



SAN JOSÉ STATE UNIVERSITY

NETWORK SECURITY

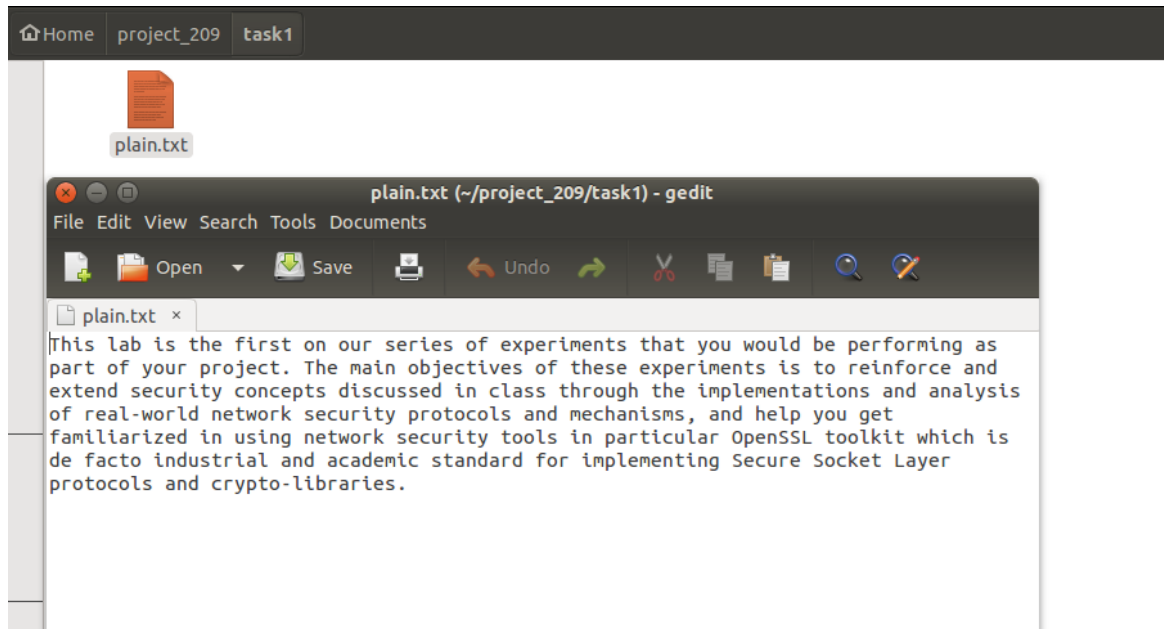
Crypto Lab 1 – Secret-Key Encryption

IFUNANYA NNOKA – 010754159

RAHUL DILEEP JADHAV - 010687638

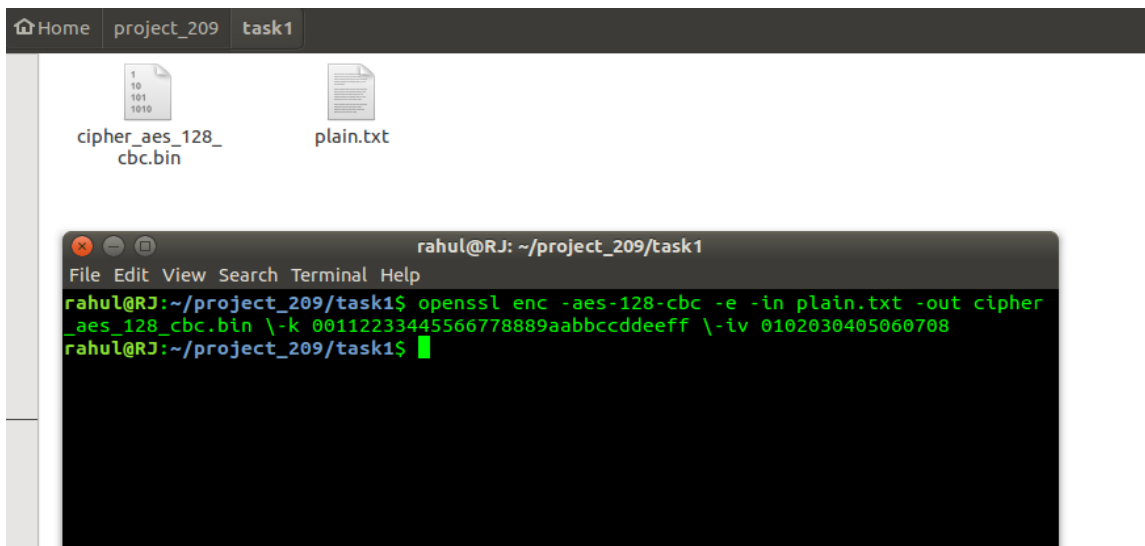
Task 1: Encryption using different ciphers and modes

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.

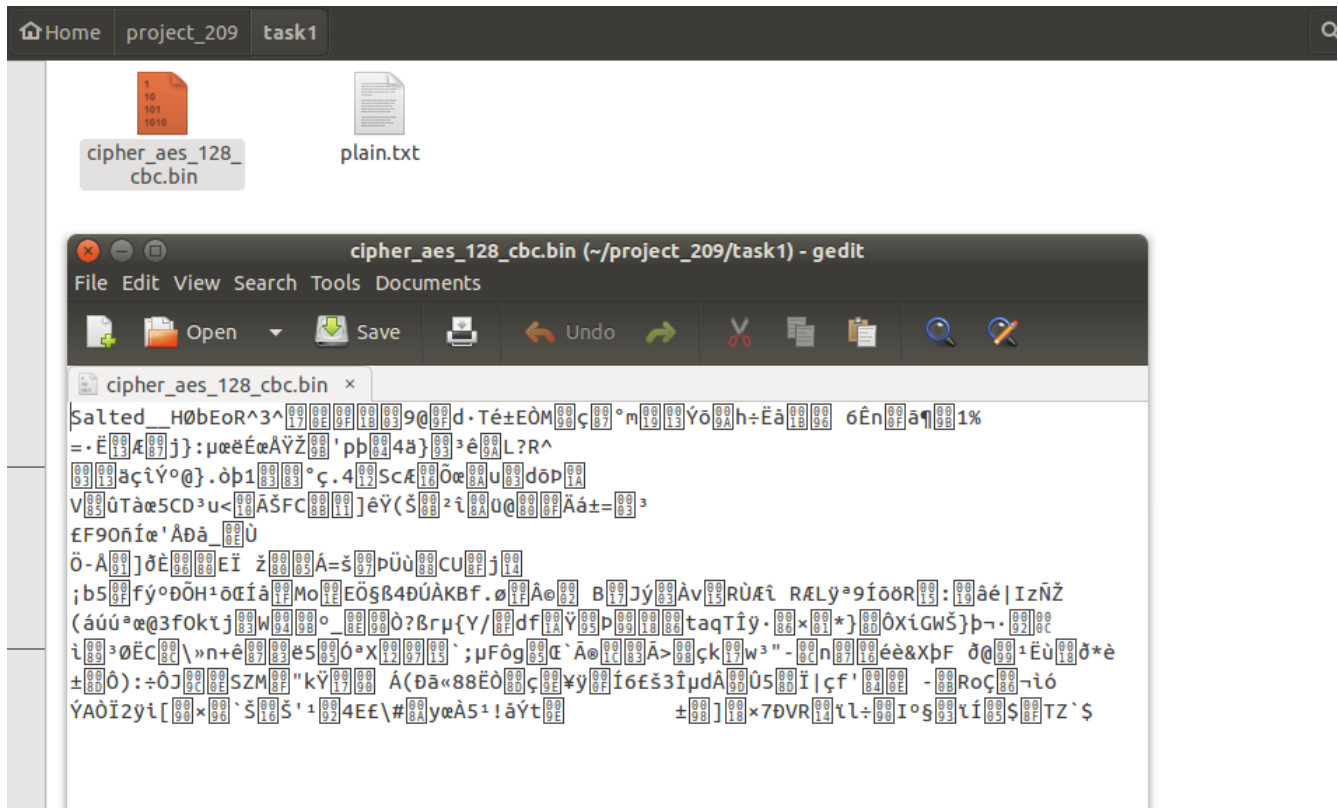


➤ For cipher and mode: **-aes-128-cbc**

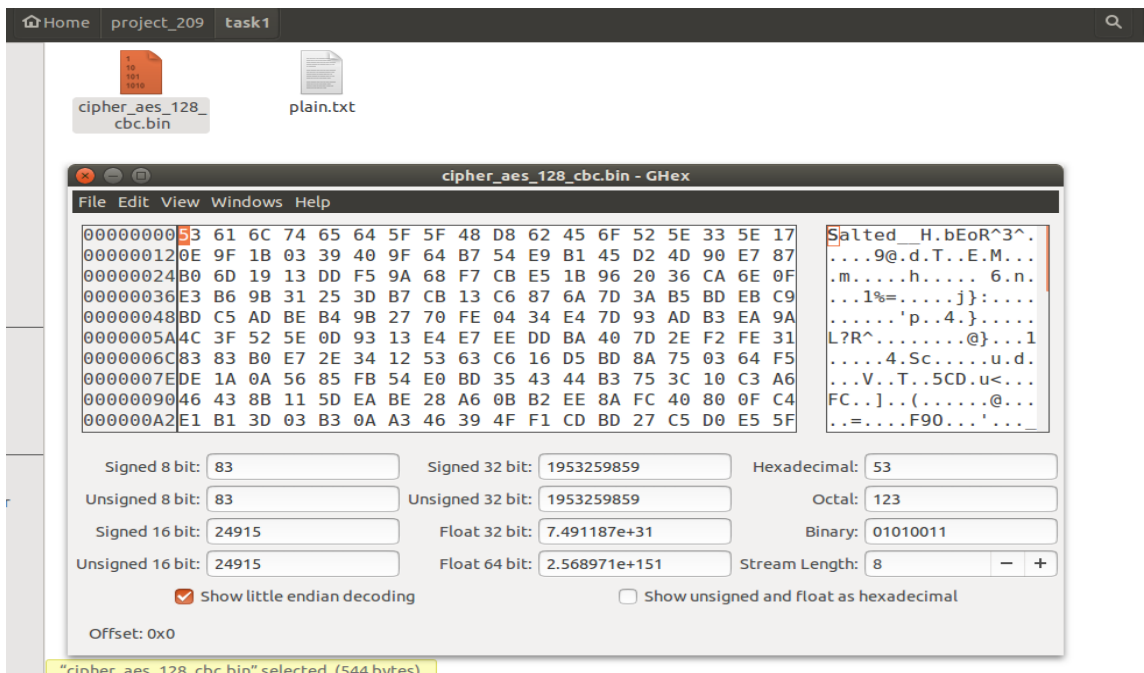
Encryption:



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

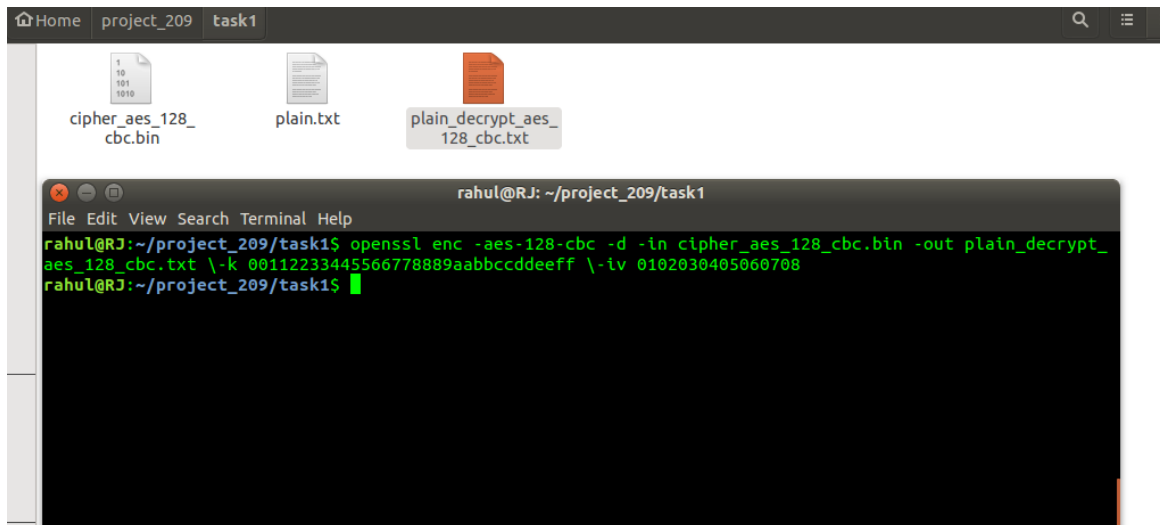


The encrypted file contains hex values so we used ghex editor to open the file.



Decryption:

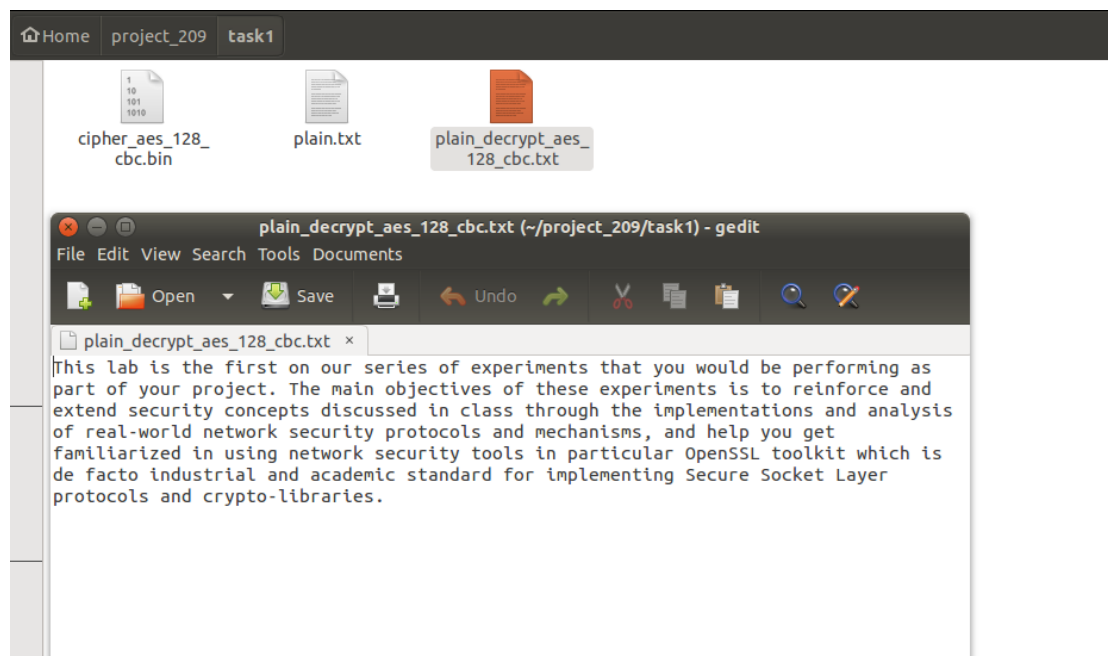
The encrypted file is decrypted to obtain the original plain text.



The screenshot shows a file manager window with three files: `cipher_aes_128_cbc.bin`, `plain.txt`, and `plain_decrypt_aes_128_cbc.txt`. Below the file manager is a terminal window titled `rahul@RJ: ~/project_209/task1`. The terminal shows the following commands and output:

```
rahul@RJ:~/project_209/task1$ openssl enc -aes-128-cbc -d -in cipher_aes_128_cbc.bin -out plain_decrypt_aes_128_cbc.txt \-k 00112233445566778889aabbccddeeff \-iv 0102030405060708
rahul@RJ:~/project_209/task1$
```

Plain Text:

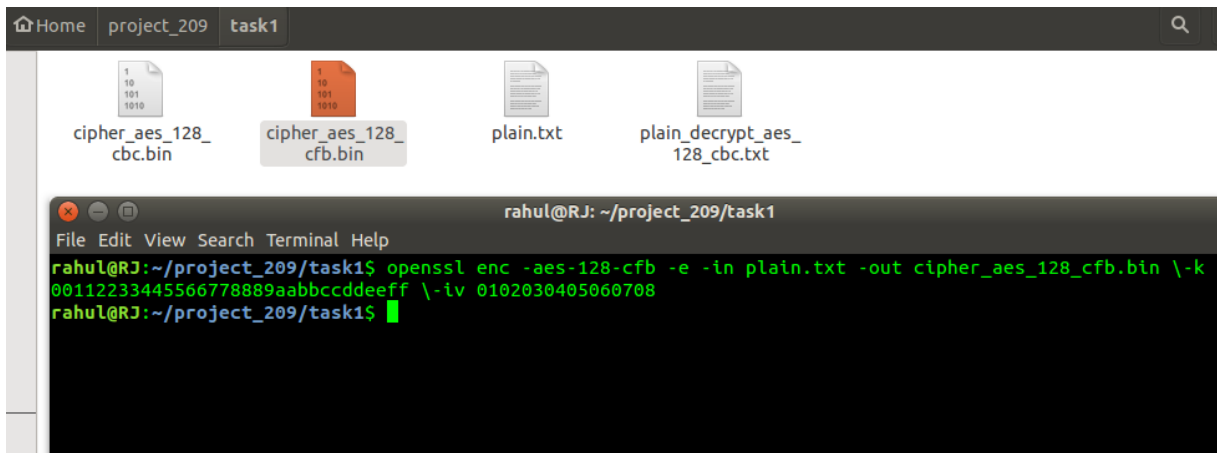


The screenshot shows the same file manager window as before. Below it is a text editor window titled `plain_decrypt_aes_128_cbc.txt (~/.project_209/task1) - gedit`. The text editor displays the following text:

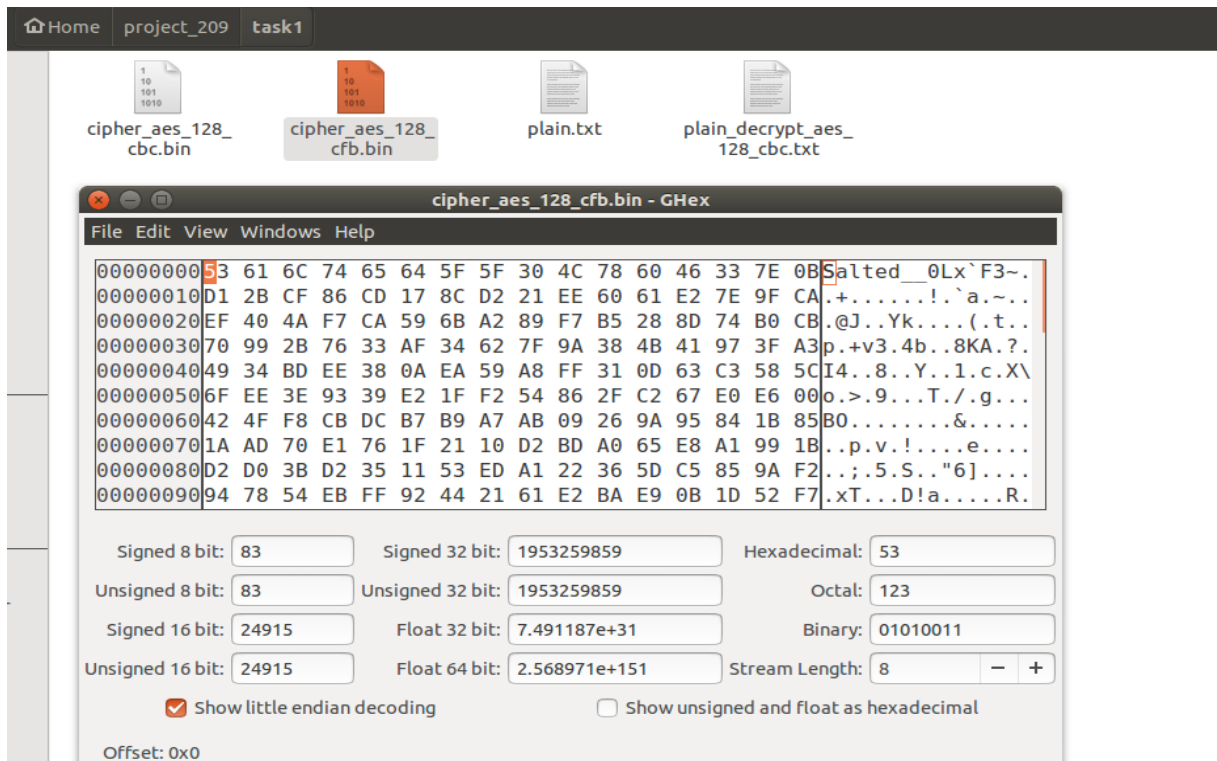
```
This lab is the first on our series of experiments that you would be performing as part of your project. The main objectives of these experiments is to reinforce and extend security concepts discussed in class through the implementations and analysis of real-world network security protocols and mechanisms, and help you get familiarized in using network security tools in particular OpenSSL toolkit which is de facto industrial and academic standard for implementing Secure Socket Layer protocols and crypto-libraries.
```

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

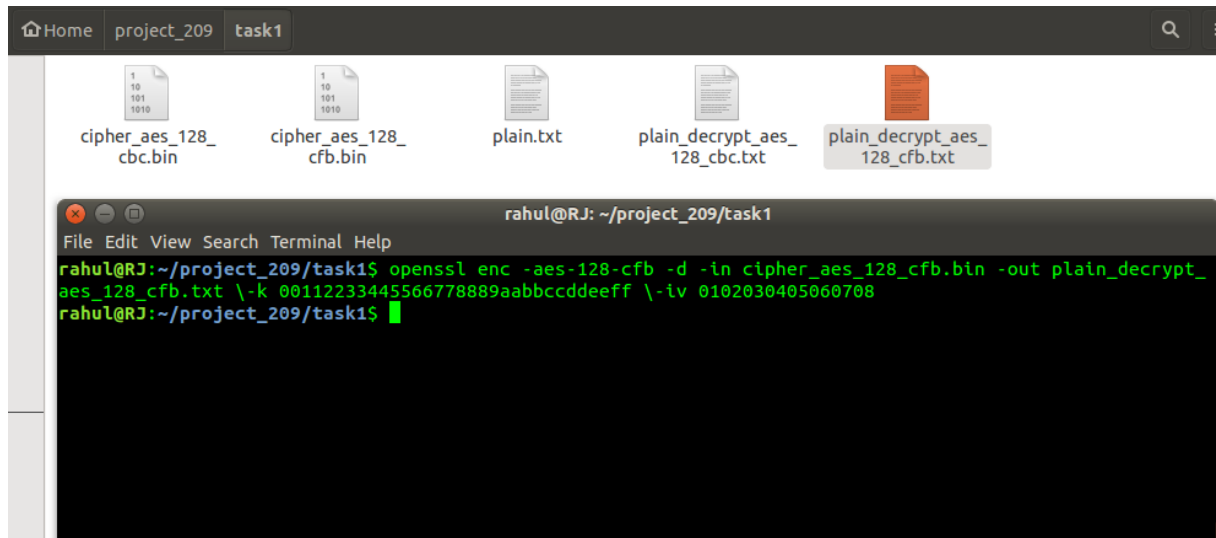
Encryption:



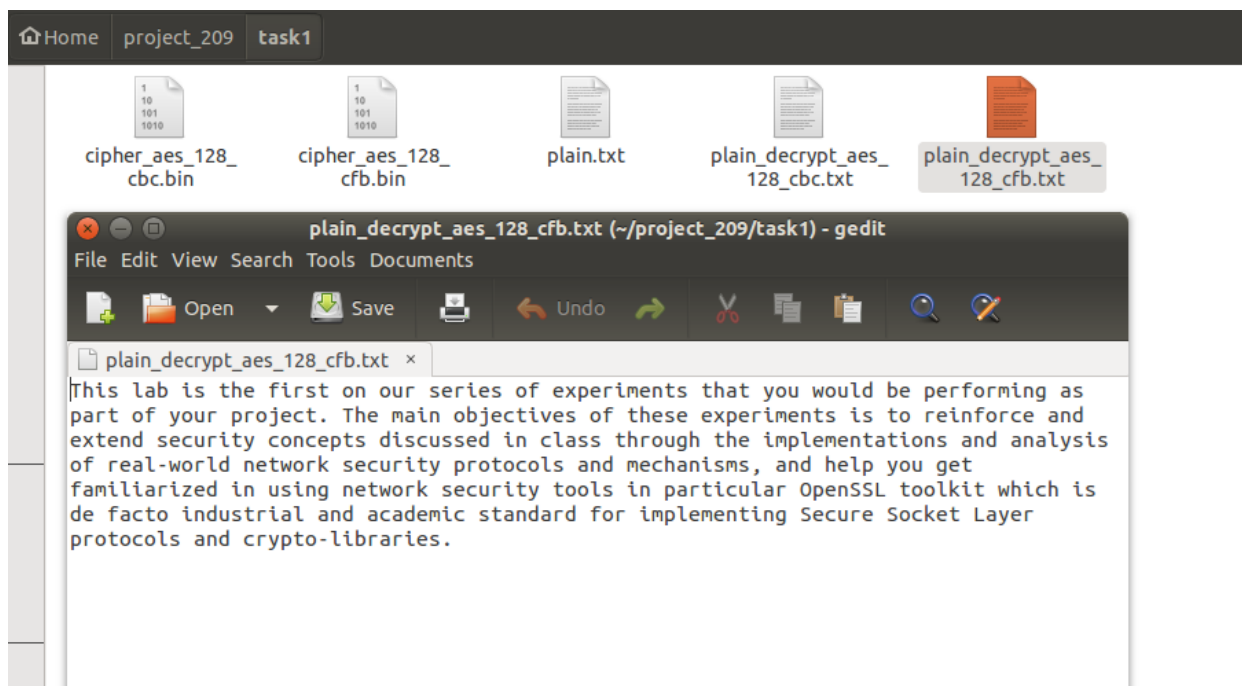
The encrypted file contains hex values so we used ghex editor to open the file.



Decryption:



Plain Text:

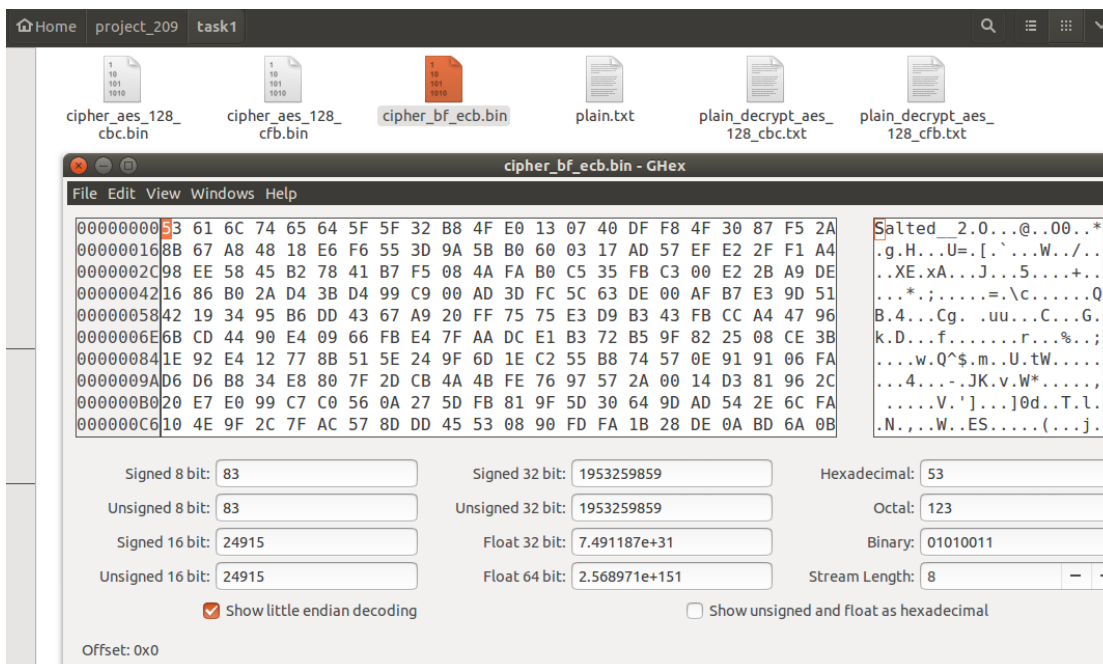


- For cipher and mode: -bf-ecb

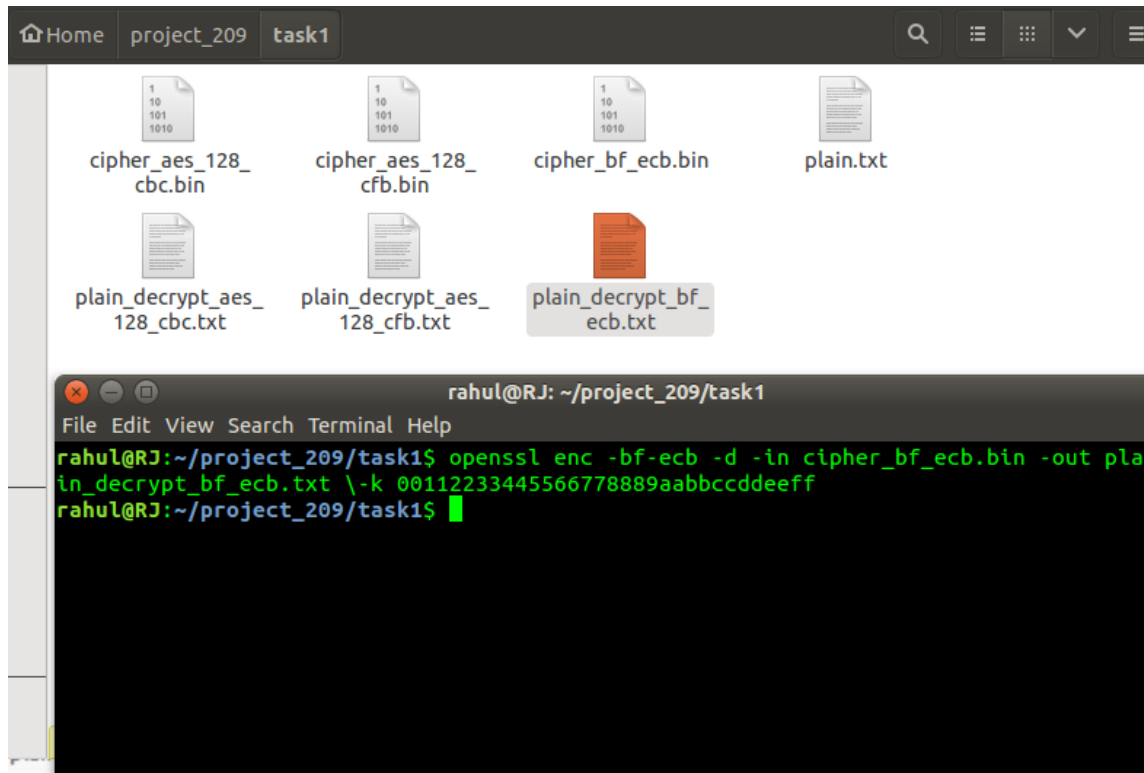
Encryption:



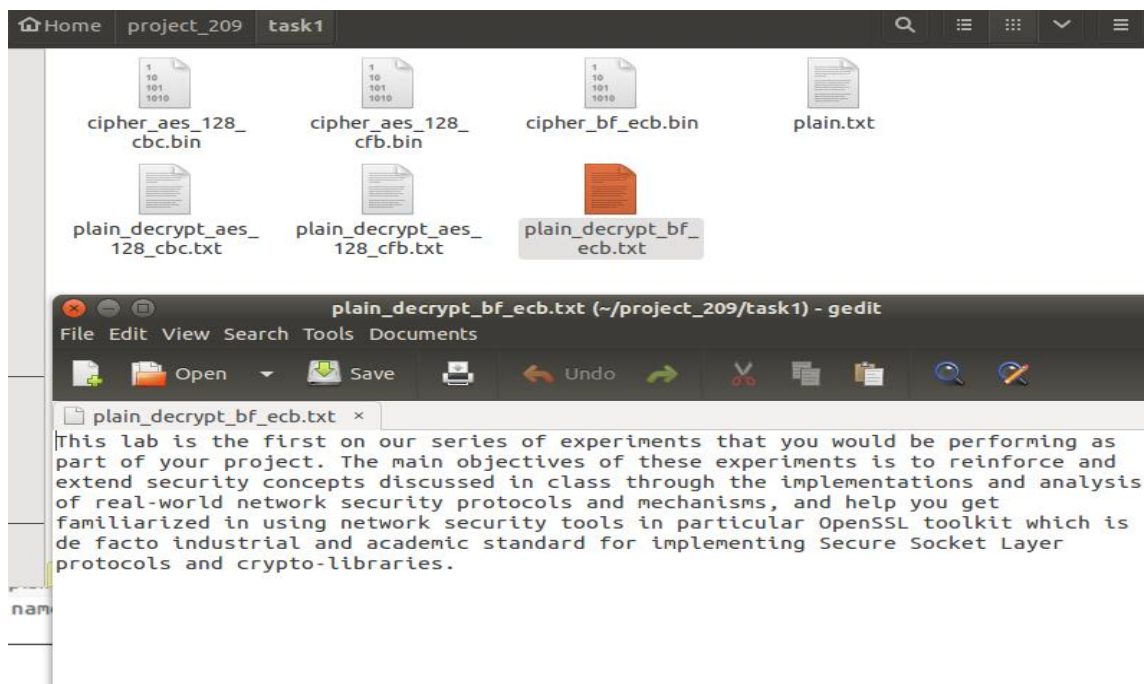
The encrypted file contains hex values so we used ghex editor to open the file.



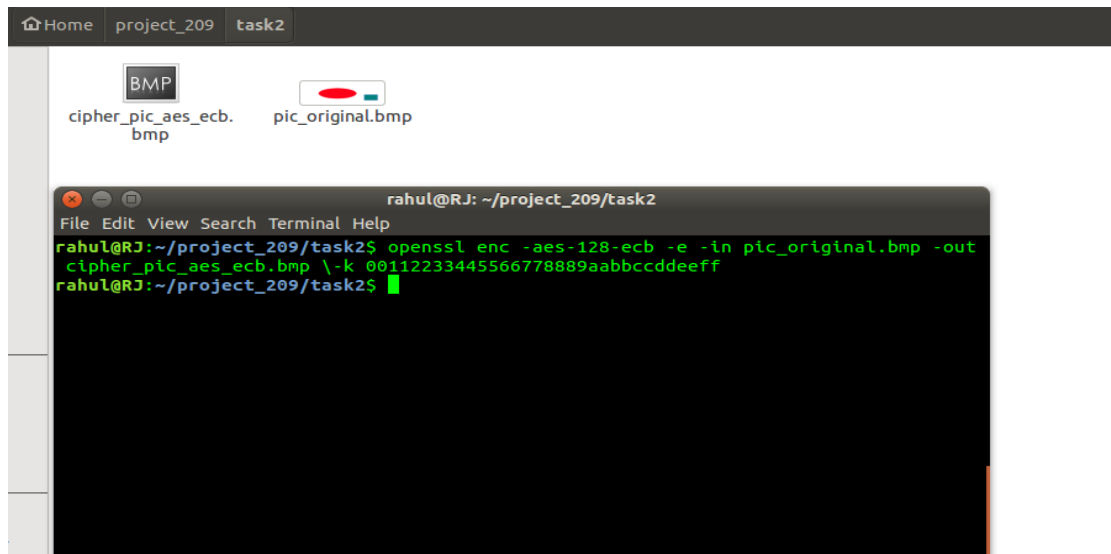
Decryption:



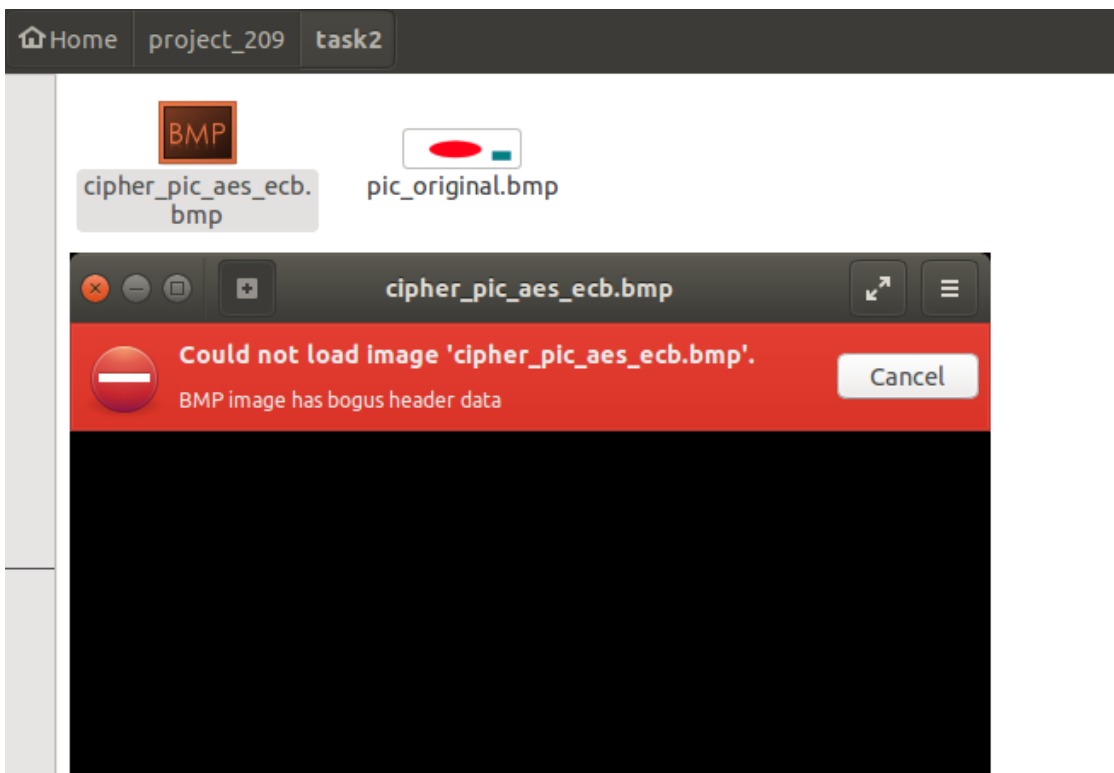
Plain Text:



Task 2: Encryption Mode – ECB vs. CBC

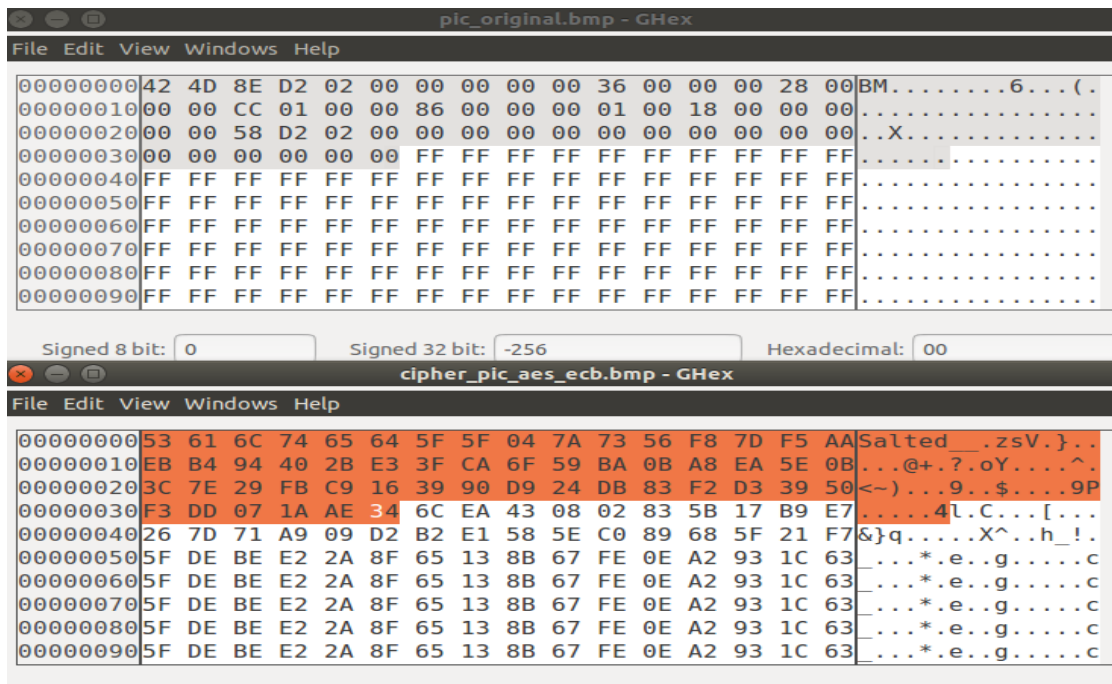


The given image “pic_original.bmp” is encrypted using electronic codebook encryption mode. But when we try to open the encrypted image it throws an error and it does not open the image.

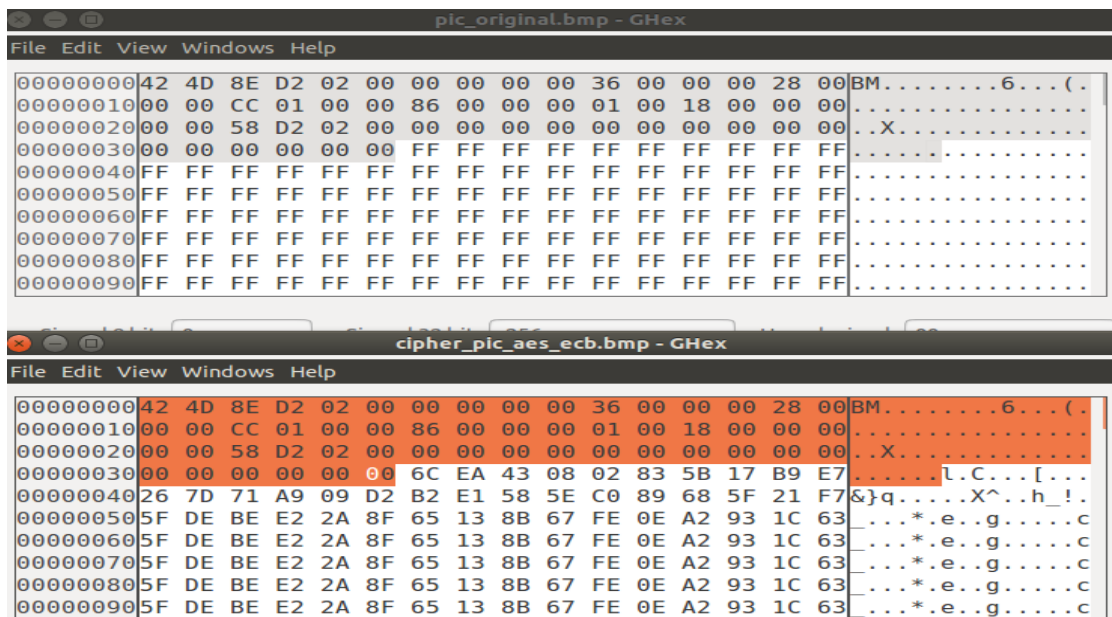


This occurred because the header of the file is not set correctly. In order to set the header, we modified the header of the encrypted file to match the header of the original file.

Before Changing the header (54 bytes)



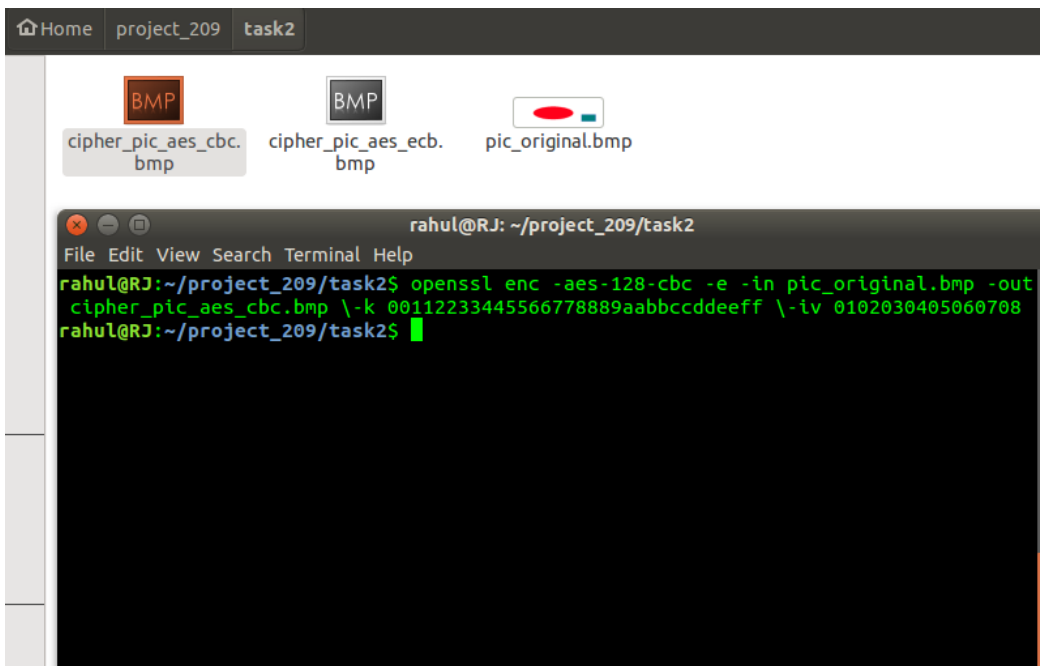
After Changing the header (54 bytes):



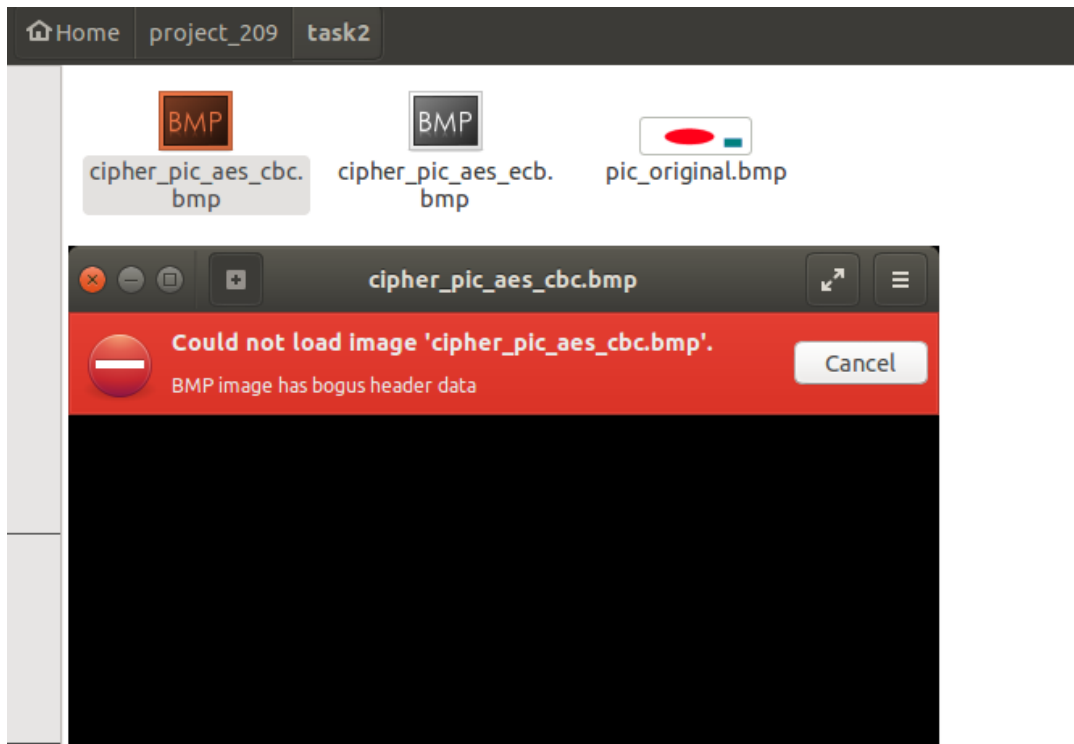
After changing the header of the encrypted image it can be viewed using any image viewing software



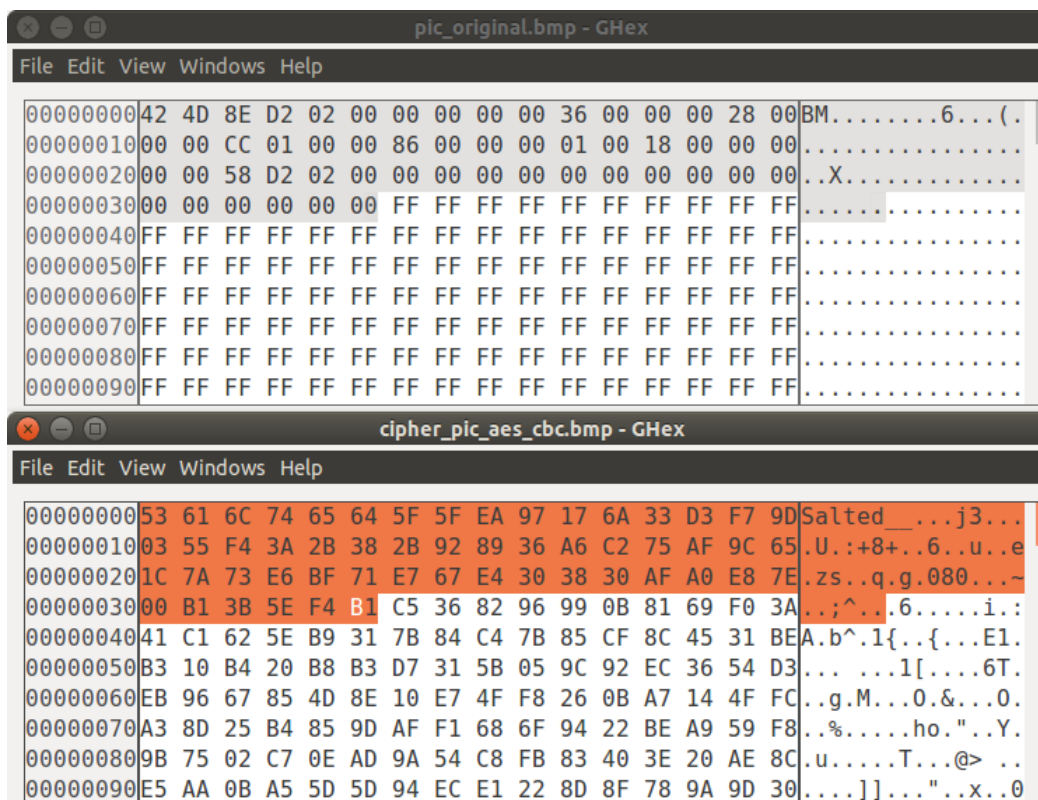
Encryption – CBC:



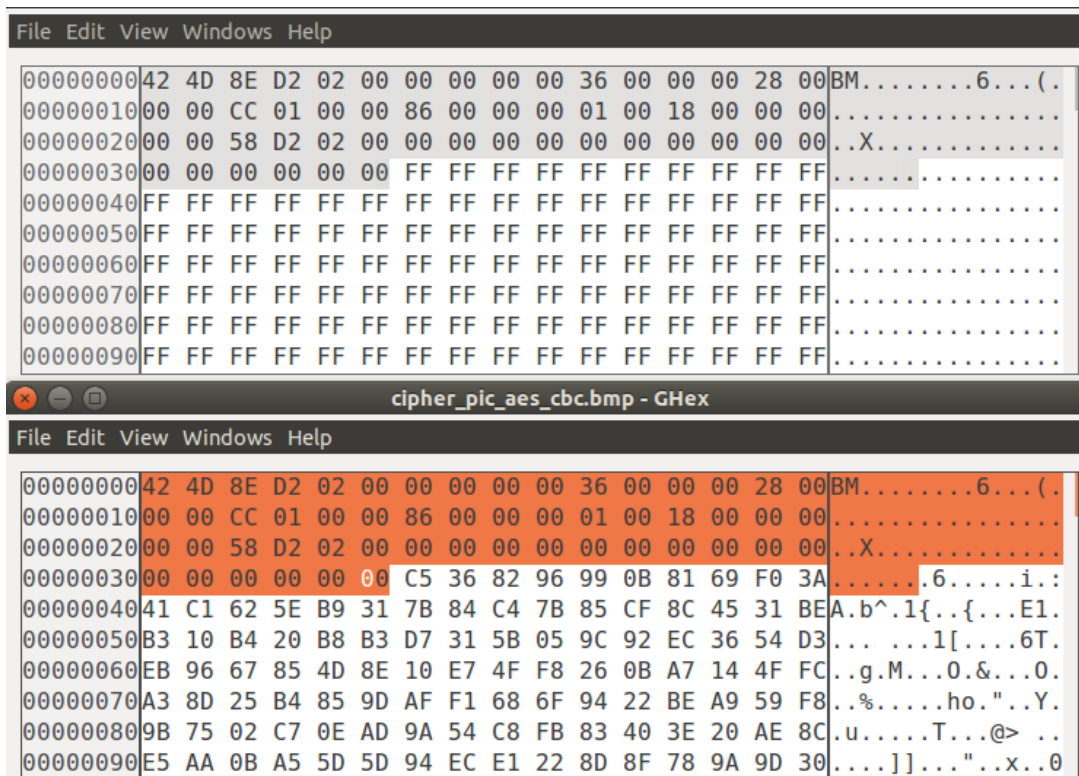
We encrypted the image using CBC mode. In order to open the encrypted image, we changed 54 bytes of the header of the image.



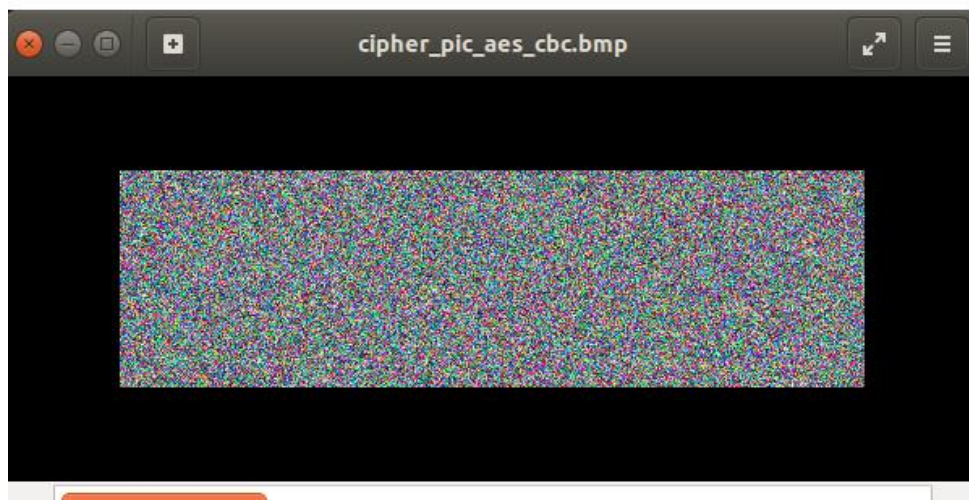
Before Changing the header (54 bytes):



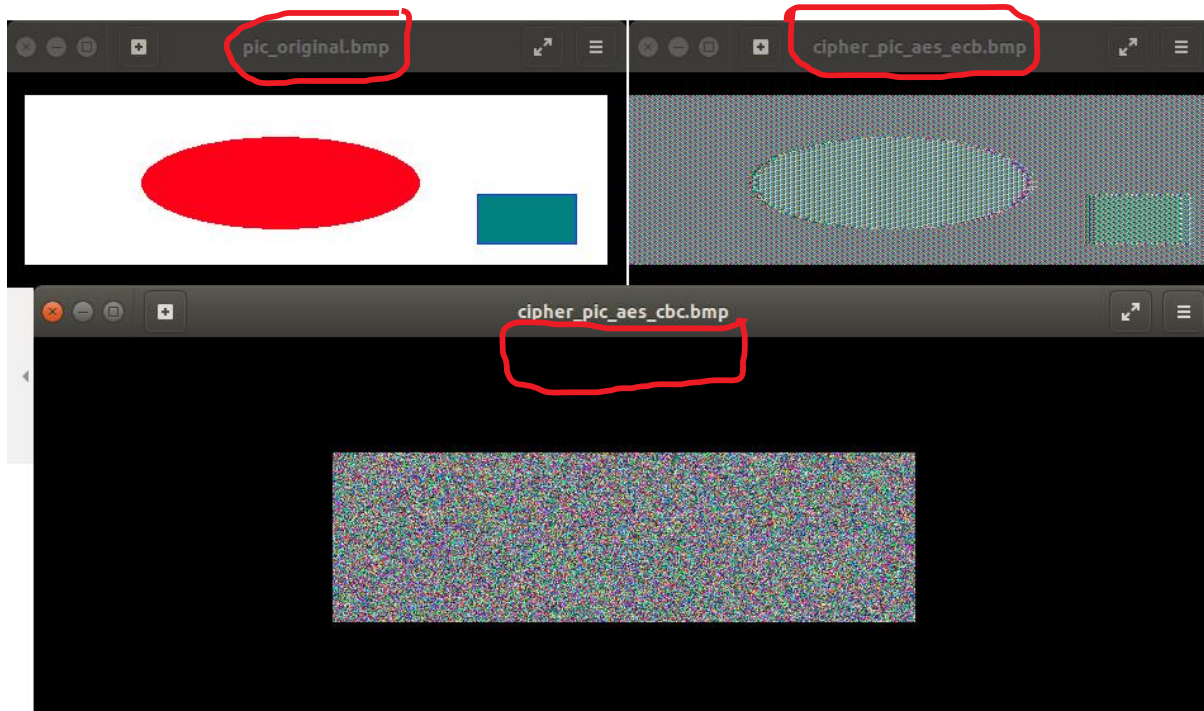
After Changing the header (54 bytes):



Encrypted Image:



Observation:



- In ECB mode the data patterns 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 00112233445566778889aabbccddeeff
Ifunanyas-MacBook-Pro:Desktop Funa$ cat cipher_ecb.bin
T??m??U?L?KH?~0?Q??#???jb
?k???_?_*/@>C?[#?H?Y?(+Q????S?r"4t5%??2b,N?.!Mc??W!y??Y???h[Q$??L???7TRr[?t[?])
~?v?%?<?)/
?
??l??
_:X??[?0Z???Ifunanyas-MacBook-Pro:Desktop Funa$
Ifunanyas-MacBook-Pro:Desktop Funa$ openssl enc -aes-128-ecb -d -in cipher_ecb_corrupted.bin -out decrypted_corrupted_ecb.txt \-K 00112233445566778889aabbccddeeff
Ifunanyas-MacBook-Pro:Desktop Funa$ cat decrypted_corrupted_ecb.txt
This is a text f0?#FW?B?9?w?t for Crypto_lab_1. Extending 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$
```

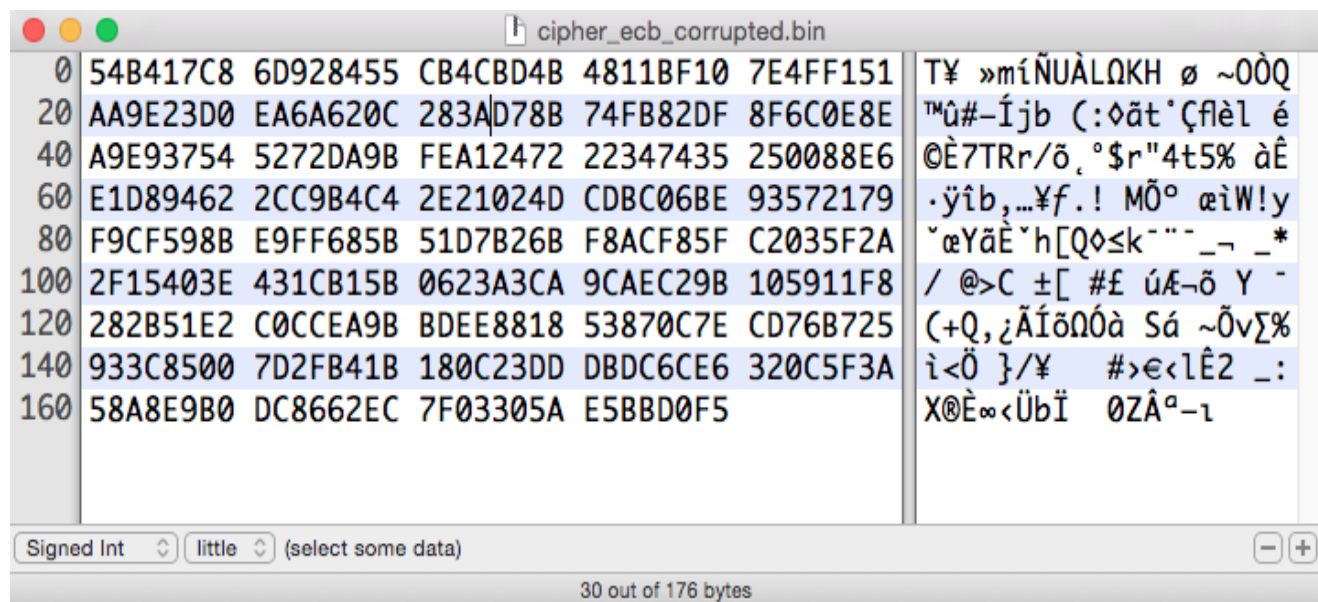
Uncorrupted ECB Cipher:

cipher_ecb.bin	
0	54B417C8 6D928455 CB4CBD4B 4811BF10 7E4FF151 T¥ »míÑUÀLQKH ø ~0ÒQ
20	AA9E23D0 EA6A620C 283BD78B 74FB82DF 8F6C0E8E ™û#-Íjb (;ðät'Çflèl é
40	A9E93754 5272DA9B FEA12472 22347435 250088E6 ©È7TRr/ö,°\$r"4t5% àÊ
60	E1D89462 2CC9B4C4 2E21024D CDBC06BE 93572179 .ÿib,...¥f.! MÕ° æiW!y
80	F9CF598B E9FF685B 51D7B26B F8ACF85F C2035F2A ~æYäË`h[Qð≤k`""- _ *
100	2F15403E 431CB15B 0623A3CA 9CAEC29B 105911F8 / @>C ±[#£ úÆ-õ Y -
120	282B51E2 C0CCEA9B BDEE8818 53870C7E CD76B725 (+Q,¿ÃÍöΩÓà Sá ~ÕvΣ%
140	933C8500 7D2FB41B 180C23DD DBDC6CE6 320C5F3A ì<Ö }/¥ #>e<lÊ2 _:
160	58A8E9B0 DC8662EC 7F03305A E5BBD0F5 X@È∞<Übİ 0ZÂª-ı

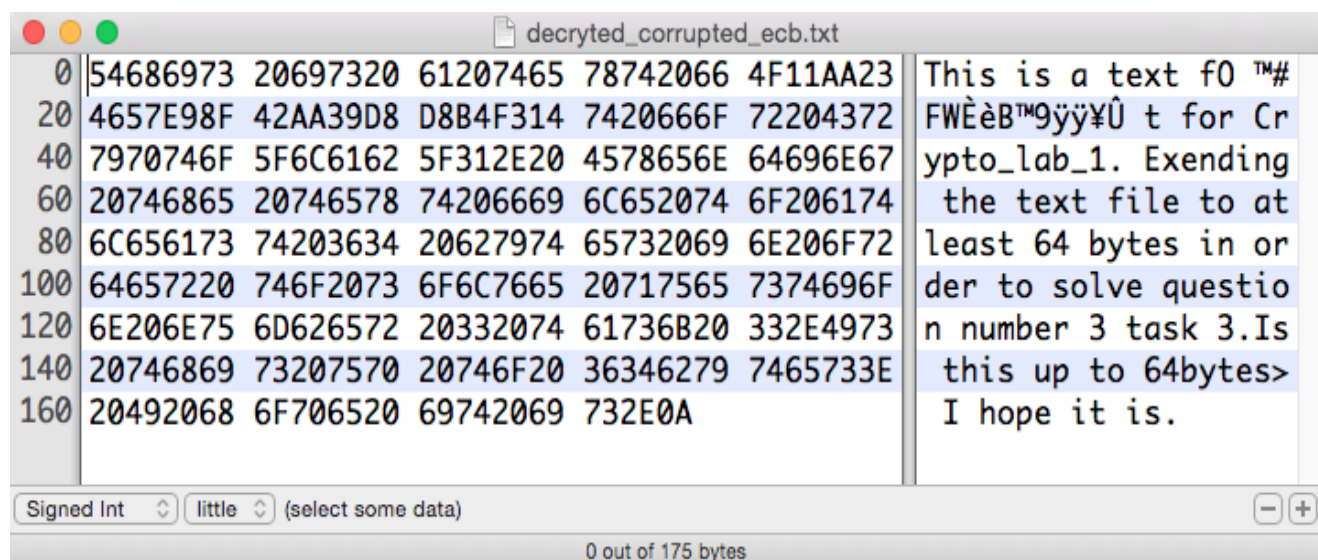
Signed Int little (select some data)

0 out of 176 bytes

Corrupted ECB Cipher:



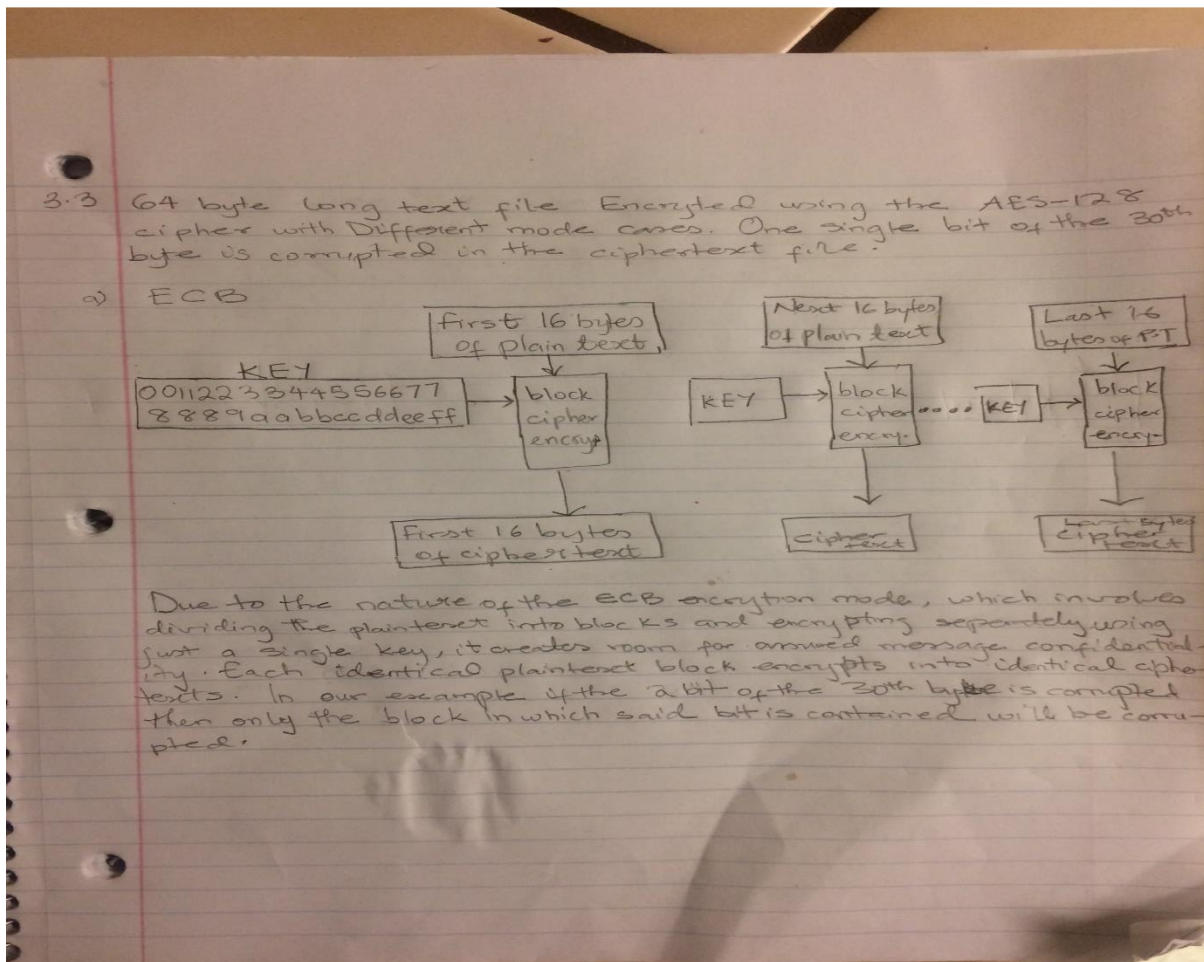
Decrypted Corrupted ECB Text file:



Original Text File:

```
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. Extending
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 64 bytes?
160 20492068 6F706520 69742069 732E0A I hope it is.
```

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 0011223344556677889aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ openssl enc -aes-128-cbc -d -in cipher_cbc_corrupted.bin -out decrypted_corrupted_cbc.txt \-K 0011223344556677889aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ cat decrypted_corrupted_cbc.txt
This is a text fP??Dg{?_uj-m?t for Crypto_mab_1. Extending 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$
```

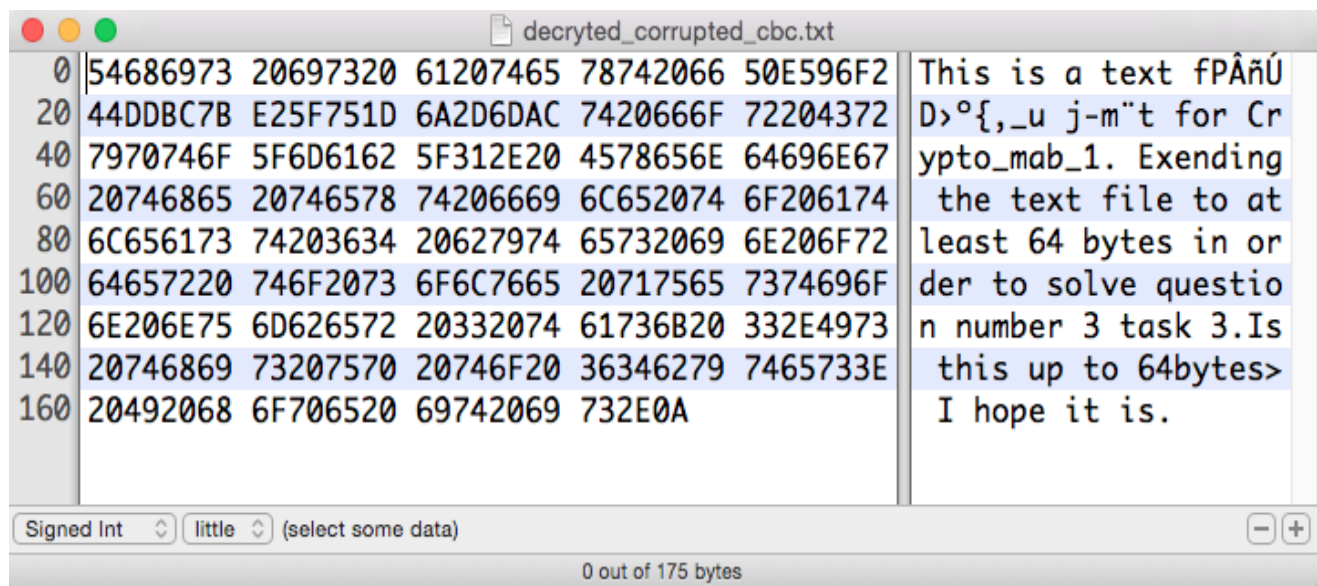
Uncorrupted CBC:

cipher_cbc.bin	
0	FECC483C 5D5A85F2 C4412CE6 7266E937 4F903789 6A5B8FCF 96D894BD 6B06D7DA 2E1015C8 85FD310B 703C2FE2 A3364E45 BFB18B4B 913ED73D CF2017AE 9EFB077A 5711983E 68264878 ACB670F4 DE3CABAC 49F0AC64 A5FECC88 FF5F0364 208DFB1B 7FBB07B5 1A9D8869 BEA13C49 1E522FC2 F3BF6275 556953F6 0F434C79 190BFE2E 2C3F663F BC8E1DF2 0EA3672B 733D4BE3 00B482B7 43C0DA81 F3529FB5
20	6A5B8FCF 96D894BD 6B06D7DA 2E1015C8 85FD310B 703C2FE2 A3364E45 BFB18B4B 913ED73D CF2017AE 9EFB077A 5711983E 68264878 ACB670F4 DE3CABAC 49F0AC64 A5FECC88 FF5F0364 208DFB1B 7FBB07B5 1A9D8869 BEA13C49 1E522FC2 F3BF6275 556953F6 0F434C79 190BFE2E 2C3F663F BC8E1DF2 0EA3672B 733D4BE3 00B482B7 43C0DA81 F3529FB5
40	703C2FE2 A3364E45 BFB18B4B 913ED73D CF2017AE 9EFB077A 5711983E 68264878 ACB670F4 DE3CABAC 49F0AC64 A5FECC88 FF5F0364 208DFB1B 7FBB07B5 1A9D8869 BEA13C49 1E522FC2 F3BF6275 556953F6 0F434C79 190BFE2E 2C3F663F BC8E1DF2 0EA3672B 733D4BE3 00B482B7 43C0DA81 F3529FB5
60	9EFB077A 5711983E 68264878 ACB670F4 DE3CABAC 49F0AC64 A5FECC88 FF5F0364 208DFB1B 7FBB07B5 1A9D8869 BEA13C49 1E522FC2 F3BF6275 556953F6 0F434C79 190BFE2E 2C3F663F BC8E1DF2 0EA3672B 733D4BE3 00B482B7 43C0DA81 F3529FB5
80	49F0AC64 A5FECC88 FF5F0364 208DFB1B 7FBB07B5 1A9D8869 BEA13C49 1E522FC2 F3BF6275 556953F6 0F434C79 190BFE2E 2C3F663F BC8E1DF2 0EA3672B 733D4BE3 00B482B7 43C0DA81 F3529FB5
100	1A9D8869 BEA13C49 1E522FC2 F3BF6275 556953F6 0F434C79 190BFE2E 2C3F663F BC8E1DF2 0EA3672B 733D4BE3 00B482B7 43C0DA81 F3529FB5
120	0F434C79 190BFE2E 2C3F663F BC8E1DF2 0EA3672B 733D4BE3 00B482B7 43C0DA81 F3529FB5
140	733D4BE3 00B482B7 43C0DA81 F3529FB5
160	

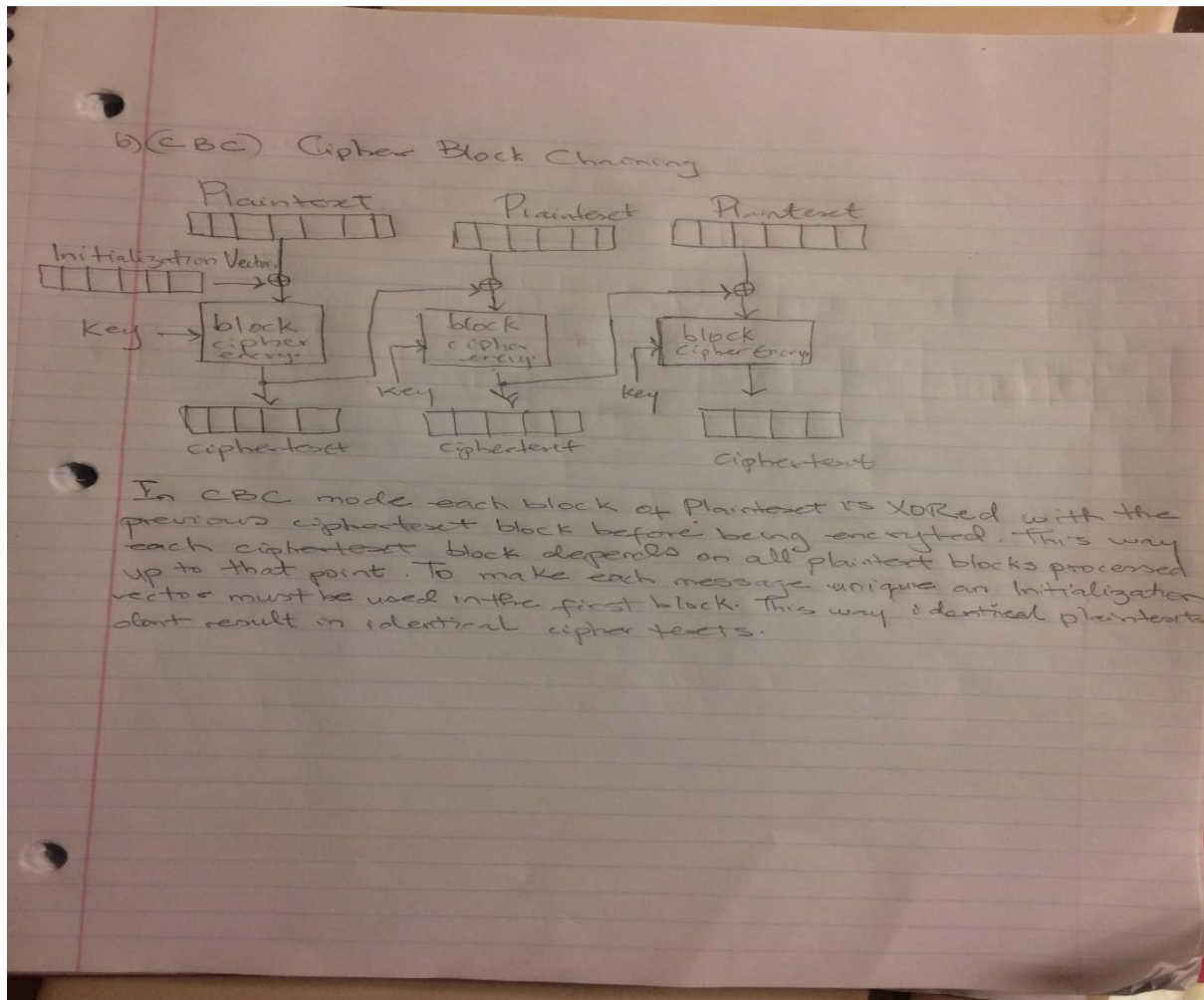
Corrupted CBC:

cipher_cfb.bin	
0	D3EEE656 E156C9F1 CEBECB78 0D647C0B ECFAA9D6 C7EBDAC2 2BC352AE 510F72BD 68EE6EBD 4F66D3C9 0524223F 63B4A325 F7F13B04 D752EEB8 8F2DA09A CEAEF859 574BC4ED 7796F8C1 0818DB14 4FBF1E0C 2482F353 A4F3CE3A A578251D 7C7BFDC6 7ACEB10B EB4119B8 67D4B3C3 9E82FD8A A2F5805C 029402D0 50EDC995 B37F457B 13BBCA4A C453C58D A5BF69C9 D0AAD18D C7346142 76245E86 65E9F175 B54109D1 7999D531 660D970A E712B9AC 9D25C0
24	2BC352AE 510F72BD 68EE6EBD 4F66D3C9 0524223F 63B4A325 F7F13B04 D752EEB8 8F2DA09A CEAEF859 574BC4ED 7796F8C1 0818DB14 4FBF1E0C 2482F353 A4F3CE3A A578251D 7C7BFDC6 7ACEB10B EB4119B8 67D4B3C3 9E82FD8A A2F5805C 029402D0 50EDC995 B37F457B 13BBCA4A C453C58D A5BF69C9 D0AAD18D C7346142 76245E86 65E9F175 B54109D1 7999D531 660D970A E712B9AC 9D25C0
48	F7F13B04 D752EEB8 8F2DA09A CEAEF859 574BC4ED 7796F8C1 0818DB14 4FBF1E0C 2482F353 A4F3CE3A A578251D 7C7BFDC6 7ACEB10B EB4119B8 67D4B3C3 9E82FD8A A2F5805C 029402D0 50EDC995 B37F457B 13BBCA4A C453C58D A5BF69C9 D0AAD18D C7346142 76245E86 65E9F175 B54109D1 7999D531 660D970A E712B9AC 9D25C0
72	0818DB14 4FBF1E0C 2482F353 A4F3CE3A A578251D 7C7BFDC6 7ACEB10B EB4119B8 67D4B3C3 9E82FD8A A2F5805C 029402D0 50EDC995 B37F457B 13BBCA4A C453C58D A5BF69C9 D0AAD18D C7346142 76245E86 65E9F175 B54109D1 7999D531 660D970A E712B9AC 9D25C0
96	7ACEB10B EB4119B8 67D4B3C3 9E82FD8A A2F5805C 029402D0 50EDC995 B37F457B 13BBCA4A C453C58D A5BF69C9 D0AAD18D C7346142 76245E86 65E9F175 B54109D1 7999D531 660D970A E712B9AC 9D25C0
120	50EDC995 B37F457B 13BBCA4A C453C58D A5BF69C9 D0AAD18D C7346142 76245E86 65E9F175 B54109D1 7999D531 660D970A E712B9AC 9D25C0
144	C7346142 76245E86 65E9F175 B54109D1 7999D531 660D970A E712B9AC 9D25C0
168	E712B9AC 9D25C0

Decrypted Corrupted CBC Text File:



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. Extending the text file to at least 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
77889aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ cat cipher_cfb.bin
d|?V?V??x
???????+?R?Q
      r?h?n?0f??$"?c??%??;?R?Q??h?YWK?w???0?
                        $$$S???:?x%|{??za
                        ?A?g?b??????\??P?b?E{??J?S0??i?b3?4aBv$^?e??u
?A      ?y??1f
????%?Ifunanyas-MacBook-Pro:Desktop openssl enc -aes-128-cfb -d -in cipher_cfb_corrupted.bin -out decrypted_corrupted
_cfb.txt \-K 0011223344556677889aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ cat decrypted_corrupted_cfb.txt
This is a text file used as ijpu?DT$?%Ss?/?_1. Extending the text file to at least 64 bytes in order to solve question
number 3 task 3.Is this up to 64bytes> I hope it is.
Ifunanyas-MacBook-Pro:Desktop Funa$
```

Uncorrupted CFB Cipher:

cipher_cfb.bin	
0	D3EEE656 E156C9F1 CEBECB78 0D647C0B ECFAA9D6 C7EBDAC2 "ÓÊV·V...ÔæÀx dI Ì'0÷«Î/-
24	2BC352AE 510B72BD 68EE6EBD 4F66D3C9 0524223F 63B4A325 +√RÆQ rΩhÓnΩOf"... \$"?c¥£%
48	F7F13B04 D752EEB8 8F2DA09A CEAEF859 574BC4ED 7796F8C1 ~0; ¢RÓΠè-†öÆ-ΥWKfİwñ~i
72	0818DB14 4FBF1E0C 2482F353 A4F3CE3A A578251D 7C7BFDC6 € 0ø \$ÇÛSŞÛÆ:•x% l{"Δ
96	7ACEB10B EB4119B8 67D4B3C3 9E82FD8A A2F5805C 029402D0 zÆ± ÎA Πg'≥√ûÇ"äφıÄ\ î -
120	50EDC995 B37F457B 13B8CA4A C453C58D A5BF69C9 D0AAD18D PÌ...î≥ E{ ° JfS≈ç•øi...™-ç
144	C7346142 76245E86 65E9F175 B54109D1 7999D531 660D970A «4aBv\$^ÛeÈòµA -yô'1f ó
168	E712B9AC 9D25C0 Á π"ù%¿

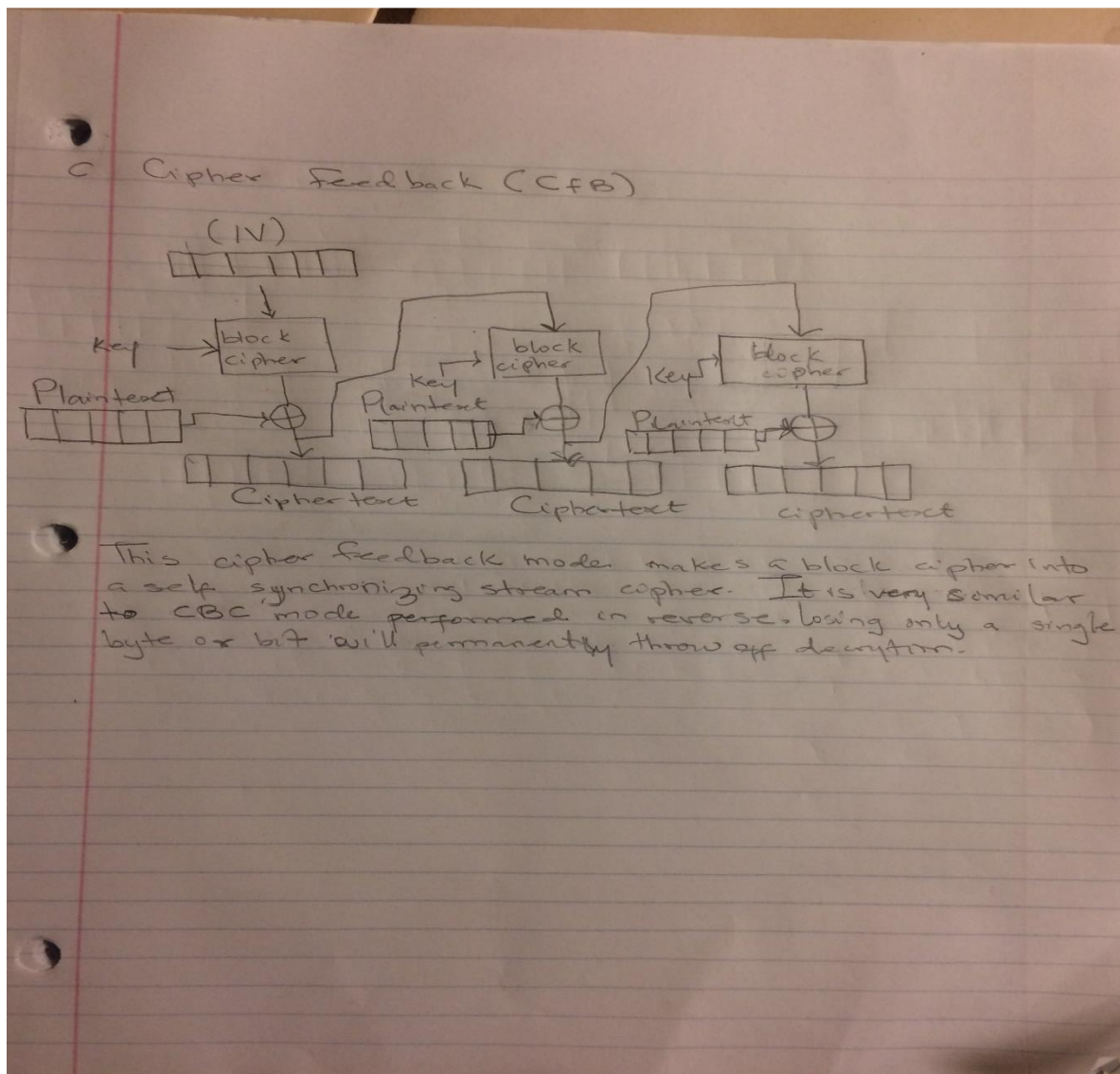
Corrupted CFB Cipher:

cipher_cfb.bin	
0	D3EEE656 E156C9F1 CEBECB78 0D647C0B ECFAA9D6 C7EBDAC2 "ÓÊV·V...ÔæÀx dI Ì'0÷«Î/-
24	2BC352AE 510F72BD 68EE6EBD 4F66D3C9 0524223F 63B4A325 +√RÆQ rΩhÓnΩOf"... \$"?c¥£%
48	F7F13B04 D752EEB8 8F2DA09A CEAEF859 574BC4ED 7796F8C1 ~0; ¢RÓΠè-†öÆ-ΥWKfİwñ~i
72	0818DB14 4FBF1E0C 2482F353 A4F3CE3A A578251D 7C7BFDC6 € 0ø \$ÇÛSŞÛÆ:•x% l{"Δ
96	7ACEB10B EB4119B8 67D4B3C3 9E82FD8A A2F5805C 029402D0 zÆ± ÎA Πg'≥√ûÇ"äφıÄ\ î -
120	50EDC995 B37F457B 13B8CA4A C453C58D A5BF69C9 D0AAD18D PÌ...î≥ E{ ° JfS≈ç•øi...™-ç
144	C7346142 76245E86 65E9F175 B54109D1 7999D531 660D970A «4aBv\$^ÛeÈòµA -yô'1f ó
168	E712B9AC 9D25C0 Á π"ù%¿

Decrypted Corrupted CFB Text File:

```
decrypted_corrupted_cfb.txt
0 | 54686973 20697320 61207465 78742066 696C6520 | This is a text file
20 | 75736564 20617320 696A7075 A0445424 B7255373 | used as inputDT$%$s
40 | 0E85142F D6007F0F 5F312E20 4578656E 64696E67 | 0 /÷ _1. Extending
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 61736820 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 00112233445566778899aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ cat cipher_ofb.bin
d|?V?V??x
  X8?'('??9u?:8?MZE'l??bQ??Q?? ?H_)?
}1]???G#?
?g???p=?LEj  ??oh???=k?E9?!?!?V'hu??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 decrypted_corrupted
_ofb.txt \-K 00112233445566778899aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ cat decrypted_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:

cipher_ofb.bin	
0 D3EEE656 E156C9F1 CEBECB78 0D647C0B 5838FC27	"ÓÊV·V...ÒËæÀx d X8,'
20 CE282797 D73975DC 3A0738EA A14D5A0E 17451C27	Æ('óó9u<: 8Í°MZ E '
40 6CD61FEA 6251C8CA 51149CBA 20EE8048 5F180F7D	l÷ ÍbQ» Q úf ÓÄH_ }
60 FF0A7D31 5D06B8A8 AC472391 0ACB6797 DCE1AA3F	´ }1] ¶@°G#ë Àgó<·™?
80 703D018C BE4C456A 099ECE12 0E6F15D1 9B7FADA3	p= áæLEj ûË o -õ ≠f
100 8D99113D 036BCBDC BCFA9145 39F821BF 21C556D2	çô = kÀ<°'ëE9~!ø!≈V“
120 906875AF D45842F8 260287FA 8EA97DAC 32AC7DE5	êhuØ‘XB~& á'é@}”2”}Â
140 7EF58199 A96FEC7E 7F267CC2 A0DE2DF2 DF2F43D7	~ıÂô@oİ~ & ~†fi-Úfi/Có
160 DDCBD816 60007B01 5A2788C0 439891	>Àÿ ` { Z'â¿Còë

Signed Int little (select some data) 0 out of 175 bytes

Corrupted OFB Cipher:

cipher_ofb.bin	
0 D3EEE656 E156C9F1 CEBECB78 0D647C0B 5838FC27	"ÓÊV·V...ÒËæÀx d X8,'
20 CE282797 D73975DC 3A0638EA A14D5A0E 17451C27	Æ('óó9u<: 8Í°MZ E '
40 6CD61FEA 6251C8CA 51149CBA 20EE8048 5F180F7D	l÷ ÍbQ» Q úf ÓÄH_ }
60 FF0A7D31 5D06B8A8 AC472391 0ACB6797 DCE1AA3F	´ }1] ¶@°G#ë Àgó<·™?
80 703D018C BE4C456A 099ECE12 0E6F15D1 9B7FADA3	p= áæLEj ûË o -õ ≠f
100 8D99113D 036BCBDC BCFA9145 39F821BF 21C556D2	çô = kÀ<°'ëE9~!ø!≈V“
120 906875AF D45842F8 260287FA 8EA97DAC 32AC7DE5	êhuØ‘XB~& á'é@}”2”}Â
140 7EF58199 A96FEC7E 7F267CC2 A0DE2DF2 DF2F43D7	~ıÂô@oİ~ & ~†fi-Úfi/Có
160 DDCBD816 60007B01 5A2788C0 439891	>Àÿ ` { Z'â¿Còë

Signed Int little (select some data) 30 out of 175 bytes

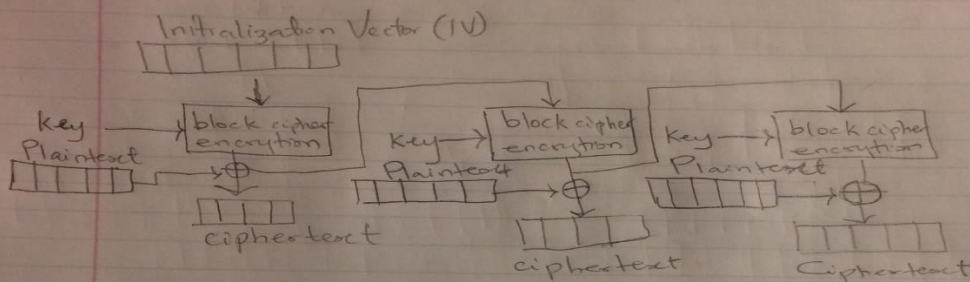
Decrypted Corrupted OFB:

decrypted_corrupted_ofb.txt

0	54686973	20697320	61207465	78742066	696C6520	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 64 bytes? I hope it is.
20	75736564	20617320	696F7075	7420666F	72204372	
40	7970746F	5F6C6162	5F312E20	4578656E	64696E67	
60	20746865	20746578	74206669	6C652074	6F206174	
80	6C656173	74203634	20627974	65732069	6E206F72	
100	64657220	746F2073	6F6C7665	20717565	7374696F	
120	6E206E75	6D626572	20332074	61736B20	332E4973	
140	20746869	73207570	20746F20	36346279	7465733E	
160	20492068	6F706520	69742069	732E0A		

Signed Int little (select some data) 0 out of 175 bytes

d) Output Feedback (OFB)

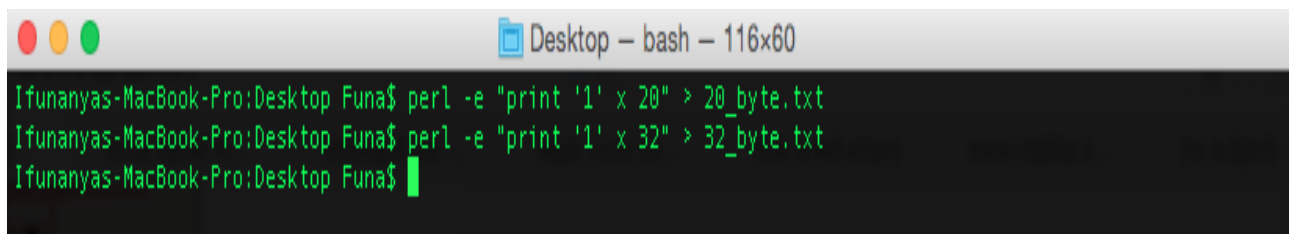


The output feedback mode makes a block cipher into a synchronous stream cipher. It generates key stream blocks which are then XOR-ed with the plaintext blocks to get the ciphertext. Flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location.

Task 4: Padding:

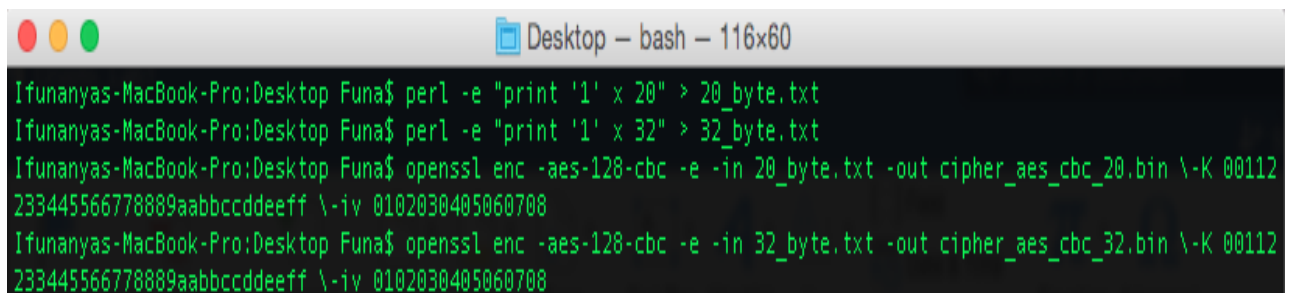
Lab description: For block ciphers, when the size of the plaintext is not the multiple of the block size, padding may be required. In this task, we will study the padding schemes. Please do the following exercises:

- 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.



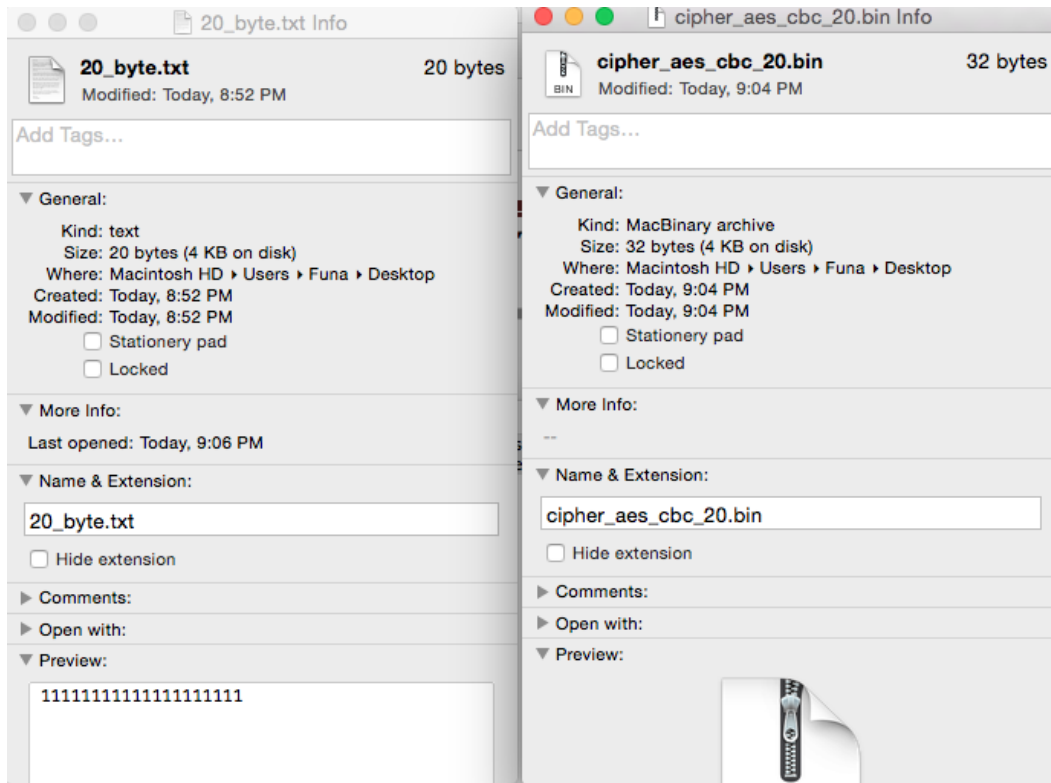
```
Desktop - bash - 116x60
Ifunanyas-MacBook-Pro:Desktop Funa$ perl -e "print '1' x 20" > 20_byte.txt
Ifunanyas-MacBook-Pro:Desktop Funa$ perl -e "print '1' x 32" > 32_byte.txt
Ifunanyas-MacBook-Pro:Desktop Funa$
```

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



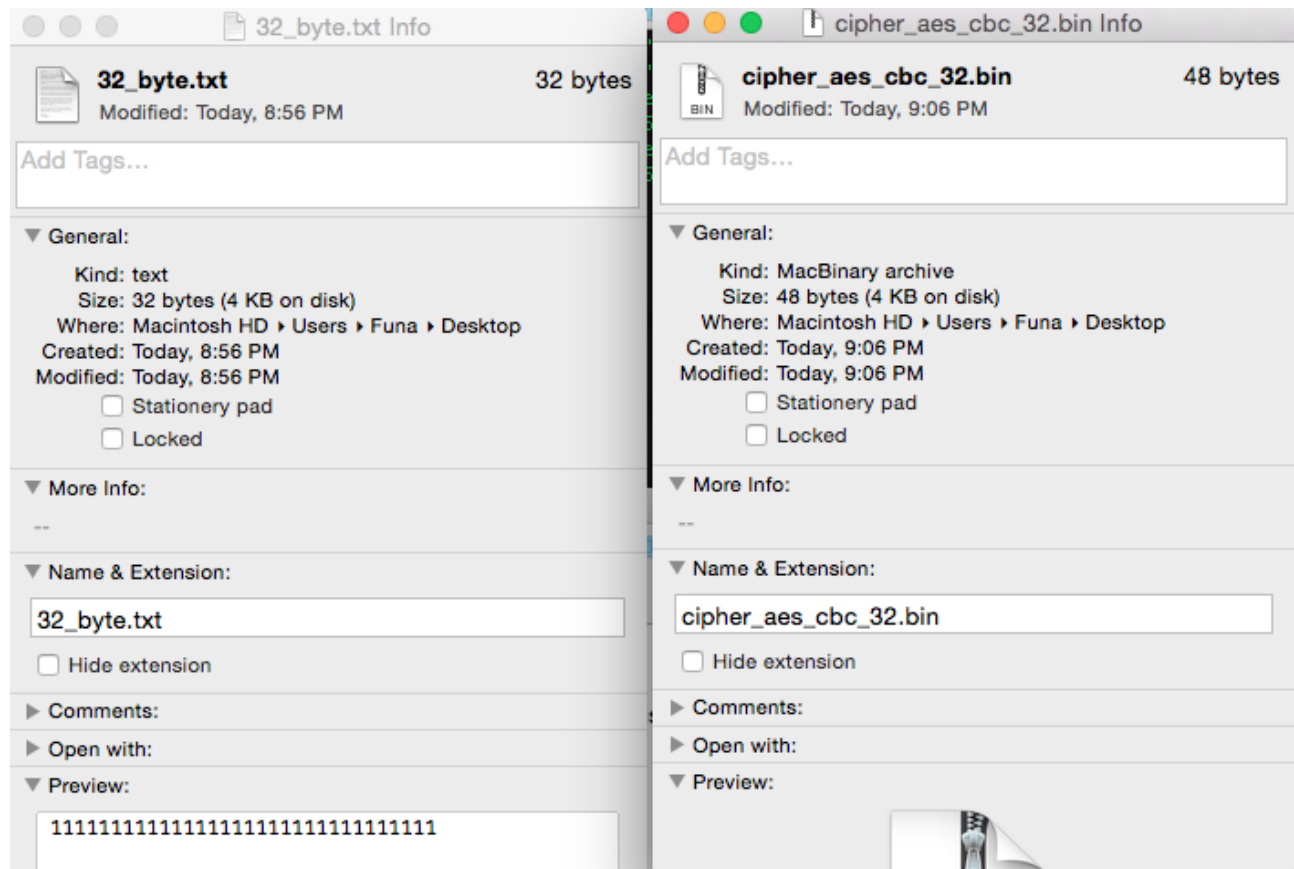
```
Desktop - bash - 116x60
Ifunanyas-MacBook-Pro:Desktop Funa$ perl -e "print '1' x 20" > 20_byte.txt
Ifunanyas-MacBook-Pro:Desktop Funa$ perl -e "print '1' x 32" > 32_byte.txt
Ifunanyas-MacBook-Pro:Desktop Funa$ openssl enc -aes-128-cbc -e -in 20_byte.txt -out cipher_aes_cbc_20.bin \-K 00112233445566778889aabbccddeeff \-iv 0102030405060708
Ifunanyas-MacBook-Pro:Desktop Funa$ openssl enc -aes-128-cbc -e -in 32_byte.txt -out cipher_aes_cbc_32.bin \-K 00112233445566778889aabbccddeeff \-iv 0102030405060708
```


- 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.

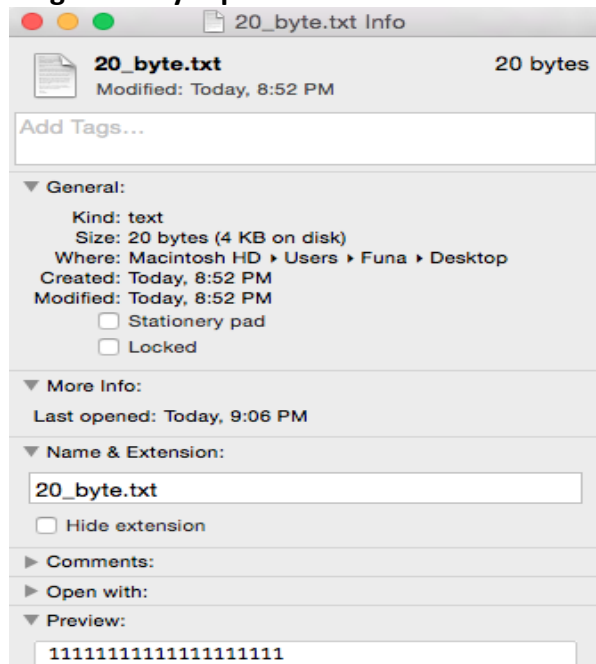
- Comparing the size of the 32byte plain text file to that of its cipher text, we notice below that some padding has been done:



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:

The image displays four macOS file information windows, each showing the details of a 20-byte AES cipher file. The files are named `cipher_aes_cfb_20.bin`, `cipher_aes_cbc_20.bin`, `cipher_aes_ofb_20.bin`, and `cipher_aes_ecb_20.bin`. Each window shows the file's size (20 bytes), its location (Macintosh HD > Users > Funa > Desktop), and its creation and modification times. The 'General' section also indicates the file is a 'MacBinary archive' and is not a 'Stationery pad' or 'Locked'. The 'Name & Extension' section shows the file name and the option to 'Hide extension'. The 'Comments', 'Open with', and 'Preview' sections are also visible at the bottom of each window.

File Name	Size	Kind	Size (on disk)	Where	Created	Modified	Stationery pad	Locked
<code>cipher_aes_cfb_20.bin</code>	20 bytes	MacBinary archive	4 KB	Macintosh HD > Users > Funa > Desktop	Today, 9:17 PM	Today, 9:17 PM	<input type="checkbox"/>	<input type="checkbox"/>
<code>cipher_aes_cbc_20.bin</code>	32 bytes	MacBinary archive	4 KB	Macintosh HD > Users > Funa > Desktop	Today, 9:04 PM	Today, 9:04 PM	<input type="checkbox"/>	<input type="checkbox"/>
<code>cipher_aes_ofb_20.bin</code>	20 bytes	MacBinary archive	4 KB	Macintosh HD > Users > Funa > Desktop	Today, 9:18 PM	Today, 9:18 PM	<input type="checkbox"/>	<input type="checkbox"/>
<code>cipher_aes_ecb_20.bin</code>	32 bytes	MacBinary archive	4 KB	Macintosh HD > Users > Funa > Desktop	Today, 9:20 PM	Today, 9:20 PM	<input type="checkbox"/>	<input type="checkbox"/>

Task 5: Pseudo Random Number Generation

5.A:

```
rahul@RJ: ~/project_209/task5
File Edit View Search Terminal Help

rahul@RJ:~/project_209/task5$ cat /proc/sys/kernel/random/entropy_avail
1844
rahul@RJ:~/project_209/task5$ cat /proc/sys/kernel/random/entropy_avail
1672
rahul@RJ:~/project_209/task5$ cat /proc/sys/kernel/random/entropy_avail
1531
rahul@RJ:~/project_209/task5$ cat /proc/sys/kernel/random/entropy_avail
1498
rahul@RJ:~/project_209/task5$
```

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:

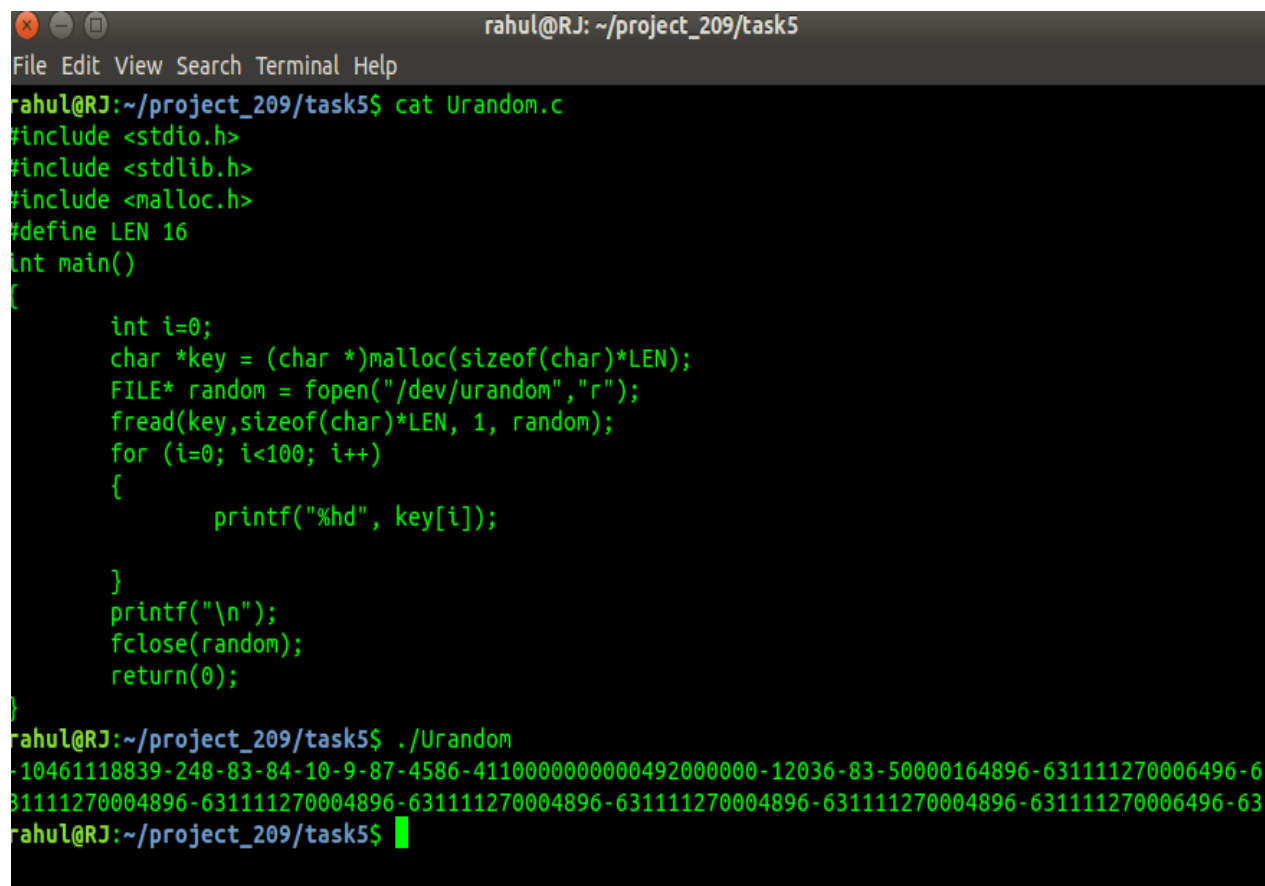
```
rahul@RJ: ~/project_209/task5
File Edit View Search Terminal Help

00000000 caf3 f0a6 8a4e abf4 1241 971d 4278 1589
00000010
rahul@RJ:~/project_209/task5$ head -c 16 /dev/random | hexdump
00000000 cdc3 3dda f792 f209 0599 fb27 8979 52a6
00000010
rahul@RJ:~/project_209/task5$ head -c 16 /dev/random | hexdump
00000000 16c6 80fa 6bf4 a37a 4f42 9c96 7e48 4479
00000010
rahul@RJ:~/project_209/task5$ head -c 16 /dev/random | hexdump
00000000 49ac 22b7 eb93 f415 7554 a4ad 26b7 7250
00000010
rahul@RJ:~/project_209/task5$ head -c 16 /dev/random | hexdump
00000000 cdb3 32c1 172f 0dcf 7642 0f4a 6315 7e56
00000010
rahul@RJ:~/project_209/task5$ head -c 16 /dev/random | hexdump
00000000 e6d0 cf90 6227 10b3 68c6 4338 3a3d 7db6
00000010
rahul@RJ:~/project_209/task5$ head -c 16 /dev/random | hexdump
00000000 763f b0aa 7052 ec2d 6639 f96b 213d 16e9
00000010
rahul@RJ:~/project_209/task5$ head -c 16 /dev/random | hexdump
00000000 5454 d8a5 f28a aae2 cfbc e34e 696f eae7
00000010
rahul@RJ:~/project_209/task5$ head -c 16 /dev/random | hexdump
00000000 1bda e0f2 6a10 f269 d23b aada 783b 75ac
00000010
rahul@RJ:~/project_209/task5$ head -c 16 /dev/random | hexdump
00000000 88a8 c77a 96e0 1052 16b3 002f 0f81 724e
00000010
rahul@RJ:~/project_209/task5$ head -c 16 /dev/random | hexdump
```

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:



```
rahul@RJ: ~/project_209/task5
File Edit View Search Terminal Help
rahul@RJ:~/project_209/task5$ cat Urandom.c
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#define LEN 16
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<100; i++)
    {
        printf("%hd", key[i]);
    }
    printf("\n");
    fclose(random);
    return(0);
}
rahul@RJ:~/project_209/task5$ ./Urandom
-10461118839-248-83-84-10-9-87-4586-4110000000000492000000-12036-83-50000164896-631111270006496-6
31111270004896-631111270004896-631111270004896-631111270004896-631111270004896-631111270006496-63
rahul@RJ:~/project_209/task5$
```

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-
31-85116-86-10637120-436541-80-844-11838105734663-7810910854-114-6078-57-101-821224-1121074781-744976-1099912524051108119-12372101455-19-50-
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-
127631106351-682189-5829-62884593-120102935447950-79871122-25121-44104-43124-92-11-514212064-46-1519253-103-7-610736-30-123-88-108-8140-40-1
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
```