**CSE 644 INTERNET SECURITY**

**Crypto Lab – Secret - Key Encryption**

**Arunbalaji Sivakumar(asivak01@syr.edu)**

**Contents:**

Task 1:………………………………………………………….. 2

Task 2:………………………………………………………….. 11

Task 3:………………………………………………………….. 14

Task 4:………………………………………………………….. 21

Task 5:………………………………………………………….. 29

**Programs:**

Lab\_edit.c: …..……...………………………………………….. 29

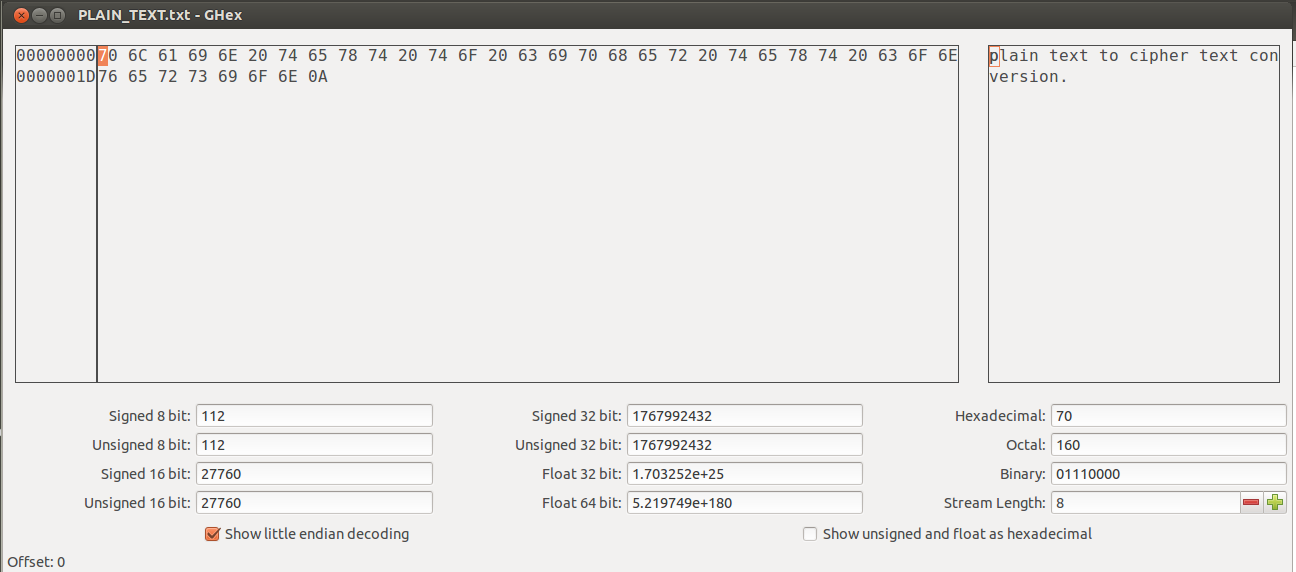
**Task 1: encryption using different ciphers and modes**

**AES:**

**Using AES-128-CBC:**

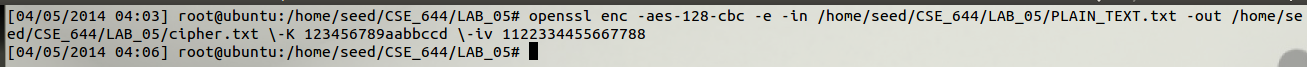
This mode requires a plain text, initialization vector and a key to begin encryption. Even if there is no text entered in the plain text a 128-bit cipher text is generated.

C:\Users\worker\Desktop\lab 5\task 1\1plaintext_cbc.png



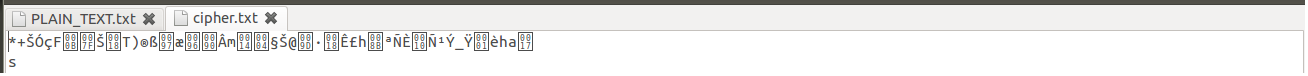
**Observation:**

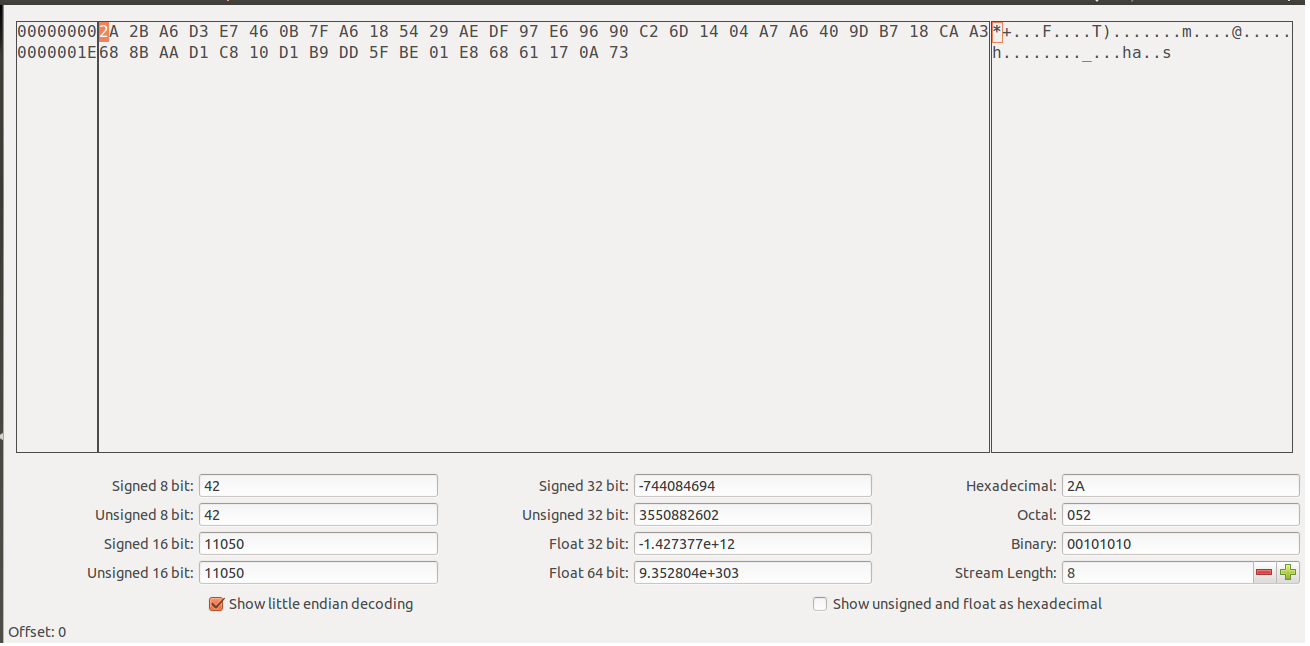
The plain text to be encrypted using AES-128-CBC and that text in hex form.



**Observation:**

The plain text is encrypted using AES-128 cipher using Cipher Block Chaining (CBC) mode.





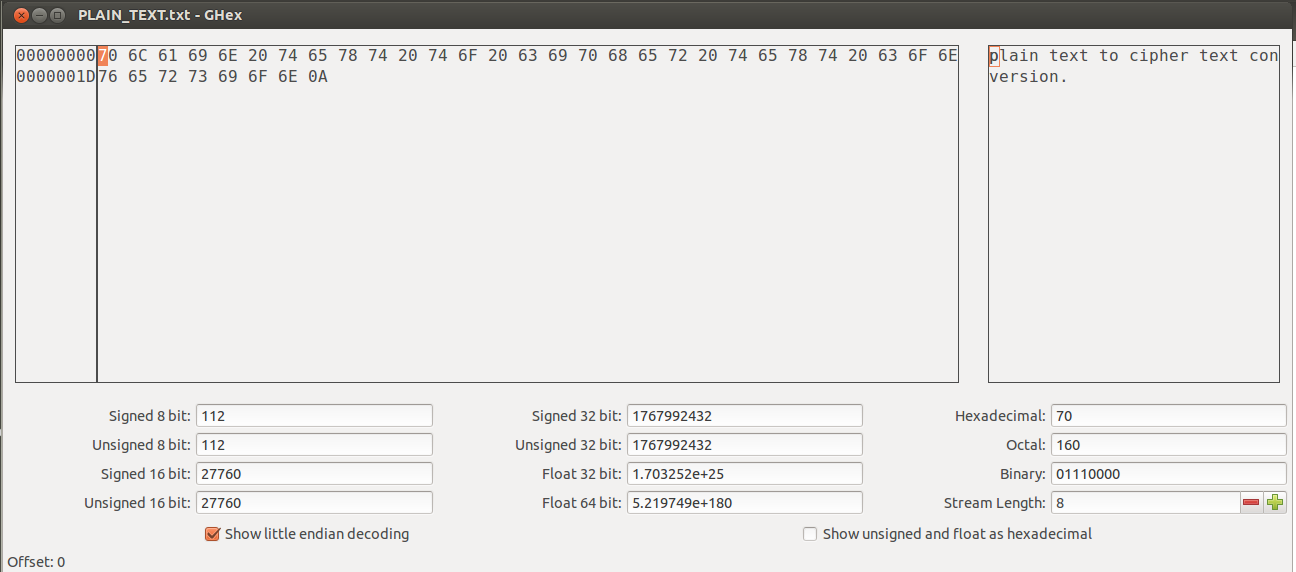
**Observation:**

The plain text contains 37 characters (including null - 15 + 15 + 7) so the cipher text contains 48 characters (16 + 16 + 16).

**Using AES-128-CFB:**

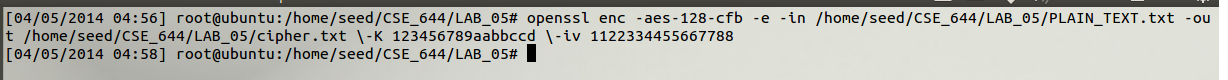
This mode requires an initialization vector and a key to begin encryption. Since CFB is a stream cipher, if there is no text entered in the plain text, cipher text will not be generated.

C:\Users\worker\Desktop\lab 5\task 1\6plaintext_cfb.png



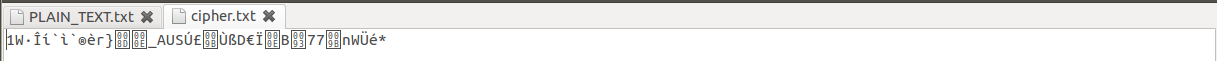
**Observation:**

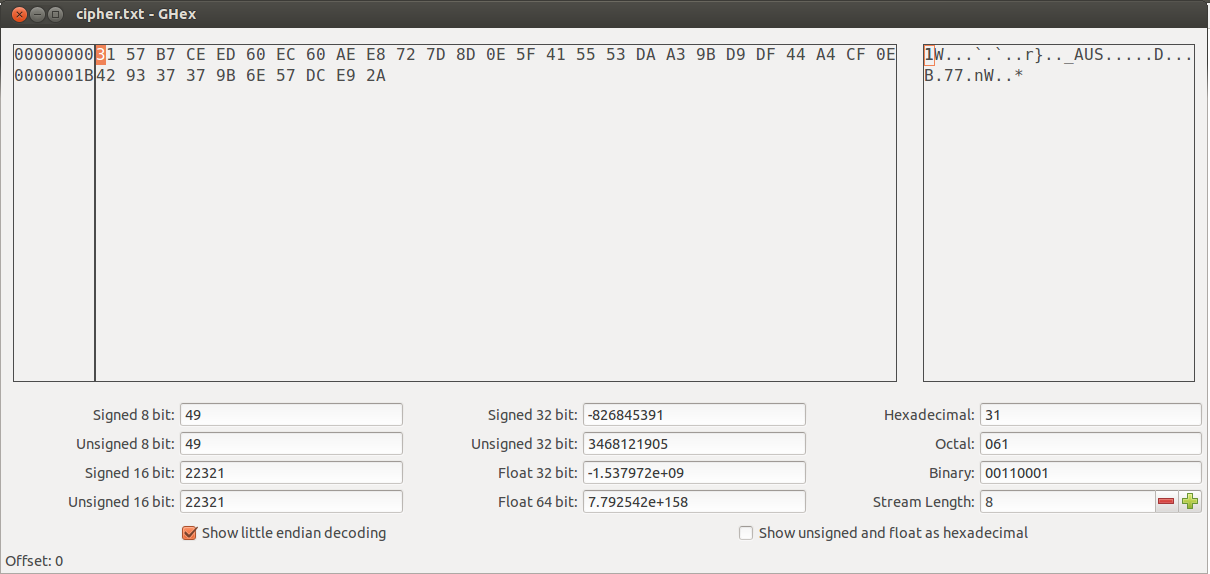
The plain text to be encrypted and that text in hex form.



**Observation:**

The plain text is encrypted using AES-128 cipher using Cipher Feedback (CFB) mode.





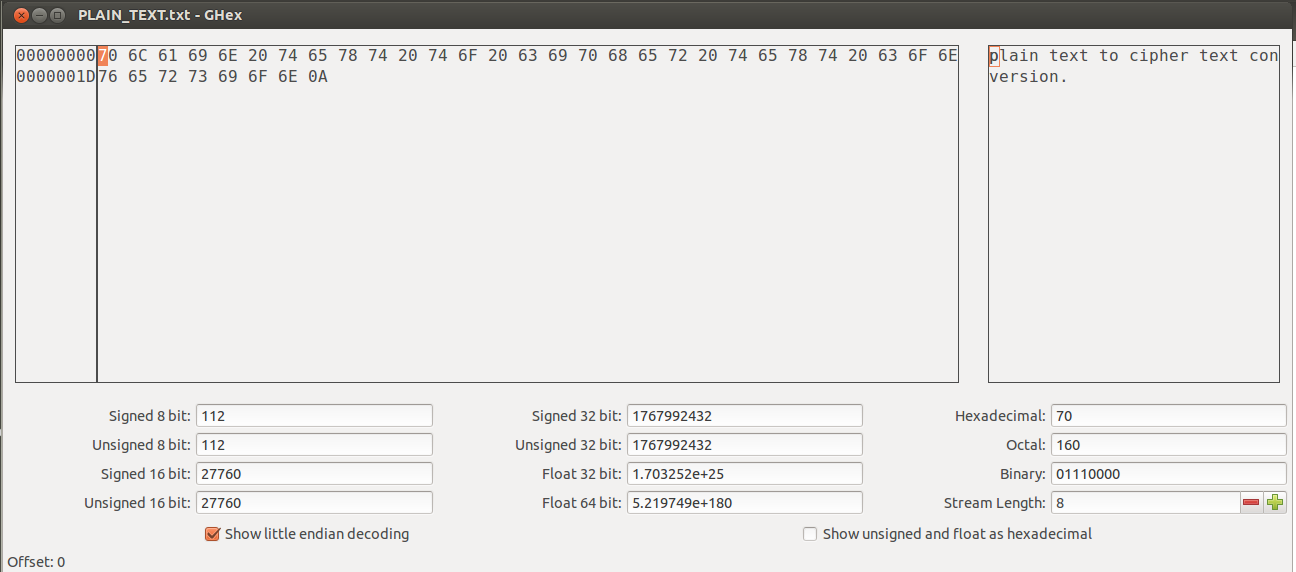
**Observation:**

The plain text contains 37 characters (including null) so the cipher text also contains 37 characters. This is because the CFB is a stream cipher.

**Using AES-128-OFB:**

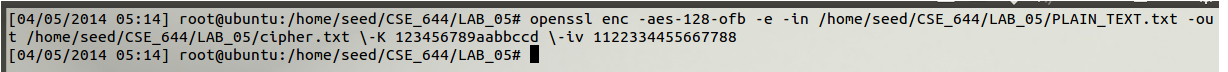
This mode requires an initialization vector and a key to begin encryption. If there is no text entered in the plain text, cipher text will not be generated. This cipher mode supports parallel encryption to form cipher text.

C:\Users\worker\Desktop\lab 5\task 1\11plaintext_ofb.png



**Observation:**

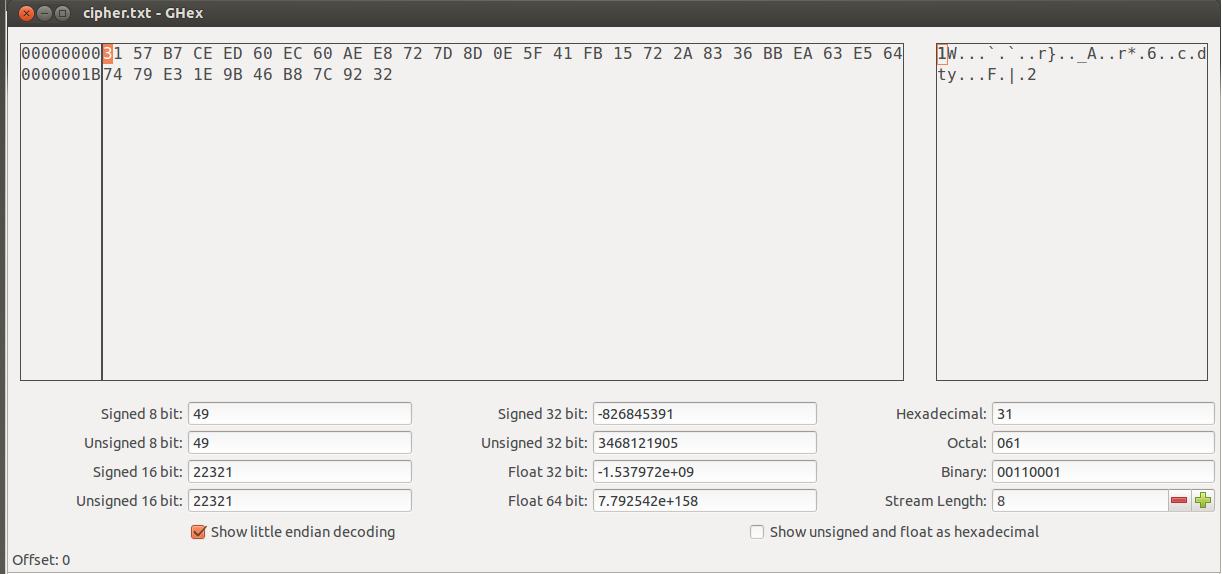
The plain text to be encrypted and that text in hex form.



**Observation:**

The plain text is encrypted using AES-128 cipher using Output Feedback (CFB) mode.

C:\Users\worker\Desktop\lab 5\task 1\14cipher text.png



**Observation:**

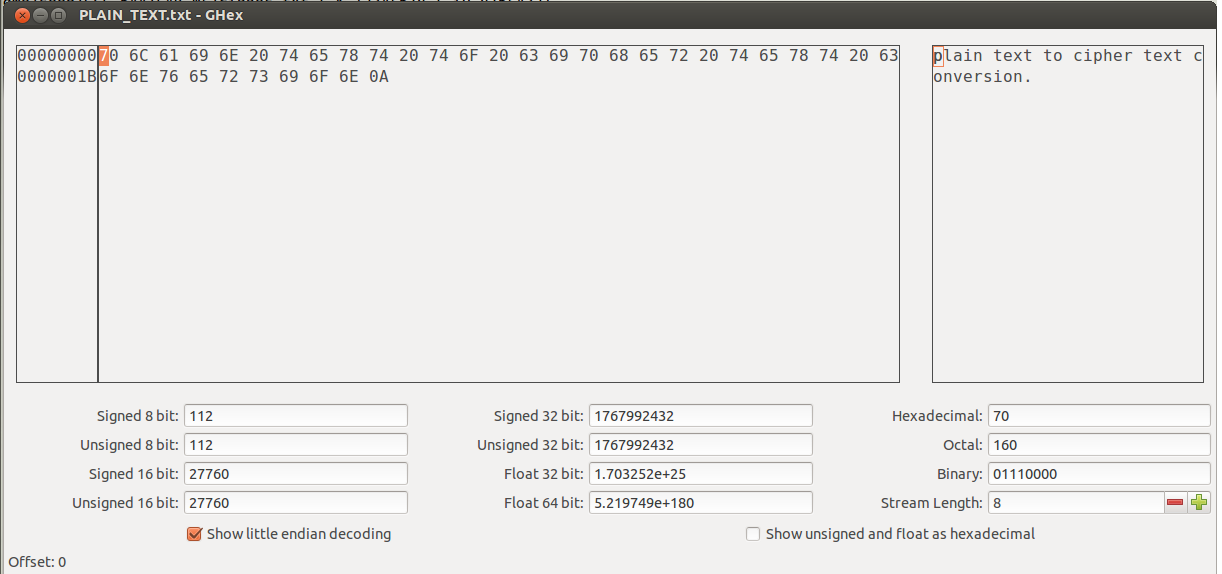
The plain text contains 37 characters (including null) so the cipher text also contains 37 characters. This is because the OFB is a stream cipher and the plain text is XORed with cipher block to get the cipher text.

**DES:**

**DES-CBC:**

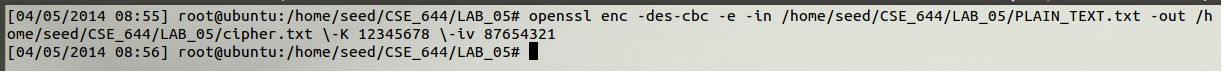
This mode requires the plain text and the 64-bit key. Even if there is no text entered in the plain text a 64-bit cipher text is generated.

C:\Users\worker\Desktop\lab 5\task 1\des\1plaintext_cbc.png



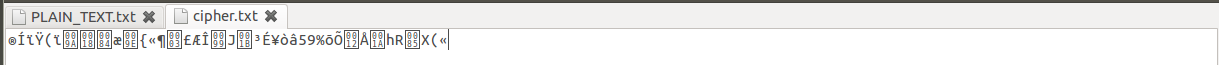
**Observation:**

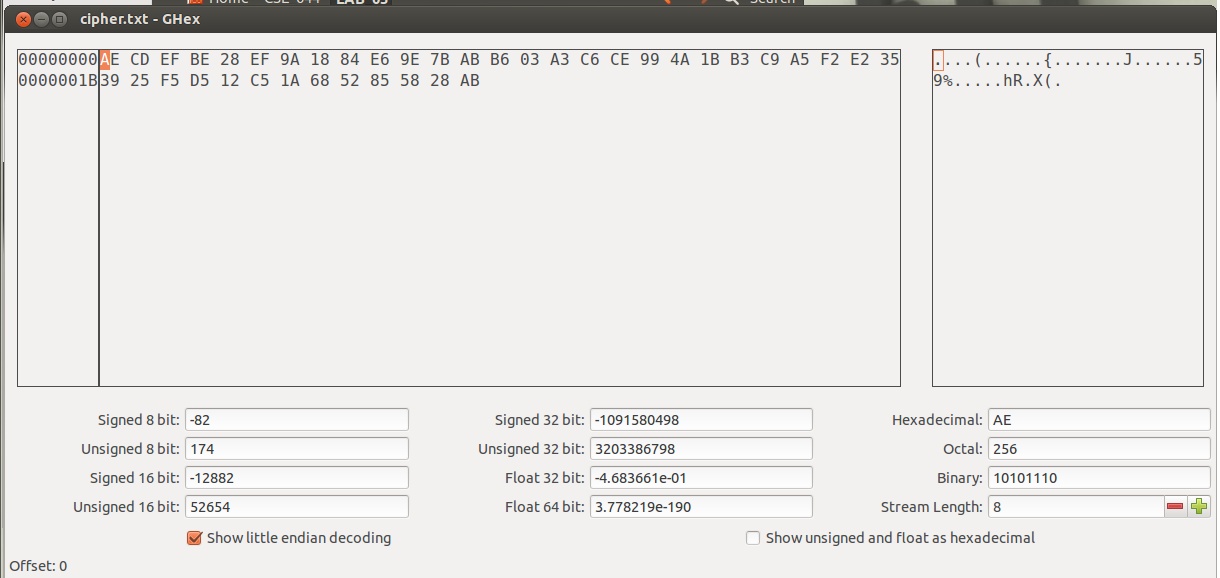
The plain text and its hex form using ghex tool.



**Observation:**

DES uses 64 bit key to perform encryption in which 56 bits are used as key and the remaining 8 bits are used as parity bits.





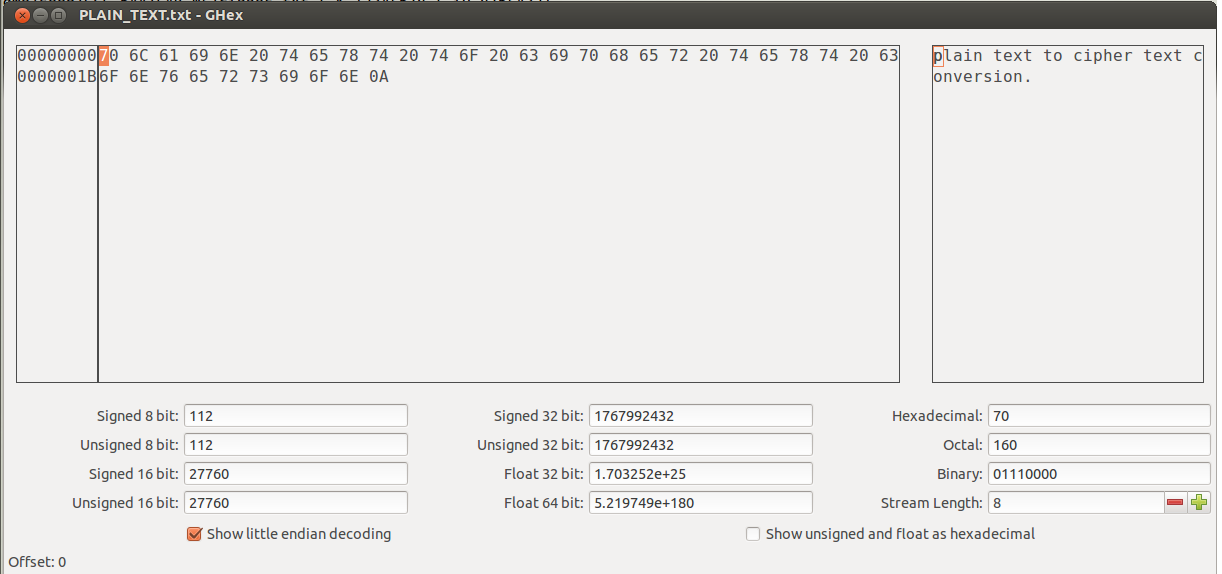
**Observation:**

The plain text is converted into cipher text using the DES encryption with 64-bit key.

**DES-CFB:**

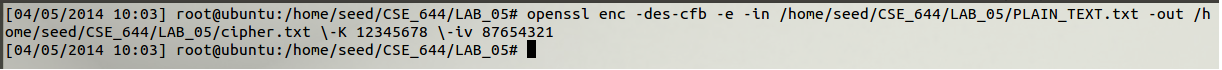
This mode requires a plain text and a key to begin encryption. Since CFB is a stream cipher, if there is no text entered in the plain text, cipher text will not be generated.

C:\Users\worker\Desktop\lab 5\task 1\des\6plaintext_cfb.png



**Observation:**

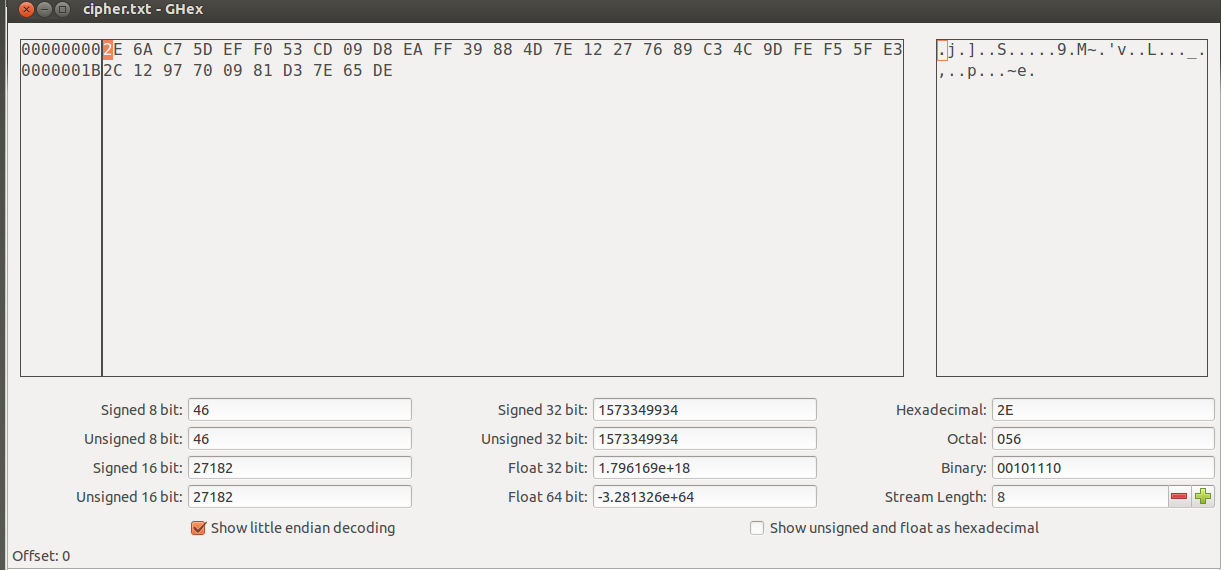
The plain text and its hex form using ghex tool.



**Observation:**

DES uses 64 bit key to perform encryption in which 56 bits are used as key and the remaining 8 bits are used as parity bits.

C:\Users\worker\Desktop\lab 5\task 1\des\9ciphertext.png



**Observation:**

The plain text is converted into cipher text using the DES encryption with 64-bit key.

**DES-OFB:**

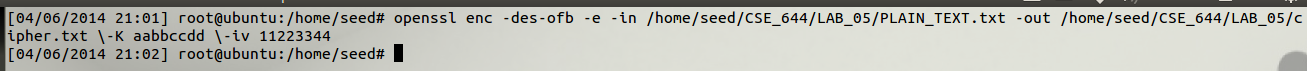
This mode requires an initialization vector and a 64-bit key to begin encryption. If there is no text entered in the plain text, cipher text will not be generated. This cipher mode supports parallel encryption to form cipher text.

C:\Users\worker\Desktop\lab 5\task 1\des\11plaintext_ofb.png



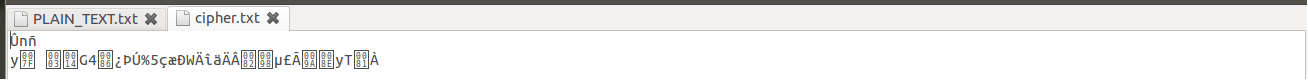
**Observation:**

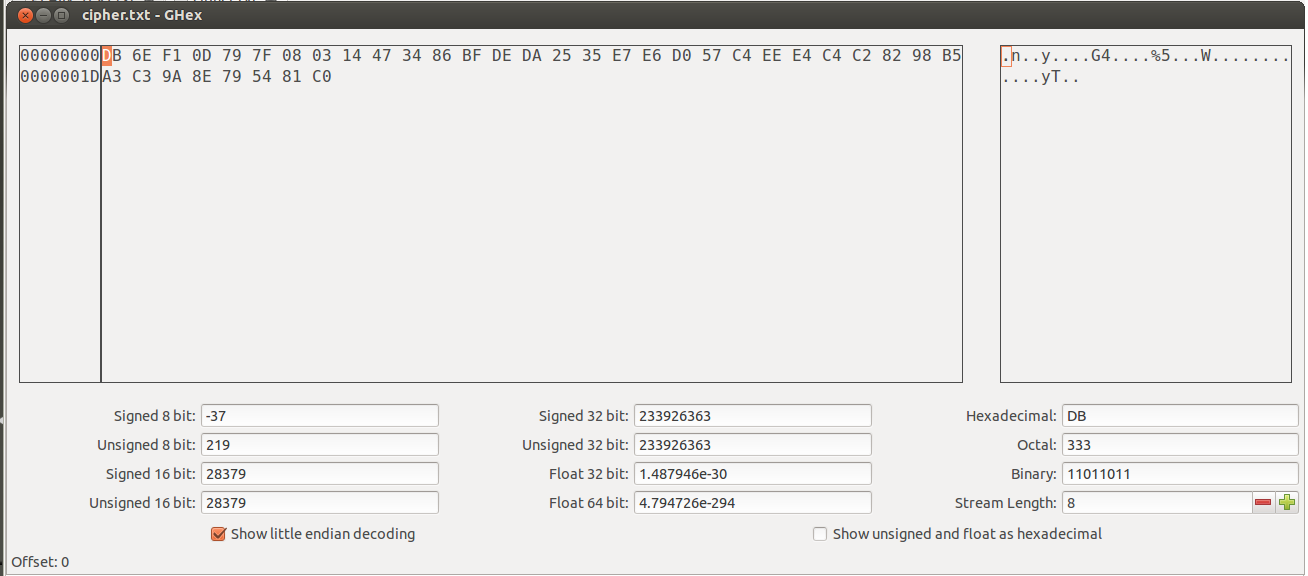
The plain text and its hex form using ghex tool.



**Observation:**

DES uses 64 bit key to perform encryption in which 56 bits are used as key and the remaining 8 bits are used as parity bits.





**Observation:**

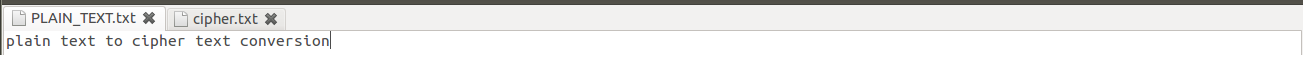
The plain text is converted into cipher text using the DES encryption with 64-bit key.

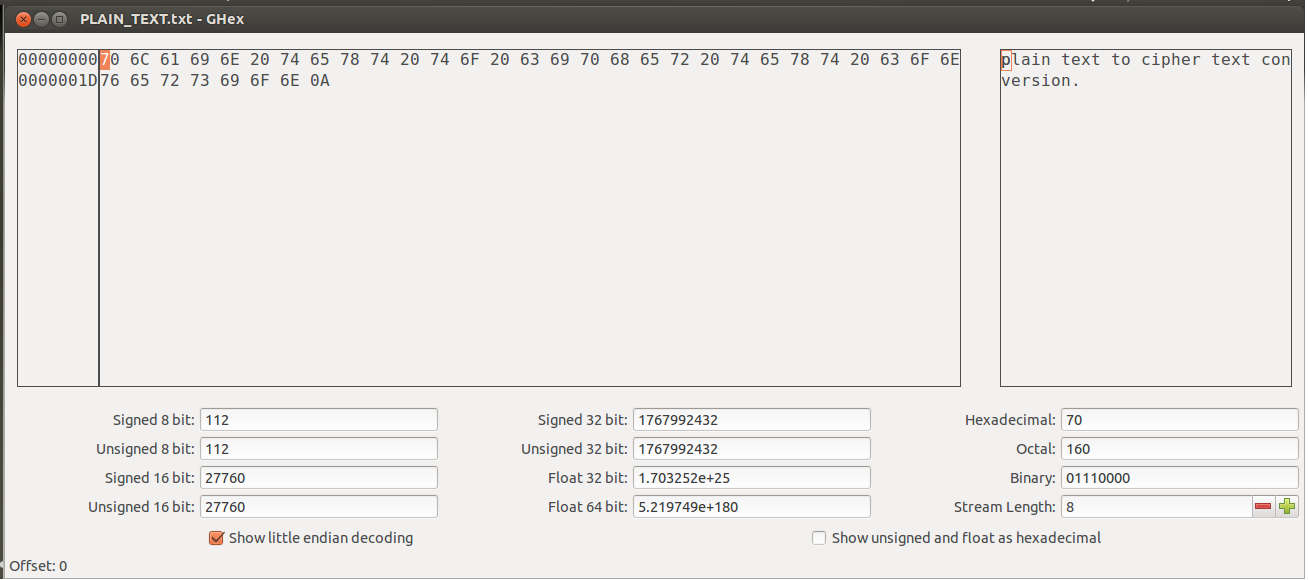
**BF:**

Blowfish is symmetric-key block cipher with the input block size as 64-bits and a variable key size from 32-bits to 48-bits.

**BF-CBC:**

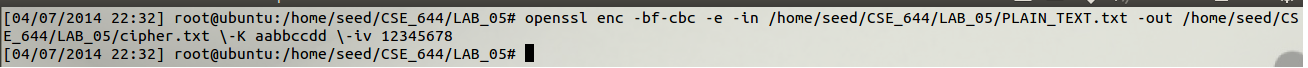
This mode requires the plain text and the 64-bit key. The key size can vary.





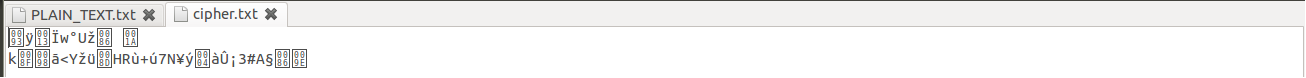
**Observation:**

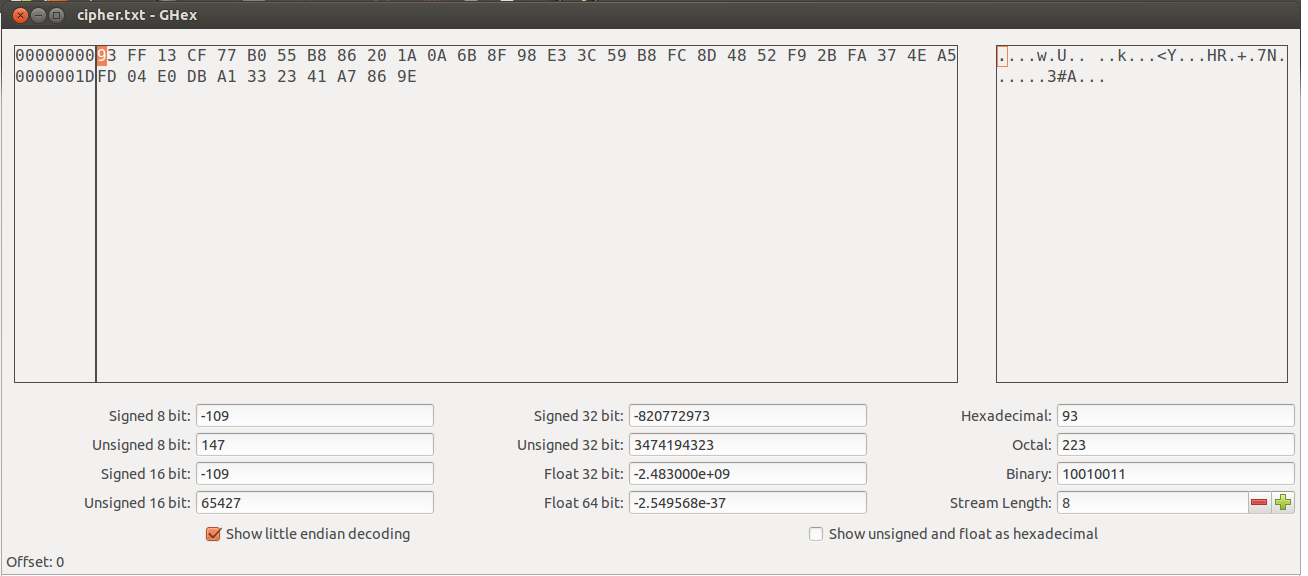
The plain text and its hex form using ghex tool.



**Observation:**

Here BF uses 64 bit key and IV to perform encryption on 64-bit block using CBC mode.



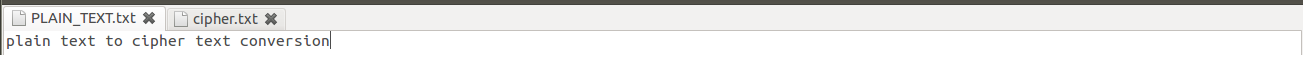


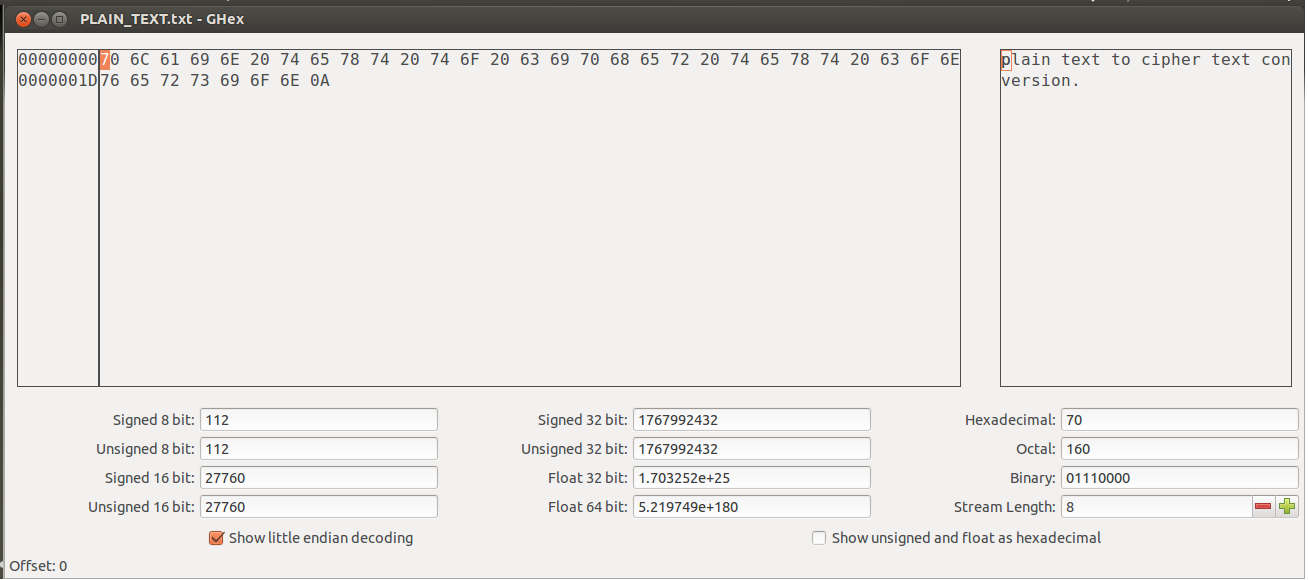
**Observation:**

The plain text is converted into cipher text using the BF-CBC encryption with 64-bit key.

**BF-CFB:**

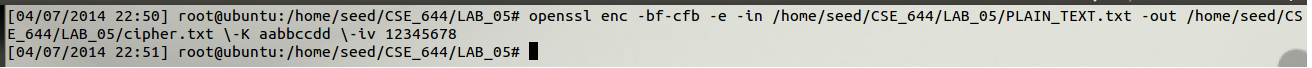
This mode requires a plain text and a key to begin encryption. Since CFB is a stream cipher, if there is no text entered in the plain text, cipher text will not be generated.





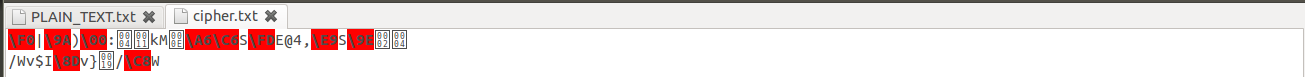
**Observation:**

The plain text and its hex form using ghex tool.



**Observation:**

Here BF uses 64 bit key and IV to perform encryption on 64-bit block using CFB mode.

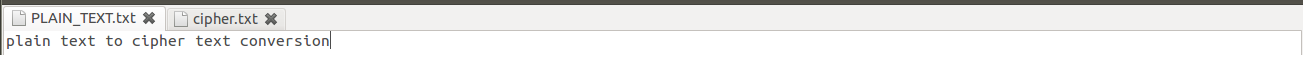


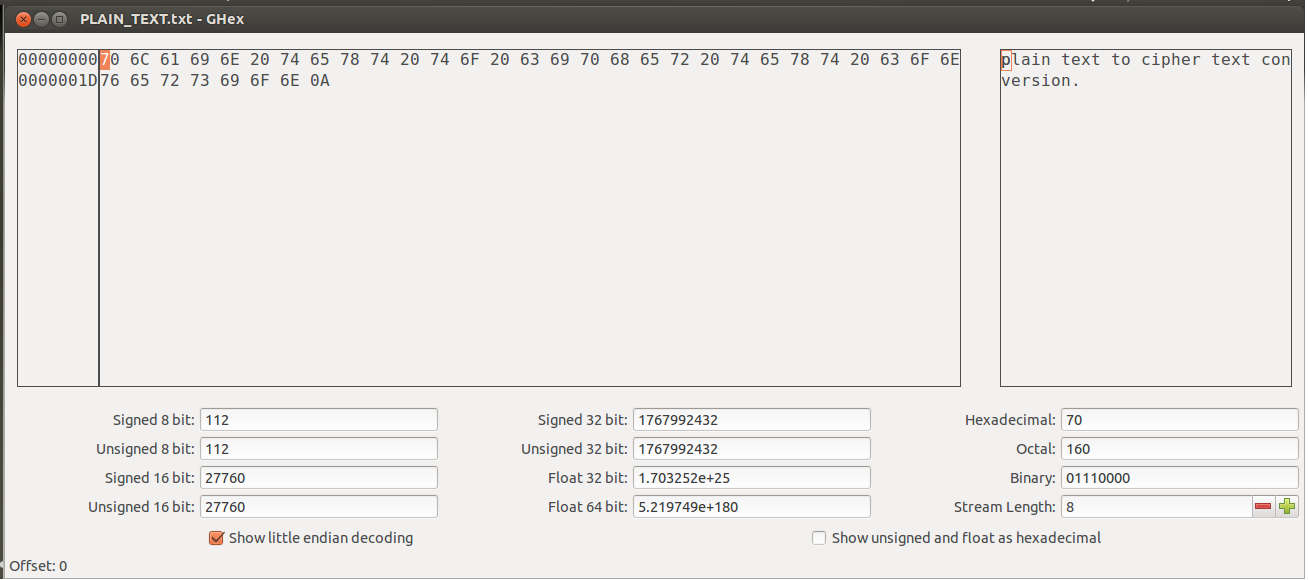
**Observation:**

The plain text is converted into cipher text using the BF-CFB encryption with 64-bit key.

**BF-OFB:**

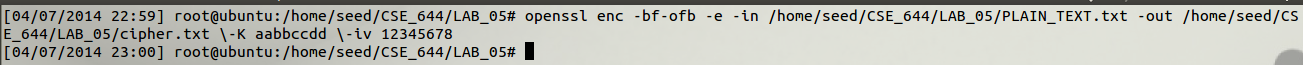
This mode requires an initialization vector and a 64-bit key to begin encryption. If there is no text entered in the plain text, cipher text will not be generated. This cipher mode supports parallel encryption to form cipher text.





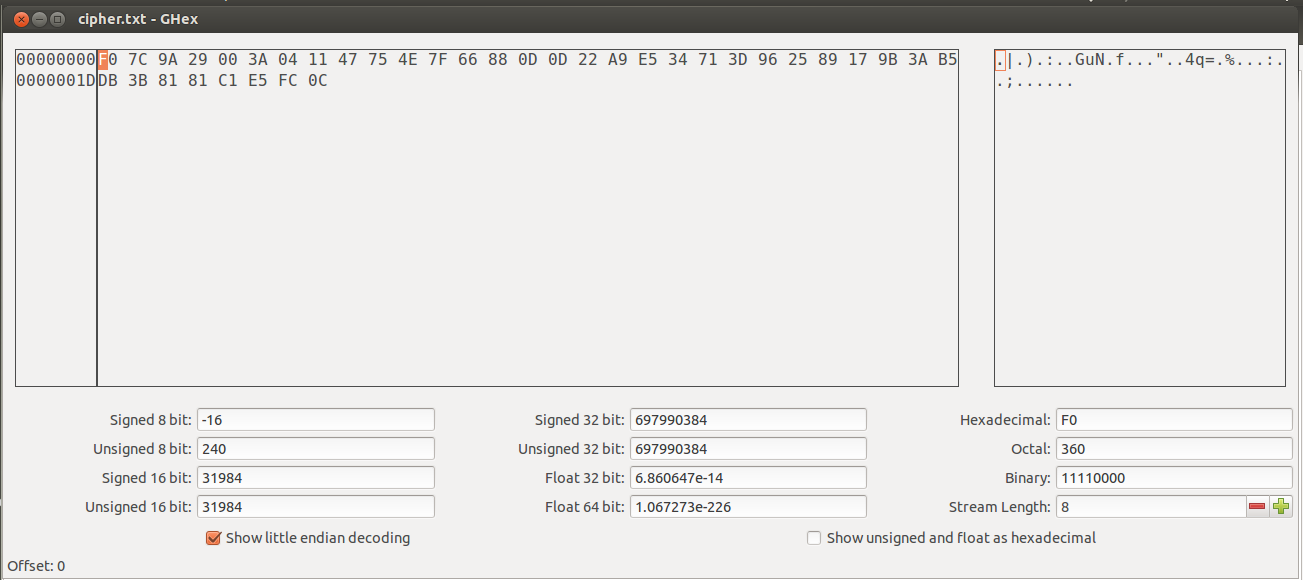
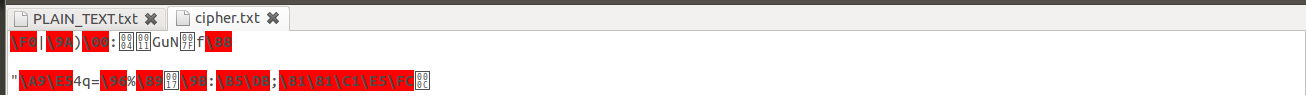
**Observation:**

The plain text and its hex form using ghex tool.



**Observation:**

Here BF uses 64 bit key and IV to perform encryption on 64-bit block using OFB mode.



**Observation:**

The plain text is converted into cipher text using the BF-OFB encryption with 64-bit key.

**Task 2: encryption mode – ECB vs. CBC**

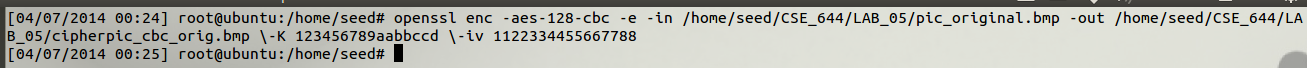
**Cipher Block chaining (CBC):**

This is a block cipher encryption in which each block of the plain text is XORed with the previously generated cipher text and then it is encrypted. So each cipher clock depends upon all plain texts that were processed.



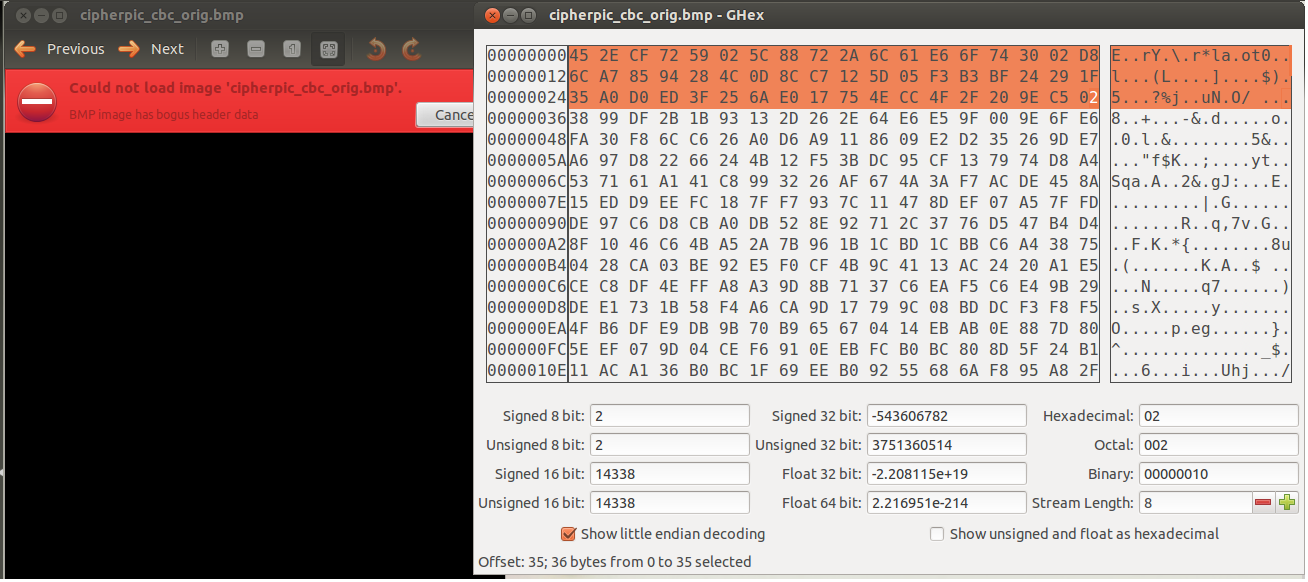
**Observation:**

The GHex tool shows the header (1st 3 rows in 2nd box) and payload part of the original bmp file.



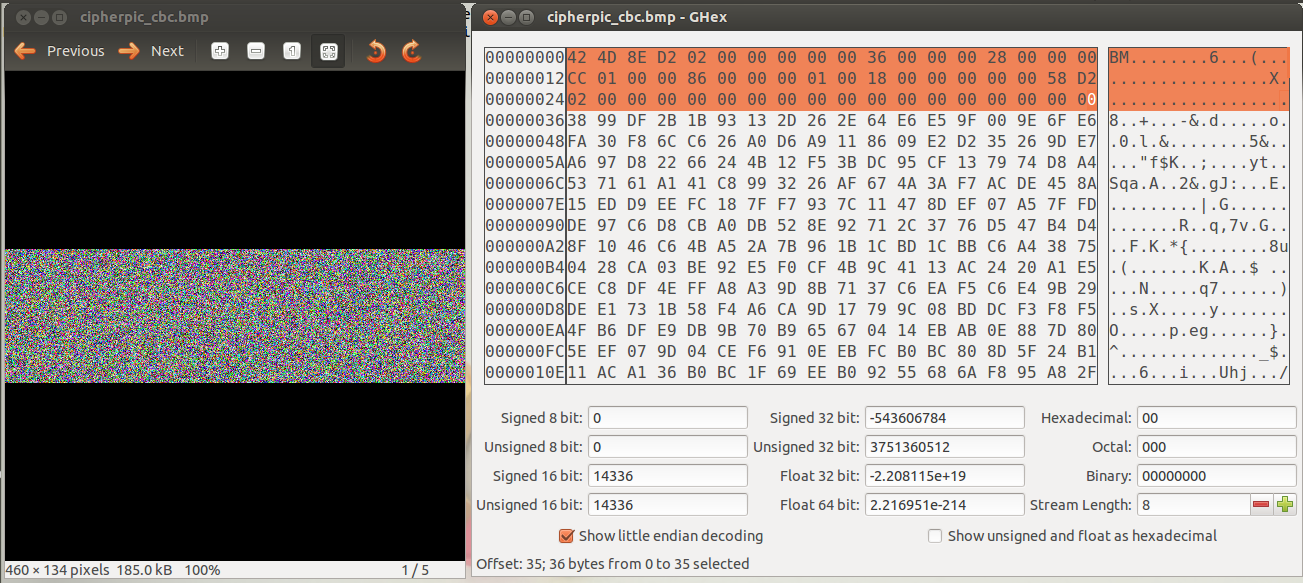
**Observation:**

The openssl uses AES-CBC mode to encrypt the original bmp file.



**Observation:**

As the bmp file is encrypted it cannot be seen through image viewer. The GHex tool shows the modified header value.



**Observation:**

The first 54 bits of the encrypted bmp file is replaced with the first 54 bits of the original bmp file. By doing this the encrypted image can be seen using image viewer. However any information about the original image cannot be observed in this encrypted bmp file. This is because CBC mode generates different cipher text for repeating plain text.

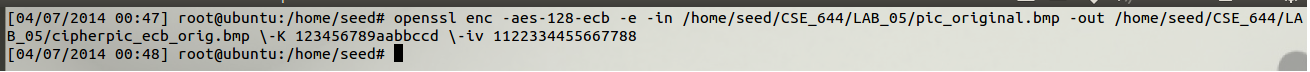
**Electronic Code Book (ECB):**

This is the simplest block cipher encryption in which a block of plain text is converted into cipher text by encrypting it with a key.



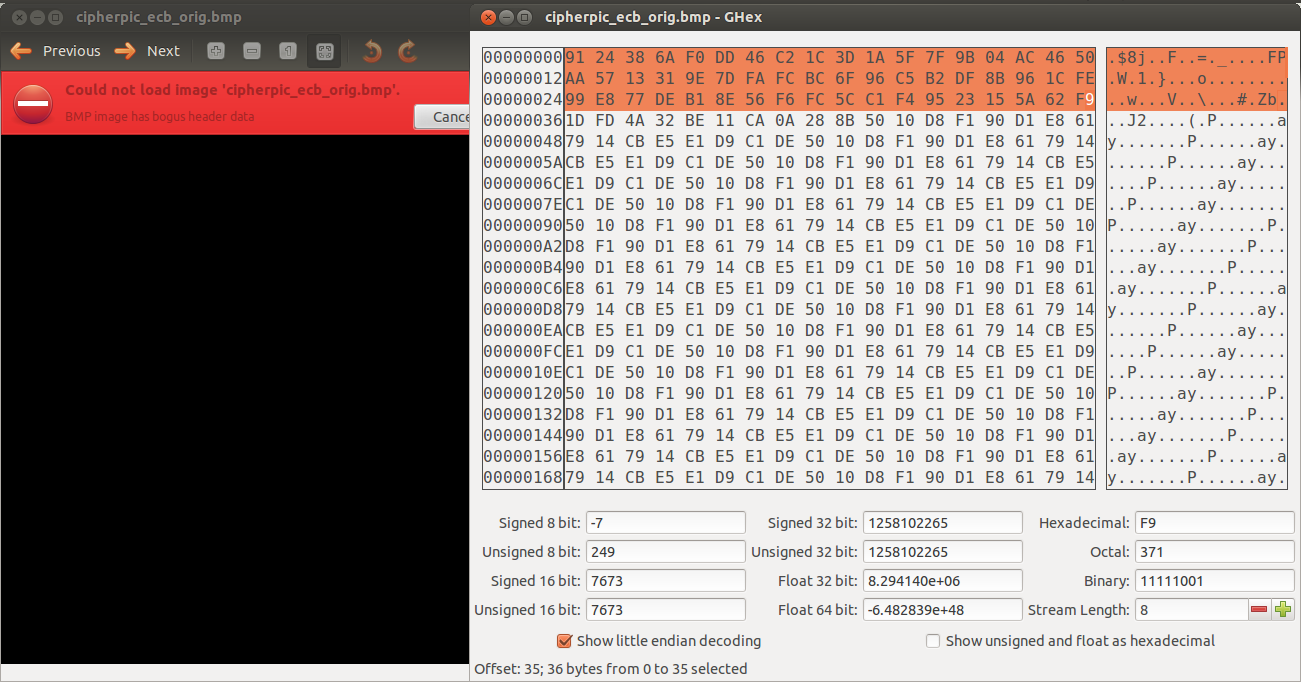
**Observation:**

The GHex tool shows the header (1st 3 rows in 2nd box) and payload part of the original bmp file.



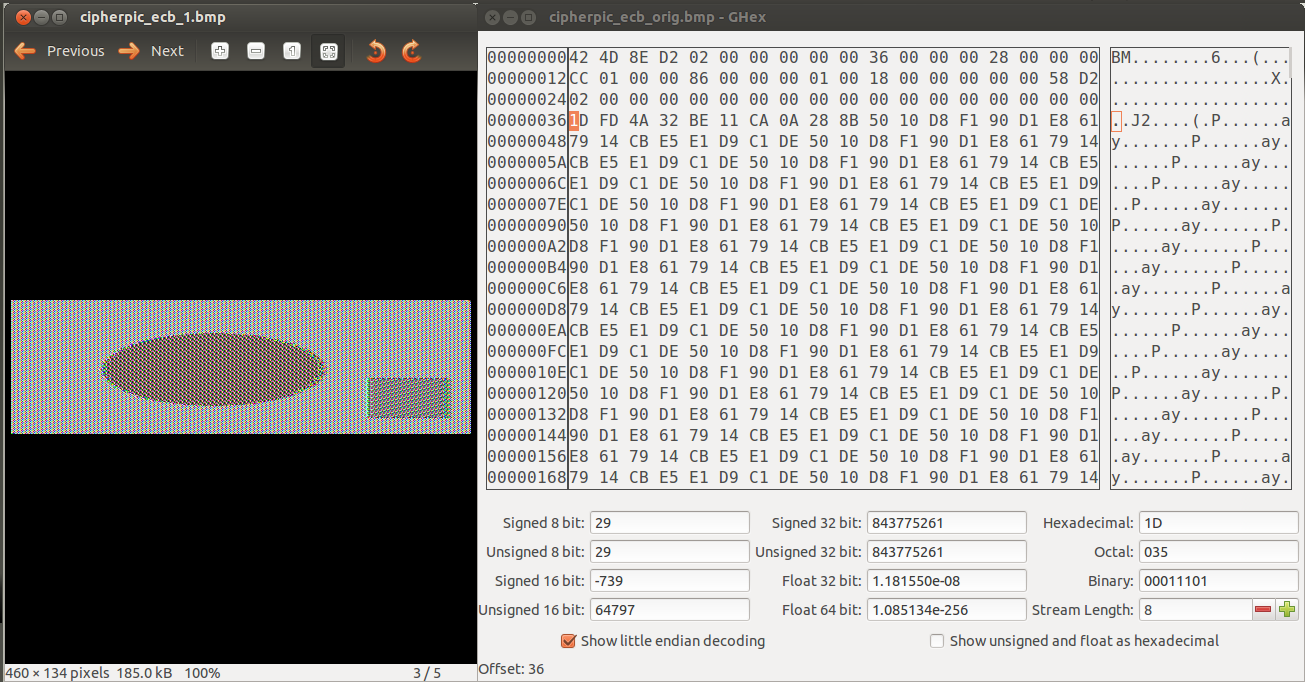
**Observation:**

The openssl uses aes-ecb mode to encrypt the original bmp file.



**Observation:**

As the bmp file is encrypted it cannot be seen through image viewer. The GHex tool shows the modified header value.



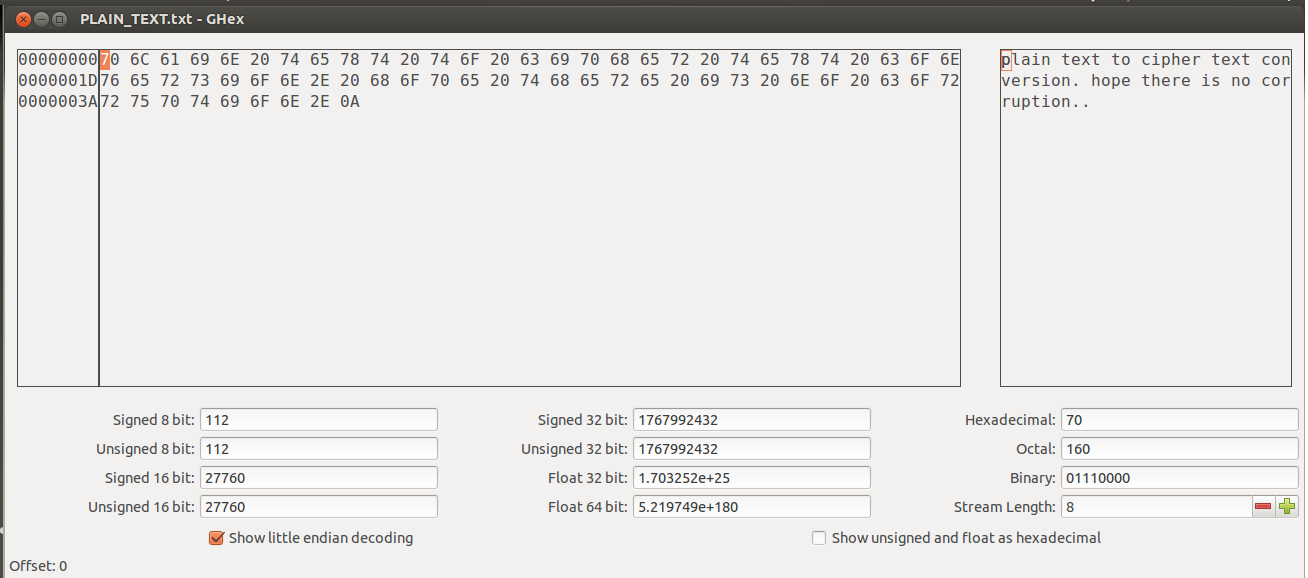
**Observation:**

The first 54 bits of the encrypted bmp file is replaced with the first 54 bits of the original bmp file. By doing this the encrypted image can be seen using image viewer. Information about the original image can be observed in this encrypted bmp file. This is because ECB mode generates same cipher text for repeating plain text. Even though the cipher image is different from the original image, most of the original image’s information can be obtained from the cipher image.

**Task 3: encryption mode – corrupted Cipher Text**

Data loss is a major concern while sending out the encrypted file. If the corrupted part in the file is not narrowed down then the intended information cannot be retrieved even with the correct key.

C:\Users\worker\Desktop\lab 5\task 3\cbc\1plain text.png



**Observation:**

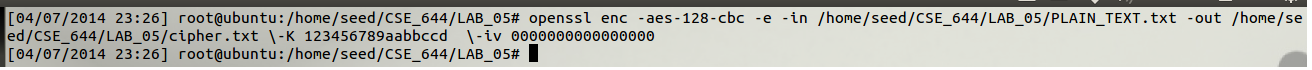
The plain text and its hex form using GHex tool. The plain text contains 67 bytes (including line feed).

**Detecting corruption in CBC mode:**

1. How much information can you recover by decrypting the corrupted file?
2. Before conducting the task:

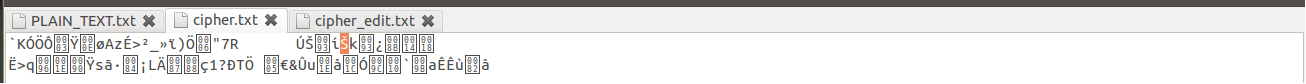
Since only a single bit of the cipher text is corrupted the following two 16-byte plain text blocks will be corrupted. The third plain text block will not be corrupted since its cipher text block is not corrupted.

1. After conducting the task:



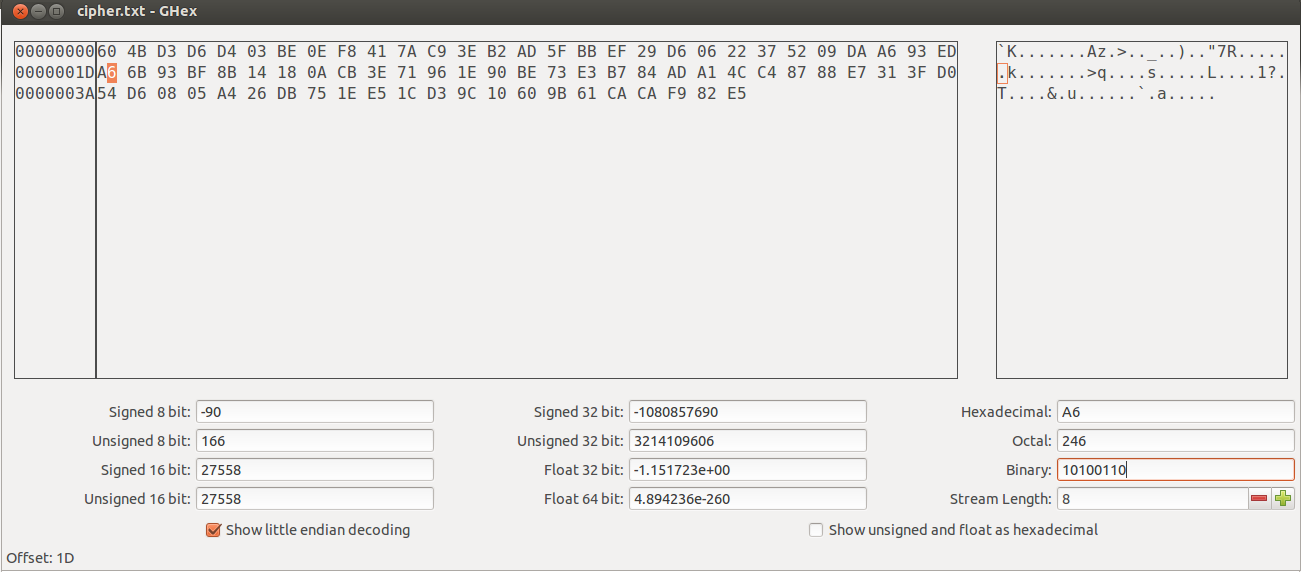
**Observation:**

The plain text is encrypted using AES-128-CBC mode and with 128-bit key and IV.



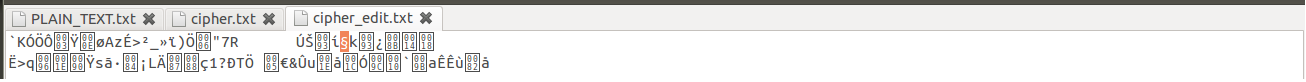
**Observation:**

The original cipher text formed during encryption. The highlighted character is where the modification is going to take place.



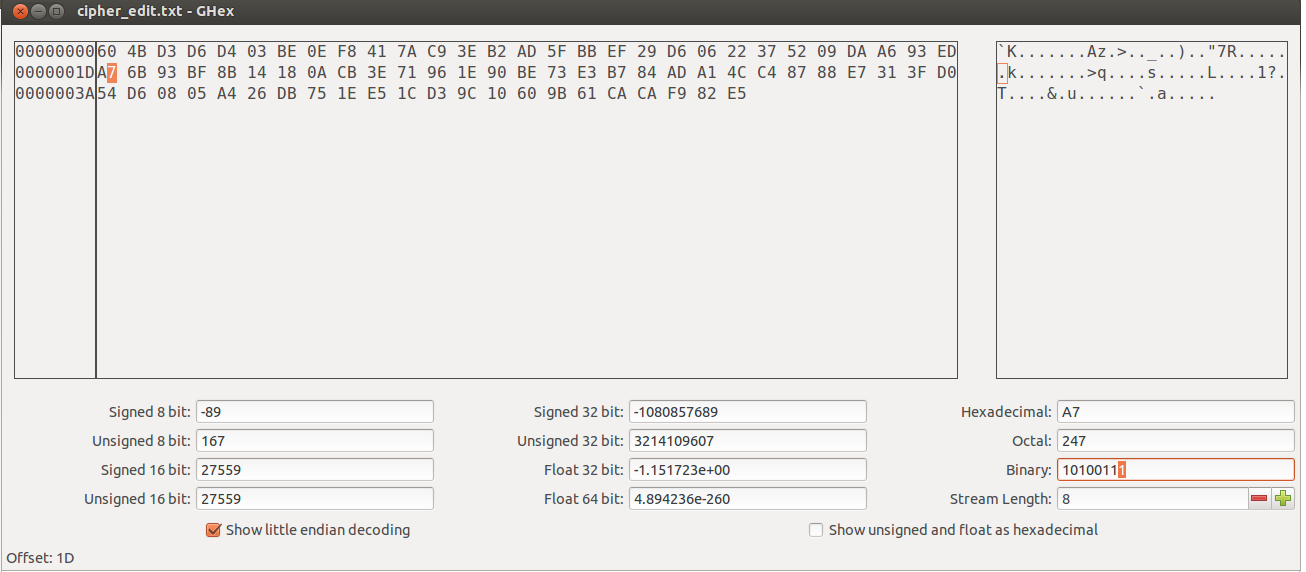
**Observation:**

The 30th byte (2nd row 1st byte) A6 in which the binary value 10100110 is going to be flipped.



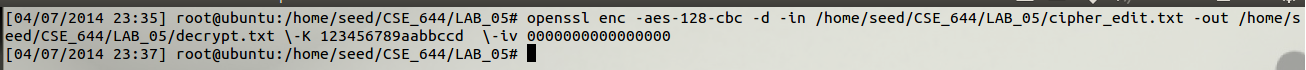
**Observation:**

The cipher text has been edited. The highlighted character is where the modification is took place.



**Observation:**

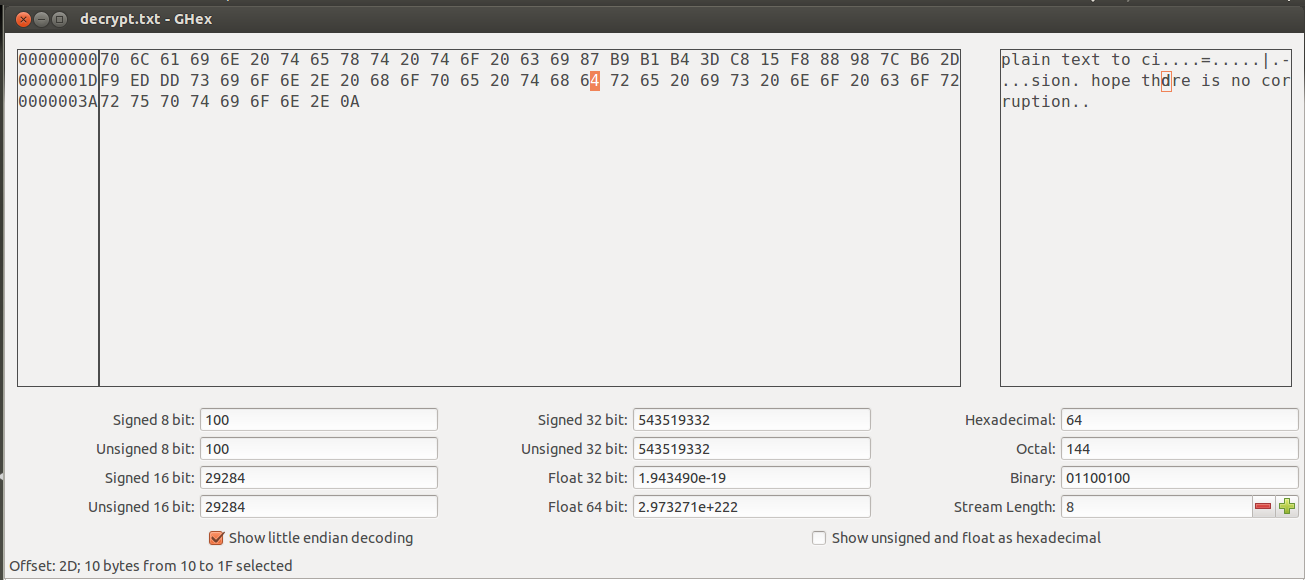
The 30th byte (2nd row 1st byte) is changed from A6 to A7 in which the binary value 10100110 was flipped to 10100111.



**Observation:**

The corrupted cipher text is decrypted using the same cipher, mode, key and IV.

C:\Users\worker\Desktop\lab 5\task 3\cbc\9decrypt text.png



**Observation:**

The entire 128-bit of the decrypted block (17-32 -> 2nd block) and the 46th byte (from 3rd block) are corrupted and the rest of the blocks are not corrupted.

1. Explain why?

The entire 2nd block and the 46th byte of the decrypted text were corrupted. This is because as the corrupted 2nd block of the cipher text was decrypted using the key it will affect the corresponding plain text formed and when the same corrupted cipher block was XORed with the decrypted cipher block only the corresponding byte in the decrypted block would be corrupted.

1. What are the implications of these differences?

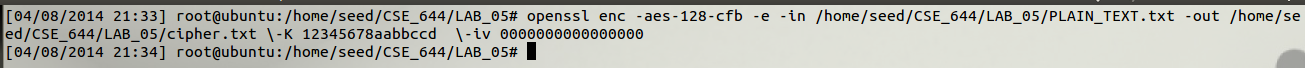
The entire decrypted block containing the 46th byte will not be corrupted since the corrupted cipher block is only XORed with the plain text block.

**Detecting corruption in CFB mode:**

1. How much information can you recover by decrypting the corrupted file?
2. Before conducting the task:

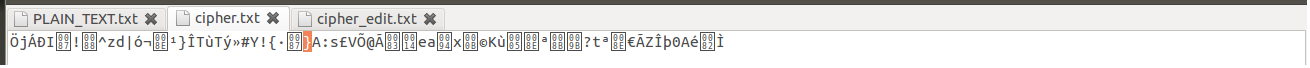
The CFB is similar to CBC. Here the corrupted cipher text is XORed with an intermediate text to form the plain text so only one bit of the plain text is erroneous and then that corrupted cipher text is passed into the block cipher encryption which is XORed with another uncorrupted cipher text so the entire plain text is corrupted.

1. After conducting the task:



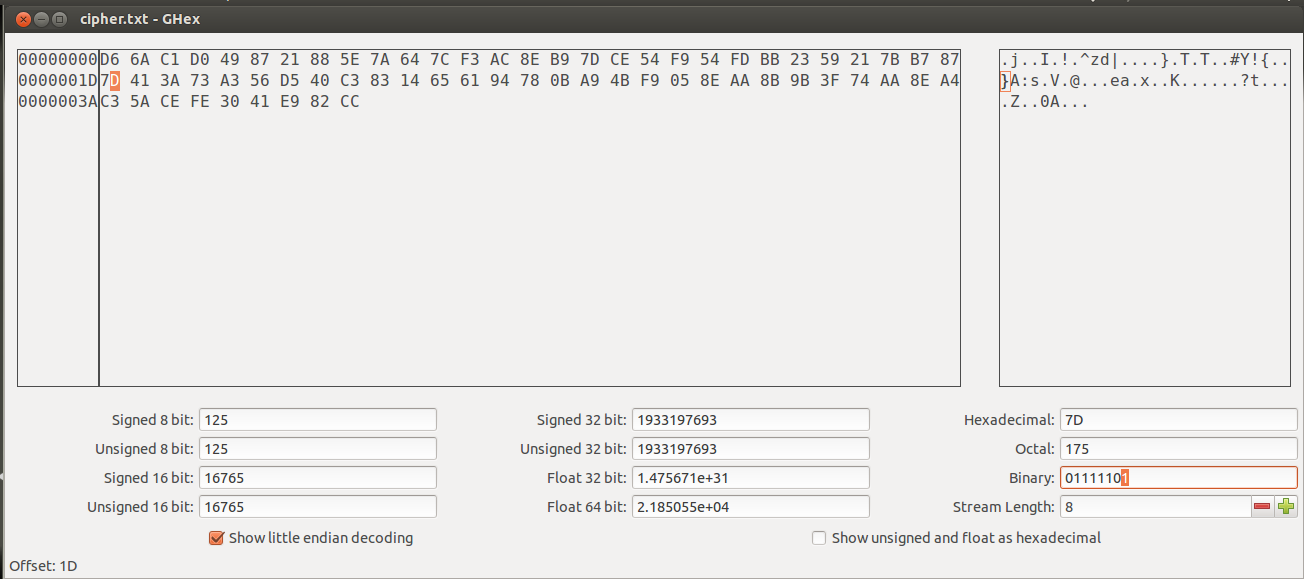
**Observation:**

The plain text is encrypted using AES-128-CFB mode and with 128-bit key and IV.



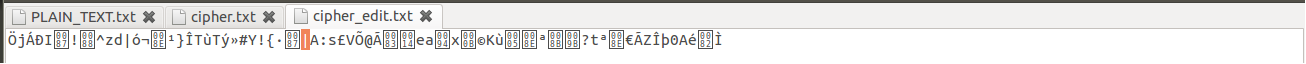
**Observation:**

The original cipher text formed during encryption. The highlighted character is where the modification is going to take place.



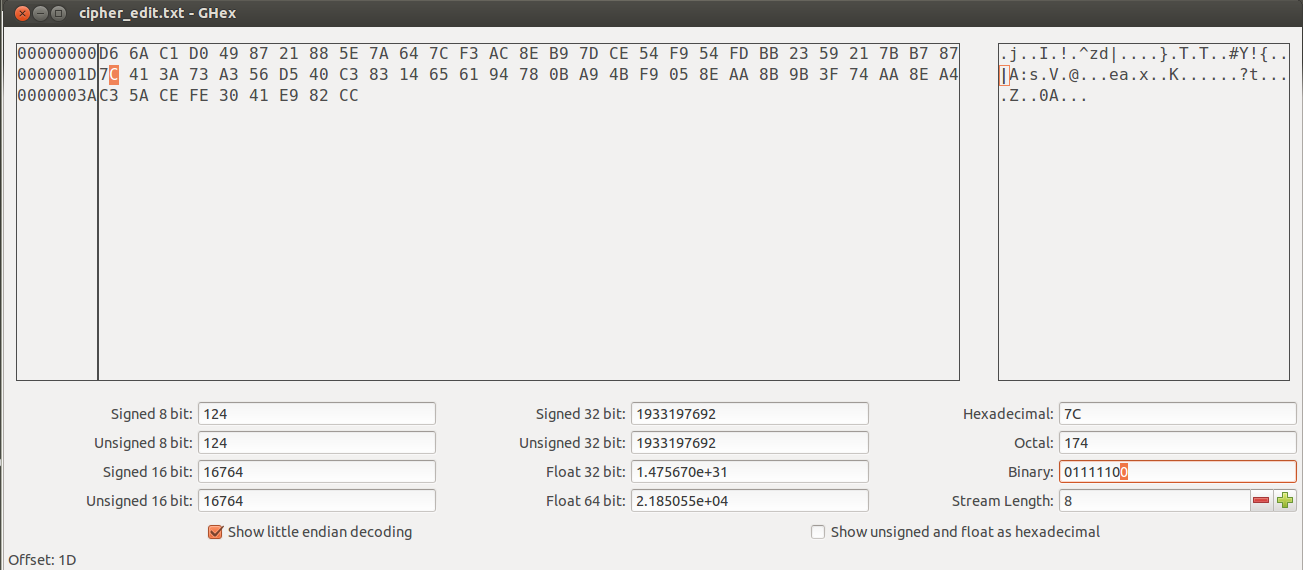
**Observation:**

The 30th byte (2nd row 1st byte) 7D in which the binary value 01111101 is going to be flipped.



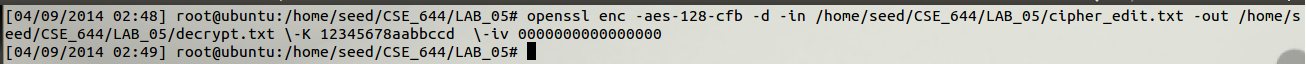
**Observation:**

The cipher text has been edited. The highlighted character is where the modification is took place.



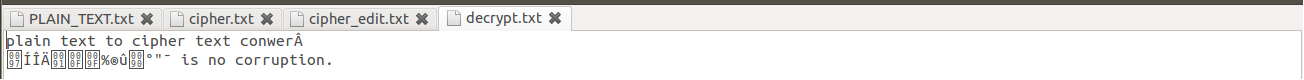
**Observation:**

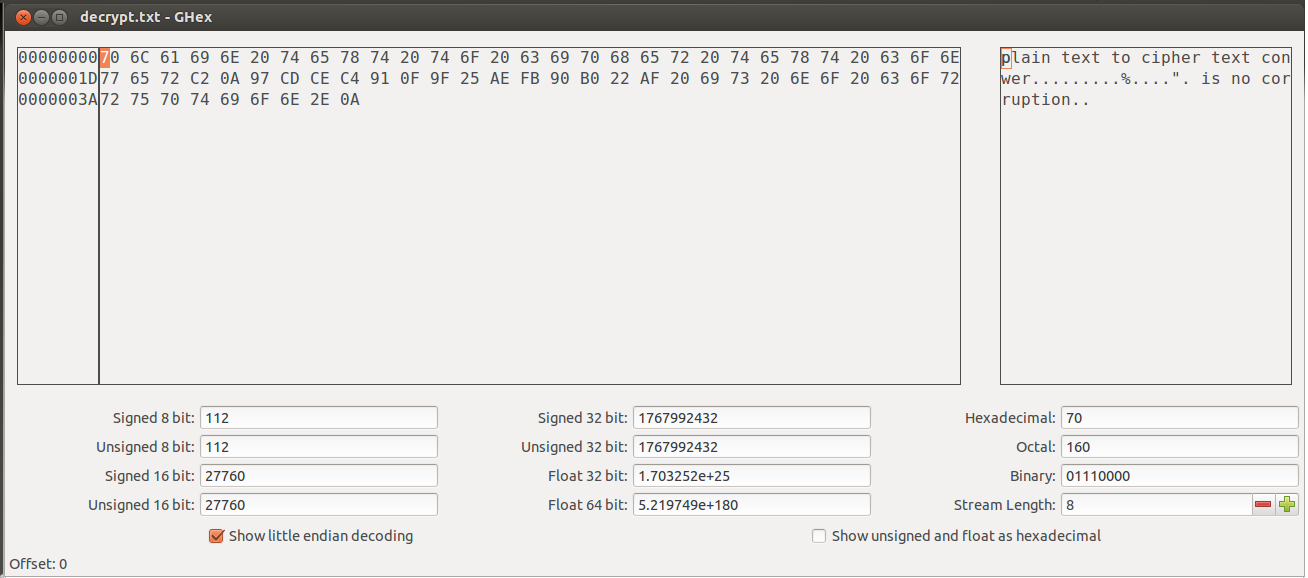
The 30th byte (2nd row 1st byte) is changed from 7D to 7C in which the binary value 01111101 was flipped to 01111100.



**Observation:**

The corrupted cipher text is decrypted using the same cipher, mode, key and IV.





**Observation:**

The entire 128-bit of the decrypted block (33-48 -> 3rd block) and the 30th byte (from 2nd block) are corrupted and the rest of the blocks are not corrupted.

1. Explain why?

The entire 3rd block and the 30th byte of the decrypted text were corrupted. This is because as the corrupted 3rd block of the cipher text was decrypted using the key it will affect the corresponding plain text formed and when the same corrupted cipher block was XORed with the decrypted cipher block only the corresponding byte in the decrypted block would be corrupted.

1. What are the implications of these differences?

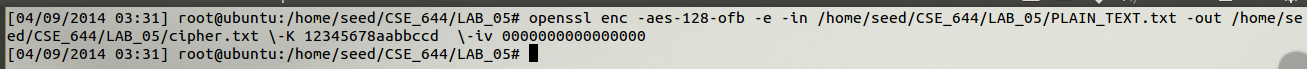
The assumption and the actual process were similar.

**Detecting corruption in OFB mode:**

1. How much information can you recover by decrypting the corrupted file?
2. Before conducting the task:

Since only a single bit of the cipher block is corrupted so when the cipher block is XORed with the output block of block cipher encryption only a single bit of that corresponding plain text block will be corrupted. The error will not occur in other plain text blocks since the corrupted cipher block is not shared by other encryption blocks.

1. After conducting the task:



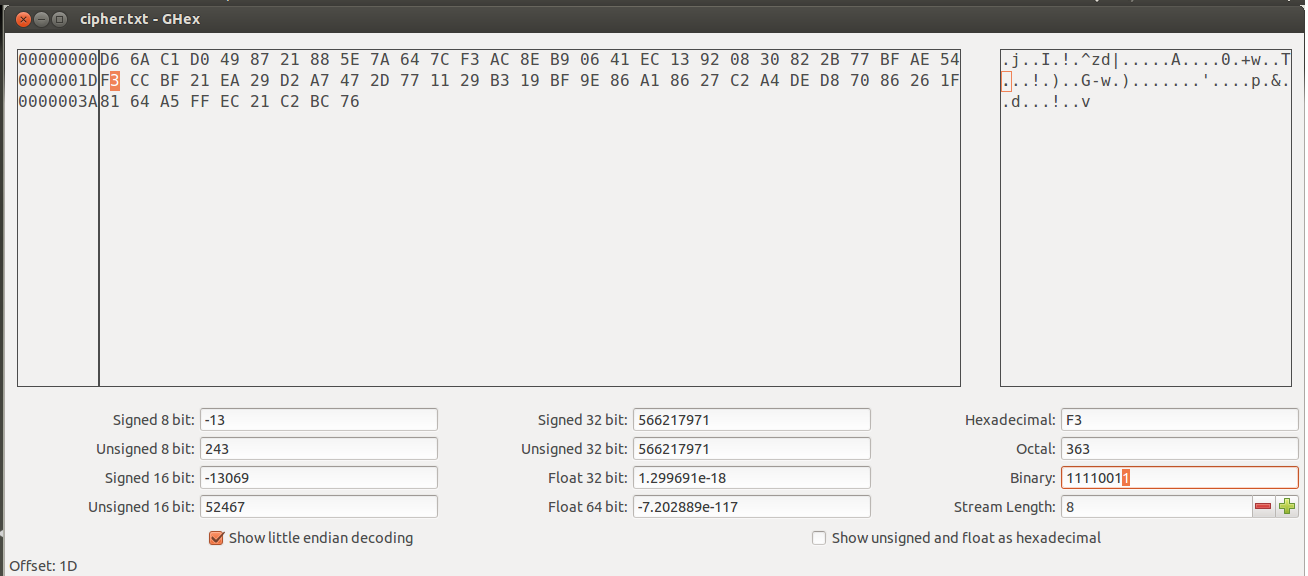
**Observation:**

The plain text is encrypted using AES-128-OFB mode and with 128-bit key and IV.



**Observation:**

The original cipher text formed during encryption.



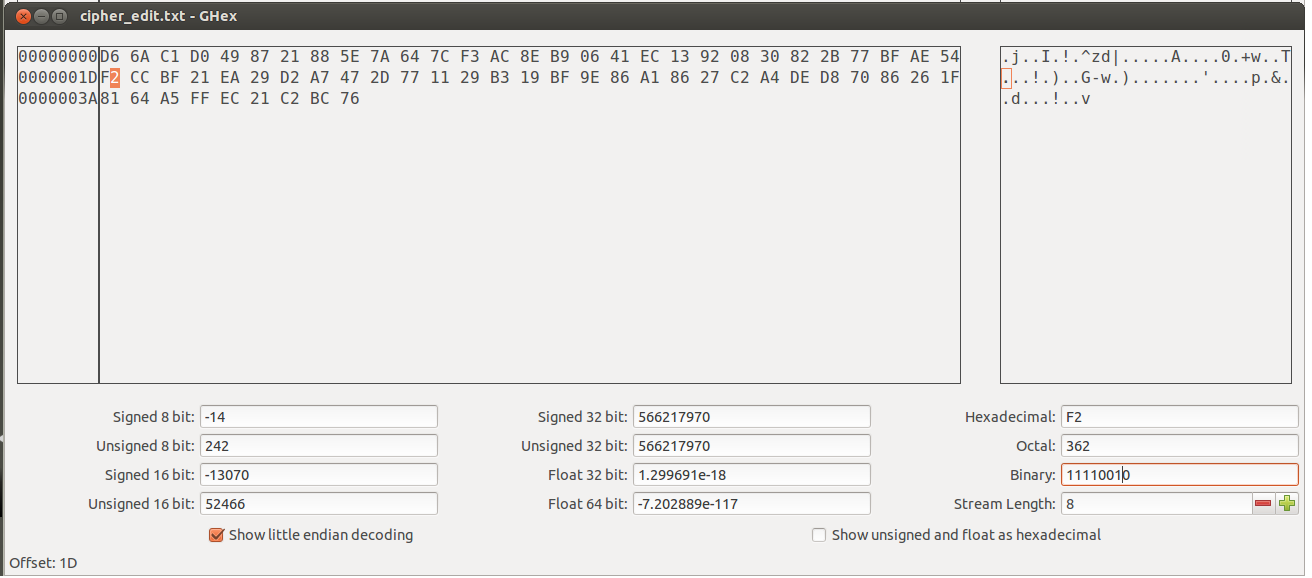
**Observation:**

The 30th byte (2nd row 1st byte) FE in which the binary value 11110011 is going to be flipped.

C:\Users\worker\Desktop\lab 5\task 3\ofb\6cipher edit.png

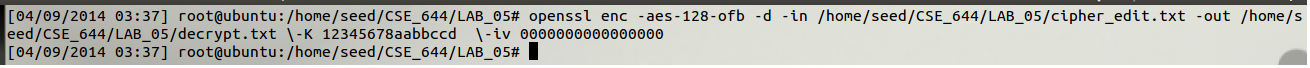
**Observation:**

The cipher text has been edited.



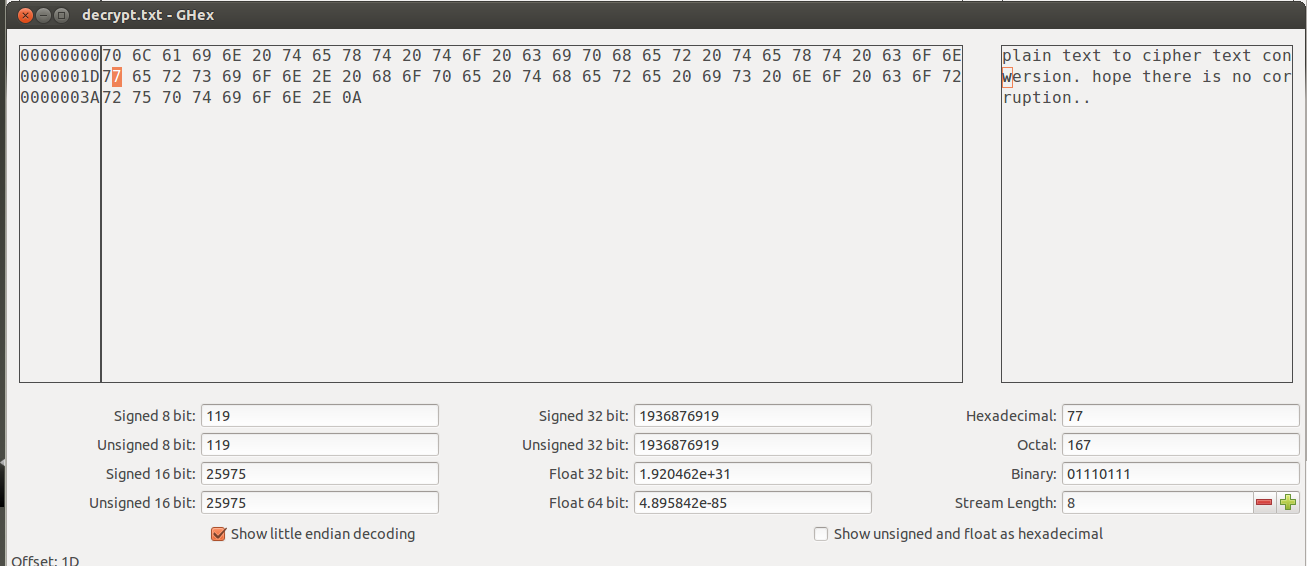
**Observation:**

The 30th byte (2nd row 1st byte) is changed from F3 to F2 in which the binary value 11110011 was flipped to 1111010.



**Observation:**

The corrupted cipher text is decrypted using the same cipher, mode, key and IV.

C:\Users\worker\Desktop\lab 5\task 3\ofb\9decrypt text.png

**Observation:**

Only the 30th byte (from 2nd block) is corrupted and the rest of the blocks are not corrupted.

1. Explain why?

Only a single bit of the cipher block is corrupted so when the cipher block is XORed with the output block of block cipher encryption only a single bit of that corresponding plain text block is corrupted. The error is contained within that plain text block.

1. What are the implications of these differences?

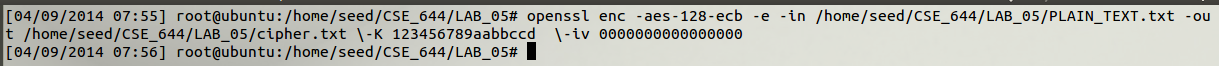
The assumption and the actual process were similar.

**Detecting corruption in ECB mode:**

1. How much information can you recover by decrypting the corrupted file?
2. Before conducting the task:

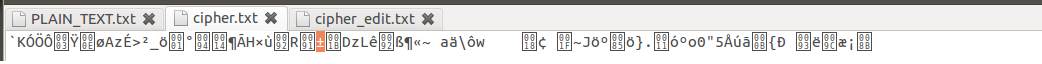
ECB is similar to OFB where a single cipher block is used get a single plain text block. So if a single bit of the cipher block is corrupted then only the corresponding plain test block will be corrupted the rest of the plain text will not get corrupted.

1. After conducting the task:



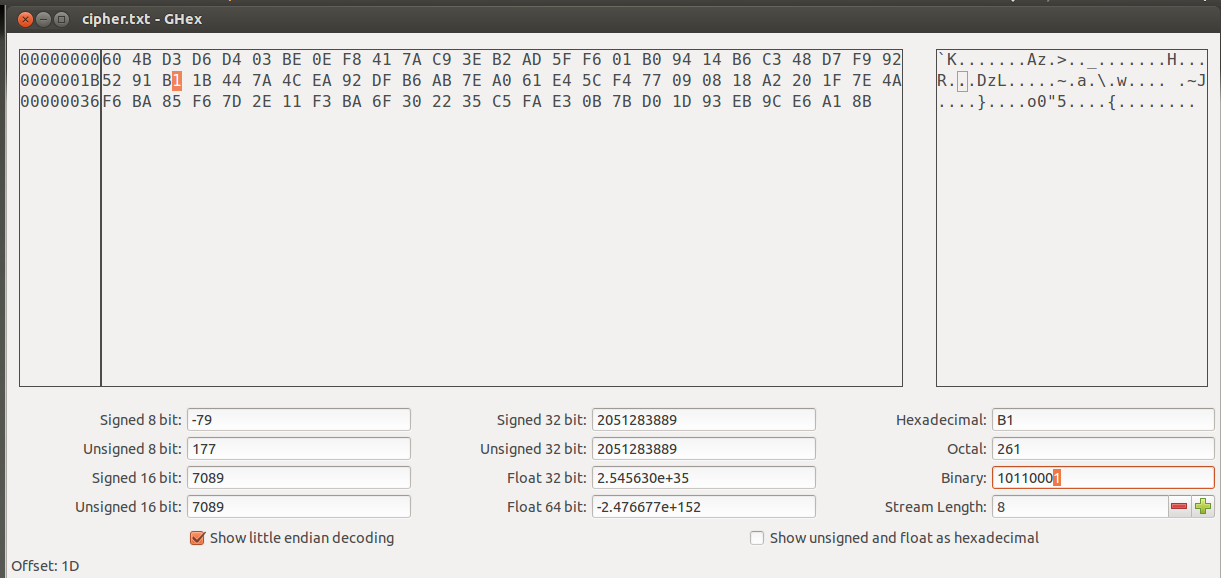
**Observation:**

The plain text is encrypted using AES-128-ECB mode and with 128-bit key and IV.



**Observation:**

The original cipher text formed during encryption. The highlighted character is where the modification is going to take place.



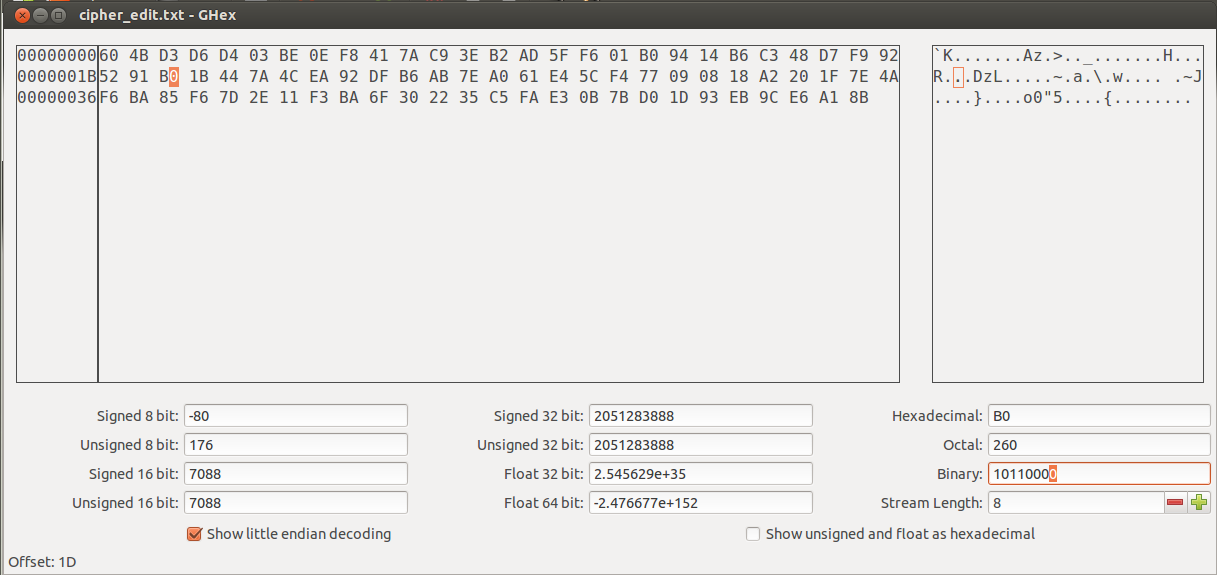
**Observation:**

The 30th byte (2nd row 3rd byte) B1 in which the binary value 10110001 is going to be flipped.

C:\Users\worker\Desktop\lab 5\task 3\ecb\6cipher edit.png

**Observation:**

The cipher text has been edited. The highlighted character is where the modification is took place.



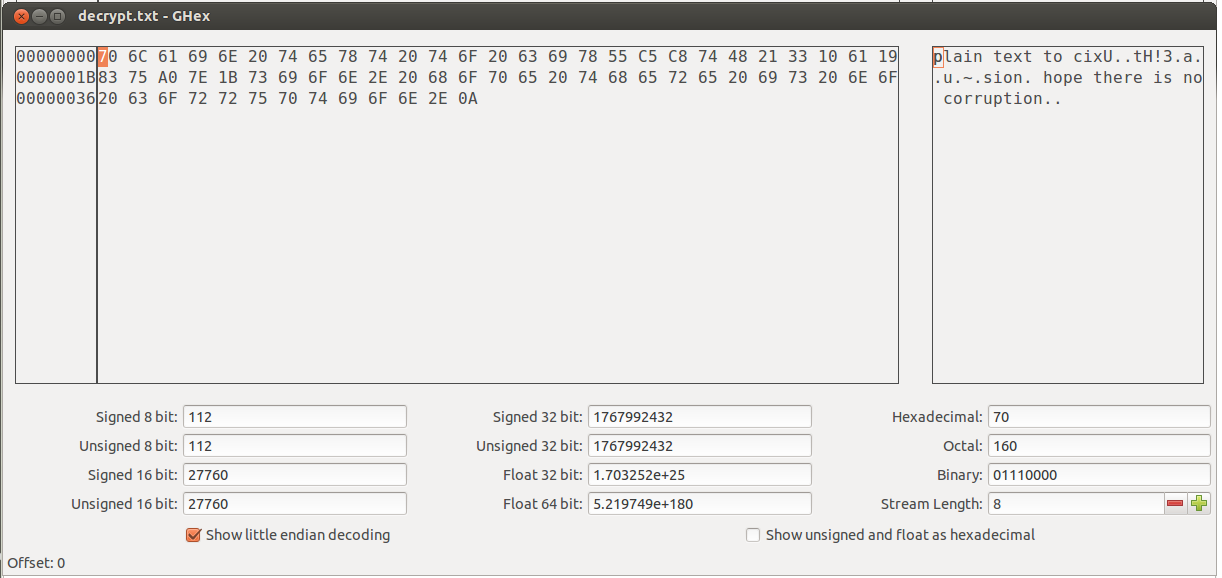
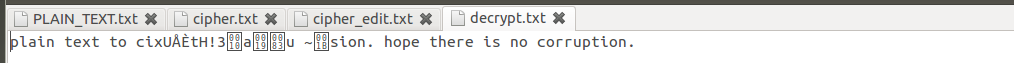
**Observation:**

The 30th byte (2nd row 3rd byte) is changed from B1 to B0 in which the binary value 10110001 was flipped to 10110000.



**Observation:**

The corrupted cipher text is decrypted using the same cipher, mode, key and IV.



**Observation:**

The entire 2nd block (17 – 32 bytes) is corrupted and the rest of the blocks are not corrupted.

1. Explain why?

Since a single bit of the cipher block is corrupted then only the corresponding plain test block will be corrupted. The rest of the plain text will not get corrupted as the corrupted text is used to get other plain texts.

1. What are the implications of these differences?

The assumption and the actual process were similar.

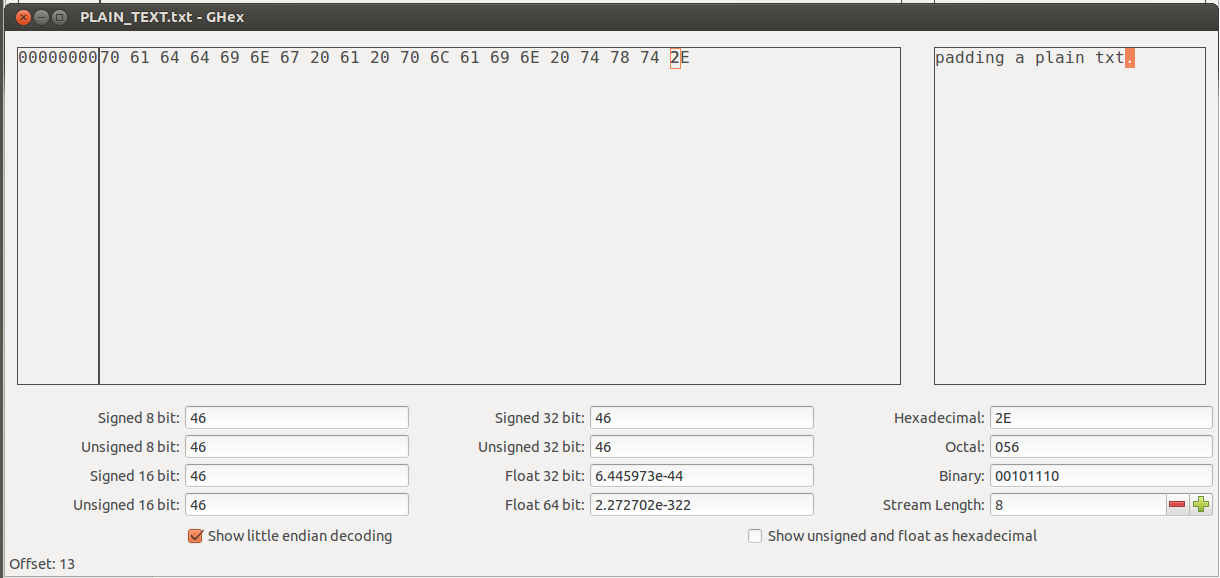
**Task 4: Padding**

Different cipher modes pad the original data in different ways to generate the cipher text. Padding is done before encryption and it is removed after decryption to get back the original data. Padding is done on the original data so that the ciphers generates can work on that data in an easier way. Padding bits usually contain some pad bits followed by the byte at the end representing the number pad bits added. Padding is usually applied by block ciphers so that they get the same block size for encrypting.

**Padding in ECB:**

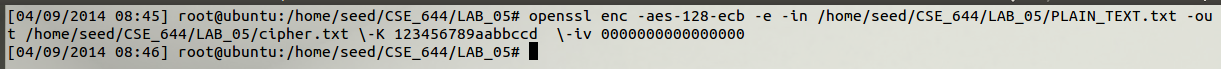
20-bytes:

C:\Users\worker\Desktop\lab 5\task 4\ecb\1plain text 20 bytes.png



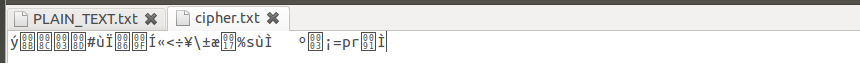
**Observation:**

The 20-byte plain text and its hex form.



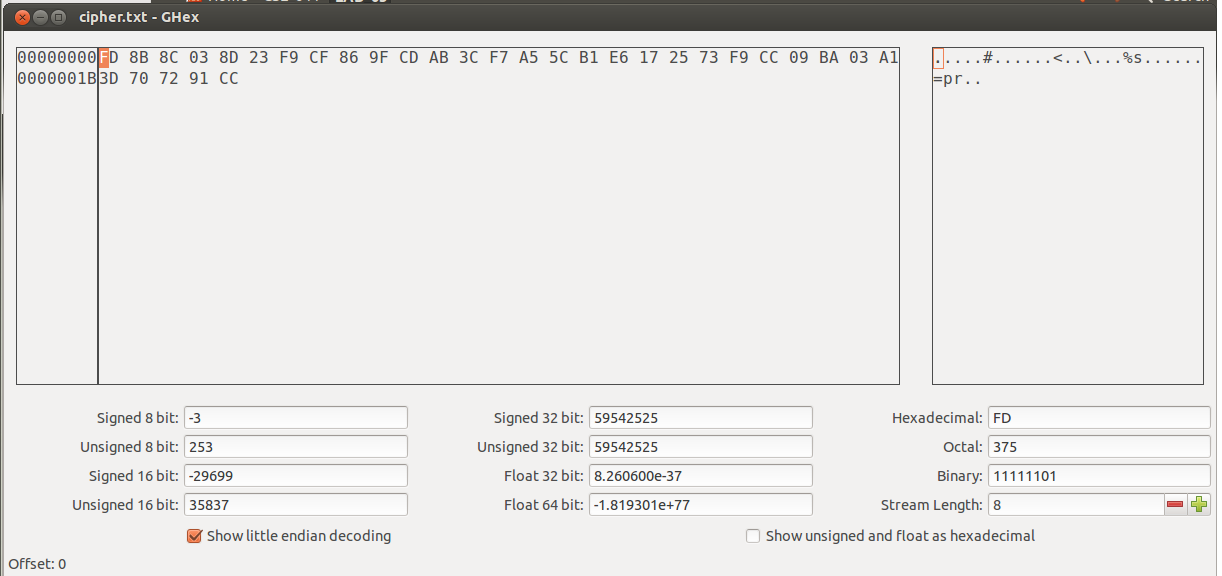
**Observation:**

The plain text is encrypted using AES-128 bit cipher with ECB mode. Padding will be done before the encryption starts.



**Observation:**

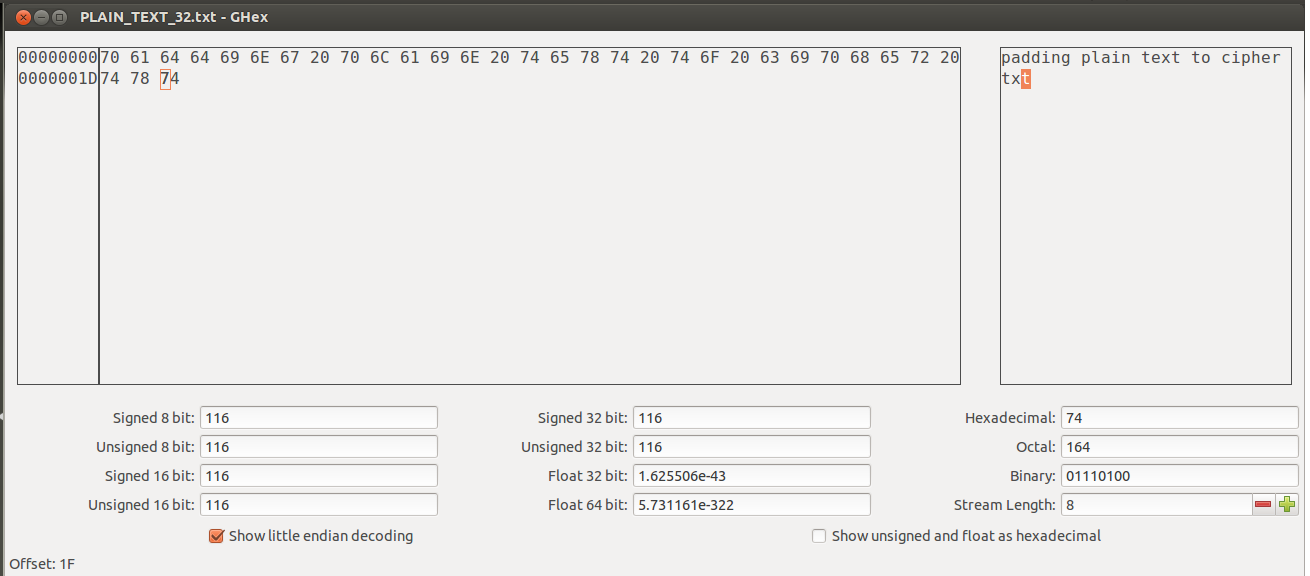
The cipher text formed from the AES-128-ECB encryption.



**Observation:**

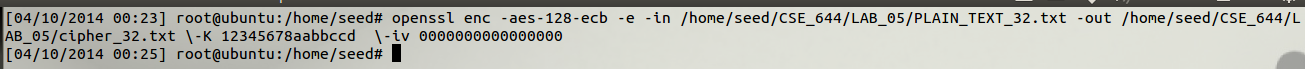
The generated cipher text contains 32-bytes which were formed from 20-byte plain text and 12-bytes of padding. As ECB is a block cipher the last 4-byte of the plain text is appended by 12-bytes of padding to get a 16-byte block of data.

32-bytes:

C:\Users\worker\Desktop\lab 5\task 4\ecb\6 plain text 32 bytes.png

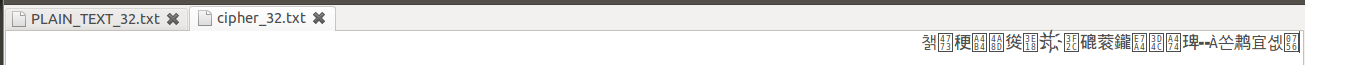
**Observation:**

The 32-byte plain text and its hex form.



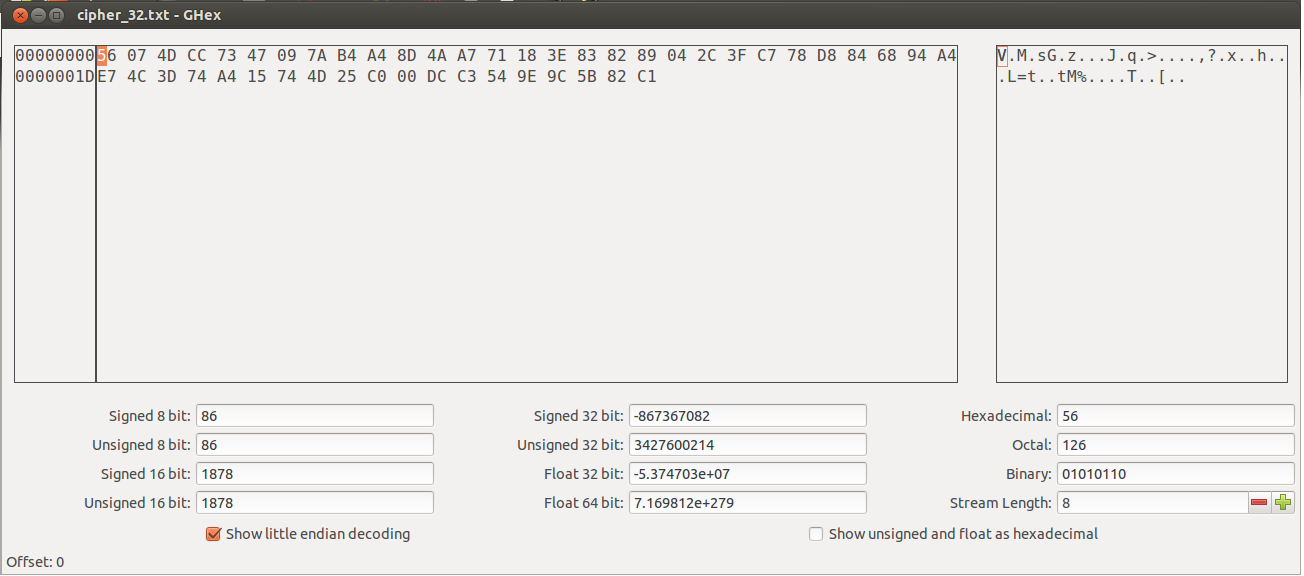
**Observation:**

The plain text is encrypted using AES-128 bit cipher with ECB mode. Padding will be done before the encryption starts.



**Observation:**

The cipher text formed from the AES-128-ECB encryption.

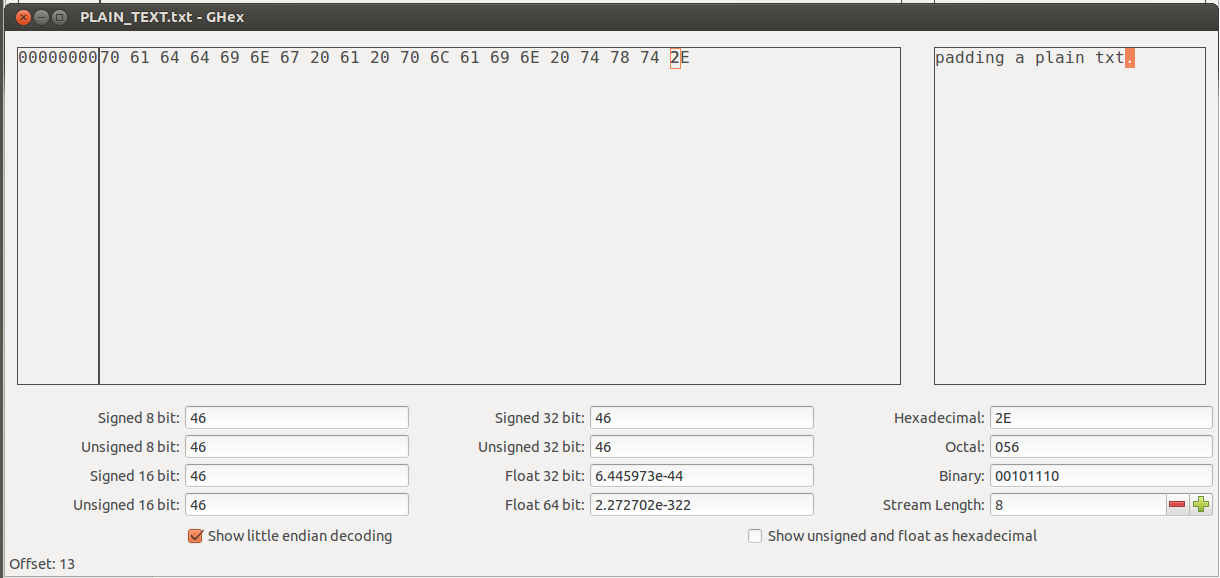


**Observation:**

The generated cipher text contains 48-bytes which were formed from 32-bytes of plain text and 16-bytes of padding. As openssl uses PKCS5 standard for its padding one byte (indicating the number of bytes added for padding) will be appended at the end of the plain text thus forming 33-bytes of plain text to be encrypted. So an extra 16-byte text is generated after encryption.

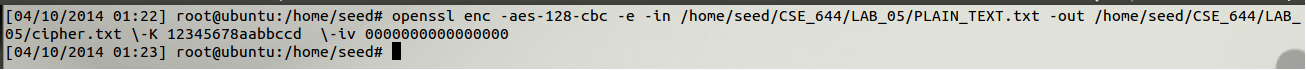
**Padding in CBC:**

20bytes:

C:\Users\worker\Desktop\lab 5\task 4\cbc\1plain text 20 bytes.png

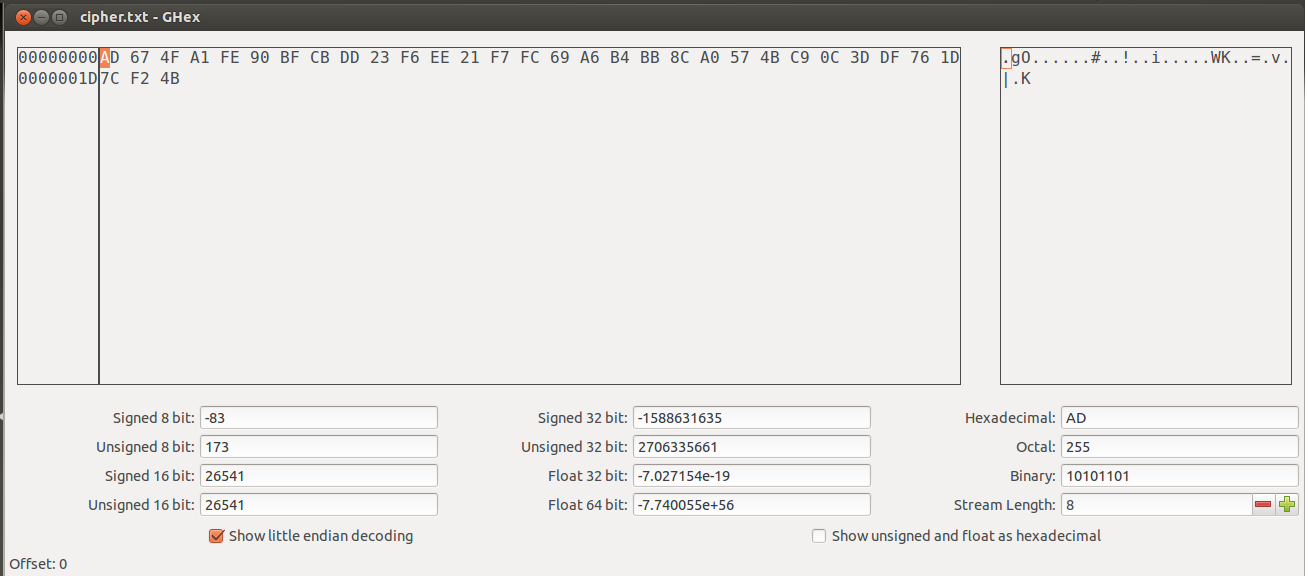
**Observation:**

The 20-byte plain text and its hex form.



**Observation:**

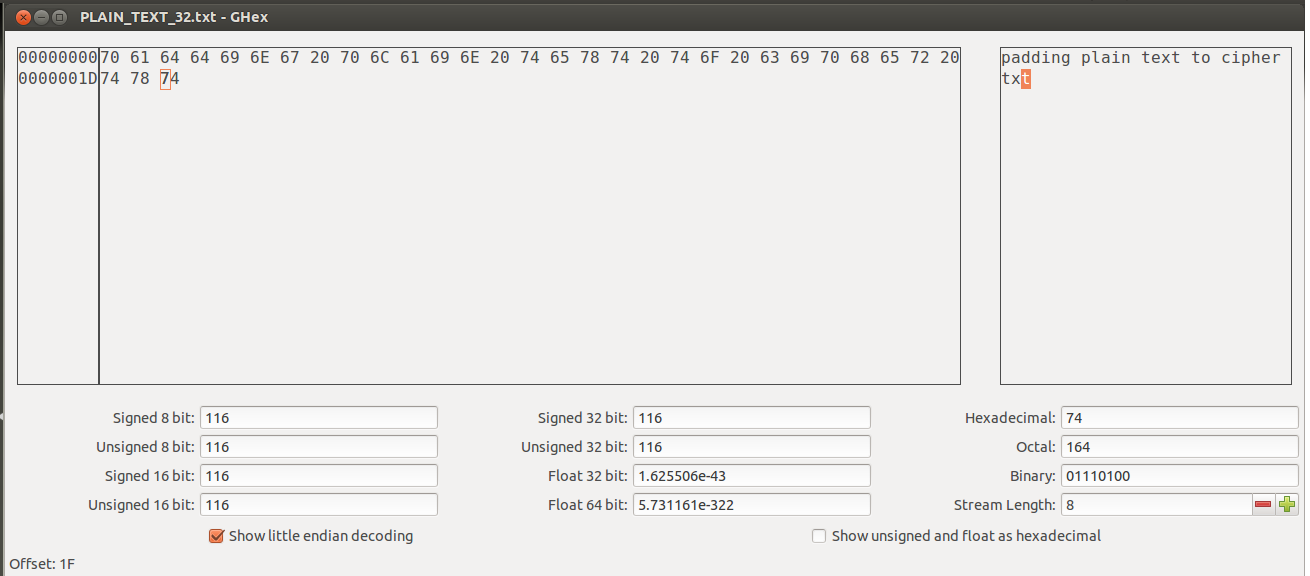
The plain text is encrypted using AES-128 bit cipher with CBC mode. Padding will be done before the encryption starts.

C:\Users\worker\Desktop\lab 5\task 4\cbc\4 cipher text.png

**Observation:**

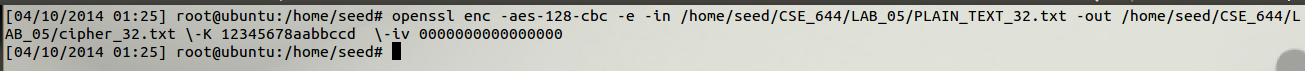
The generated cipher text contains 32-bytes which were formed from 20-byte plain text and 12-bytes of padding. As CBC is a block cipher the last 4-byte of the plain text is appended by 12-bytes of padding to get a 16-byte block of data.

32-bytes:

C:\Users\worker\Desktop\lab 5\task 4\cbc\6 plain text 32 bytes.png

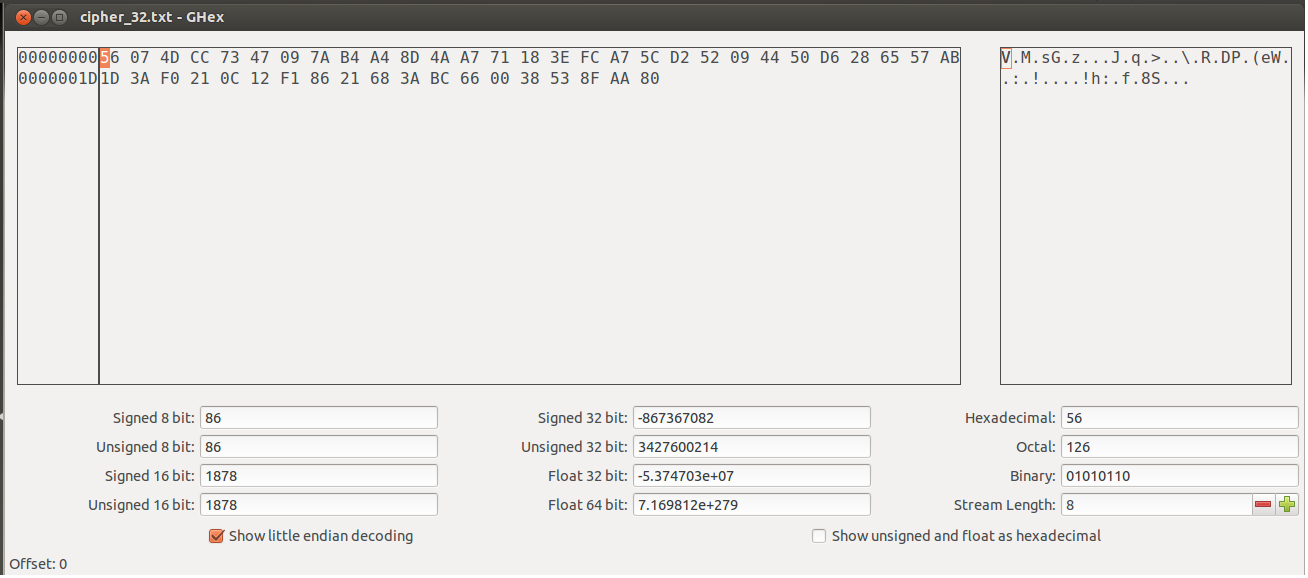
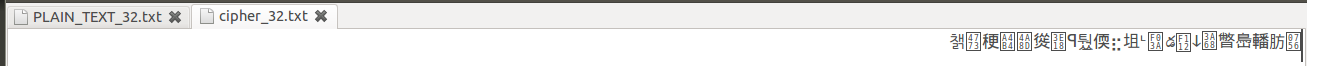
**Observation:**

The 32-byte plain text and its hex form.



**Observation:**

The plain text is encrypted using AES-128 bit cipher with ECB mode. Padding will be done before the encryption starts.

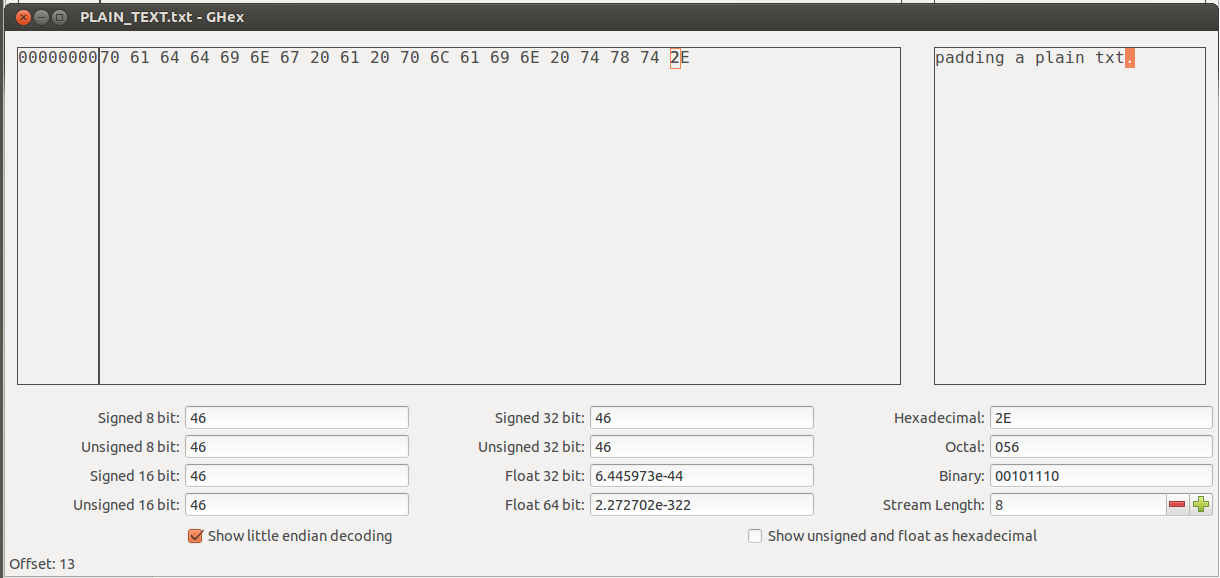


**Observation:**

The generated cipher text contains 48-bytes which were formed from 32-bytes of plain text and 16-bytes of padding. As openssl uses PKCS5 standard for its padding one byte (indicating the number of bytes added for padding) will be appended at the end of the plain text thus forming 33-bytes of plain text to be encrypted. So an extra 16-byte text is generated after encryption.

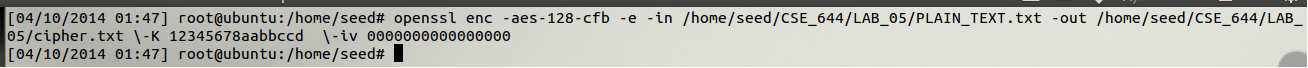
**Padding in CFB:**

20-bytes:

C:\Users\worker\Desktop\lab 5\task 4\cfb\1plain text 20 bytes.png

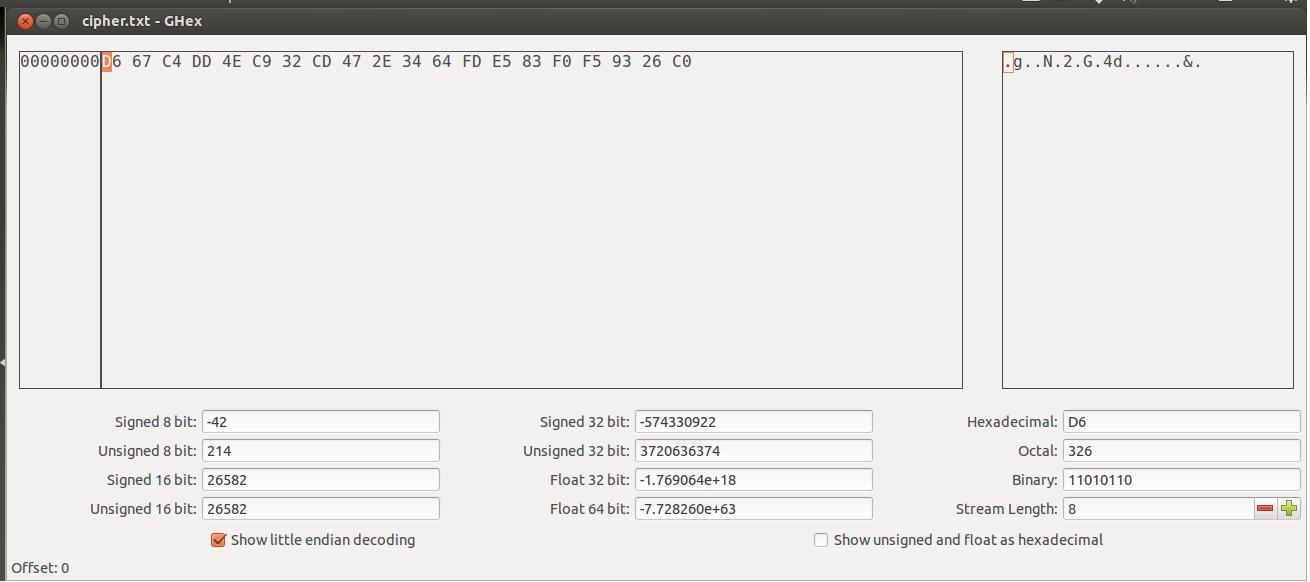
**Observation:**

The 20-byte plain text and its hex form.



**Observation:**

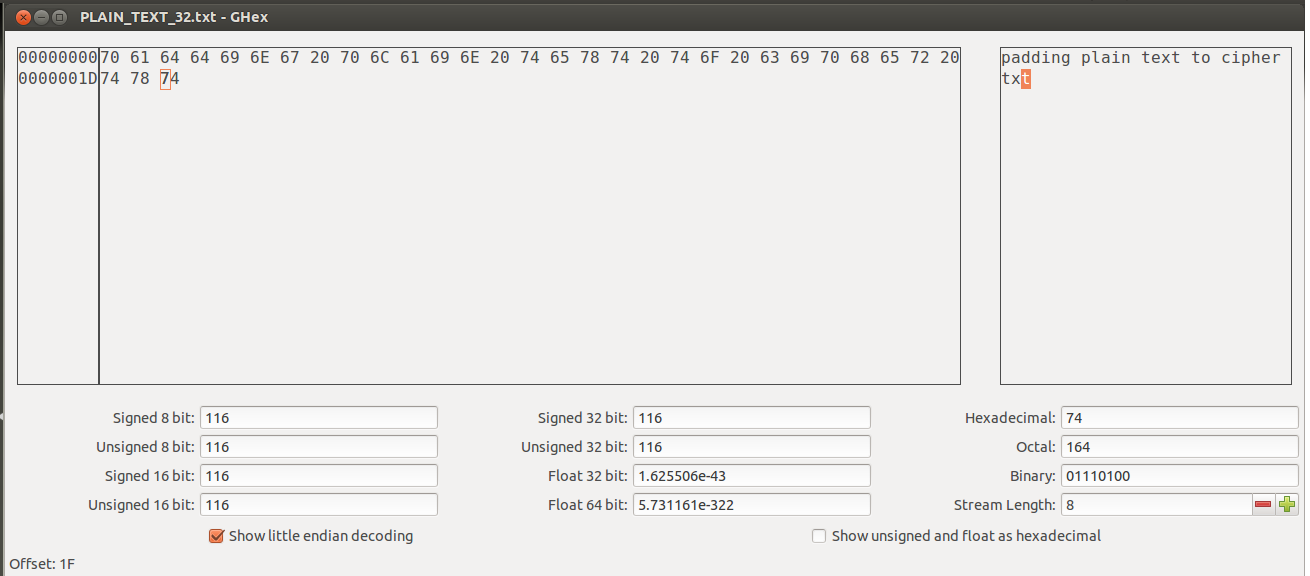
The plain text is encrypted using AES-128 bit cipher with CFB mode.

C:\Users\worker\Desktop\lab 5\task 4\cfb\4 cipher text.png

**Observation:**

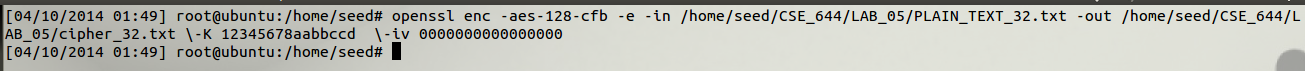
The generated cipher text contains 20-bytes which were formed from 20-byte plain text. Padding is not done since the CFB mode works by XORing the plain text the output of the block cipher. So the last 4-bytes of the plain text is XORed with the first 4-bytes of the last key stream block, thus producing the 4-byte cipher text.

32-bytes:

C:\Users\worker\Desktop\lab 5\task 4\cfb\6 plain text 32 bytes.png

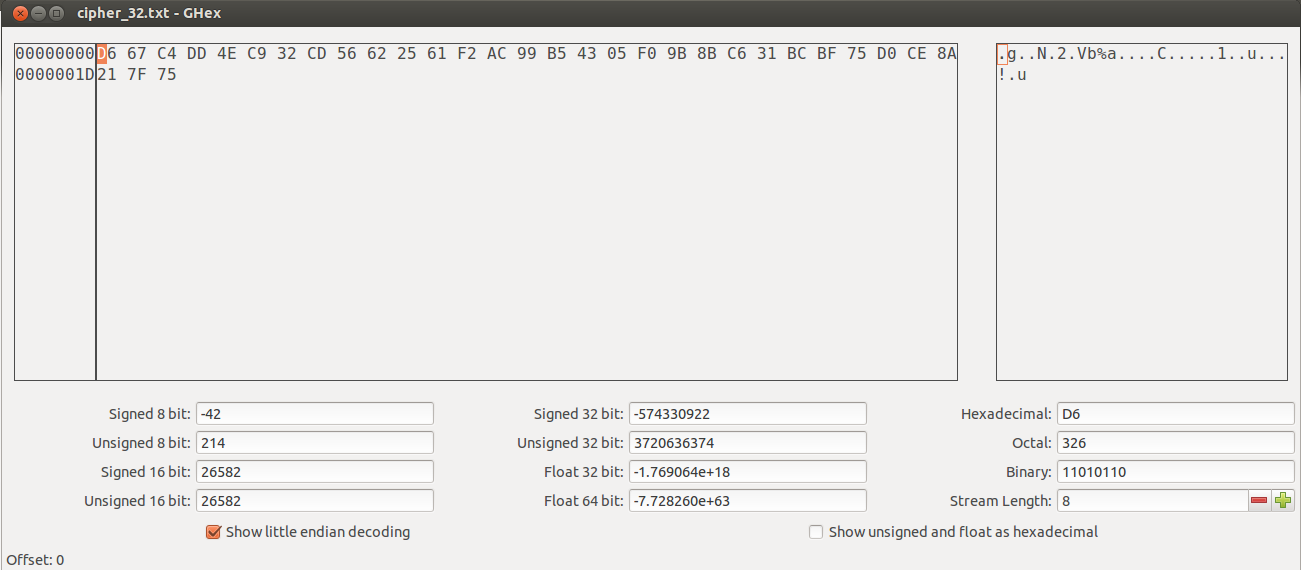
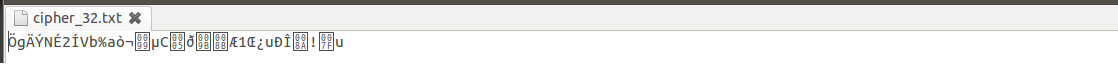
**Observation:**

The 32-byte plain text and its hex form.



**Observation:**

The plain text is encrypted using AES-128 bit cipher with CFB mode.

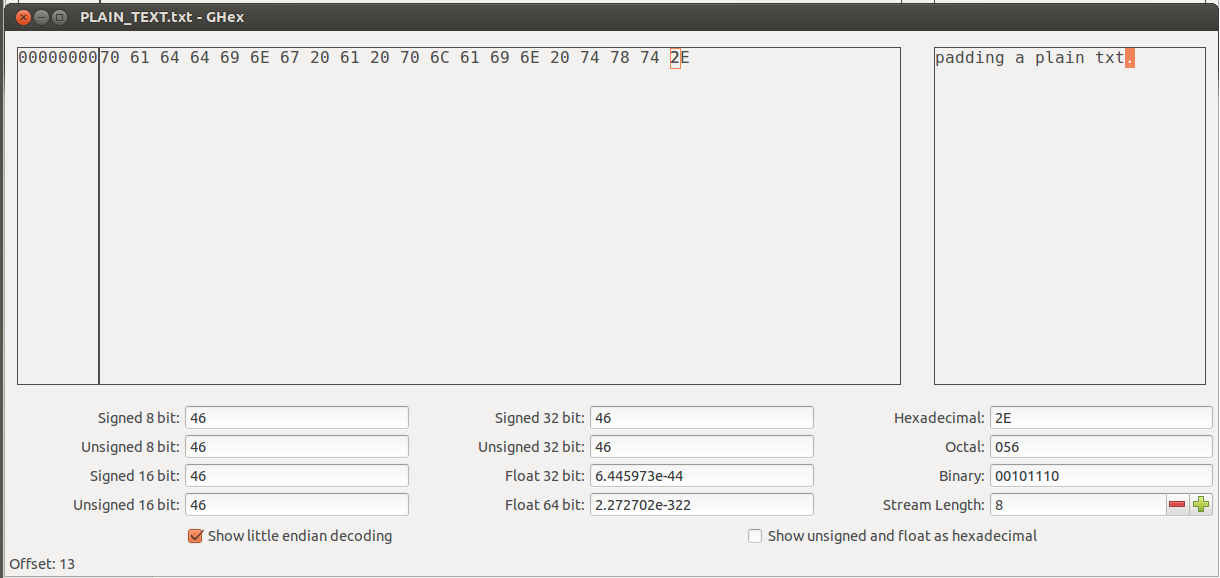


**Observation:**

The generated cipher text contains 32-bytes which were formed from 32-byte plain text. Padding is not done since the CFB mode works by XORing the plain text the output of the block cipher.

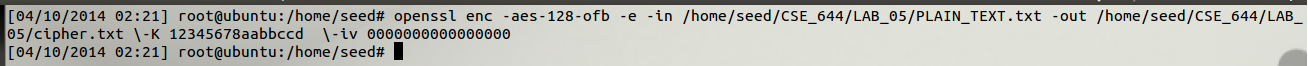
**Padding in OFB:**

20-bytes:

C:\Users\worker\Desktop\lab 5\task 4\ofb\1plain text 20 bytes.png

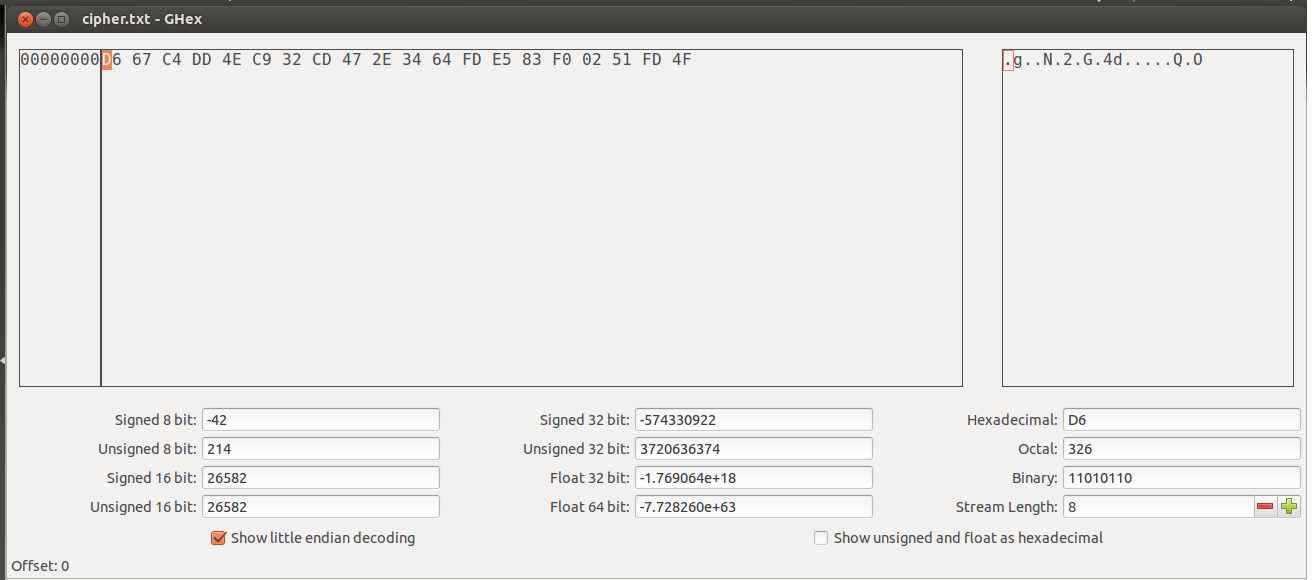
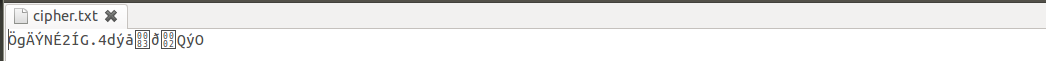
**Observation:**

The 20-byte plain text and its hex form.



**Observation:**

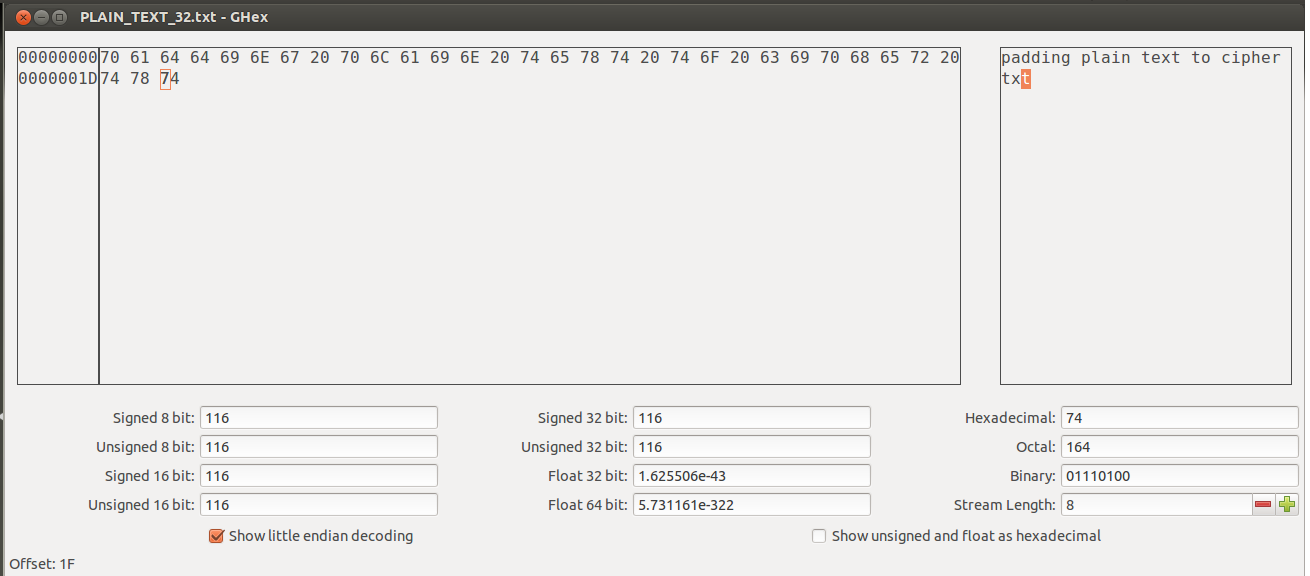
The plain text is encrypted using AES-128 bit cipher with OFB mode.



**Observation:**

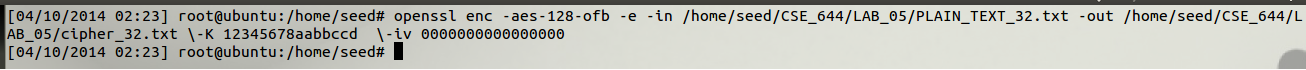
The generated cipher text contains 20-bytes which were formed from 20-byte plain text. Padding is not done since the OFB mode works by XORing the plain text the output of the block cipher. So the last 4-bytes of the plain text is XORed with the first 4-bytes of the last key stream block, thus producing the 4-byte cipher text.

32-bytes:

C:\Users\worker\Desktop\lab 5\task 4\ofb\6 plain text 32 bytes.png

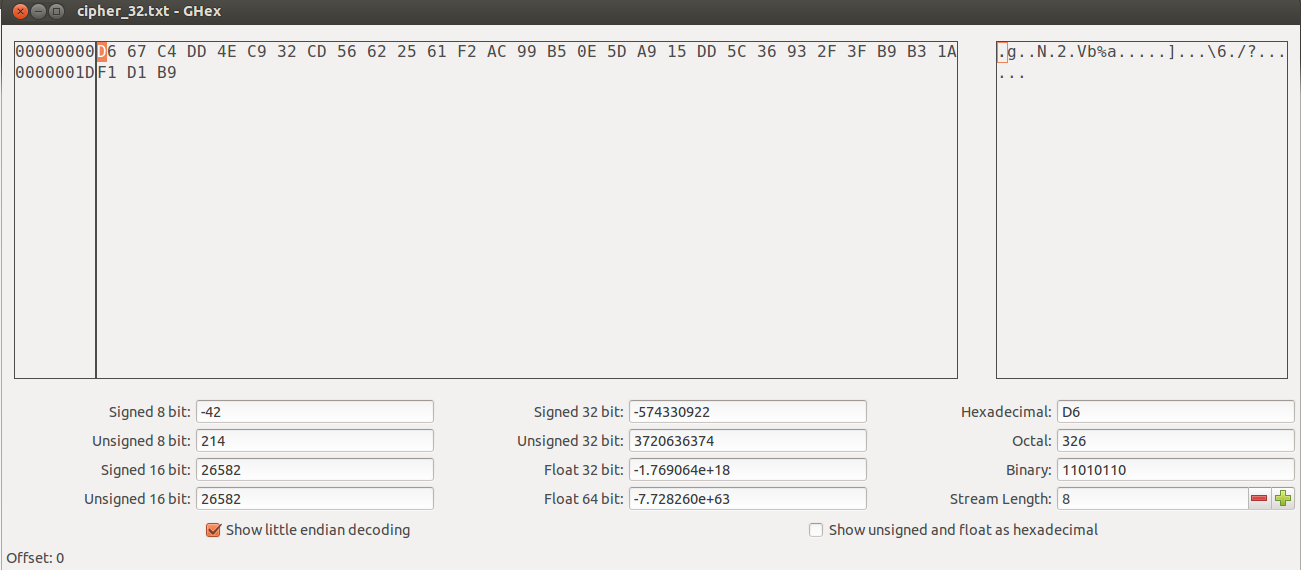
**Observation:**

The 32-byte plain text and its hex form.



**Observation:**

The plain text is encrypted using AES-128 bit cipher with OFB mode.

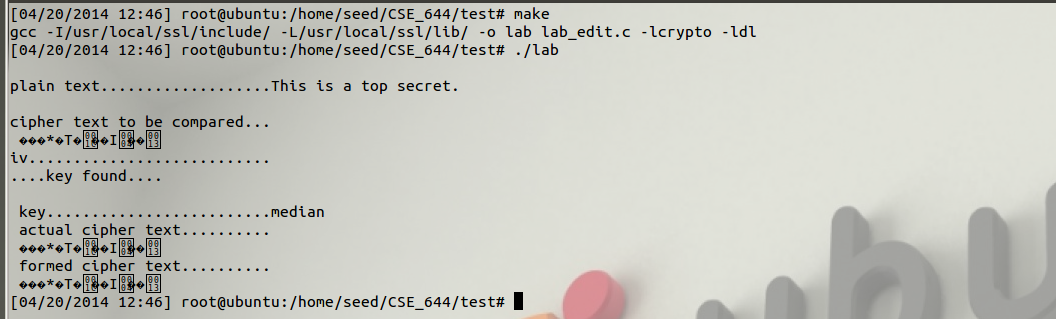
C:\Users\worker\Desktop\lab 5\task 4\ofb\9 cipher text 32.png

**Observation:**

The generated cipher text contains 32-bytes which were formed from 32-byte plain text. Padding is not done since the CFB mode works by XORing the plain text the output of the block cipher.

**Task 5: programming using the Crypto Library**

This task uses the Envelope (EVP) interface and links it to the openssl libraries where several encryption algorithms are in place.



Observation:

There is no key in the word list that can generate that cipher text.

**Programs:**

**Lab\_edit.c:**

#include <openssl/conf.h>

#include <openssl/evp.h>

#include <openssl/err.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

int hex\_to\_int(char c){

int first = c / 16 - 3;

int second = c % 16;

int result = first\*10 + second;

if(result > 9) result--;

return result;

}

int hex\_to\_ascii(char c, char d){

int high = hex\_to\_int(c) \* 16;

int low = hex\_to\_int(d);

return high+low;

}

int main(int arc, char \*argv[])

{

unsigned char outbuf[1024];

unsigned char cipher[1024];

unsigned char temp, key[16];

int outlen, tmplen, l, i, length, count, found =0, k = 0;

size\_t nread, len;

FILE \*in;

unsigned char iv[17];

for(i = 0; i < 17; i++)

iv[i] = 0;

iv[16] = '\0';

char intext[] = "This is a top secret.";

char st[] = "8d20e5056a8d24d0462ce74e4904c1b513e10d1df4a2ef2ad4540fae1ca0aaf9";

i = 0;

while(i < 64)

{

if(st[i] >= 'a' && st[i] <= 'z')

st[i] = st[i] - 32;

i++;

}

length = strlen(st);

char buf = 0;

for(i = 0; i < length; i++)

{

if(i % 2 != 0)

{

cipher[k] = hex\_to\_ascii(buf, st[i]);

k++;

}

else

{

buf = st[i];

}

}

cipher[k] = '\0';

in = fopen("/home/seed/CSE\_644/LAB\_05/english\_wrd\_list.txt", "r");

if(in == NULL)

{

printf("\n cannot open file");

exit(1);

}

EVP\_CIPHER\_CTX ctx;

EVP\_CIPHER\_CTX\_init(&ctx);

while(fgets(key, sizeof(key), in) != NULL)

{

l = 0;

if(strlen(key) < 16)

{

l = strlen(key)-1;

while(l < 16)

{

key[l] = ' ';

l++;

}

key[l] = '\0';

}

else

key[16] = '\0';

EVP\_EncryptInit\_ex(&ctx, EVP\_aes\_128\_cbc(), NULL, key, iv);

if(!EVP\_EncryptUpdate(&ctx, outbuf, &outlen, intext, strlen(intext)))

{

/\* Error \*/

return 0;

}

/\* Buffer passed to EVP\_EncryptFinal() must be after data just

\* encrypted to avoid overwriting it.

\*/

if(!EVP\_EncryptFinal\_ex(&ctx, outbuf + outlen, &tmplen))

{

/\* Error \*/

return 0;

}

outlen += tmplen;

EVP\_CIPHER\_CTX\_cleanup(&ctx);

count = 0;

for(i = 0; i < 32; i++)

{

if(cipher[i] == outbuf[i])

count++;

}

//printf("\n actual cipher text..........%s", cipher);

//printf("\n formed cipher text..........%s\n", outbuf);

if(count == 32)

{

printf("\nplain text...................%s\n",intext);

printf("\ncipher text to be compared...\n%s", cipher);

printf("\niv...........................%s",iv);

printf("\n....key found....\n");

printf("\n key.........................%s",key);

printf("\n actual cipher text..........\n%s", cipher);

printf("\n formed cipher text..........\n%s\n", outbuf);

found = 1;

break;

}

}

fclose(in);

if(found == 0)

{

printf("\niv...........................%s",iv);

printf("\nplain text...................%s\n",intext);

printf("cipher text..........%s\n", cipher);

printf("cipher text in hex...%s\n",st);

printf("\n\n key cannot be found for the above cipher text\n");

}

return 0;

}

**Makefile:**

INC=/usr/local/ssl/include/

LIB=/usr/local/ssl/lib/

all:

gcc -I$(INC) -L$(LIB) -o lab lab\_edit.c -lcrypto -ldl