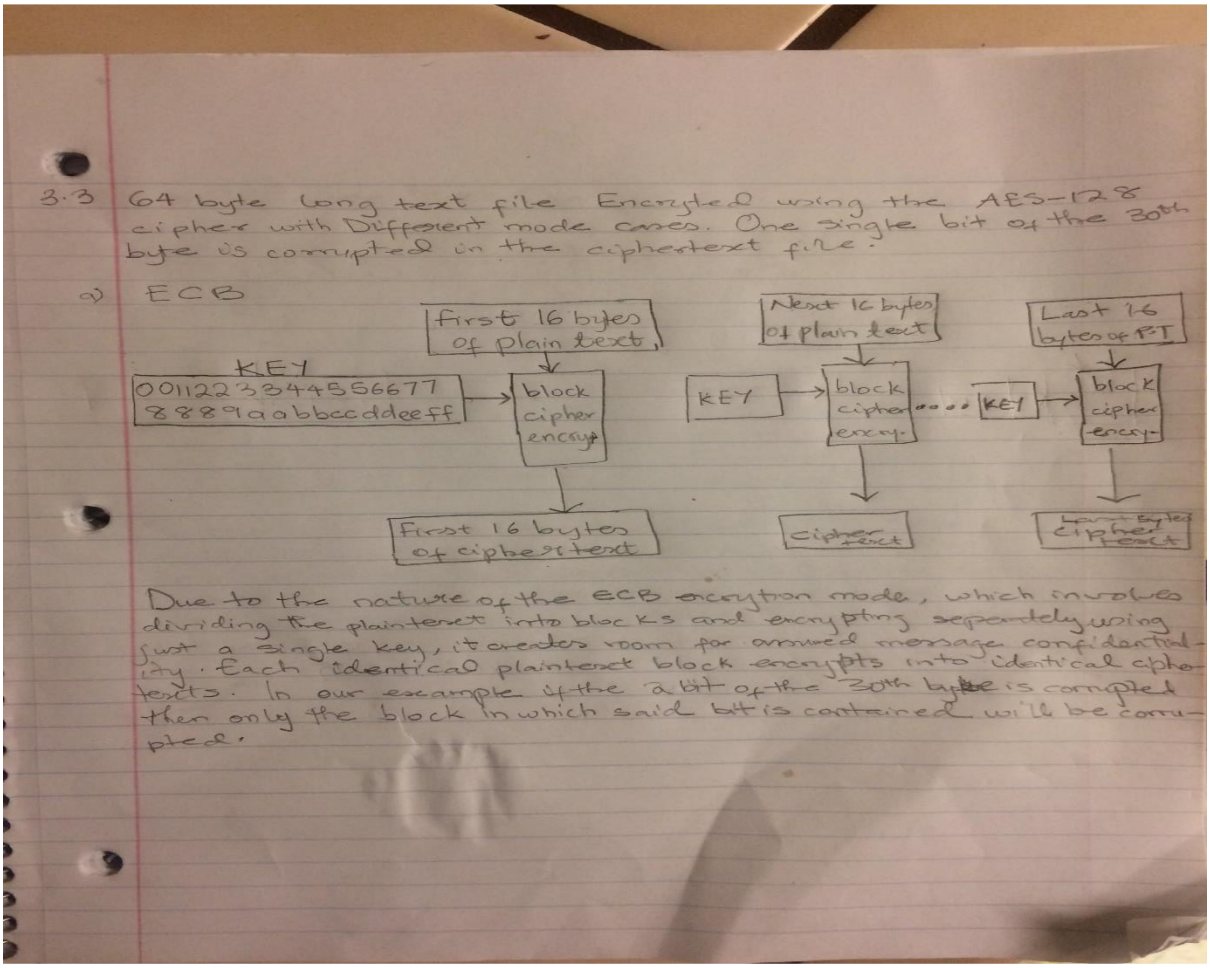
input.txt

0	54686973 20697320 61207465 78742066 696C6520	This is a text file
20	75736564 20617320 696E7075 7420666F 72204372	used as input for Cr
40	7970746F 5F6C6162 5F312E20 4578656E 64696E67	ypto_lab_1. Exending
60	20746865 20746578 74206669 6C652074 6F206174	the text file to at
80	6C656173 74203634 20627974 65732069 6E206F72	least 64 bytes in or
100	64657220 746F2073 6F6C7665 20717565 7374696F	der to solve questio
120	6E206E75 6D626572 20332074 61736B20 332E4973	n number 3 task 3.Is
140	20746869 73207570 20746F20 36346279 7465733E	this up to 64bytes>
160	20492068 6F706520 69742069 732E0A	I hope it is.

Signed Int | little | (select some data)

0 out of 175 bytes

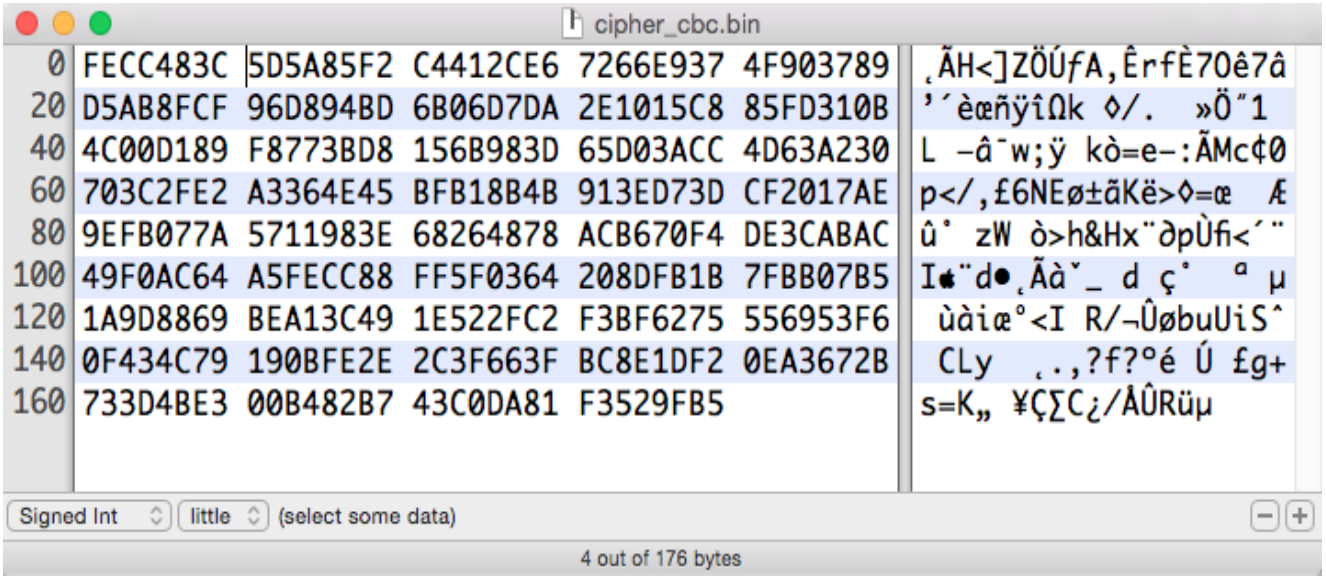
In ECB compared with the input.txt, we can see that decrypted_ecb.txt has a section corrupted.



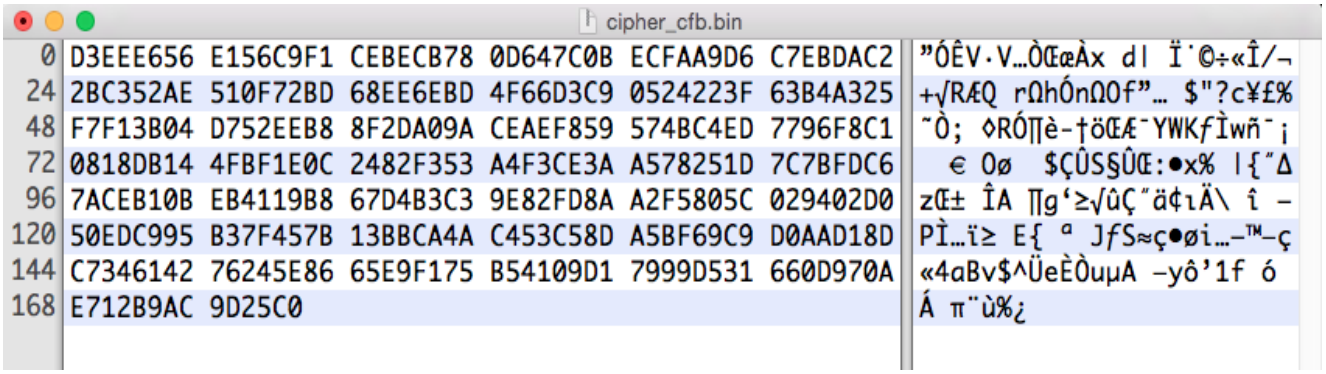
Encryption/Decryption Algorithm CBC:

```
Ifunanyas-MacBook-Pro:Desktop Funa$ openssl enc -aes-128-cbc -e -in input.txt -out cipher_cbc.bin \-K 00112233445566
778889aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ openssl enc -aes-128-cbc -d -in cipher_cbc_corrupted.bin -out decryted_corrupted
_cbc.txt \-K 00112233445566778889aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ cat decryted_corrupted_cbc.txt
This is a text fP??D(?_uj-m?t for Crypto_mab_1. Exending the text file to atleast 64 bytes in order to solve questi
on number 3 task 3.Is this up to 64bytes> I hope it is.
Ifunanyas-MacBook-Pro:Desktop Funa$
```

Uncorrupted CBC:



Corrupted CBC:



Decrypted Corrupted CBC Text File:

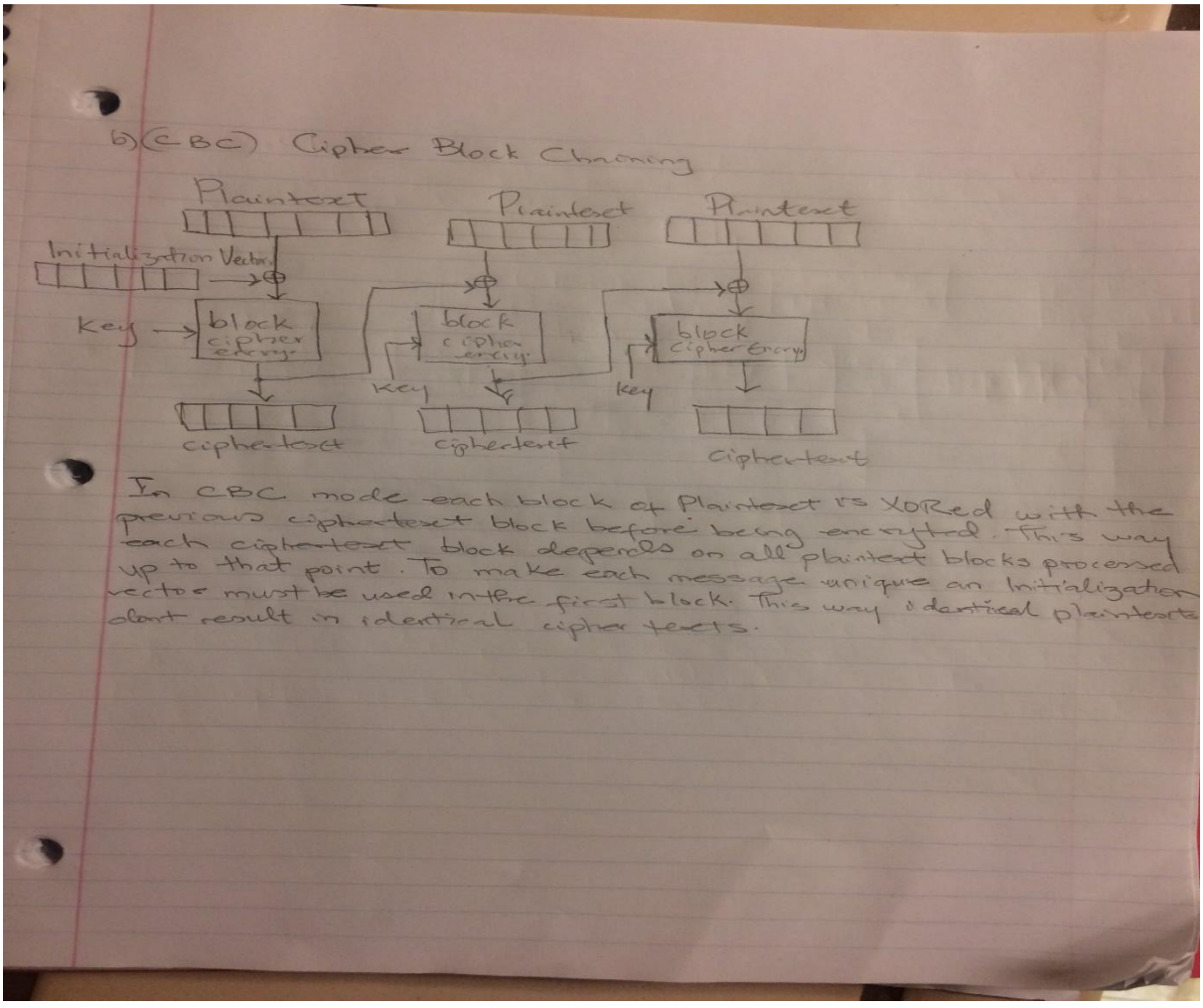
decryted_corrupted_cbc.txt

0	54686973	20697320	61207465	78742066	50E596F2	This is a text fPÂñÚ
20	44DDBC7B	E25F751D	6A2D6DAC	7420666F	72204372	D›°{,_u j-m¨t for Cr
40	7970746F	5F6D6162	5F312E20	4578656E	64696E67	ypto_mab_1. Exending
60	20746865	20746578	74206669	6C652074	6F206174	the text file to at
80	6C656173	74203634	20627974	65732069	6E206F72	least 64 bytes in or
100	64657220	746F2073	6F6C7665	20717565	7374696F	der to solve questio
120	6E206E75	6D626572	20332074	61736B20	332E4973	n number 3 task 3.Is
140	20746869	73207570	20746F20	36346279	7465733E	this up to 64bytes>
160	20492068	6F706520	69742069	732E0A		I hope it is.

Signed Int | little | (select some data)

0 out of 175 bytes

Here we can see there is more corruption to the decrypted text file compared to ECB.



Encryption/Decryption Algorithm CFB:

```
Ifunanyas-MacBook-Pro:Desktop Funa$ cat input.txt
This is a text file used as input for Crypto_lab_1. Exending the text file to atleast 64 bytes in order to solve que
stion number 3 task 3.Is this up to 64bytes> I hope it is.
Ifunanyas-MacBook-Pro:Desktop Funa$ openssl enc -aes-128-cfb -e -in input.txt -out cipher_cfb.bin \-K 00112233445566
778889aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ cat cipher_cfb.bin
d|?V?V??ξ?x
  ????????+?R?Q
             r?h?n?Of??$"?c??%??;?R??ή?YWK??w???O?
                                                $??S???:?x%|{??zα
                                                                  ?A?g?Þ??????\??P?ϱ?E{??J?Sδ??i?b3?4aBv$^?e??u
?A      ?y??1f
????%?Ifunanyas-MacBook-Pro:Desktop openssl enc -aes-128-cfb -d -in cipher_cfb_corrupted.bin -out decryted_corrupted
_cfb.txt \-K 00112233445566778889aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ cat decryted_corrupted_cfb.txt
This is a text file used as ijpu?DT$?%Ss?/?_1. Exending the text file to atleast 64 bytes in order to solve question
 number 3 task 3.Is this up to 64bytes> I hope it is.
Ifunanyas-MacBook-Pro:Desktop Funa$
```

Uncorrupted CFB Cipher:

cipher_cfb.bin

0	D3EEE656	E156C9F1	CEBECB78	0D647C0B	ECFAA9D6	C7EBDAC2	”ÓÊV·V…ÒŒæÀx d| Ï˙©÷«Î/¬
24	2BC352AE	510B72BD	68EE6EBD	4F66D3C9	0524223F	63B4A325	+√RÆQ rΩhÓnΩOf”… $"?c¥£%
48	F7F13B04	D752EEB8	8F2DA09A	CEAEF859	574BC4ED	7796F8C1	˜Ò; ◊RÓ∏è-†öŒÆ¯YWKƒÌwñ¯¡
72	0818DB14	4FBF1E0C	2482F353	A4F3CE3A	A578251D	7C7BFDC6	€ Oø $ÇÛS§ÛŒ:•x% |{˝Δ
96	7ACEB10B	EB4119B8	67D4B3C3	9E82FD8A	A2F5805C	029402D0	zŒ± ÎA ∏g‘≥√ûÇ˝ä¢ıÄ\ î –
120	50EDC995	B37F457B	13BBCA4A	C453C58D	A5BF69C9	D0AAD18D	PÌ…ï≥ E{ ª JƒS≈ç•øi…–™–ç
144	C7346142	76245E86	65E9F175	B54109D1	7999D531	660D970A	«4aBv$^ÜeÈÒuµA –yô’1f ó
168	E712B9AC	9D25C0					Á π¨ù%¿

Corrupted CFB Cipher:

cipher_cfb.bin

0	D3EEE656	E156C9F1	CEBECB78	0D647C0B	ECFAA9D6	C7EBDAC2	”ÓÊV·V…ÒŒæÀx d| Ï˙©÷«Î/¬
24	2BC352AE	510F72BD	68EE6EBD	4F66D3C9	0524223F	63B4A325	+√RÆQ rΩhÓnΩOf”… $"?c¥£%
48	F7F13B04	D752EEB8	8F2DA09A	CEAEF859	574BC4ED	7796F8C1	˜Ò; ◊RÓ∏è-†öŒÆ¯YWKƒÌwñ¯¡
72	0818DB14	4FBF1E0C	2482F353	A4F3CE3A	A578251D	7C7BFDC6	€ Oø $ÇÛS§ÛŒ:•x% |{˝Δ
96	7ACEB10B	EB4119B8	67D4B3C3	9E82FD8A	A2F5805C	029402D0	zŒ± ÎA ∏g‘≥√ûÇ˝ä¢ıÄ\ î –
120	50EDC995	B37F457B	13BBCA4A	C453C58D	A5BF69C9	D0AAD18D	PÌ…ï≥ E{ ª JƒS≈ç•øi…–™–ç
144	C7346142	76245E86	65E9F175	B54109D1	7999D531	660D970A	«4aBv$^ÜeÈÒuµA –yô’1f ó
168	E712B9AC	9D25C0					Á π¨ù%¿

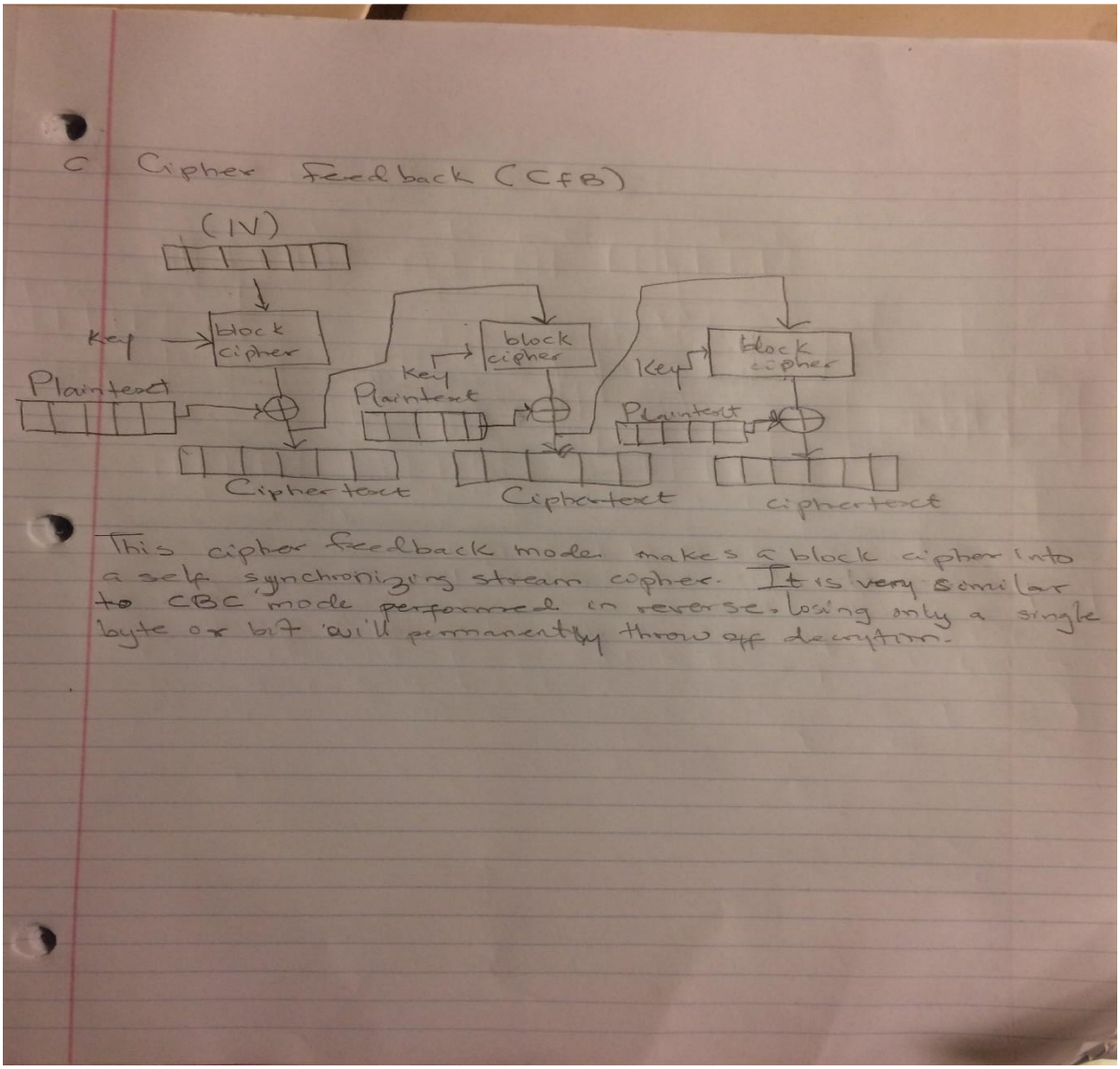
Decrypted Corrupted CFB Text File:

decryted_corrupted_cfb.txt

0	54686973	20697320	61207465	78742066	696C6520	This is a text file
20	75736564	20617320	696A7075	A0445424	B7255373	used as ijpu†DT$∑%Ss
40	0E85142F	D6007F0F	5F312E20	4578656E	64696E67	Ö /÷ _1. Exending
60	20746865	20746578	74206669	6C652074	6F206174	the text file to at
80	6C656173	74203634	20627974	65732069	6E206F72	least 64 bytes in or
100	64657220	746F2073	6F6C7665	20717565	7374696F	der to solve questio
120	6E206E75	6D626572	20332074	61736B20	332E4973	n number 3 task 3.Is
140	20746869	73207570	20746F20	36346279	7465733E	this up to 64bytes>
160	20492068	6F706520	69742069	732E0A		I hope it is.

Signed Int | little | (select some data)

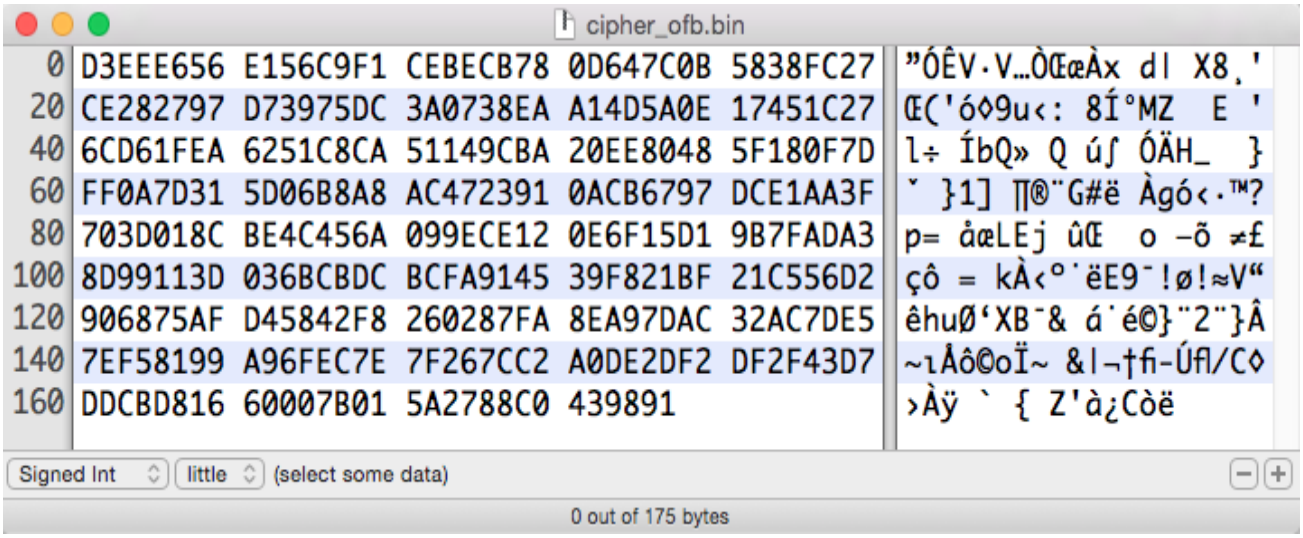
0 out of 175 bytes



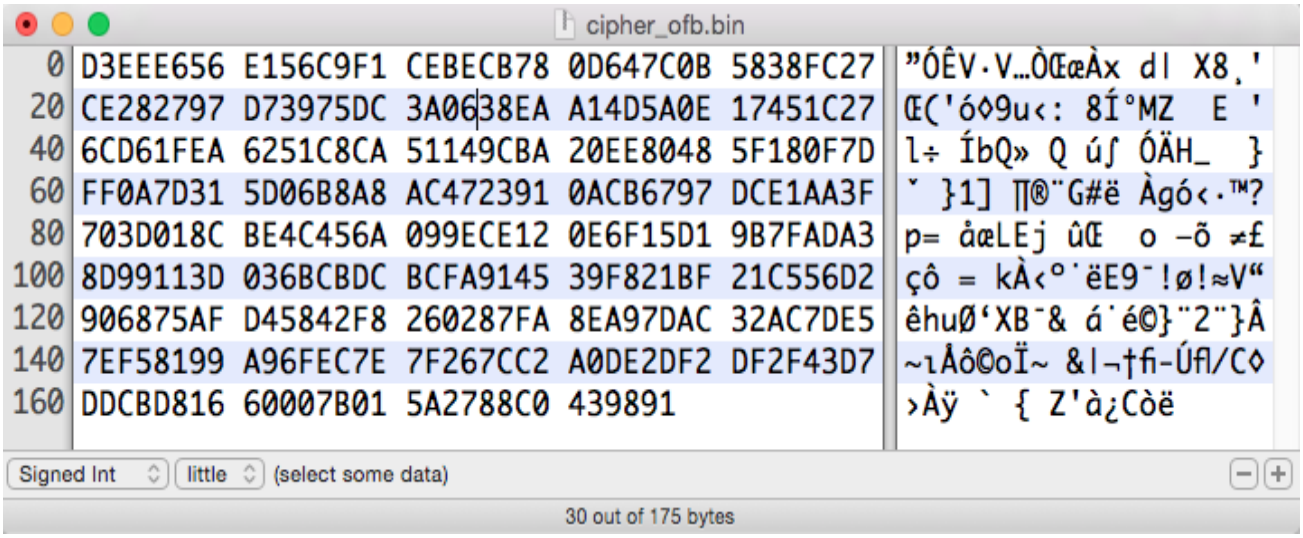
Encryption/Decryption Algorithm OFB:

```
Ifunanyas-MacBook-Pro:Desktop Funa$ openssl enc -aes-128-ofb -e -in input.txt -out cipher_ofb.bin \-K 00112233445566
778889aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ cat cipher_ofb.bin
d|?V?V??ξ?x
  X8?'?('??9u?:8?MZE'l??bQ??Q?? ?H_}?
}1]???G#?
?g????p=??LEj   ??oh????=k?E9?!?!?Vhu??XB?&????}?2?}?~????o?~&| ?-??/C????`(Z'??C??Ifunanyas-MacBook-Pro:Desktop
Funa$
Ifunanyas-MacBook-Pro:Desktop Funa$ openssl enc -aes-128-ofb -d -in cipher_ofb_corrupted.bin -out decryted_corrupted
_ofb.txt \-K 00112233445566778889aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ cat decryted_corrupted_ofb.txt
This is a text file used as ioput for Crypto_lab_1. Exending the text file to atleast 64 bytes in order to solve que
stion number 3 task 3.Is this up to 64bytes> I hope it is.
Ifunanyas-MacBook-Pro:Desktop Funa$
```

Uncorrupted OFB Cipher:



Corrupted OFB Cipher:



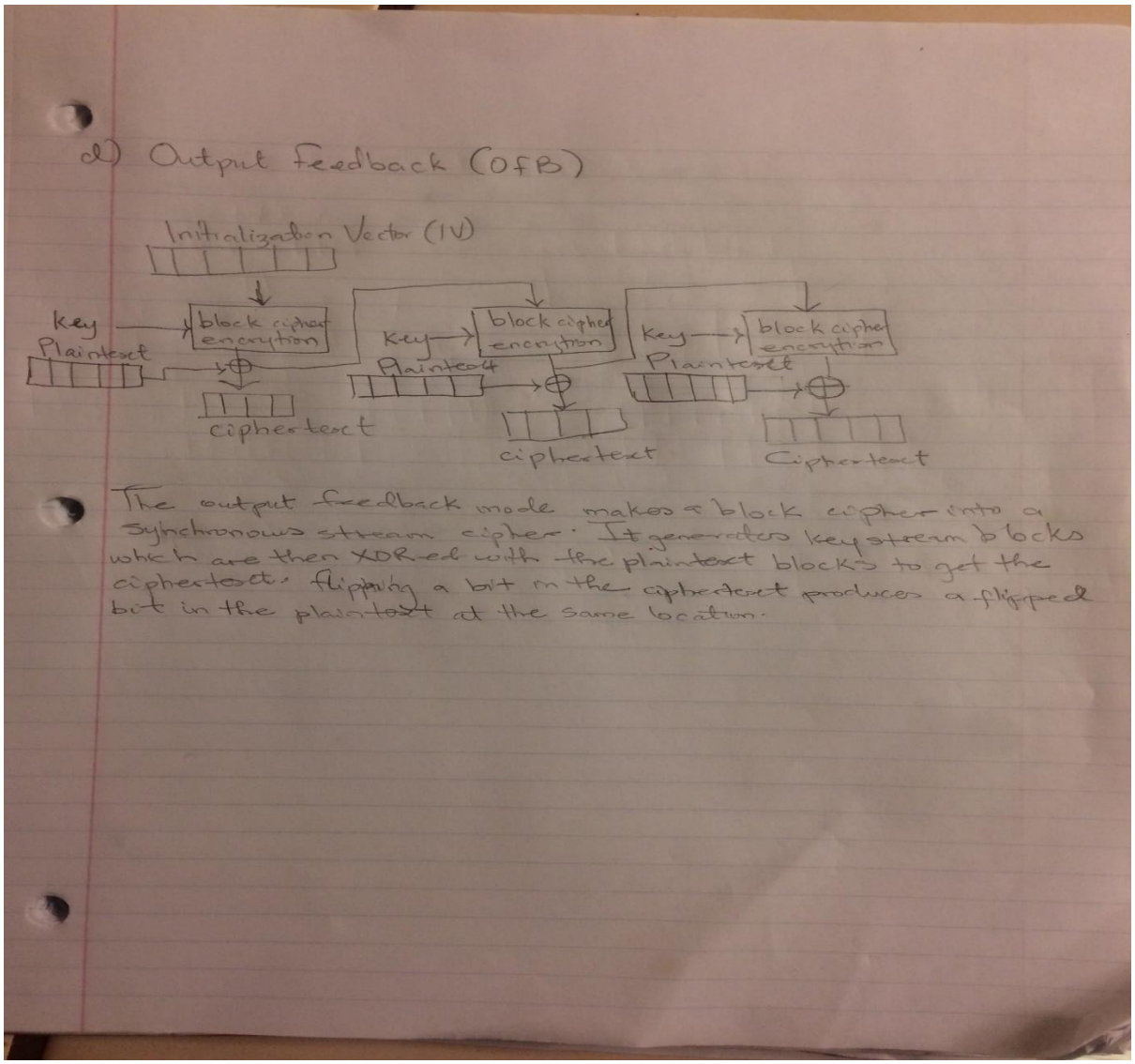
Decrypted Corrupted OFB:

decryted_corrupted_ofb.txt

0	54686973 20697320 61207465 78742066 696C6520	This is a text file
20	75736564 20617320 696F7075 7420666F 72204372	used as ioput for Cr
40	7970746F 5F6C6162 5F312E20 4578656E 64696E67	ypto_lab_1. Exending
60	20746865 20746578 74206669 6C652074 6F206174	the text file to at
80	6C656173 74203634 20627974 65732069 6E206F72	least 64 bytes in or
100	64657220 746F2073 6F6C7665 20717565 7374696F	der to solve questio
120	6E206E75 6D626572 20332074 61736B20 332E4973	n number 3 task 3.Is
140	20746869 73207570 20746F20 36346279 7465733E	this up to 64bytes>
160	20492068 6F706520 69742069 732E0A	I hope it is.

Signed Int | little | (select some data)

0 out of 175 bytes



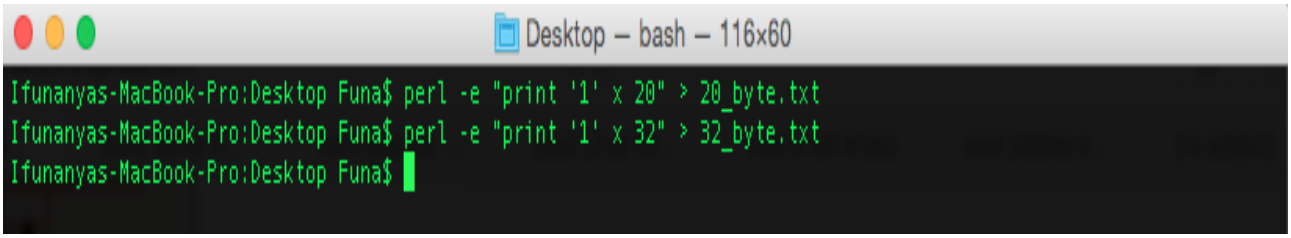
Task 4: Padding:

Lab description: 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:

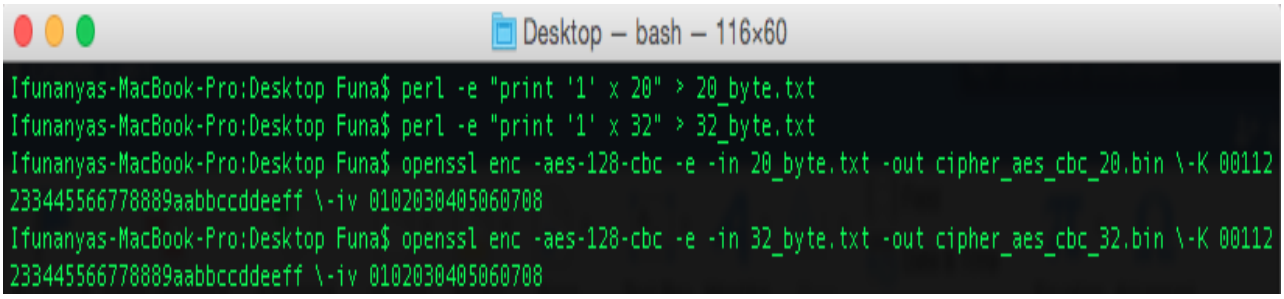
- The openssl manual says that openssl uses PKCS5 standard for its padding. Please design an experiment to verify this. In particular, use your experiment to figure out the paddings in the AES encryption when the length of the plaintext is 20 octets and 32 octets.
- Please use ECB, CBC, CFB, and OFB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

A. According to the openssl manual all the block ciphers normally use PKCS5 padding (also known as standard block padding), and cipher blocks of 8 bytes (DES & Blowfish) or 16 bytes (AES).

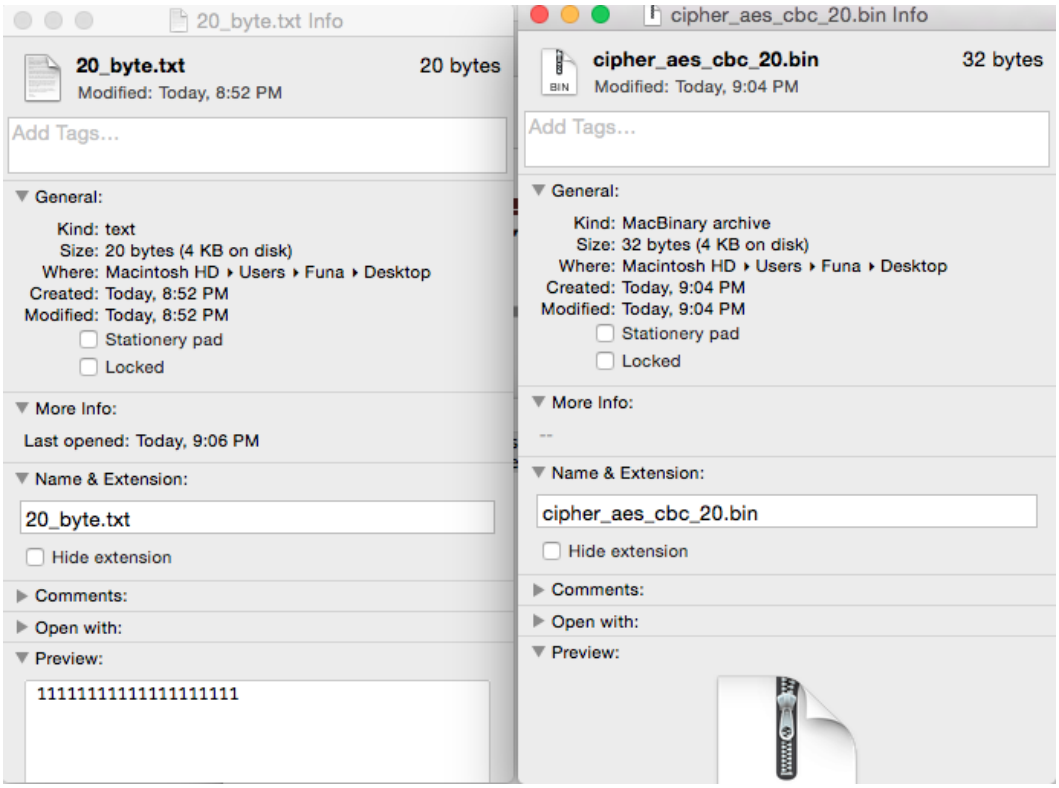
- First I created two files; 20 bytes and 32 bytes.



- and these files are encrypted by AES-128-CBC.

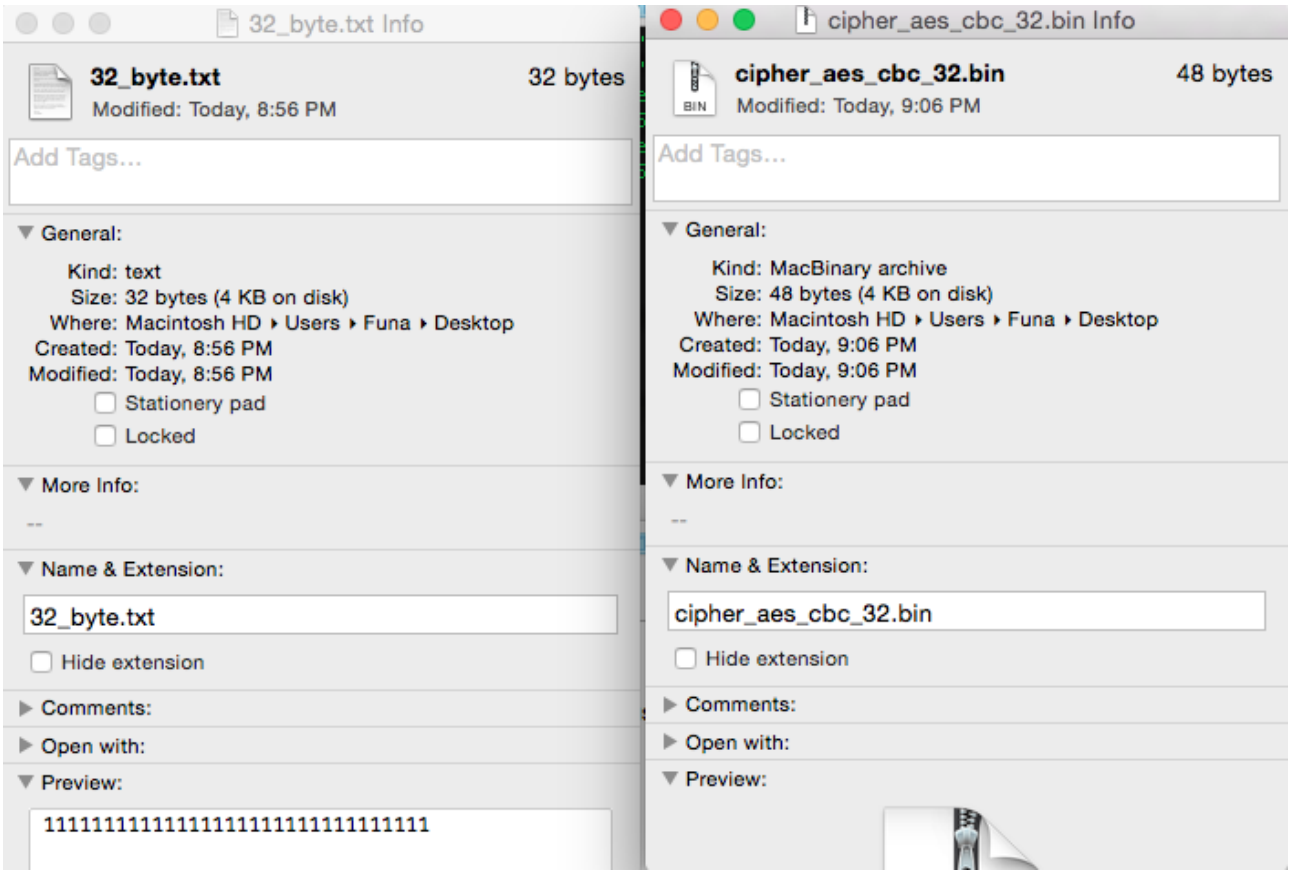


- The block size of 128 bits is equal to 128/8 = 16 bytes. Comparing the size of the 20byte plain text file to that of its cipher text, we notice below that some padding has been done:



As can be seen, the size of the encrypted/cipher text file is 32 bytes, this is 12 bytes more than the original 20byte file. We see here that 12 bytes of padding has been added to the encrypted file. This is because AES-128-CBC was used, therefore each block is 16 bytes. Since the content of the file is 20 bytes, and we know that the first block has to be 16 bytes, this means that the second block which is 4 bytes needs an additional 12bytes.

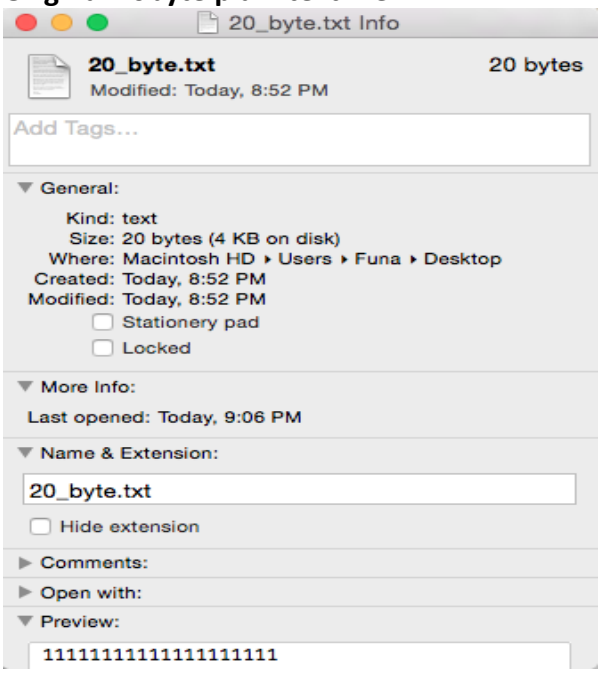
- Comparing the size of the 32byte plain text file to that of its cipher text, we notice below that some padding has been done:



As can be seen, the size of the encrypted/cipher text file is 48 bytes, this is 16 bytes more than the original 32byte file.

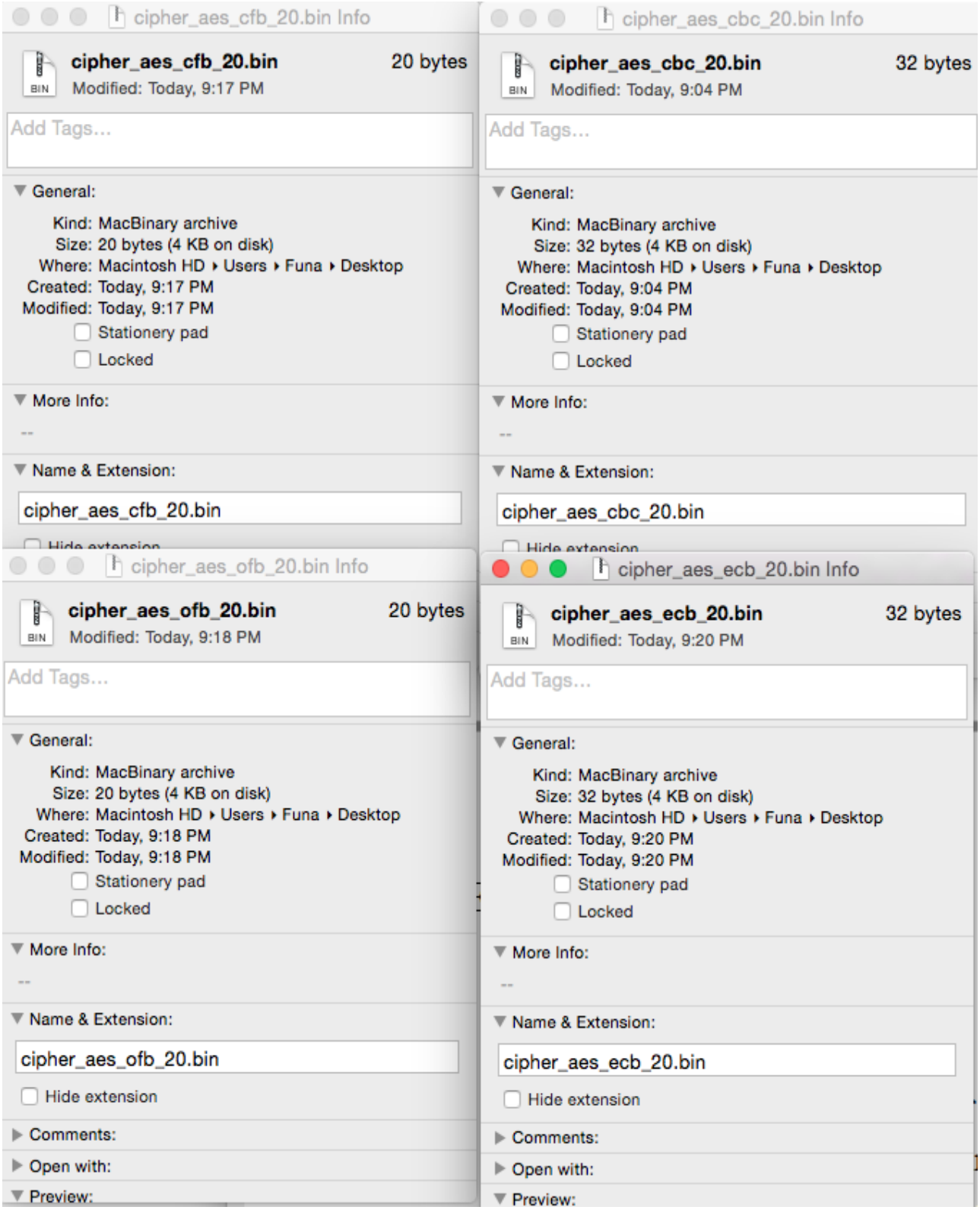
B. In this next part we compare all four modes of encryption and observe which modes utilize padding and which do not. Here I have used AES-128 to encrypt the 20byte plain text file using all four modes.

Original 20byte plain text file



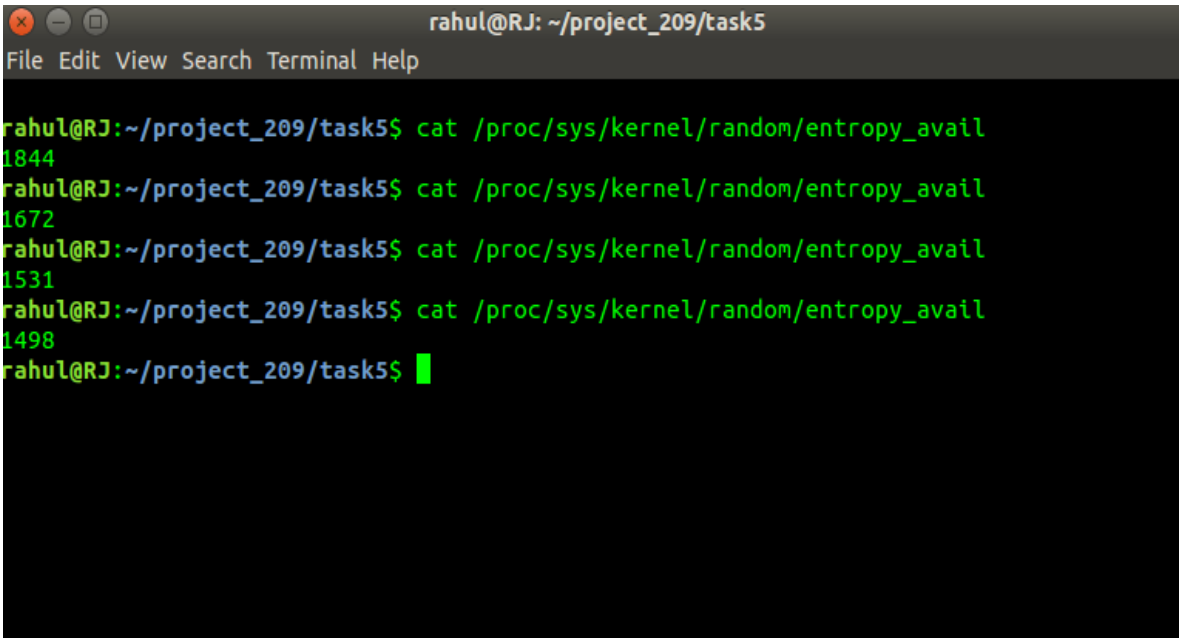
- Notice below that modes CBC and ECB utilize padding of their cipher text files while CFB and OFB do not. This is because Padding is needed for ECB and CBC encryption modes due to the fact that their encryption algorithm uses the cipher “blocks” mechanism to encrypt plain text input, thus padding is required to ensure that each block is the required size depending on the mode. AES uses 16 byte blocks while Blowfish and 3DES use 8-byte blocks. On the other hand, in the case of CFB and OFB padding isn’t required because they are stream ciphers, where each character is enciphered on its own.

CFC vs CBC vs OFB vs ECB cipher file size comparison:



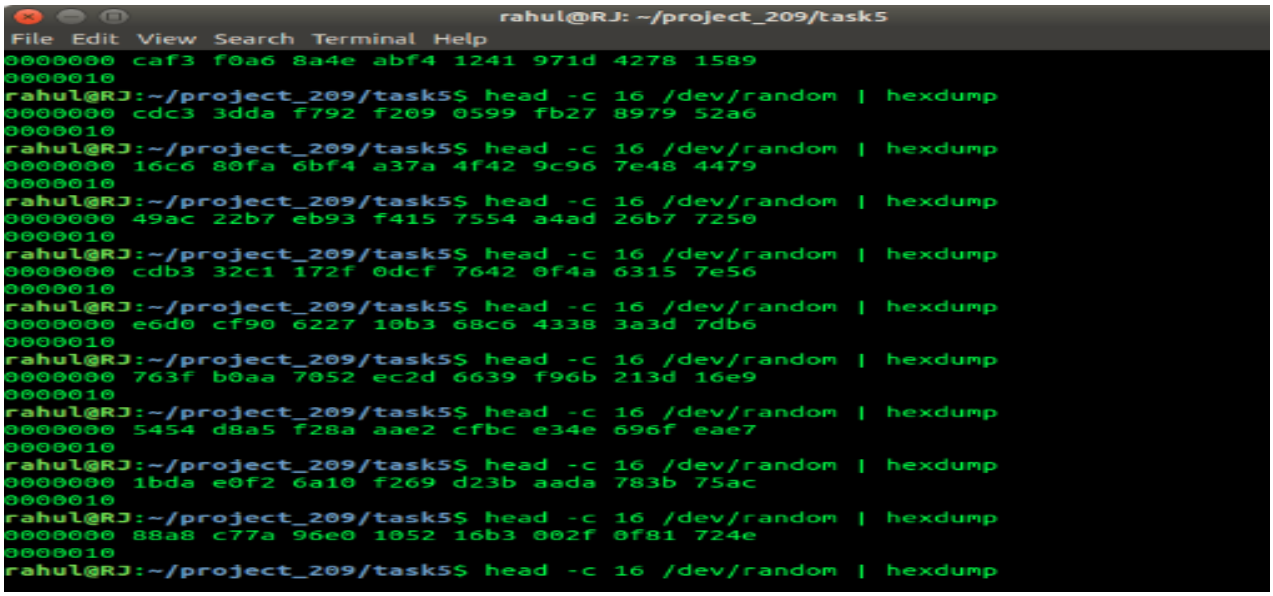
Task 5: Pseudo Random Number Generation

5.A:



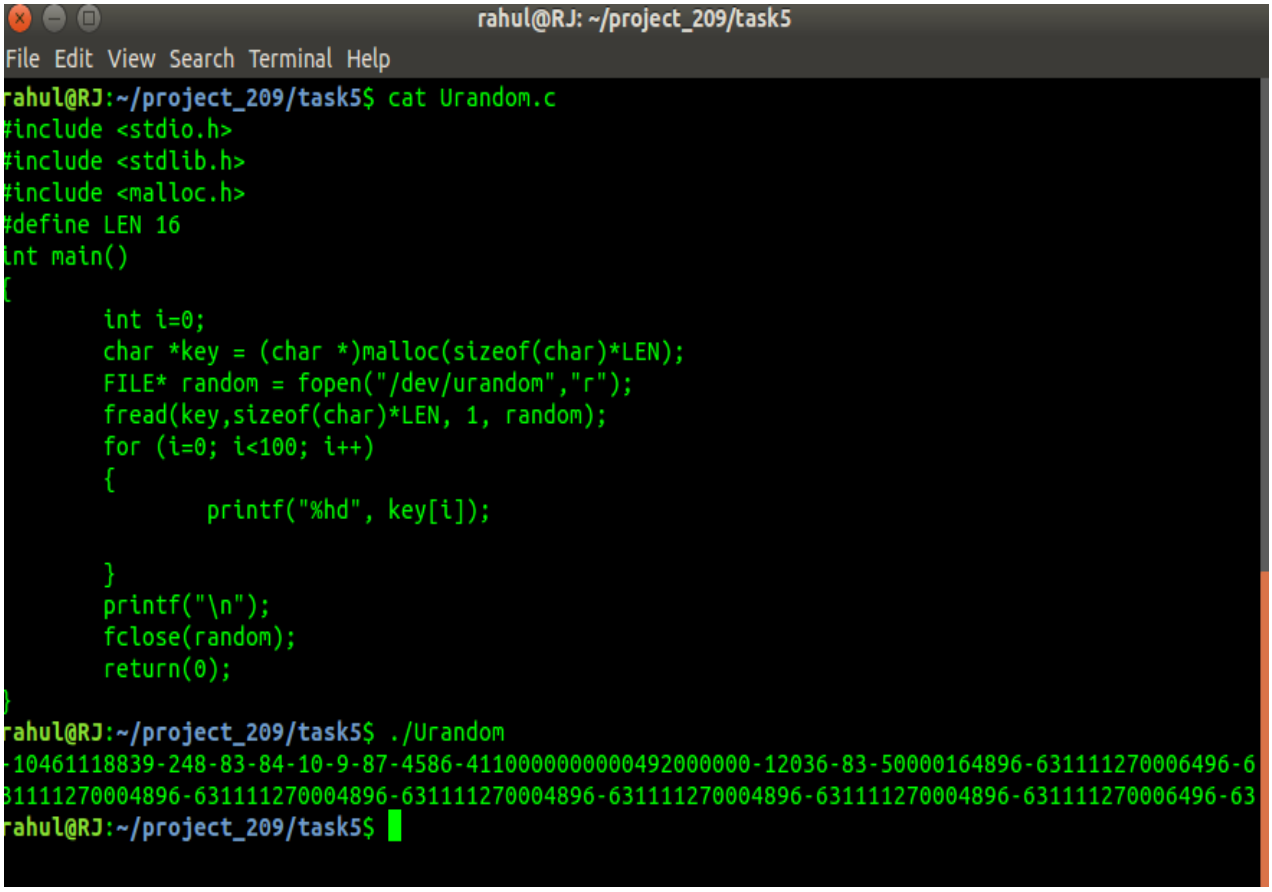
Upon executing the given command "`% cat /proc/sys/kernel/random/entropy_avail`", we got a result which shows how many bits of random numbers the system currently has. The more you move the mouse or more you type the lesser the entropy availability is. If we wait for few seconds, the entropy pool gathers more randomness and again the number random numbers it can generate is increased.

5B:



Upon executing the given command "`% head -c 16 /dev/random | hexdump`" several times, at some point it will get blocked. This happens as a result of inadequacy of numbers in random pool. Once it gathers more randomness over time, it again functions properly.
Data getting blocked for a particular amount of time may lead to denial of service, This limitation can be overcome by using NONBLOCKING mode of random or by using /dev/urandom which generates unlimited random numbers.

5C:



The above program read the random numbers from the file directly. The output of the program totally depends on the length of the random numbers we want to read from the file. The more the size, the more number of random numbers are generated. For instance, LEN = 16 in the current code which means 128 bits of random numbers.
We also changed the length to 1600 to see the difference in output. We also had to change the range of the output to print the generated random numbers.

```
rahul@RJ:~/project_209/task5$ cat Urandom.c
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#define LEN 1600
int main()
{
        int i=0;
        char *key = (char *)malloc(sizeof(char)*LEN);
        FILE* random = fopen("/dev/urandom","r");
        fread(key,sizeof(char)*LEN, 1, random);
        for (i=0; i<1000; i++)
        {
                printf("%hd", key[i]);

        }
        printf("\n");
        fclose(random);
        return(0);
}
rahul@RJ:~/project_209/task5$ ./Urandom
-10-121111-9182-117112350-121-39-7549497259-2763534-60110-10992-53-1-109-52-63-10593-45-34-7927295-101-7121-10011188101-1196921434218-1-1857
-71-1027761-119281566-19100-510012760-3560-84-15-11272-36-56-38-571059-104110-78-4011-384313-885-1119923100-7168-109-1284-10-14-574312-9084-
31-85116-86-10637120-436541-80-844-11838105734663-7810910854-114-6078-57-101-821224-1121074781-744976-10999125240511081l9-12372101455-19-50-
83-35-127-9174117-20-10643-97844-5149-102-89104-70-110-43-2730108-126-100217310072-9478-55-10443123-87-111-42-61117-71-48-67-25325-98-79-69-
58-457550-70-72-62481055735119-105119119-3539-3495-86-9124683395-8566-6199-75-149550-91117-46-43-697610634998102-164172-7071101-110-117-8785
-6093-402030-30-541-6377-56-2112-96-1168214239102-1958-31-8019-121-106-11974-9397-57491379-118-53-73-4-25-117-9118-15-111-103-361449-544225-
1231-20-49-125-126-103-1195311106-3611213-9-50-6072-113-31-872-121-93102-65-3125-128-3440-699-2094-7771-119-722-928-5555-784681-22-334-83551
119-81-9710259-123124101-105369-76-1232-92-42116108-18-18-80478615114118-12238-17-12449-1191112129-5419127-172619575920-55-55-9-7398-128-33-
127631106351-682189-5829-62884593-120102935447950-79871122-25121-44104-43124-92-11-514212064-46-1519253-103-7-610736-38-123-88-108-8140-48-1
12622-3629-241154-8466871871-2168517414345110366-529-671764-11269831-93117531118-59-91-127-124107-4-403-111-100-1058108120-12867587384-46113
-11106-7-11861-83-5291-1-1226012110247-113-9-703123-92-66-18228-127-100-125-84100-10-16-532299-71-49-16-94-14-113-103-49878926-40-20-109-8-4
682116-12116-47-10870-40-4346-3026488-80-21-38-103399411512193-298897-116-9026102-41-32506771-36-11949-493-2668-7754127382-47-31-10871120111
8897113-27-115-39-1483-78-931174-3646-105-3410991124-120-989-109-1543-28-58-10-116-92121-6742456011279-11714892211228-2093-12-5077-55-49-118
```