# Introduction to Information Security [CSP 458 - 02]

Lab-2: Secret Key Encryption [100 points]

Anusha Sonte Parameshwar [A20518694]

---

**Overview:**

The learning objective of this lab is to get familiar with the concepts in the secret-key encryption by performing the tasks listed below:

**Lab Tasks:**

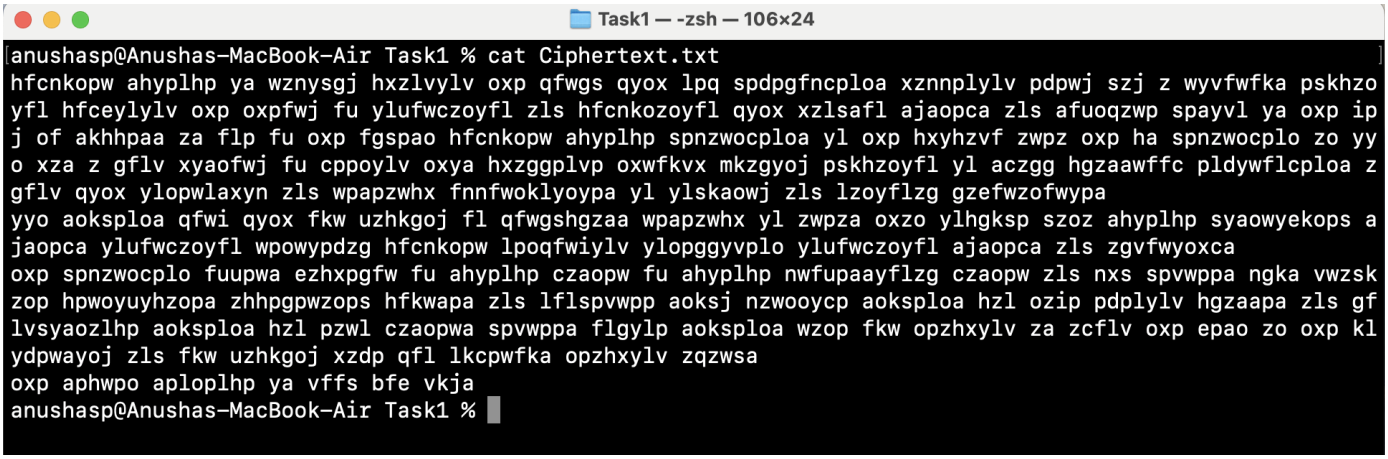

1. Frequency Analysis ……2
2. Encryption using Different Ciphers and Modes ……..6
   - 2.1 Using cipher type AES-128-CBC ……..6
   - 2.2 Using cipher type AES-128-CFB ……7
   - 2.3 Using cipher type BF-CBC ……..7
   - 2.4 Using cipher type AES-128-ECB ……8
   - 2.5 Using cipher type AES-128-OFB ……9
3. Encryption Mode: ECB vs CBC ……..10
   - 3.1 Using ECB Mode ……..10
   - 3.2 Using CBC Mode ……..11
   - 3.3 Testing with the picture of our choice ……..12
     - 3.3.1 Using ECB Mode ……13
     - 3.3.2 Using CBC Mode ……..14
4. Padding ……..15
   - 4.1 Using ECB Mode ……..16
   - 4.2 Using CBC Mode ……..17
   - 4.3 Using CFB Mode ……..19
   - 4.4 Using OFB Mode ……..21
5. Error propagation: Corrupted Ciphertext ……...23
   - 5.1 Using AES-128 ECB Mode ……..23
   - 5.2 Using AES-128 CBC Mode ……..25
   - 5.3 Using AES-A28 CFB Mode ……..26
   - 5.4 Using AES-A28 OFB Mode ……..28
6. Initial Vector (IV) ……...30
   - 6.1. Uniqueness of the IV ……...30
     - 6.1.1 Using two different IVs ……..30
     - 6.1.2 Using the same IV ……..32
   - 6.2. Common mistakes: Use the same IV ……33
   - 6.3. Common mistakes: Use a predictable IV ……34
7. Programming using the Crypto library [Extra Credit] ……...35

**Task 1: Frequency Analysis**

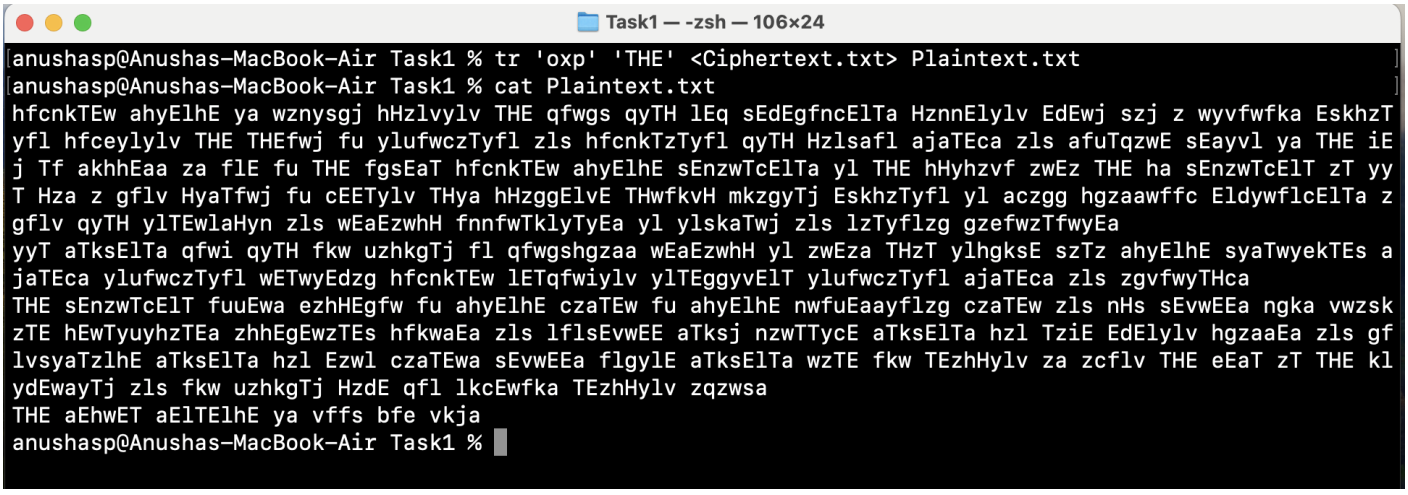
**Goal:** The goal of this task is to find out the original text from the mono-alphabetic cipher text using frequency analysis technique.

**Steps Performed:**

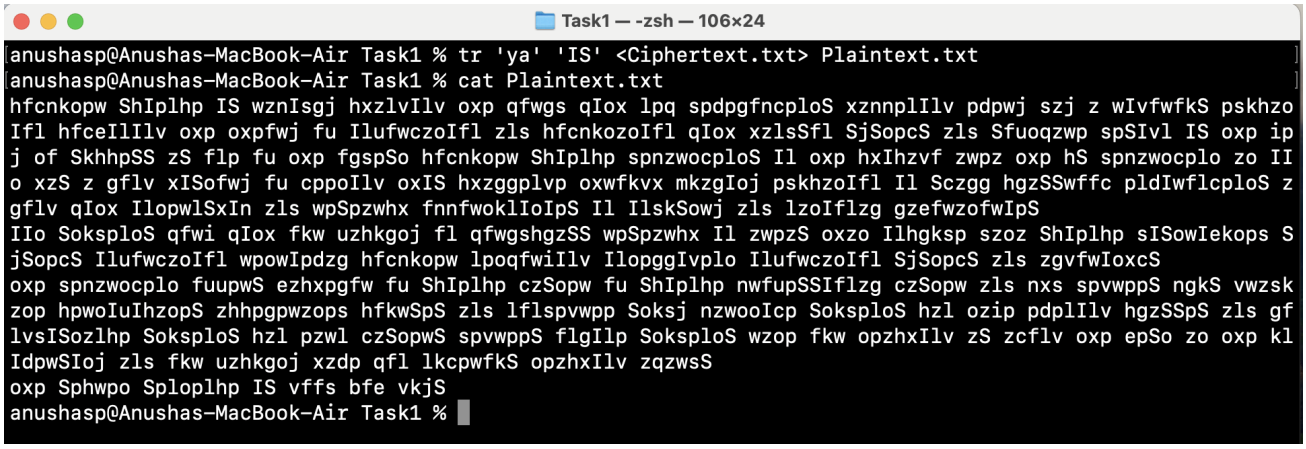
- Initially, I created a file named ‘Ciphertext.txt’ and copied the given ciphertext into this file.



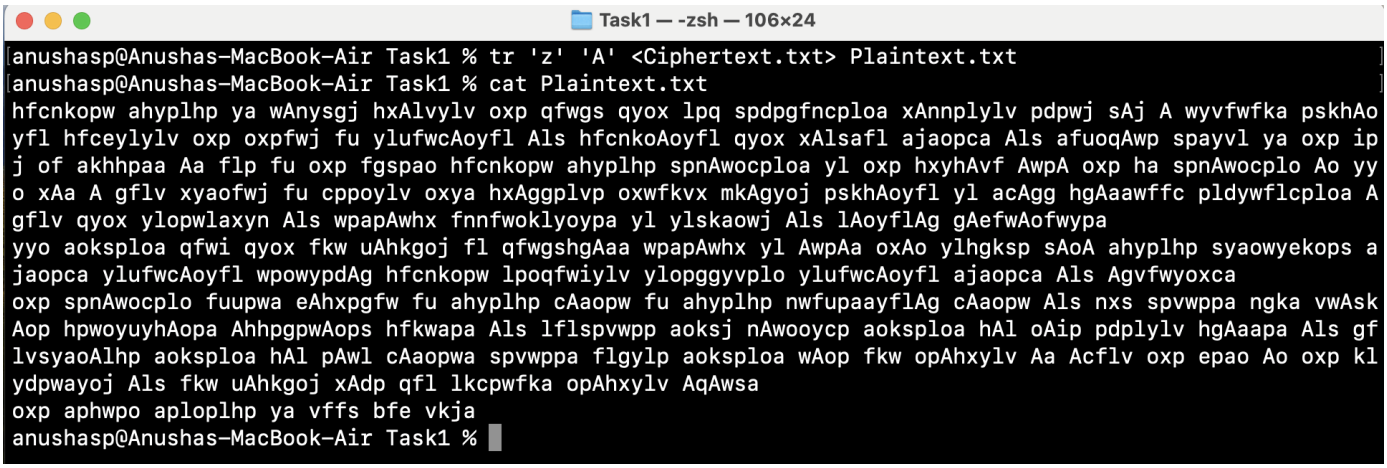

- Analyzing the most used English three letter words. The most used three letter word is ‘THE’, so replacing ‘oxp’ from ciphertext with ‘THE’ using the tr command and creating the output file as ‘Plaintext.txt’.



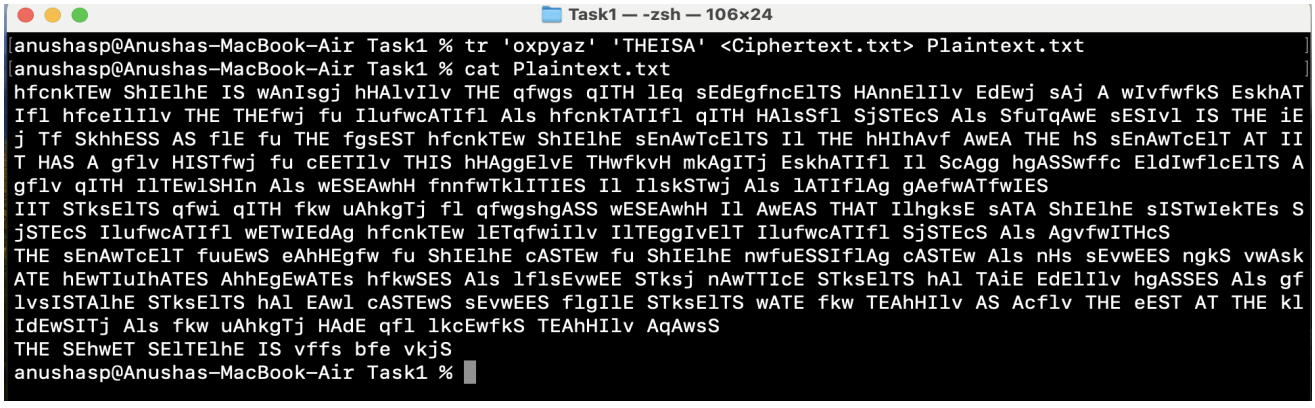

- Analyzing the most used English two letter words. The most used two letter word is ‘IS’, so replacing ‘ya’ from ciphertext with ‘IS’ using the tr command and creating the output file as ‘Plaintext.txt’.




- Analyzing the most used English one letter words. The most used one letter word is ‘A’, so replacing ‘z’ of cipher text with ‘A’.



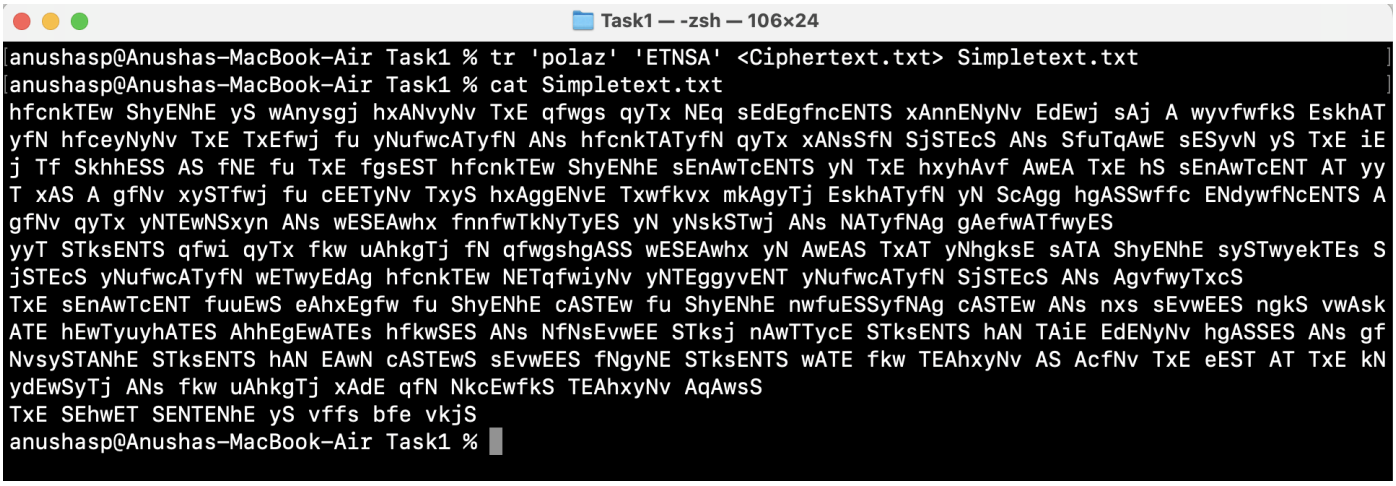

- Combining all the above, replacing ‘oxpyaz’ with ‘THEISA’.



- Using the [frequency analysis tool], analyze the frequency of letters present in the given ciphertext.

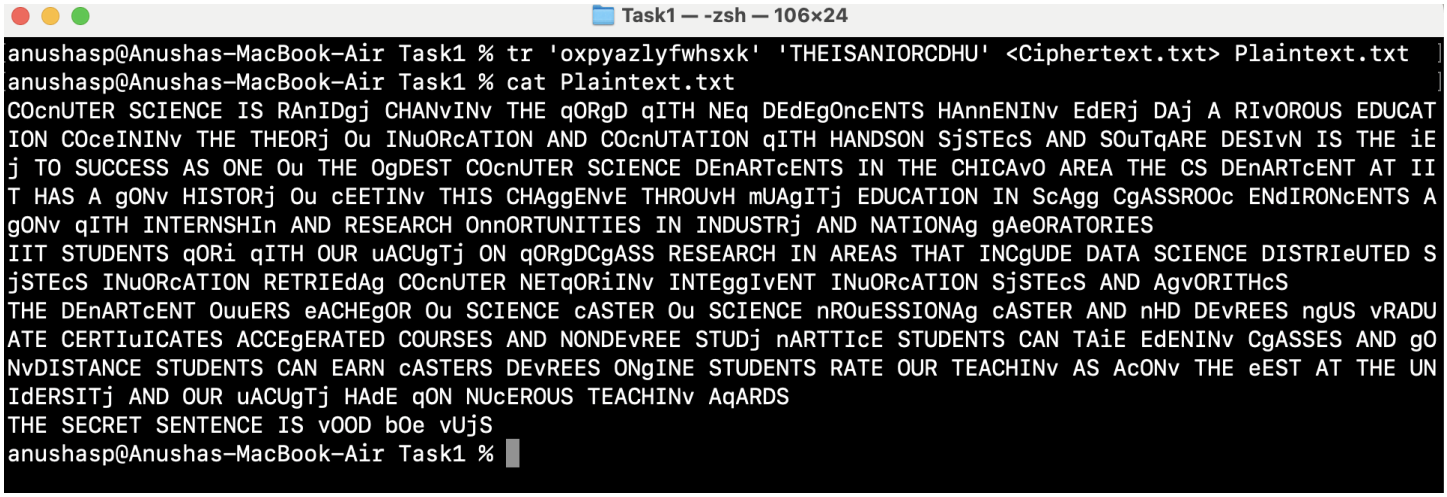
| P | O | L | A | Z | Y | F | W | H | S | X | K | G |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12.07 | 9.68 | 8.84 | 8.43 | 8.22 | 7.08 | 6.66 | 6.35 | 4.68 | 4.27 | 3.64 | 3.23 | 3.12 |

| C | V | N | J | U | Q | D | E | I | M | B |
|---|---|---|---|---|---|---|---|---|---|---|
| 2.71 | 2.71 | 1.87 | 1.66 | 1.56 | 1.25 | 0.73 | 0.62 | 0.42 | 0.1 | 0.1 |

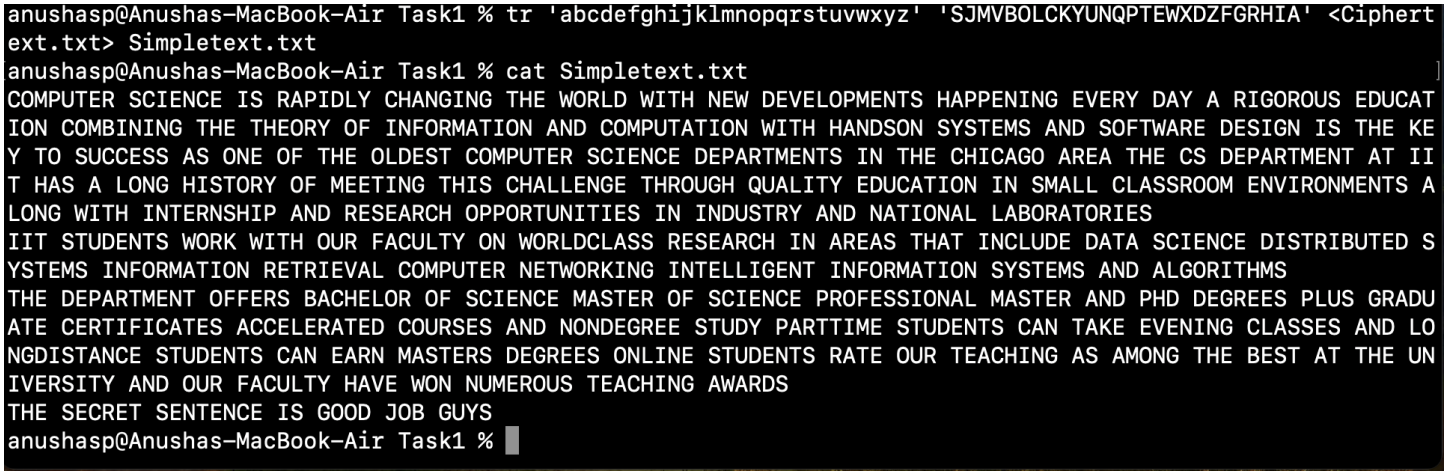



- Replacing the letters of ciphertext with plaintext based on observation:

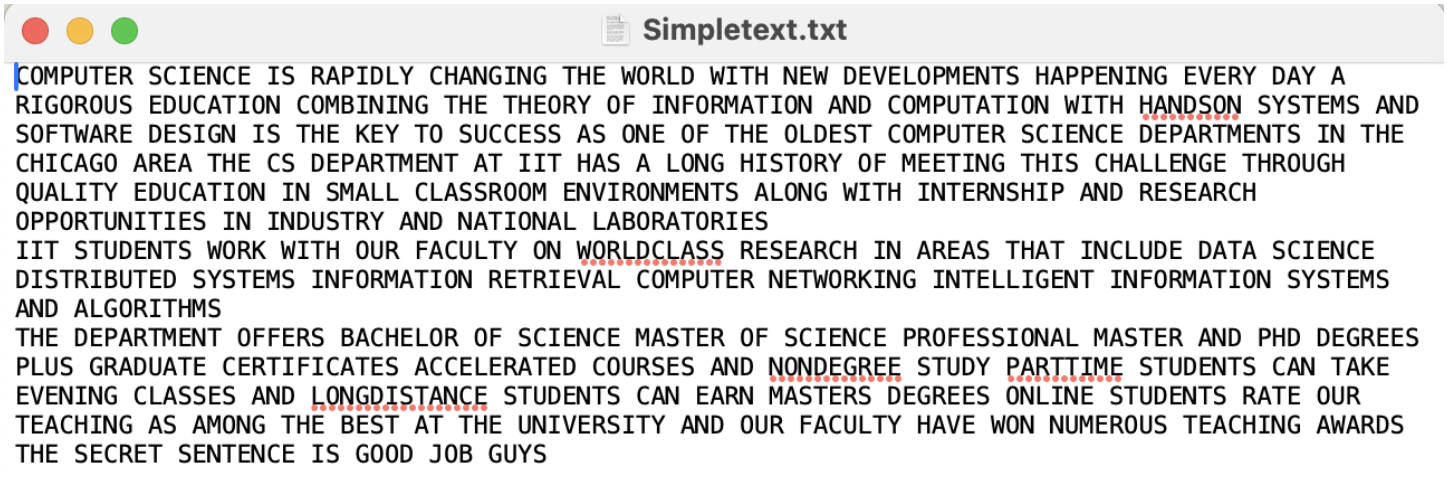
```
anushasp@Anushas-MacBook-Air Task1 % tr 'yfwhsxk' 'IORCDHU' <Ciphertext.txt> Simpletext.txt
anushasp@Anushas-MacBook-Air Task1 % cat Simpletext.txt
COcnUopR aCIplCp Ia RznIDgj CHzlvIlv oHp qORgD qIoH lpq DpdpgOncploa HznnplIlv pdpRj Dzj z RIvOROUa pDUCzo
IOl COceIlIlv oHp oHpORj Ou IluORczoIOl zlD COcnUozoIOl qIoH HzlDaOl ajaopca zlD aOuoqzRp DpaIvl Ia oHp ip
j oO aUCCpaa za Olp Ou oHp OgDpao COcnUopR aCIplCp DpnzRocploa Il oHp CHICzvO zRpz oHp Ca DpnzRocplo zo II
o Hza z gOlv HIaoORj Ou cppoIlv oHIa CHzggplvp oHROUvH mUzgIoj pDUCzoIOl Il aczgg CgzaaROOc pldIROlcploa z
gOlv qIoH IlopRlaHIn zlD RpapzRCH OnnORoUlIoIpa Il IlDUaoRj zlD lzoIOlzg gzeORzoORIpa
IIo aoUDploa qORi qIoH OUR uzCUgoj Ol qORgDCgzaa RpapzRCH Il zRpza oHzo IlCgUDp Dzoz aCIplCp DIaoRIeUopD a
jaopca IluORczoIOl RpoRIpdzg COcnUopR lpoqORiIlv IlopggIvplo IluORczoIOl ajaopca zlD zgvORIoHca
oHp DpnzRocplo OuupRa ezCHpgOR Ou aCIplCp czaopR Ou aCIplCp nROupaaIOlzg czaopR zlD nHD DpvRppa ngUa vRzDU
zop CpRoIuICzopa zCCpgpRzopD COURapa zlD lOlDpvRpp aoUDj nzRooIcp aoUDploa Czl ozip pdplIlv Cgzaapa zlD gO
lvDIaozlCp aoUDploa Czl pzRl czaopRa DpvRppa OlgIlp aoUDploa Rzop OUR opzCHIlv za zcOlv oHp epao zo oHp Ul
IdpRaIoj zlD OUR uzCUgoj Hzdp qOl lUcpROUa opzCHIlv zqzRDa
oHp apCRpo aploplCp Ia vOOD bOe vUja
anushasp@Anushas-MacBook-Air Task1 %
```




- Replacing all the alphabets at once based on observations:



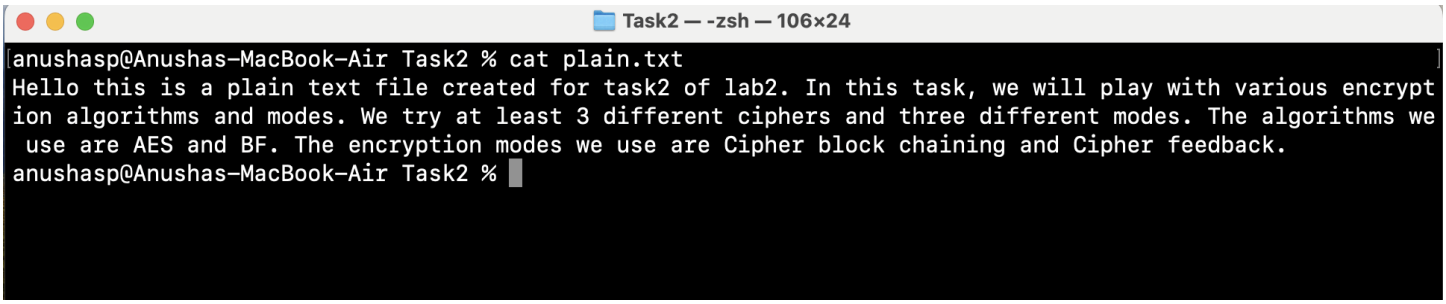

Decrypted plain text:



## Task 2: Encryption using Different Ciphers and Modes

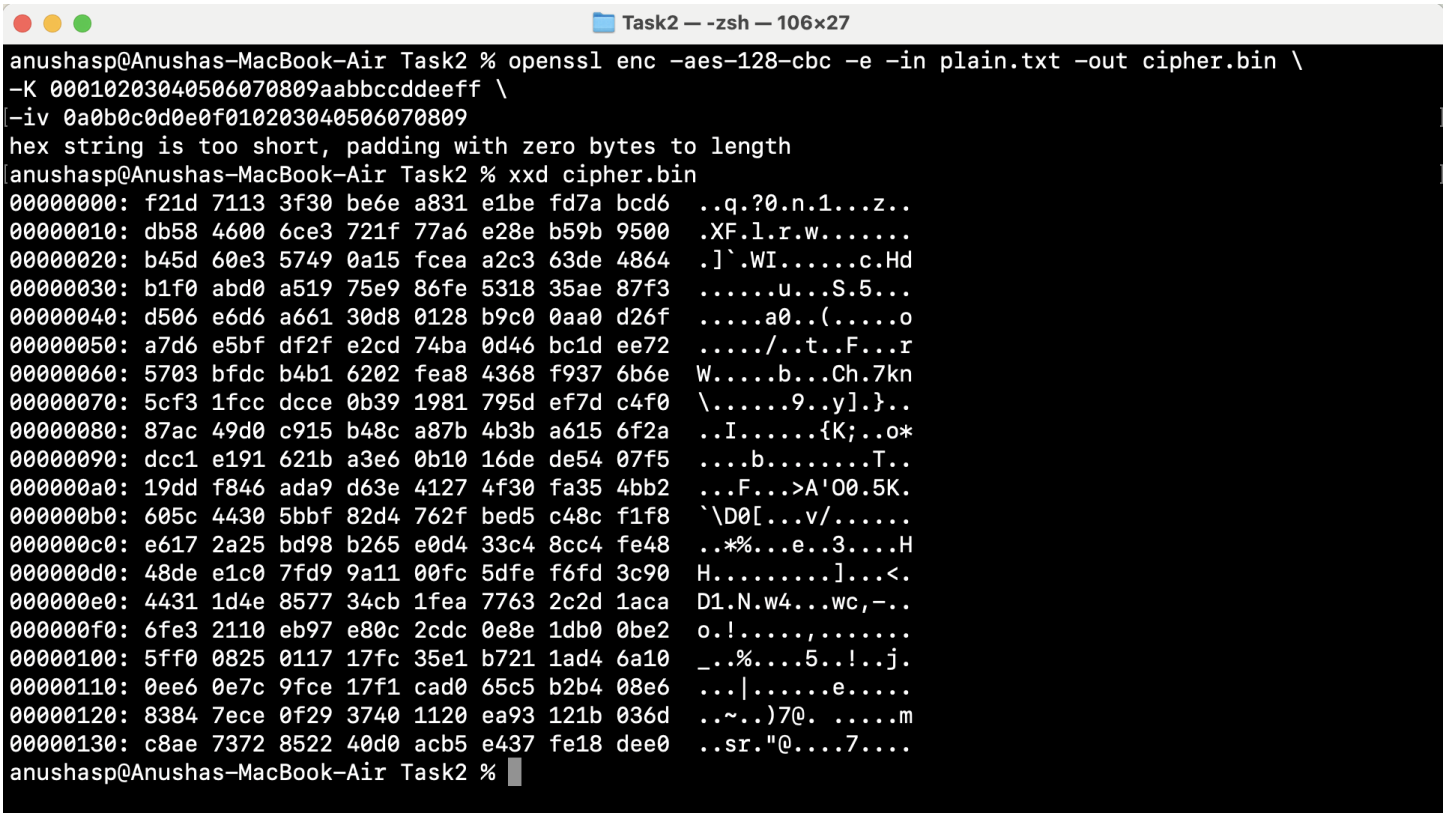
**Goal:** The goal of this task is to encrypt the text using various encryption algorithms and modes.

Step1: Generate a plain.txt file and write some custom message into it as shown below:



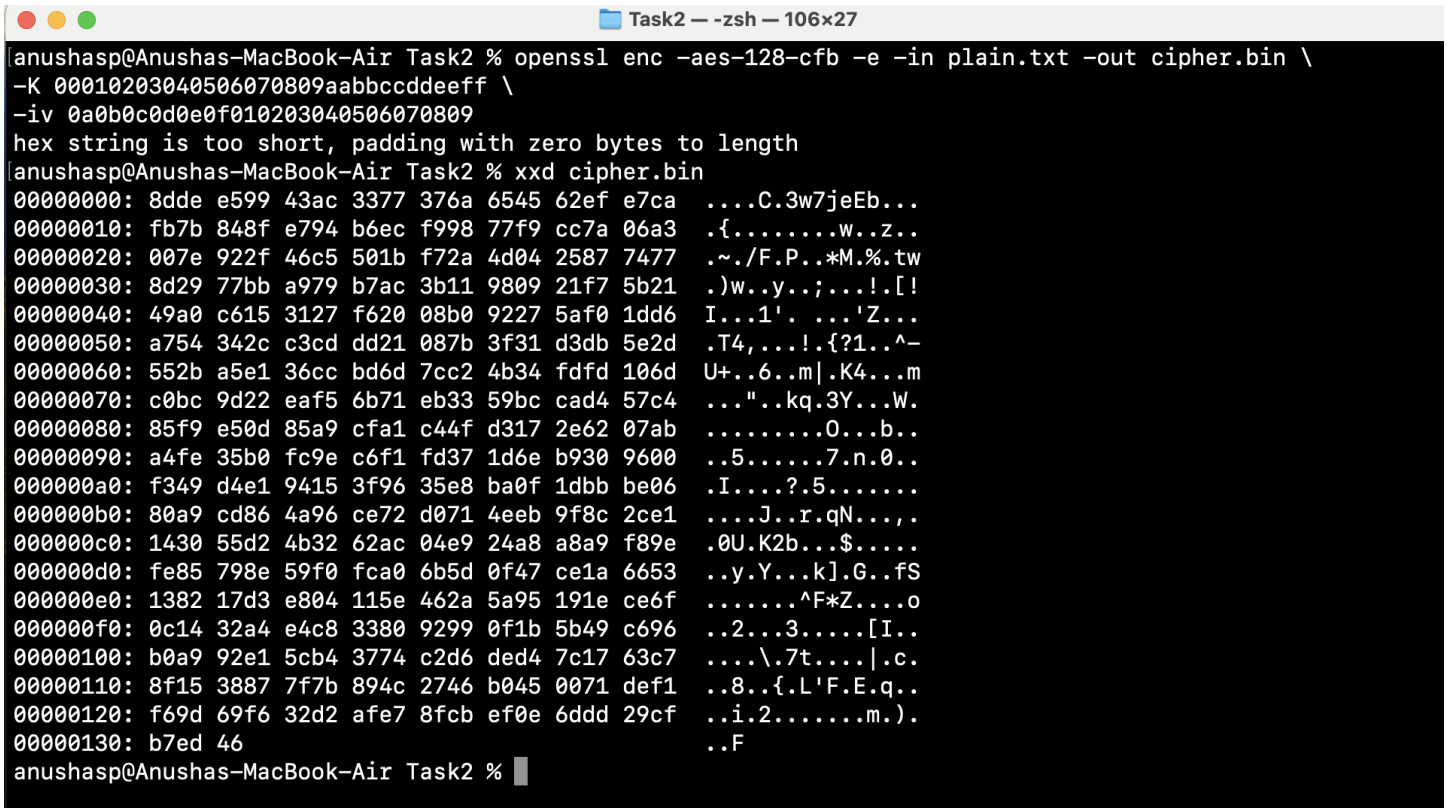

### 2.1 Using cipher type AES-128-CBC :

- Using openssl command to encrypt the given plain text using AES encryption with CBC mode. CBC stands for Cipher block chaining.
- The input file is 'plain.txt' and the output is generated in the form of .bin file which can be viewed by using xxd command as shown below.



### 2.2 Using cipher type AES-128-CFB :

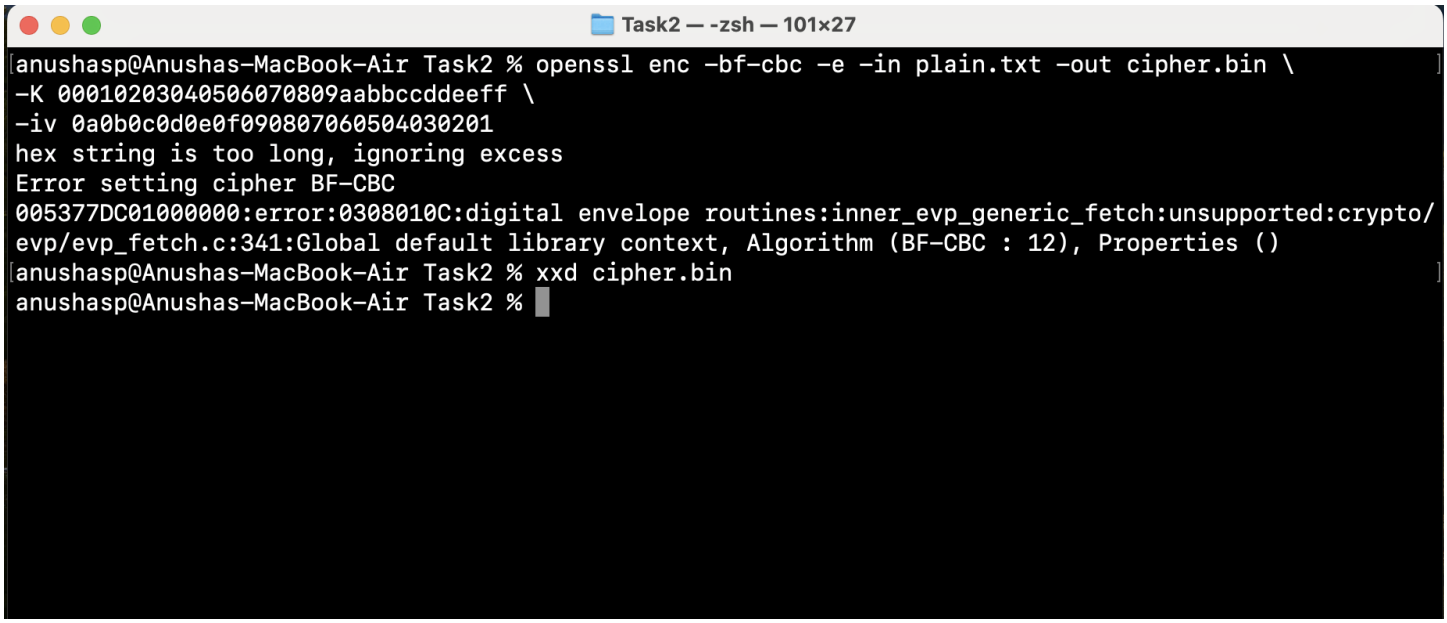
- Using openssl command to encrypt the given plain text using AES encryption with CFB mode. CFB stands for Cipher FeedBack.
- It is most identical to CBC encryption but is performed in reverse.
- The input file is ‘plain.txt’ and the output is generated in the form of .bin file which can be viewed by using xxd command as shown below.




### 2.3 Using cipher type BF-CBC :

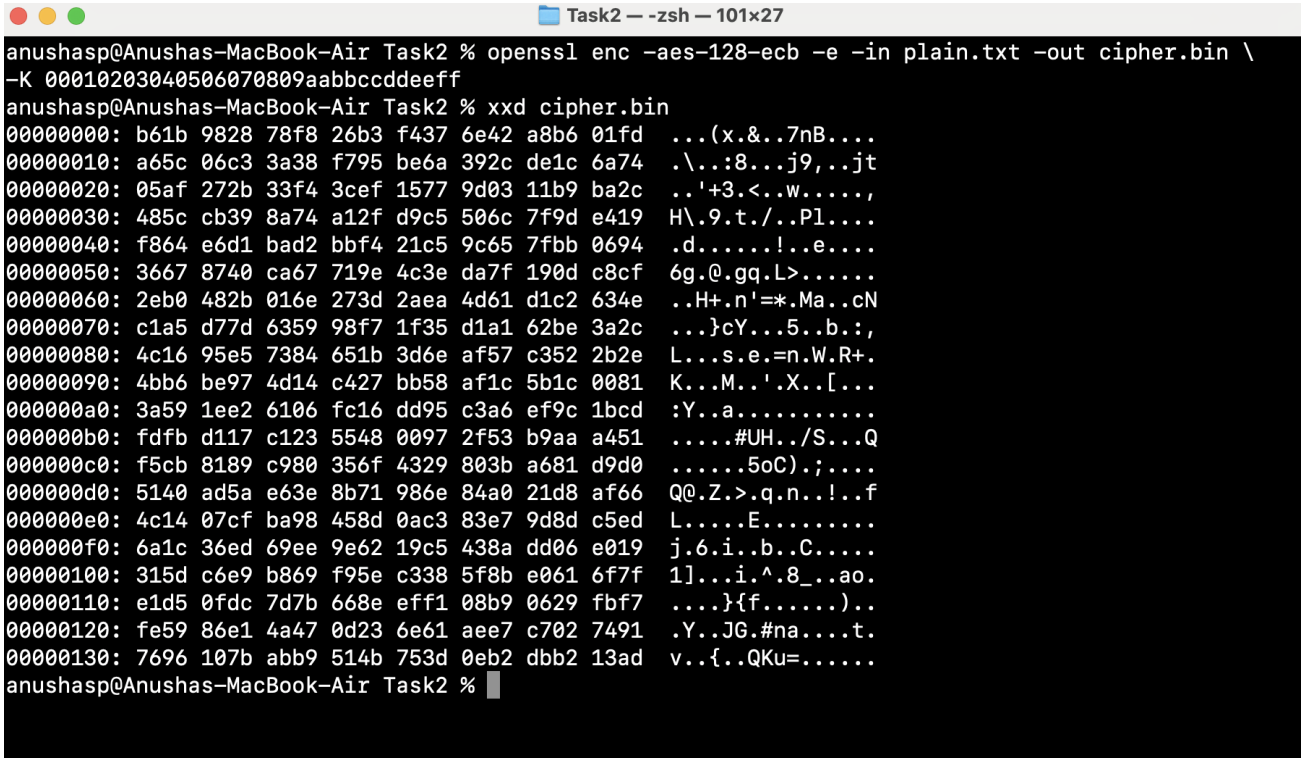
- Using openssl command to encrypt the given plain text using BF encryption with CBC mode.
- The input file is ‘plain.txt’ and the output is generated in the form of a .bin file which can be viewed by using xxd command as shown below.

*Note: -bf-cbc was not working on MAC and was throwing ‘Unsupported crypto’ error, attached is the screenshot for your reference. So I tried two other encryption modes.



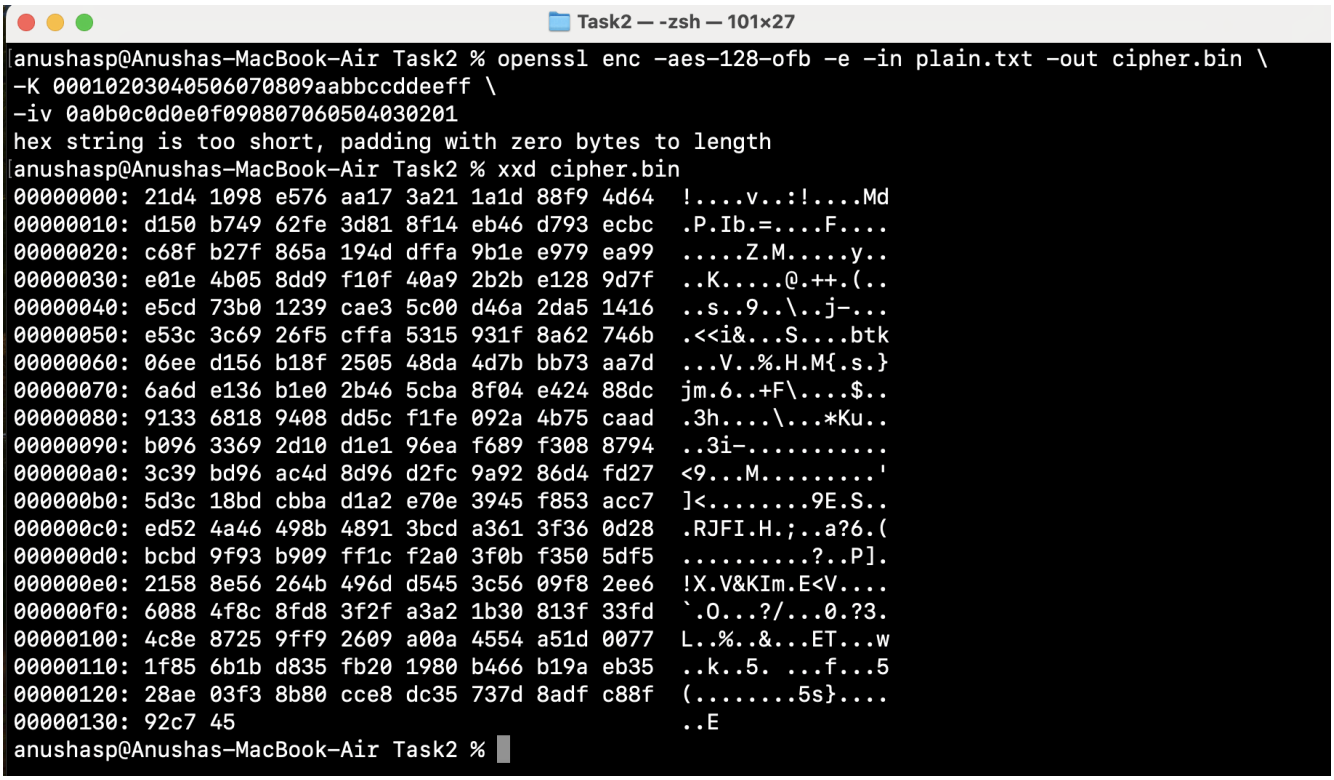

### 2.4 Using cipher type AES-128-ECB :

- Using openssl command to encrypt the given plain text using AES encryption with ECB mode.
- The input file is ‘plain.txt’ and the output is generated in the form of a .bin file which can be viewed by using xxd command as shown below.



### 2.5 Using cipher type AES-128-OFB :

- Using openssl command to encrypt the given plain text using AES encryption with OFB mode.
- The input file is ‘plain.txt’ and the output is generated in the form of a .bin file which can be viewed by using xxd command as shown below.



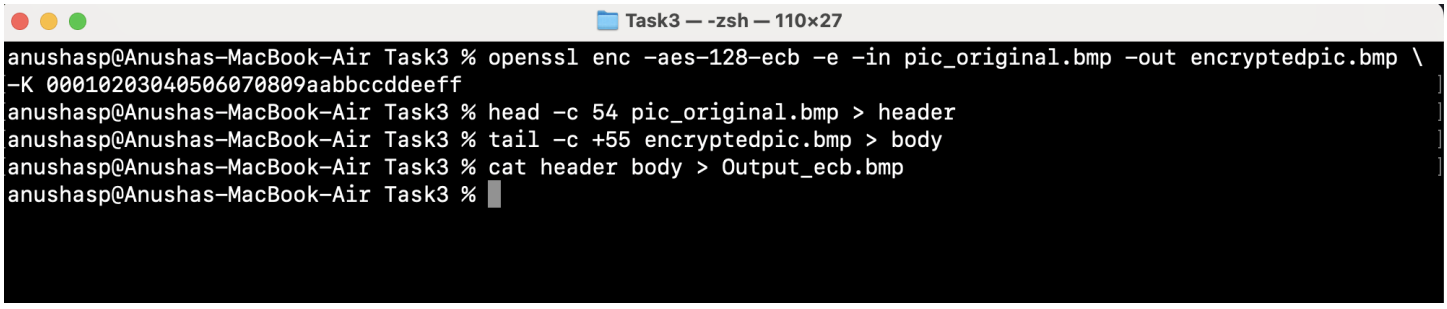
**Task 3: Encryption Mode: ECB vs CBC**

**Goal:** The goal of this task is to encrypt the given picture 'pic_original.bmp' using ECB and CBC modes.

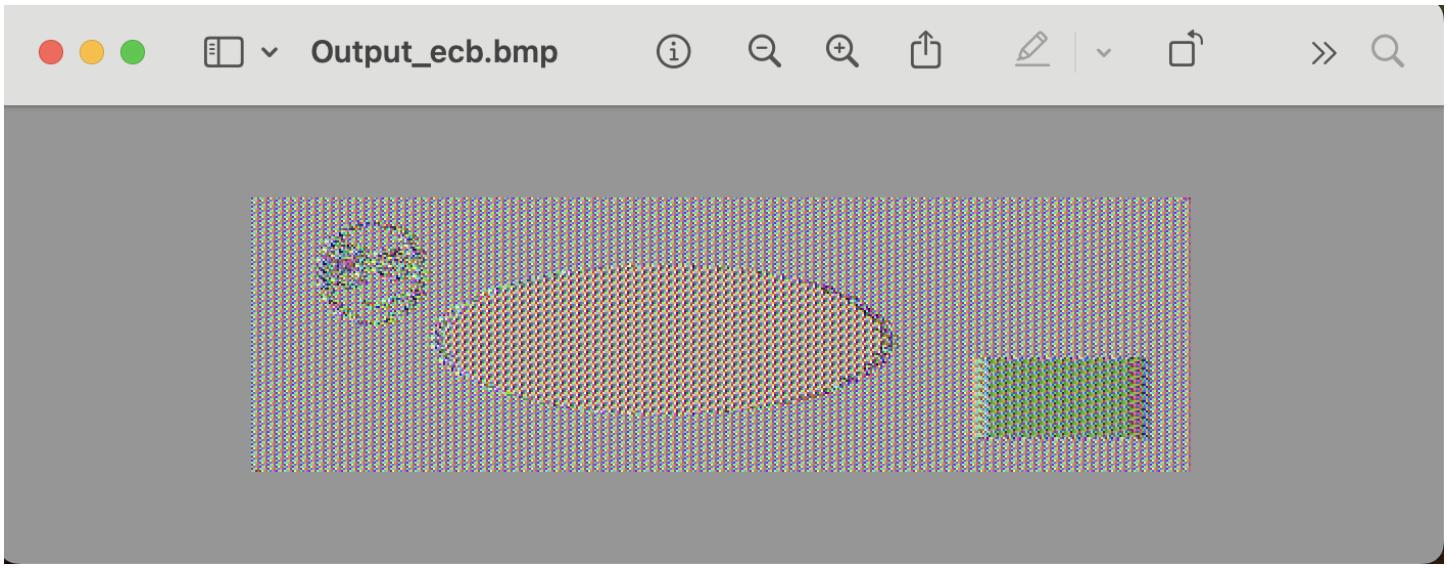
Given input picture [pic_original.bmp]:

**3.1 Using ECB Mode:**

- ECB stands for Electronic Code Book. In ECB Mode, each block is treated independently and the same length plain text blocks produce same length ciphertext blocks.
- We use openssl command to encrypt the given pic_orignal.bmp in ecb mode and the output is stored in the form of encryptedpic.bmp file.
- To view this output: we need to create head and tail and store it in the Output_ecb.bmp file.

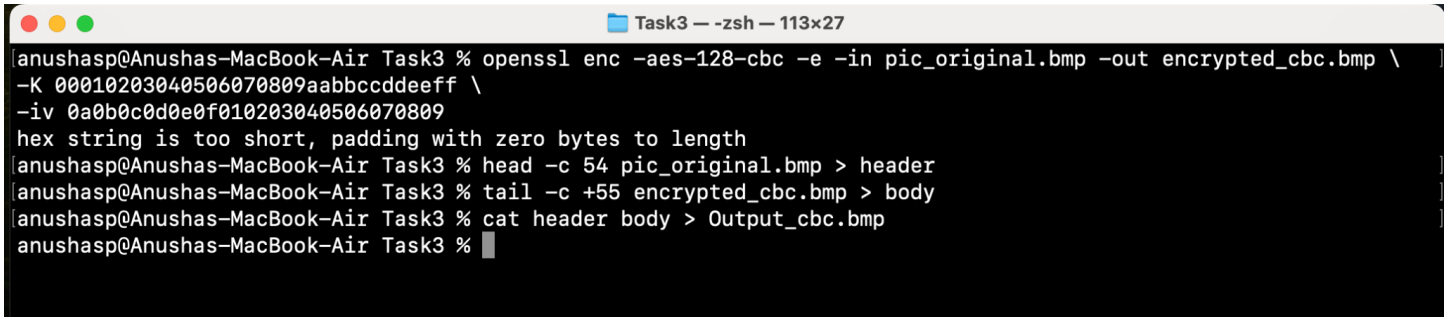


The Output_ecb.bmp file looks as below:

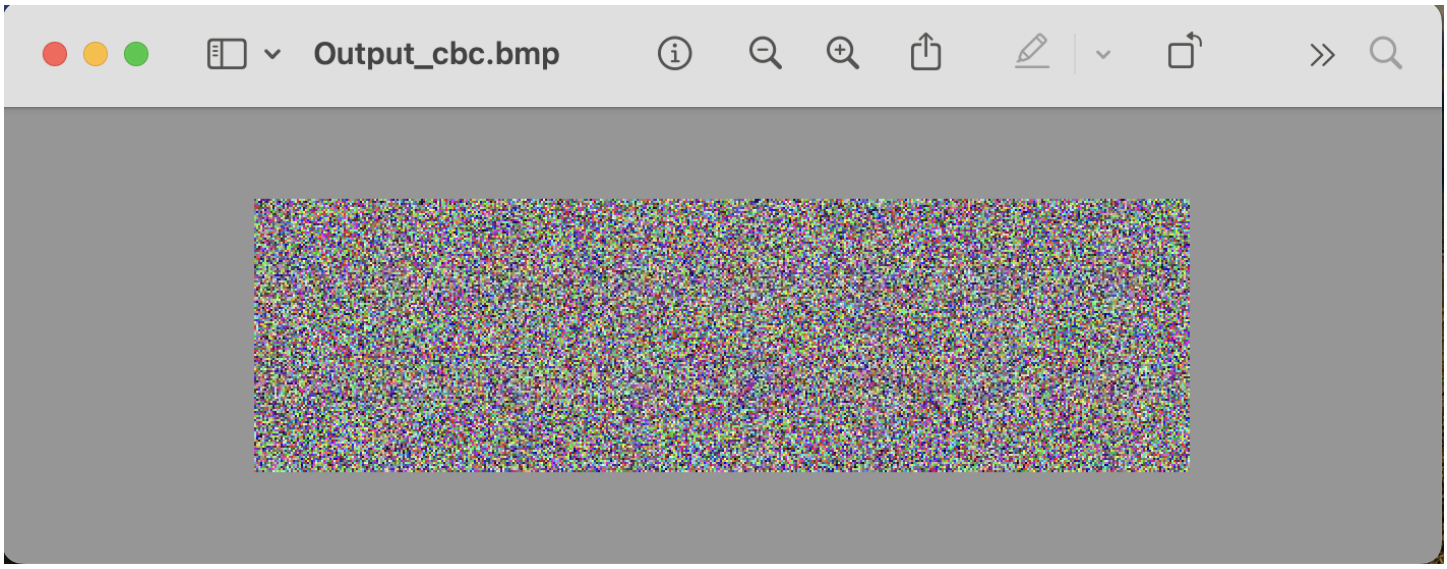



### 3.2 Using CBC Mode:

- CBC stands for Cipher block chaining and the Ciphertext C2 is produced using C1 and so on. The initial ciphertext is produced using an initialization vector IV. So, even for the same messages, the ciphertext produced would be different.
- We use openssl command to encrypt the given pic_orignal.bmp in cbc mode and the output is stored in the form of encryptedpic.bmp file.
- To view this output: we need to create head and tail and store it in the Output_cbc.bmp file.

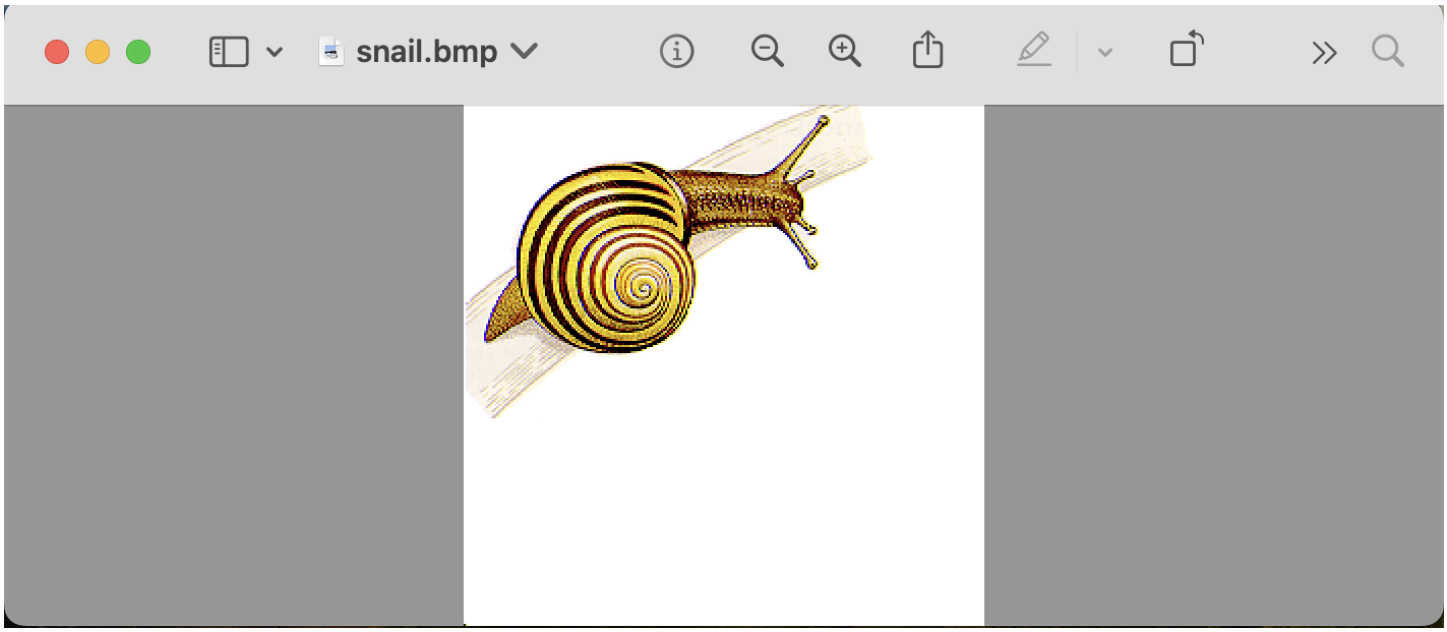


The Output_ecb.bmp file looks as below:




**3.3 Testing with the picture of our choice:**

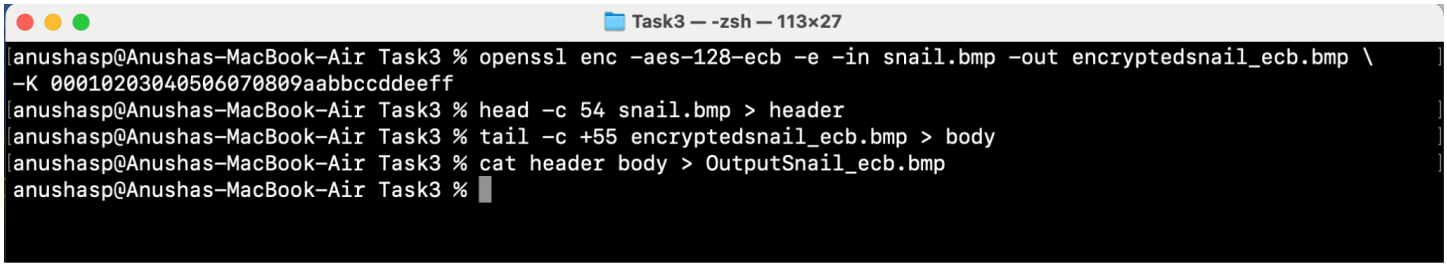
Let us consider the below picture tiger.bmp and let's try to encrypt this using ECB and CBC mode as shown above. [Input: snail.bmp]



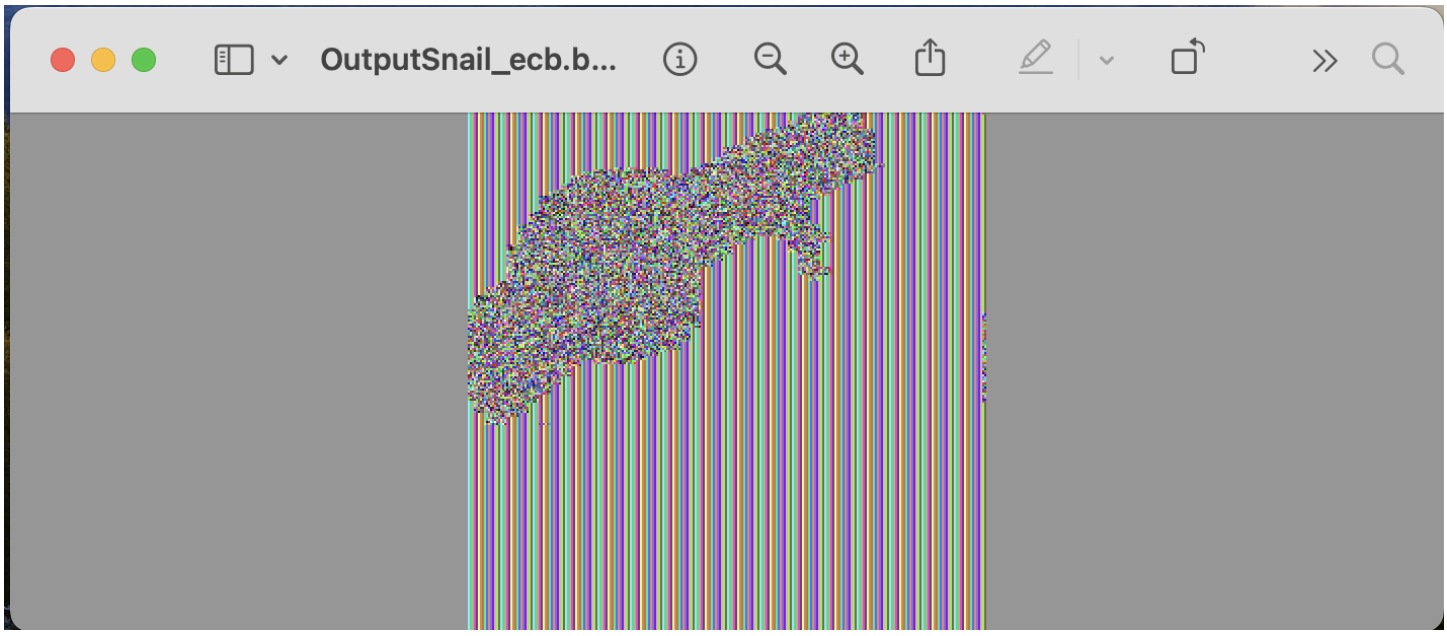
### 3.3.1 Using ECB Mode:

**Steps Performed:**

- Use openssl command with AES algorithm in ECB mode to generate an encrypted bmp file. As this is not viewable, we use head and tail and output the file.

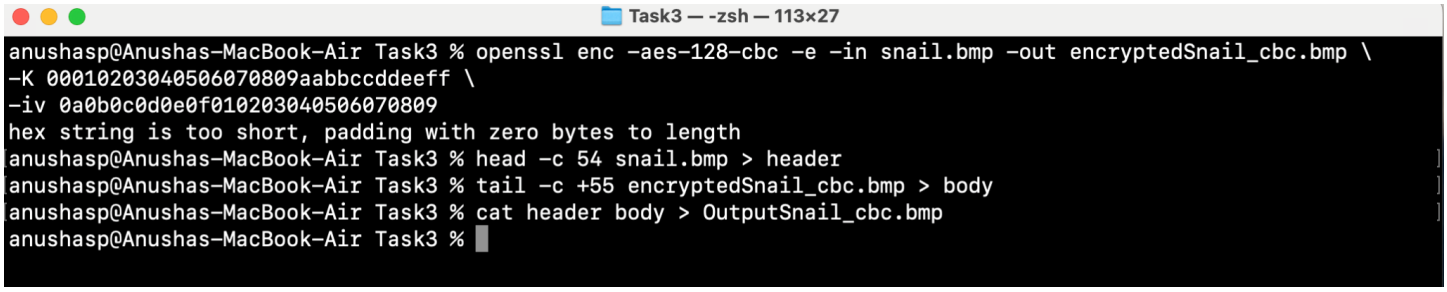



The output file OutputSnail_ecb.bmp is:

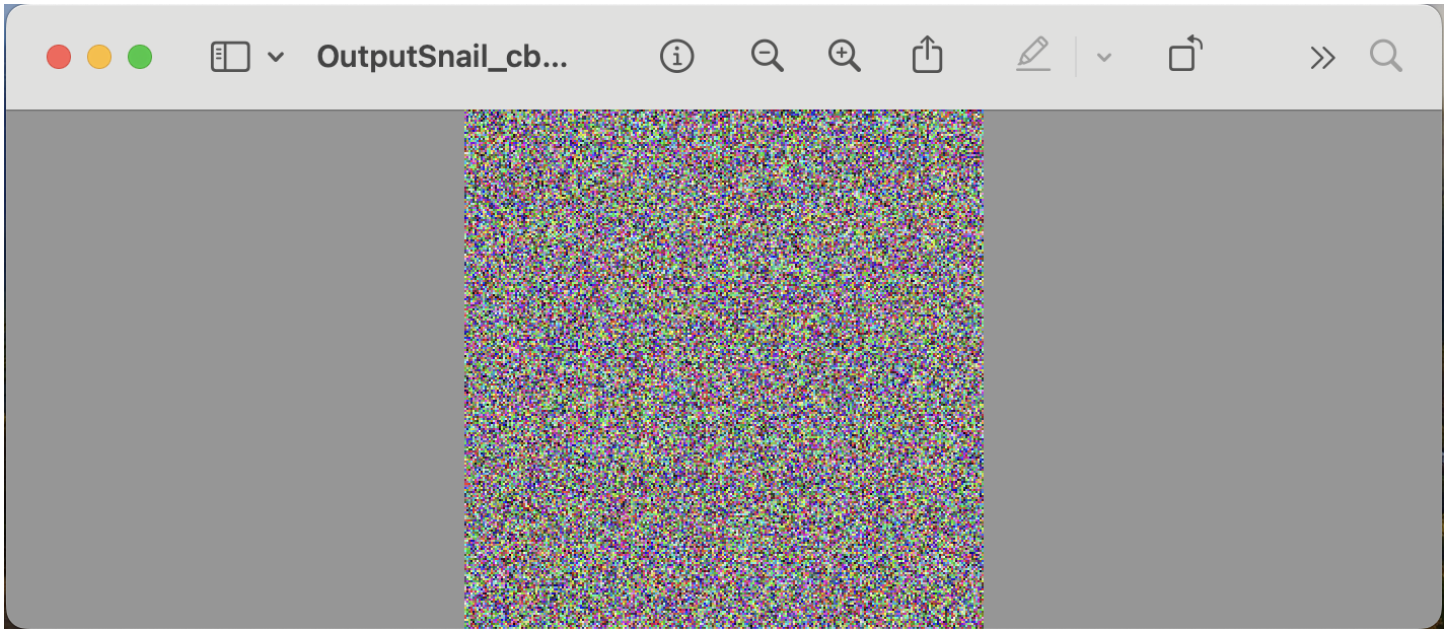


### 3.3.2 Using CBC Mode:

- Use openssl command with AES algorithm in CBC mode to generate an encrypted bmp file. As this is not viewable, we use head and tail and output the file. Here, we need an add an IV in the command.




The output file OutputSnail_cbc.bmp is:

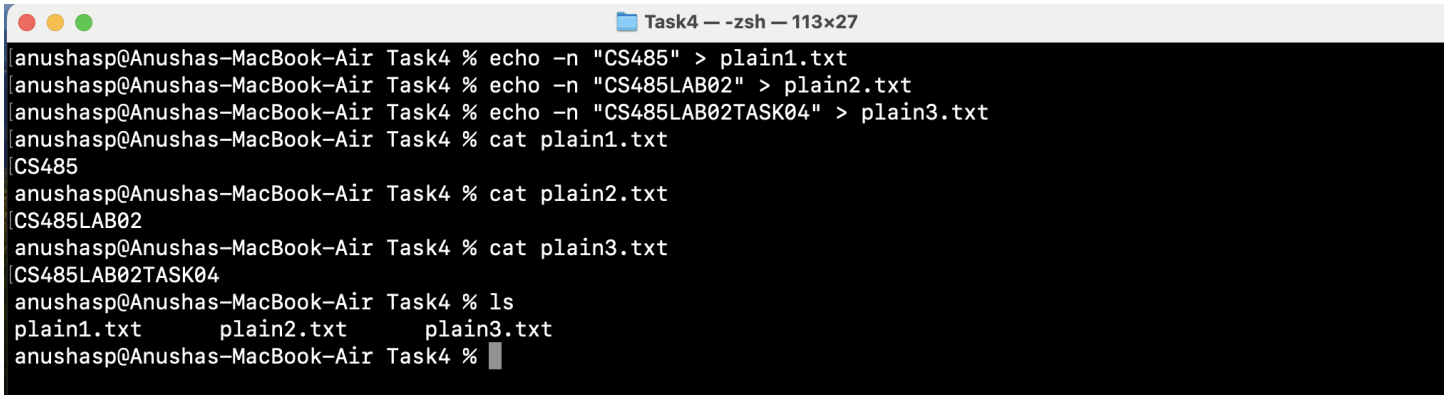


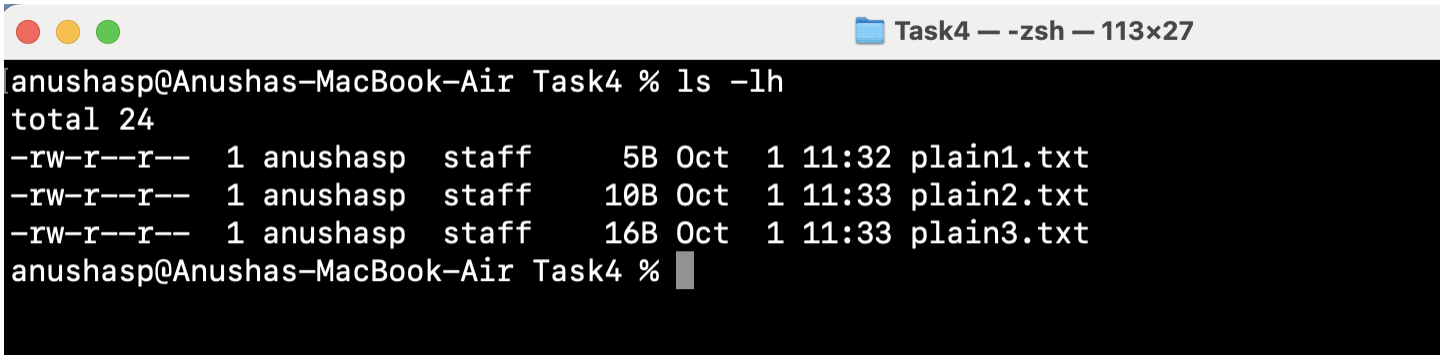
**Task 4: Padding**

**Goal:** The goal of this task is to understand the concept of padding using various encryption modes. When the size of a plaintext is not a multiple of the block size, padding may be required. All the block ciphers normally use PKCS#5 padding, which is known as standard block padding.

Steps:

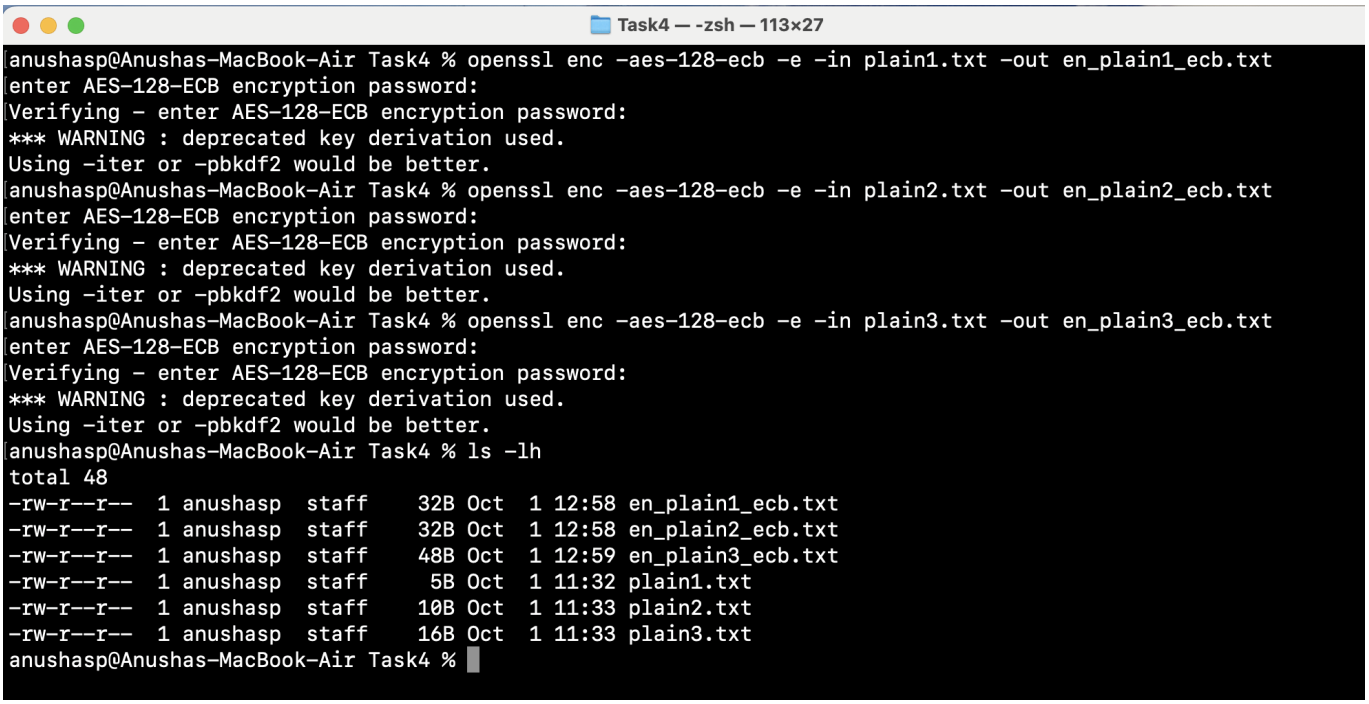
1. Create three files plain1.txt, plain2.txt and plain3.txt with 5, 10 and 16 bytes respectively.



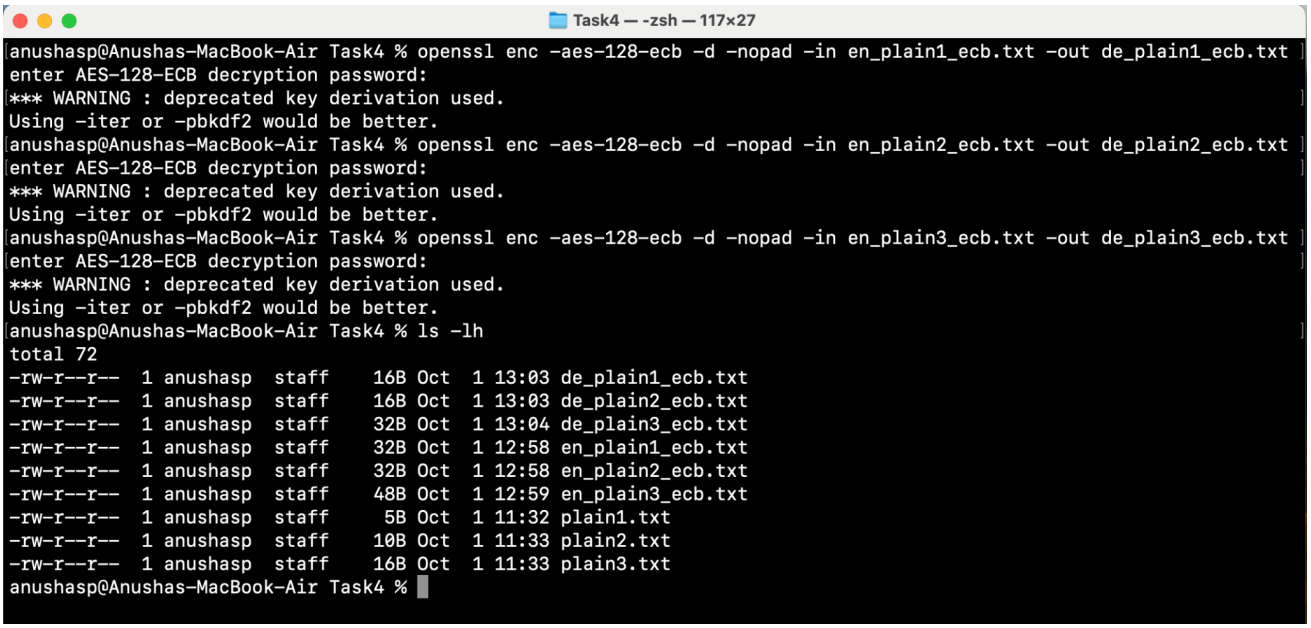



### 4.1 Using ECB Mode:

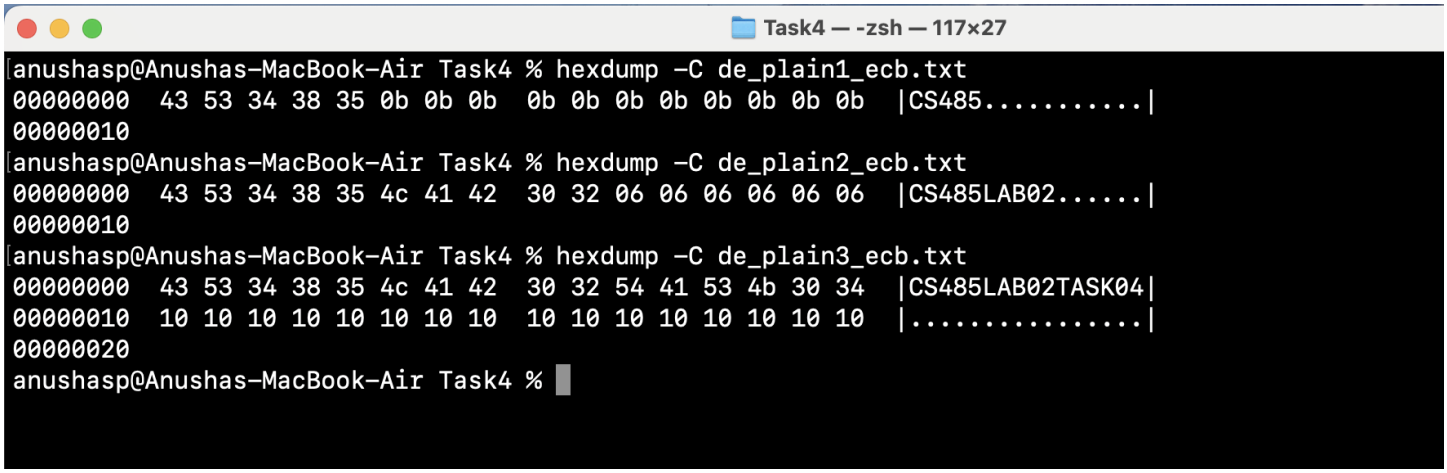
- We will encrypt the files using ECB mode and compare the file sizes.
- The first file plain1.txt of 5B length is now converted into 32B and the second file and third file plain2.txt, plain3.txt are converted into 32B and 48B respectively.




- Now let us decrypt and check the padding, for this we use nopad in the openssl command.



The decrypted files content is as below:




**Analysis [EBC]:**

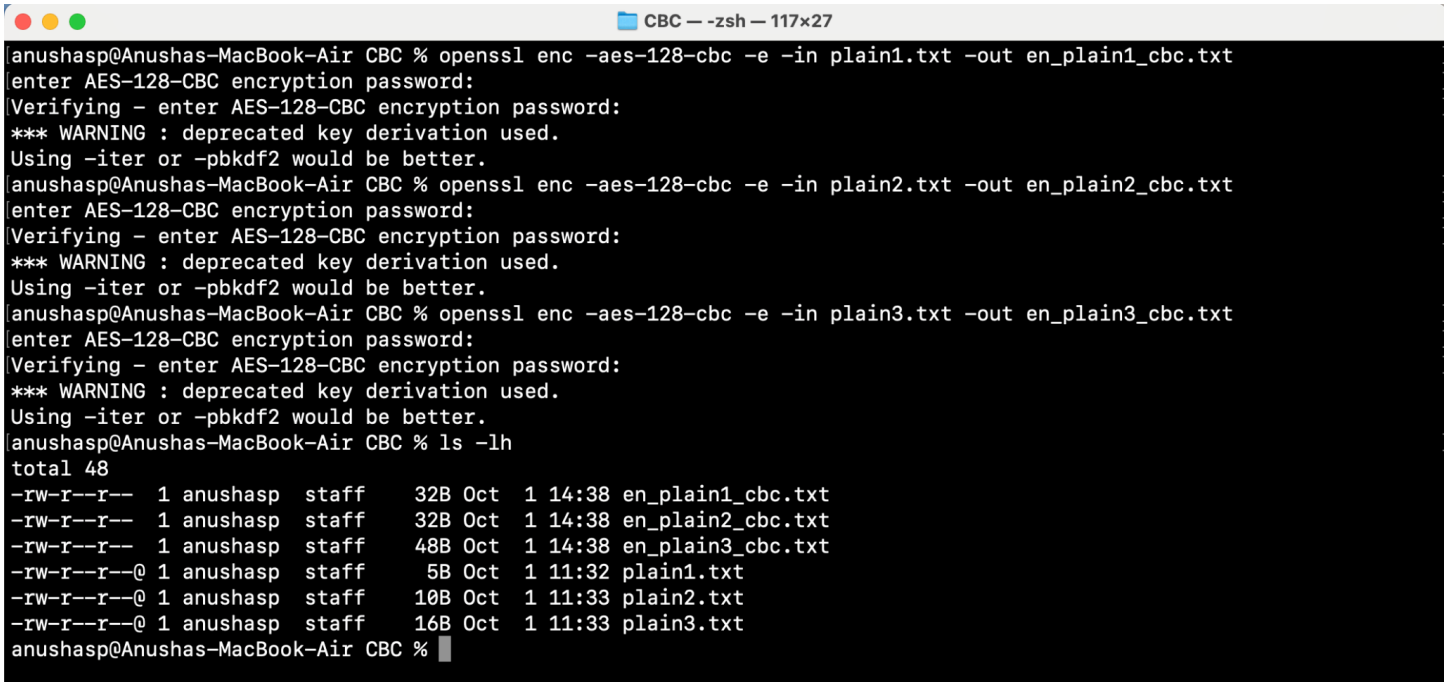
| File Name | Original File Size | Encrypted File Size | Decrypted File Size |
|---|---|---|---|
| plain1.txt | 5 Bytes | 32B | 16B |
| plain2.txt | 10 Bytes | 32B | 16B |
| plain3.txt | 16 Bytes | 48B | 32B |

**4.2 Using CBC Mode:**

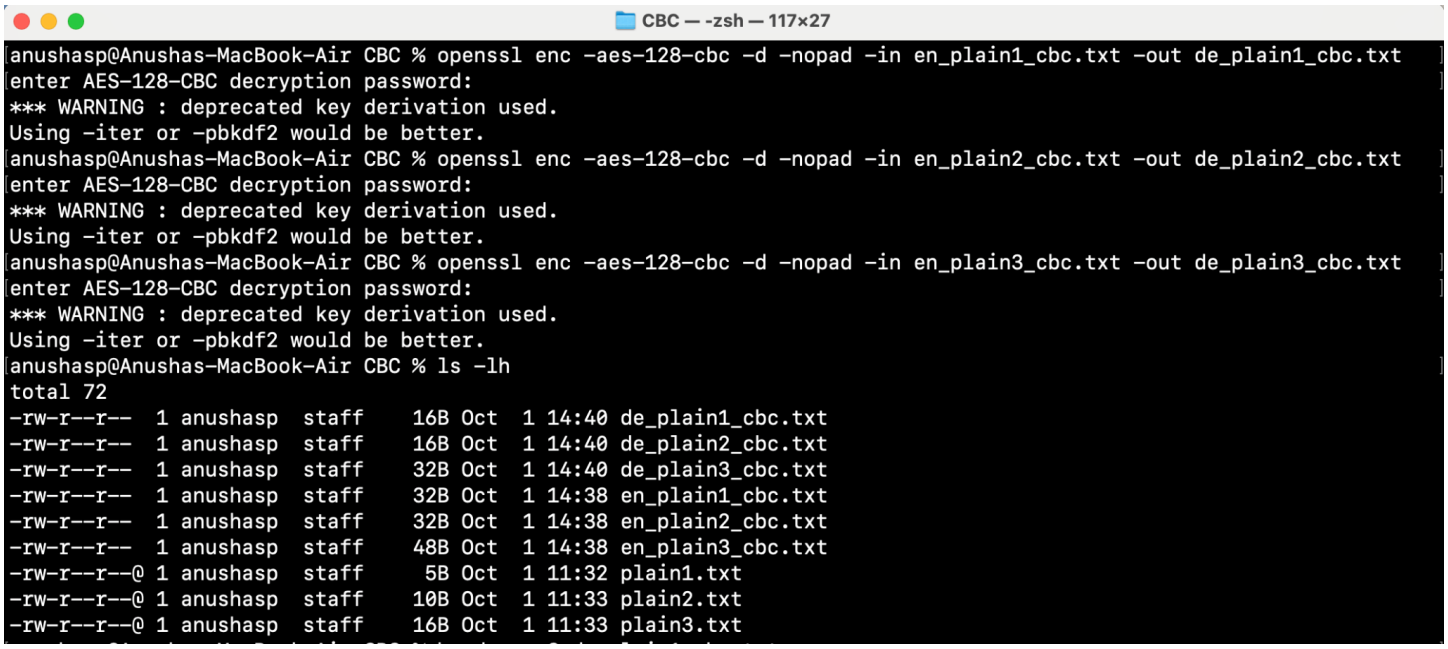
- We will encrypt the files using CBC mode and compare the file sizes.
- The first file plain1.txt of 5B length is now converted into 32B and the second file and third file plain2.txt, plain3.txt are converted into 32B and 48B respectively.

**Analysis [CBC]:**

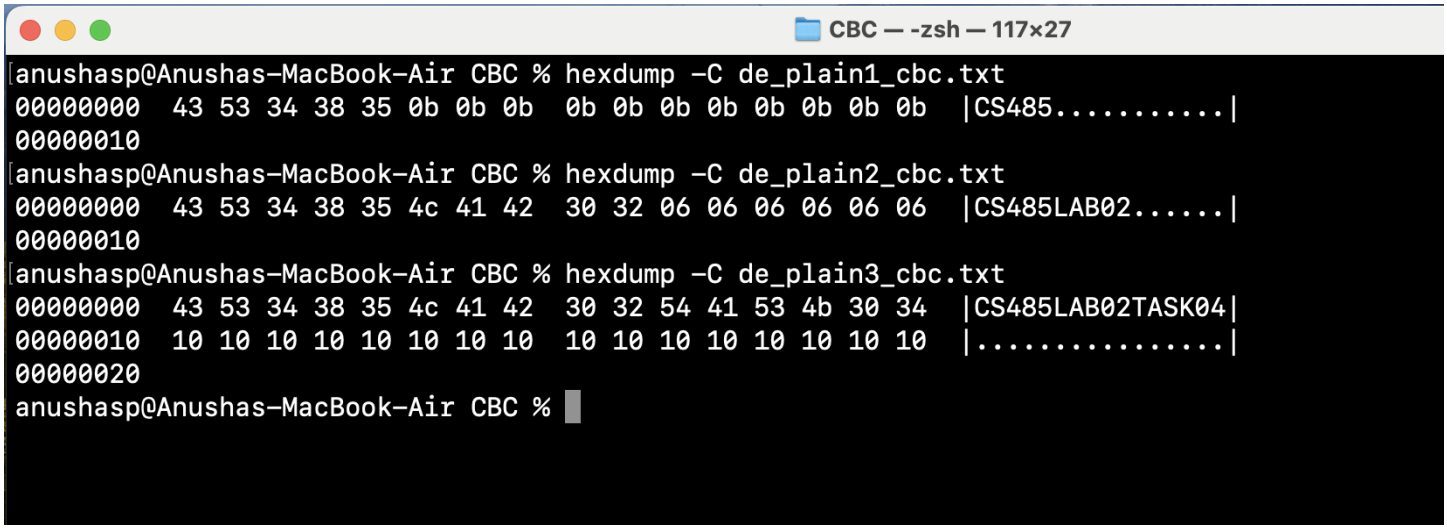
| File Name | Original File Size | Encrypted File Size | Decrypted File Size |
|---|---|---|---|
| plain1.txt | 5 Bytes | 32B | 16B |
| plain2.txt | 10 Bytes | 32B | 16B |
| plain3.txt | 16 Bytes | 48B | 32B |




- Now let us decrypt and check the padding, for this we use no pad in the openssl command.

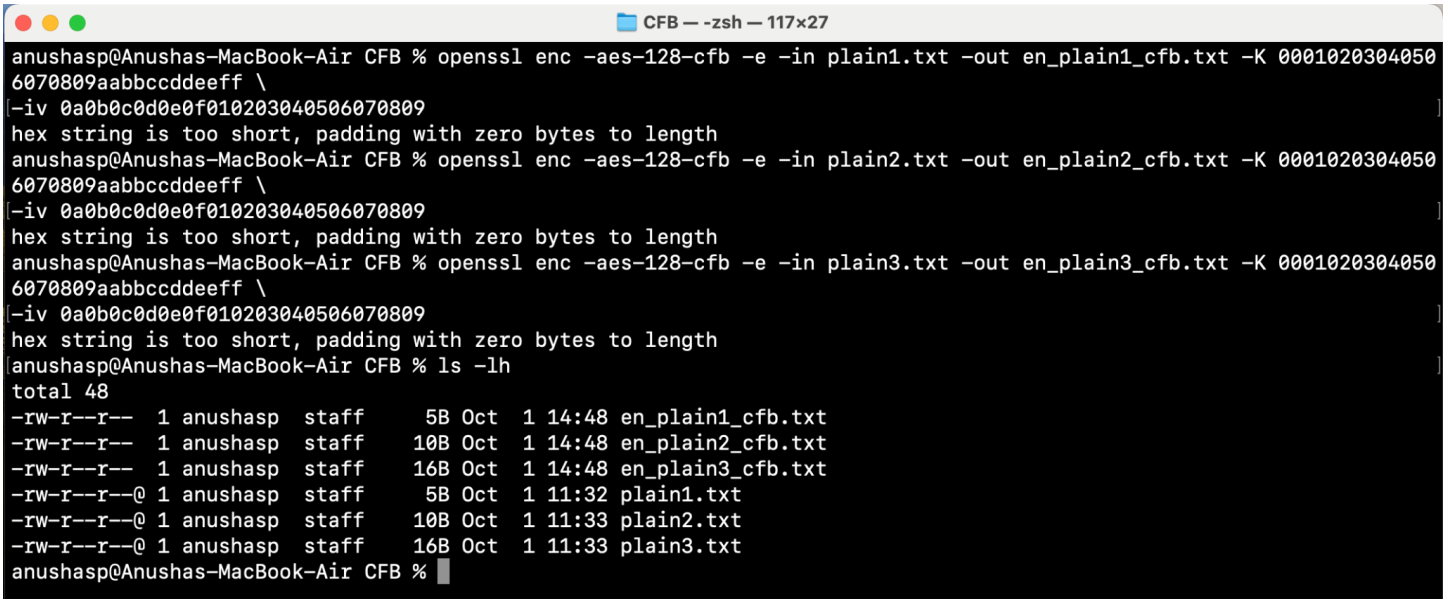


The decrypted files content is as below:

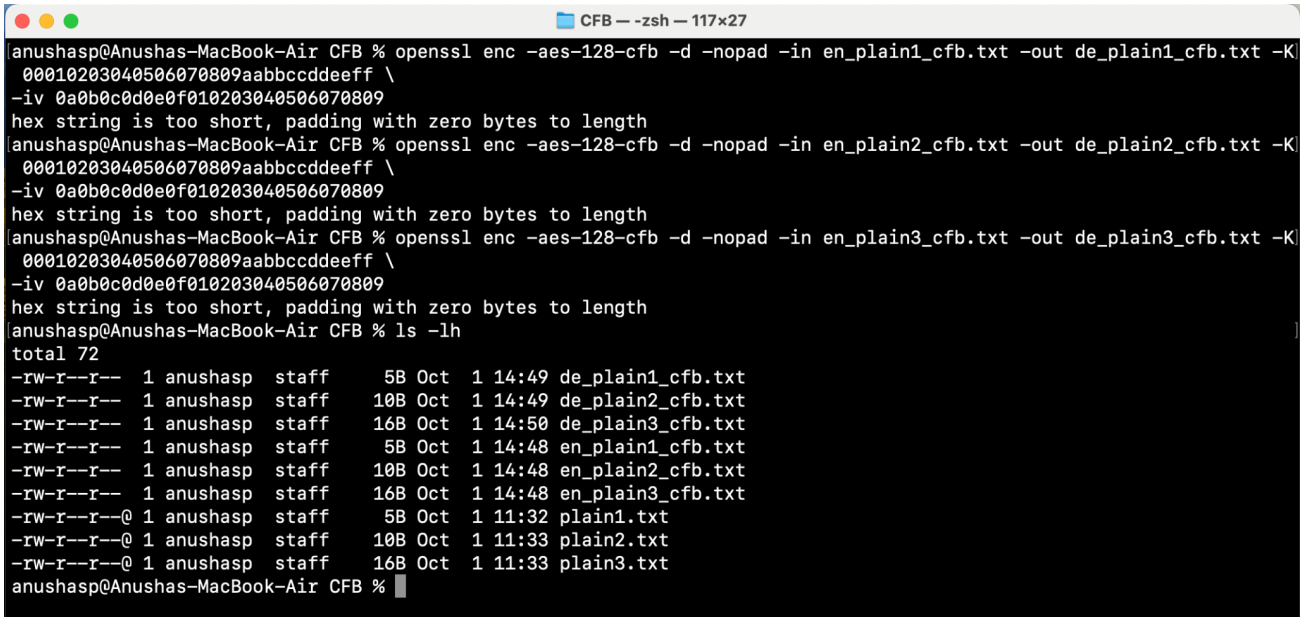



### 4.3 Using CFB Mode:

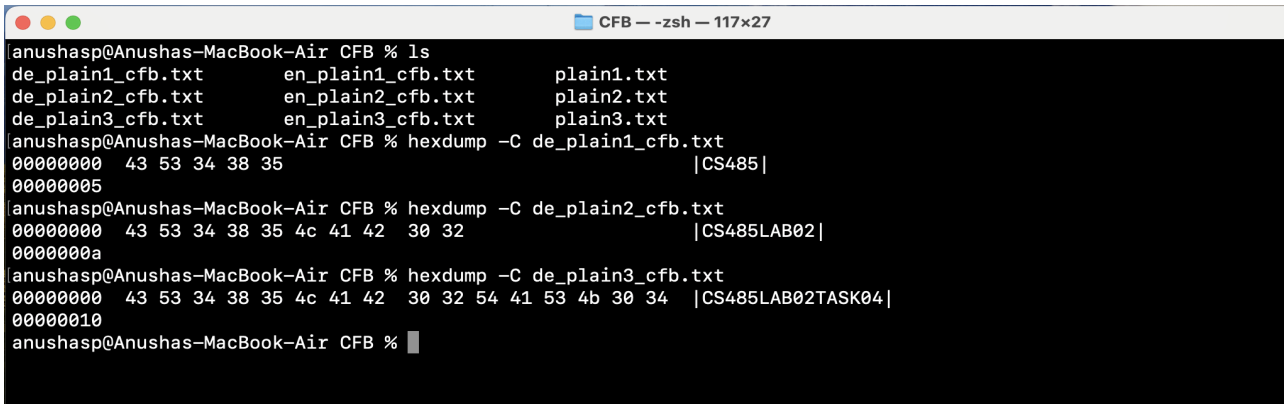
- We will encrypt the files using CFB mode and compare the file sizes.
- After encryption, there is no change in their file sizes as shown below, so there was NO padding.



- Now let us decrypt and check the padding, for this we use no pad in the openssl command.




The decrypted files content is as below:

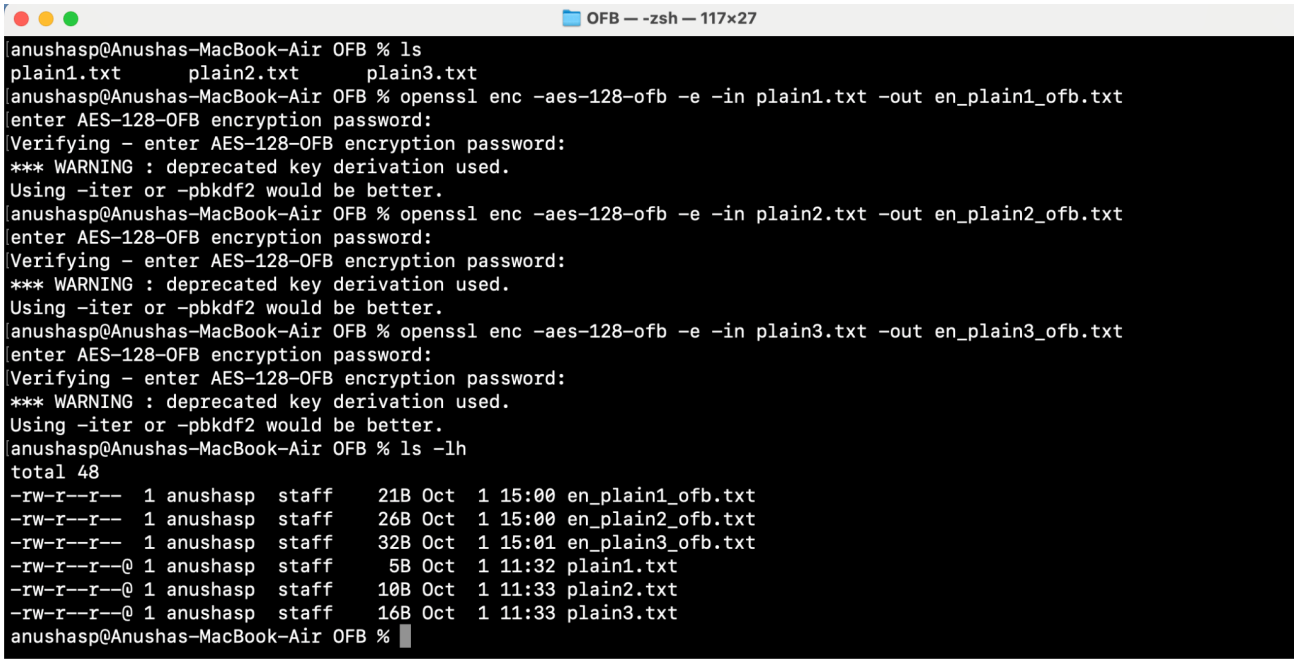



**Analysis [CBC]:**

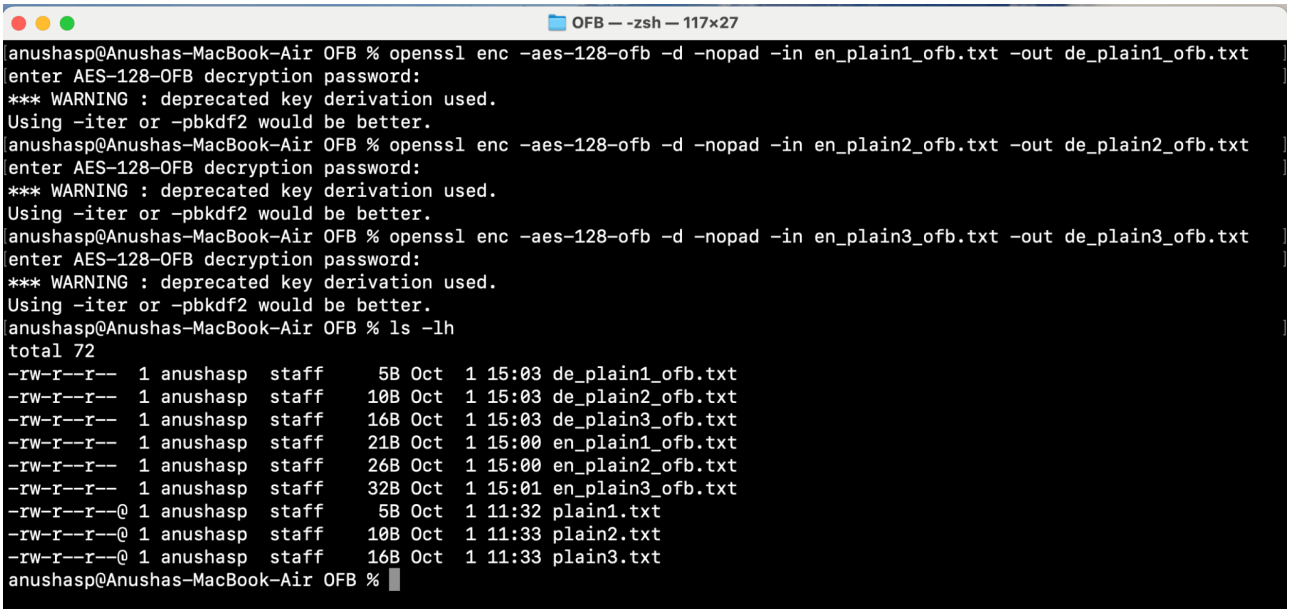
| File Name | Original File Size | Encrypted File Size | Decrypted File Size |
|---|---|---|---|
| plain1.txt | 5 Bytes | 5 Bytes | 5 Bytes |
| plain2.txt | 10 Bytes | 10 Bytes | 10 Bytes |
| plain3.txt | 16 Bytes | 16 Bytes | 16 Bytes |

### 4.4 Using OFB Mode:

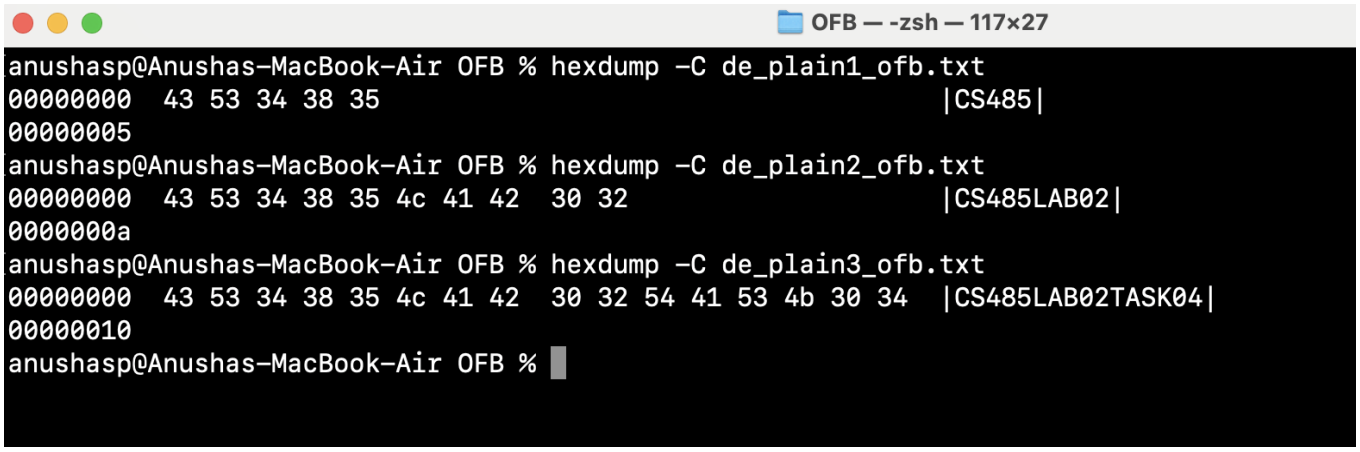
- We will encrypt the files using OFB mode and compare the file sizes.
- After encryption, there is no change in their file sizes as shown below, so there was NO padding.




- Now let us decrypt and check the padding, for this we use no pad in the openssl command.



The decrypted files content is as below:




**Analysis [OFB]:**

| File Name | Original File Size | Encrypted File Size | Decrypted File Size |
|---|---|---|---|
| plain1.txt | 5 Bytes | 21 Bytes | 5 Bytes |
| plain2.txt | 10 Bytes | 26 Bytes | 10 Bytes |
| plain3.txt | 16 Bytes | 32 Bytes | 16 Bytes |

**Conclusion:**

For ECB and CBC there was padding but for CFB and OFB there was NO padding.

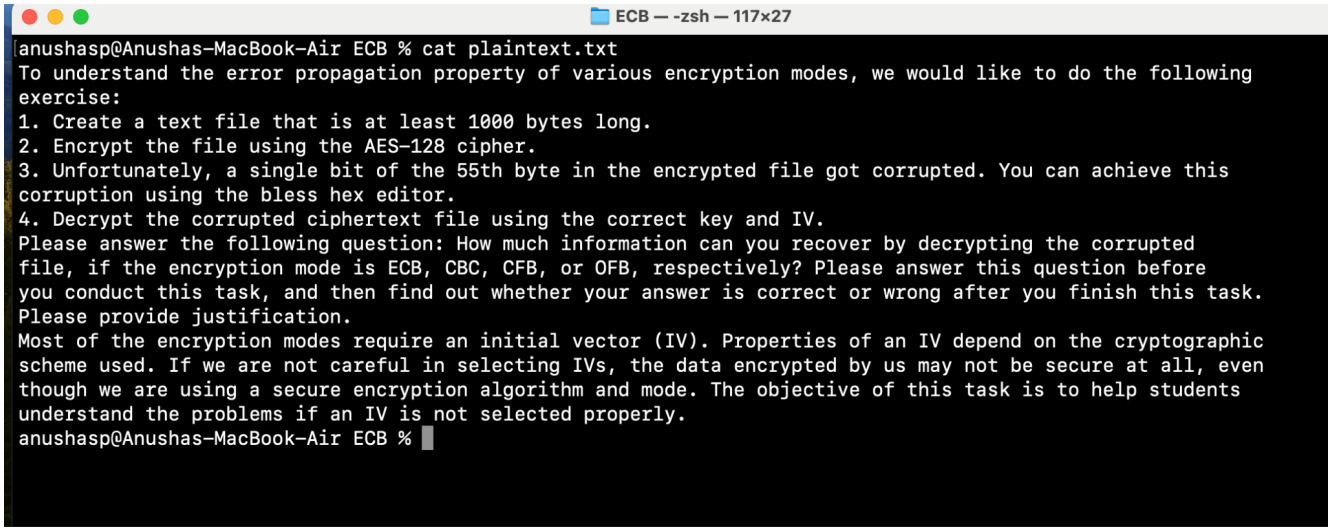
ECB and CBC modes need to be paired with a deterministic padding mode such as PKCS#5 or PKCD#7 compatible padding to operate on messages of any size.

OFB and CFB modes do not require padding. In OFB and CFB modes, the encryption and decryption processes are identical.

## Task 5: Error propagation: Corrupted Cipher Text

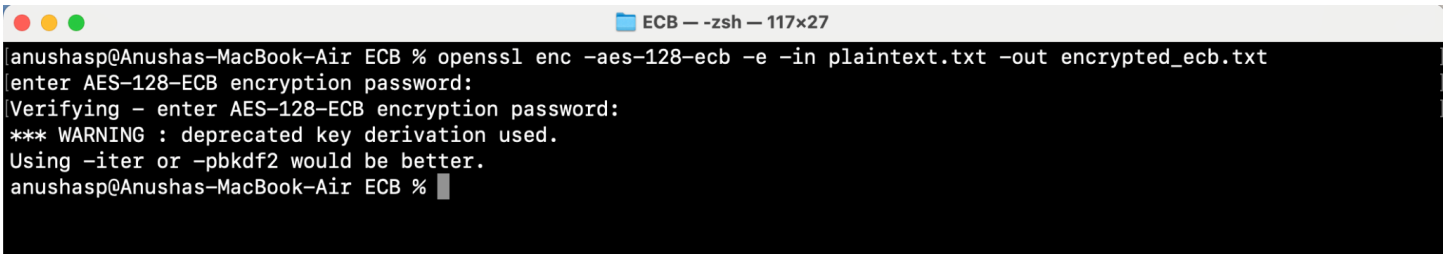
**Goal:** The goal of this task is to understand the error propagation property of various encryption modes.

Step 1: Create a plaintext file of at least 1000 bytes long.

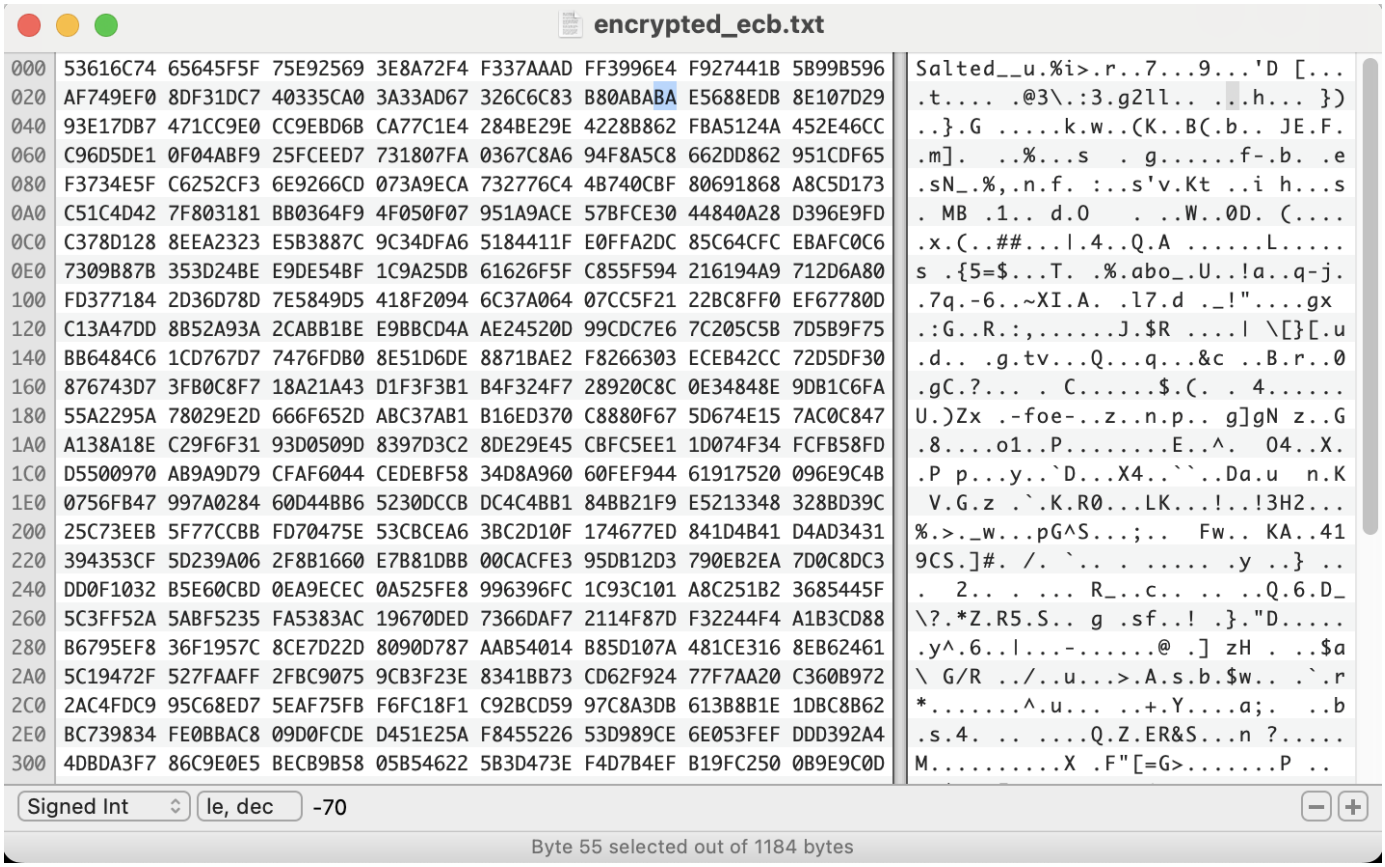



### 5.1 Using AES-128 ECB Mode:

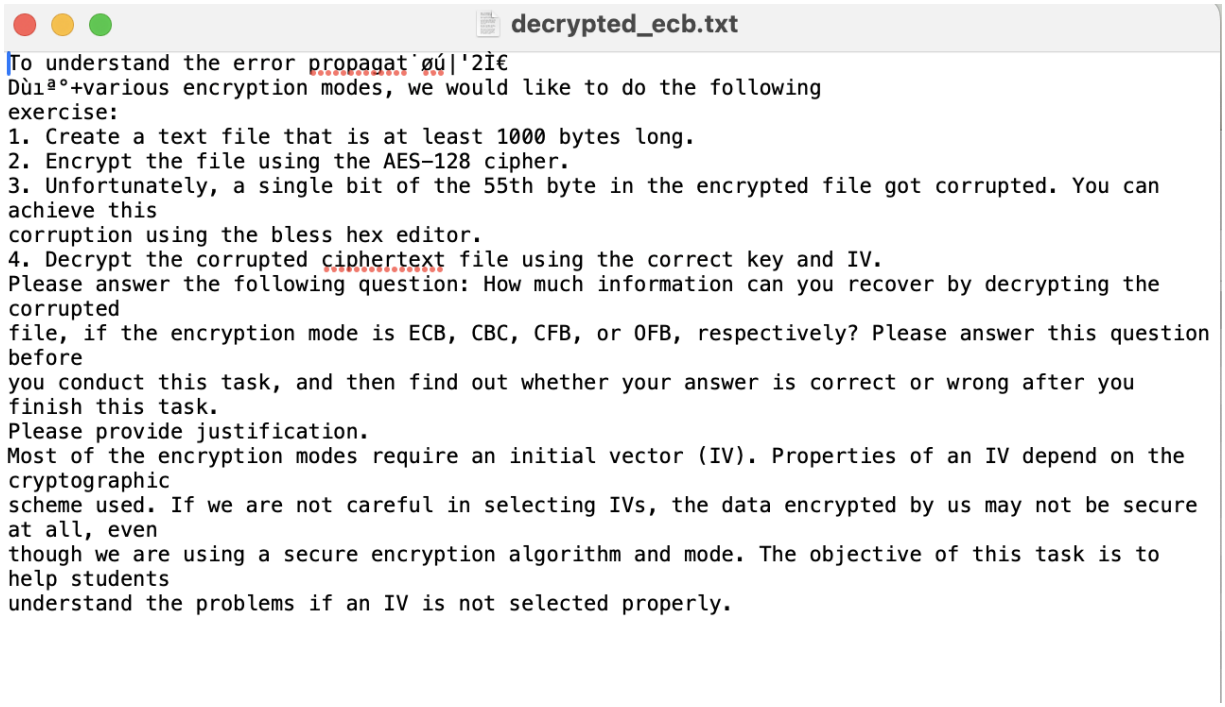
- The plaintext is divided into fixed-sized blocks (usually 128 bits or 16 bytes in the case of AES).
- Each block is treated independently during encryption.
- The same encryption key is used to encrypt each block of plaintext.
- Encrypt the above plaintext file using AES algorithm and ECB encryption mode.



- Corrupt the 55th bit using HEX Fiend as shown below.




- Now decrypt the file after corruption and check the output.

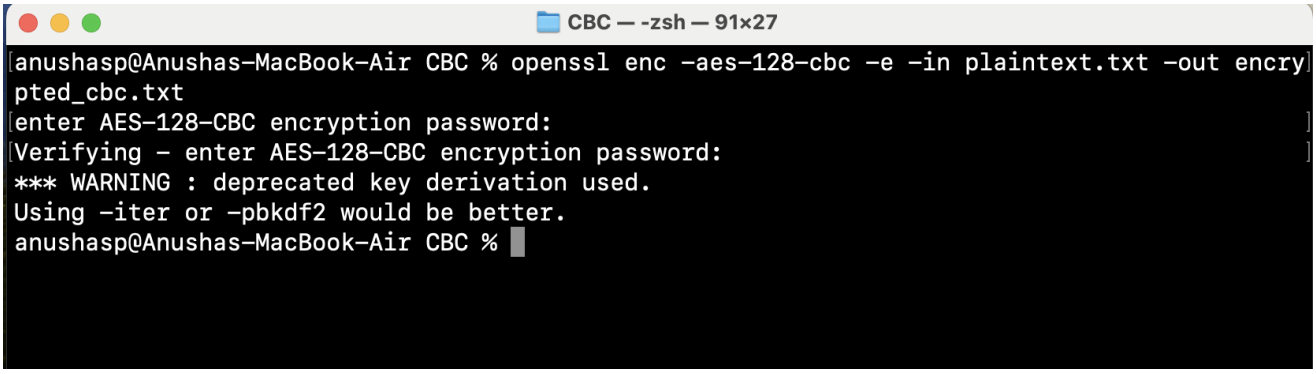


**Observation:**

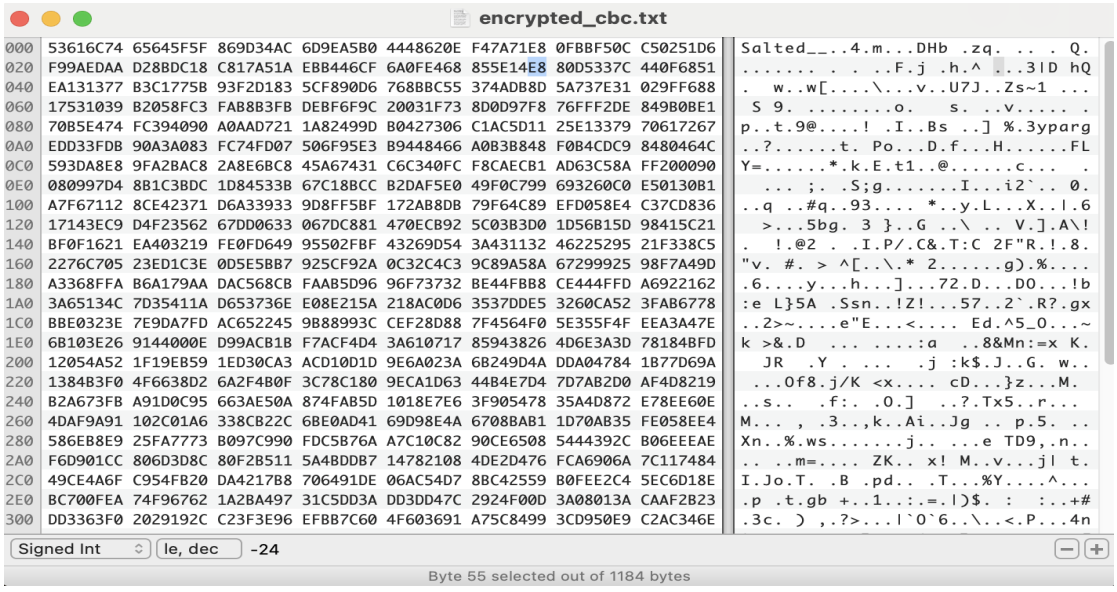
In ECB mode of encryption, each block of plain text is encrypted separately. So, on corrupting the 55th byte only the block containing that corrupted byte will be affected. This is one of the drawbacks of ECB mode, as it does not provide any inherent error propagation or diffusion, making it less secure and susceptible to patterns in the plaintext. [So impacted number of blocks = 1]

**5.2 Using AES-128 CBC Mode:**

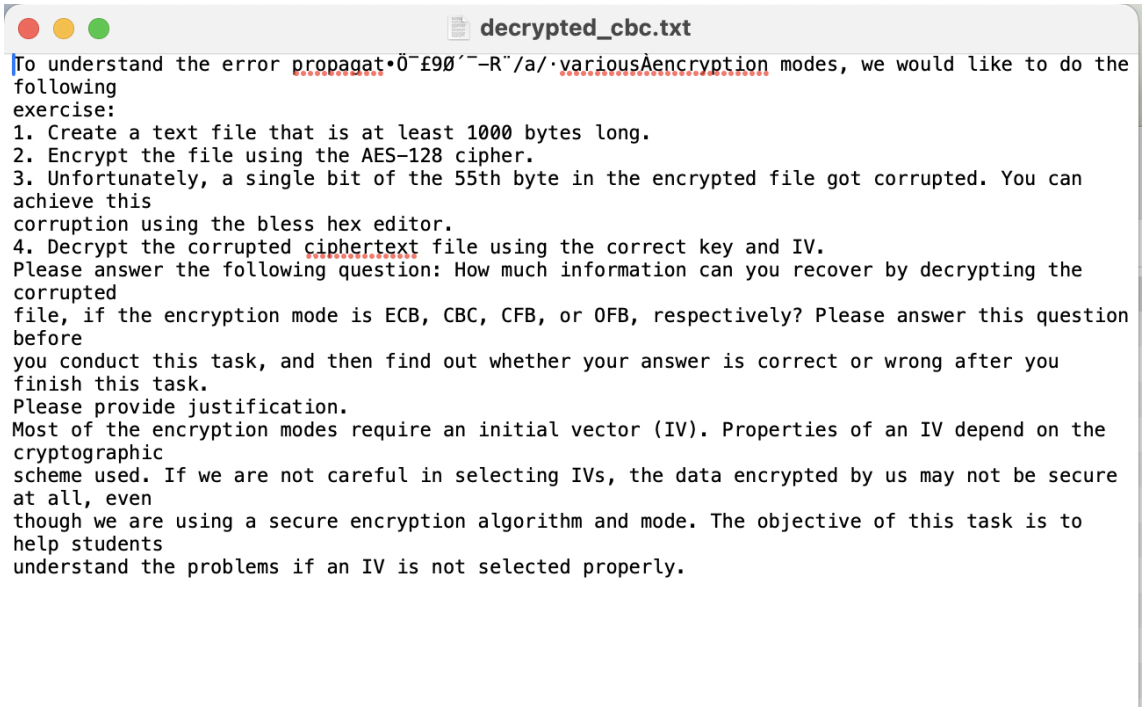
- In CBC mode, each plaintext block is XORed (bitwise exclusive OR) with the previous ciphertext block before encryption.
- This chaining of blocks adds an element of feedback into the encryption process.
- Let us encrypt the above plaintext file using AES algorithm and CBC encryption mode.




- Corrupt the 55th bit using HEX Fiend as shown below



- Now decrypt the file after corruption and check the output.

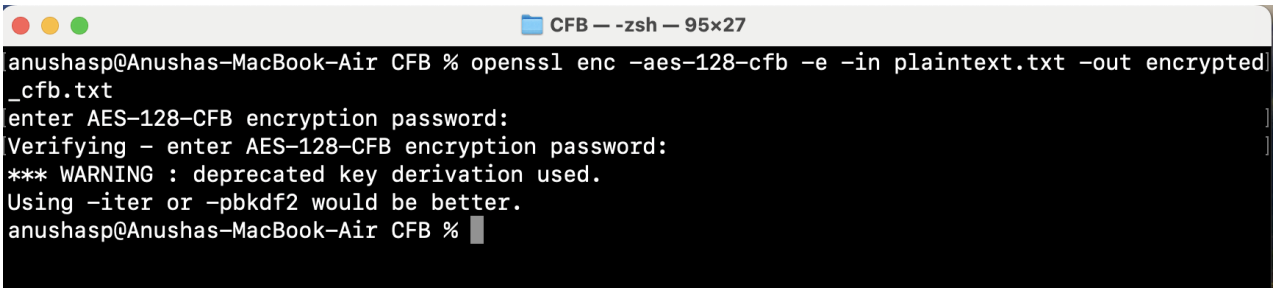



**Observation:**

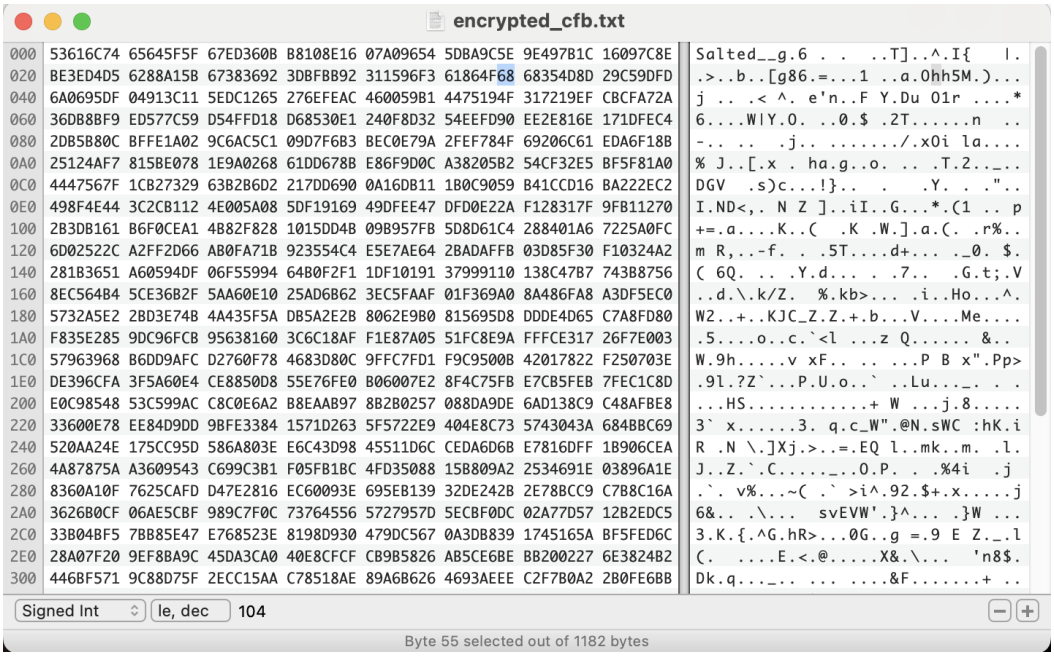
In CBC (Cipher Block Chaining) mode of encryption, if one byte is corrupted, the impact will be localized to the block containing the corrupted byte and the subsequent block during decryption. One damaged block propagates to two blocks as we use the first cipher text in generating the second and so on. [So impacted = 2 blocks]

**5.3 Using AES-A28 CFB Mode:**

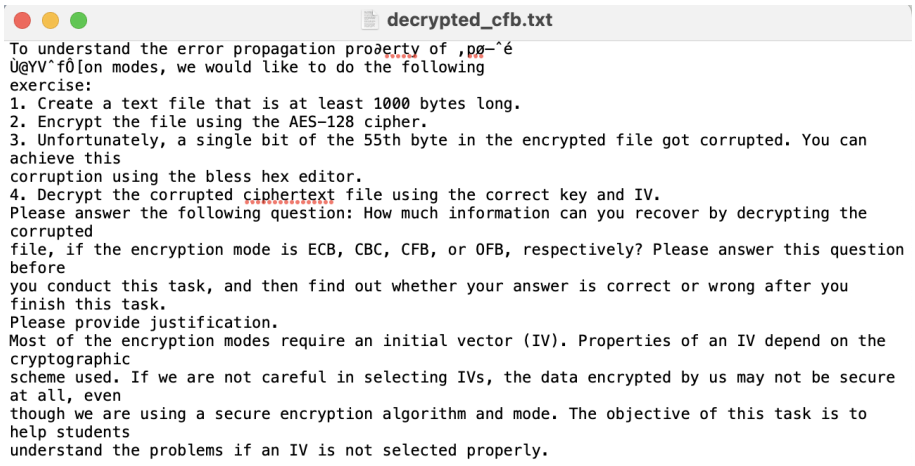
- CFB mode operates in a manner that resembles a stream cipher, generating a stream of pseudorandom bits that are XORed with the plaintext to produce the ciphertext.
- Let us encrypt the above plaintext file using AES algorithm and CFB encryption mode.



- Corrupt the 55th bit using HEX Fiend as shown below:




- Now decrypt the file after corruption and check the output.



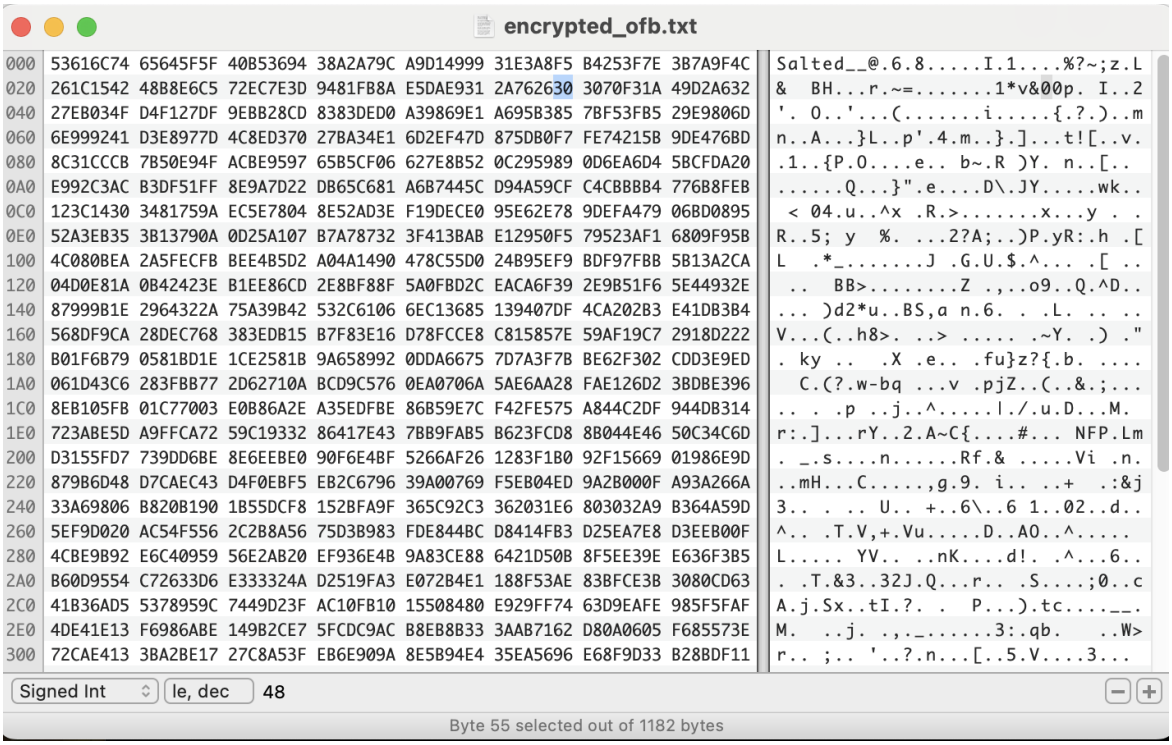

**Observation:**

In CFB mode, if one byte is corrupted, the impact is typically limited to the block containing the corrupted byte and the subsequent blocks during decryption. [So impact ~ 2 blocks]

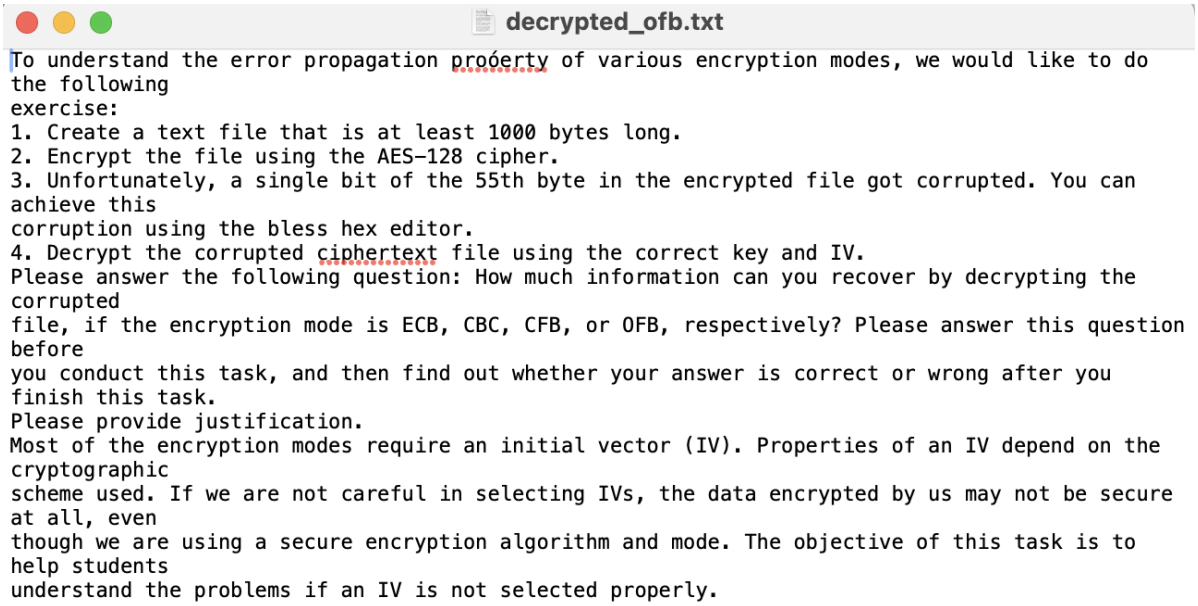
### 5.4 Using AES-128 OFB Mode:

- OFB mode operates in a manner that resembles a stream cipher. It generates a pseudorandom bit stream, which is then XORed with the plaintext to produce the ciphertext.
- Let us encrypt the above plaintext file using AES algorithm and OFB encryption mode.


- Corrupt the 55th bit using HEX Fiend as shown below



- Now decrypt the file after corruption and check the output.




**Observation:**

In OFB mode, the error does not propagate beyond the corrupted byte. It is designed for streaming, making it suitable for applications where error recovery is important.

So impact = only corrupted byte!

**Summary:**

| Mode | Observation | Impact |
|---|---|---|
| **ECB** | Only the block containing the corrupted byte is affected. | 1 block |
| **CBC** | Current block and subsequent block. | 2 blocks |
| **CFB** | Current block and subsequent block. | 2 blocks |
| **OFB** | Only the corrupted byte is affected. | 1 byte |

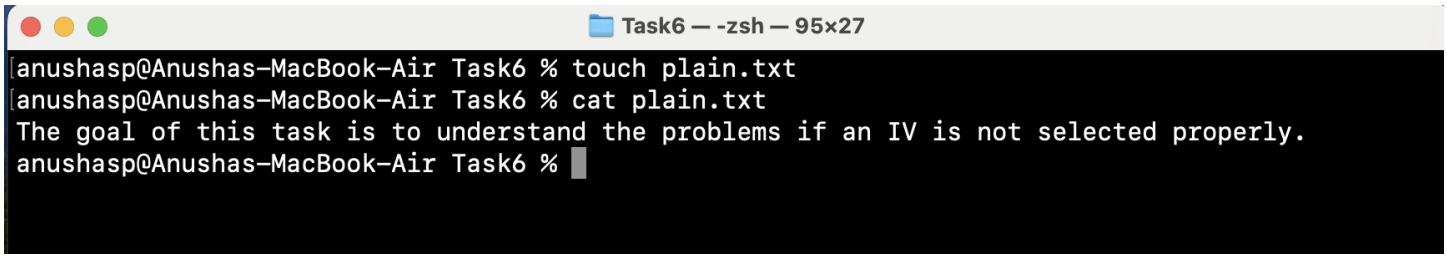
## Task 6: Initial Vector (IV)

**Goal:** The goal of this task is to understand the problems if an IV is not selected properly.

### 6.1. Uniqueness of the IV:

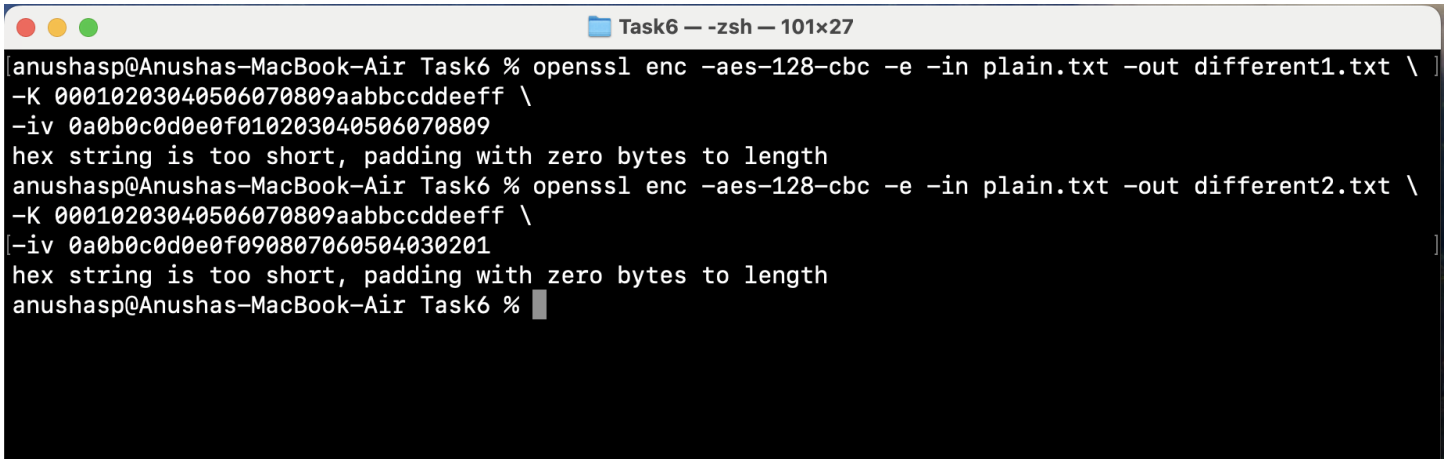
Initialization Vectors (IVs) are used in various cryptographic algorithms, such as block ciphers in Cipher Block Chaining (CBC) mode, to add randomness and prevent patterns in the ciphertext.

Step 1: Creating the plaintext file plain.txt



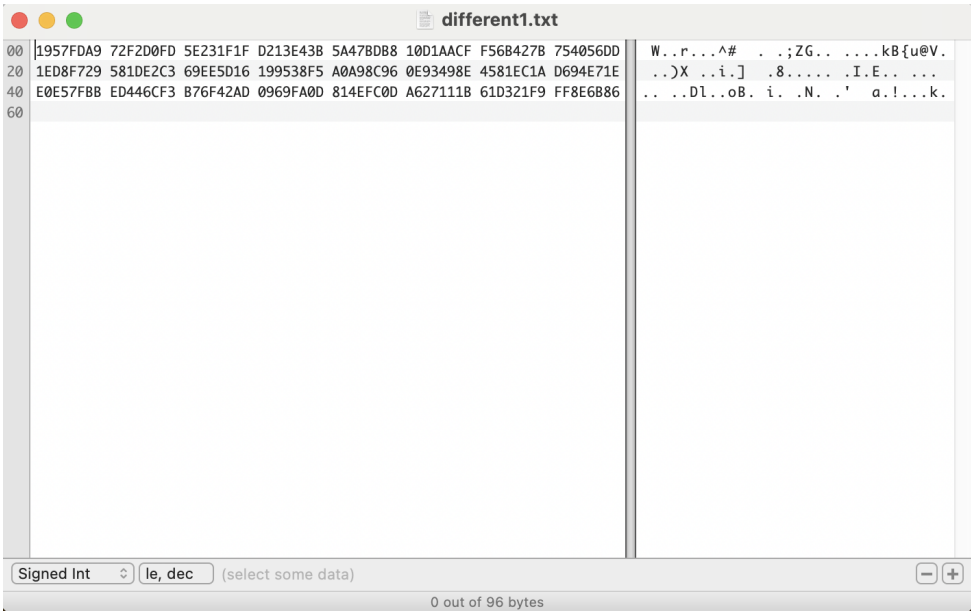

### 6.1.1 Using two different IVs:

- Encrypting the same plain.txt using two different IVs.

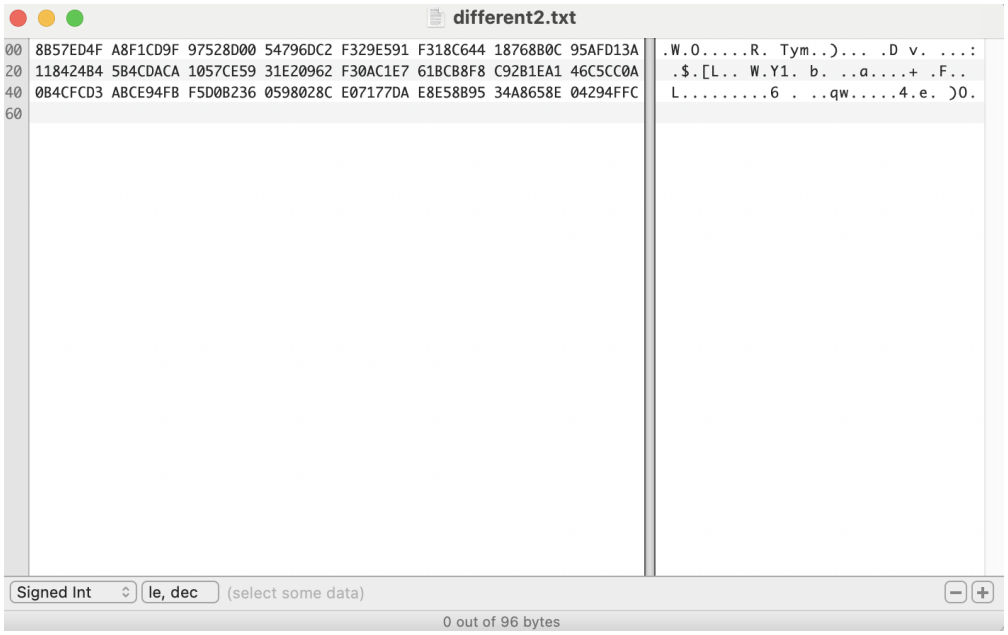


**Output files:**

different1.txt



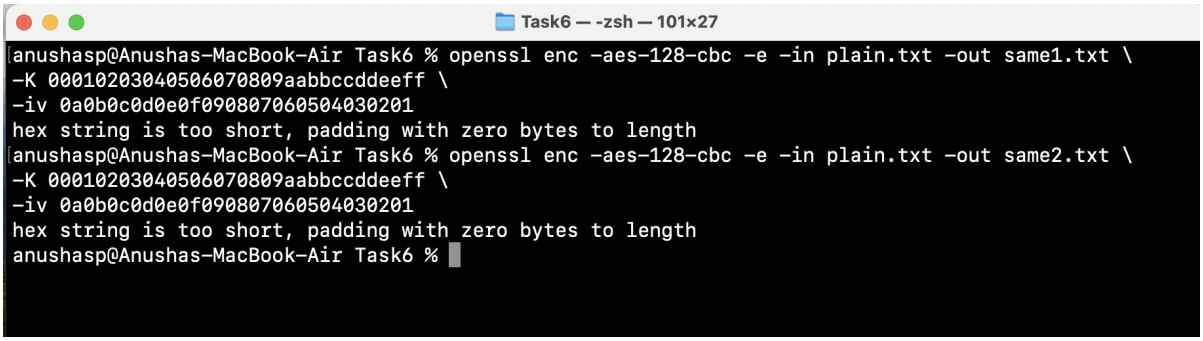

different2.txt




**Observation:** The ciphertext generated for the same plain text is different when different IV is used.

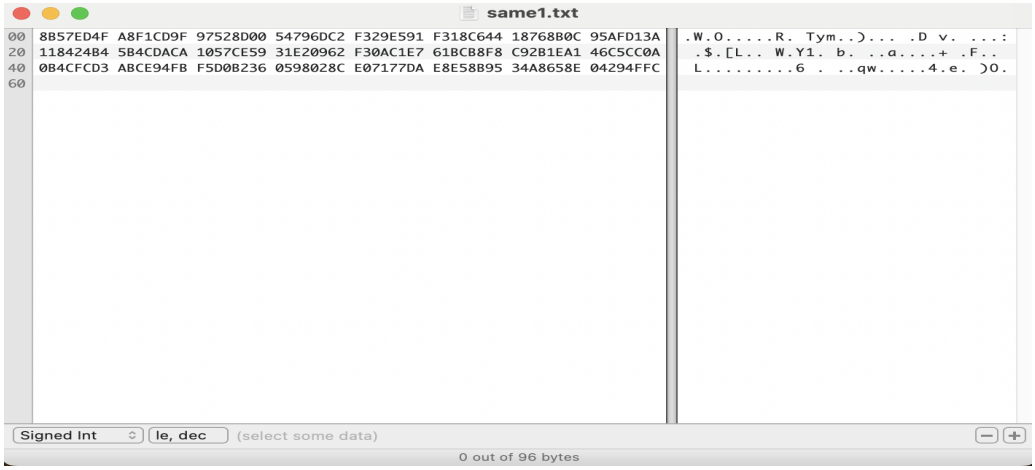
### 6.1.2 Using the same IV:

- Encrypting the same plain.txt using the same IV twice.

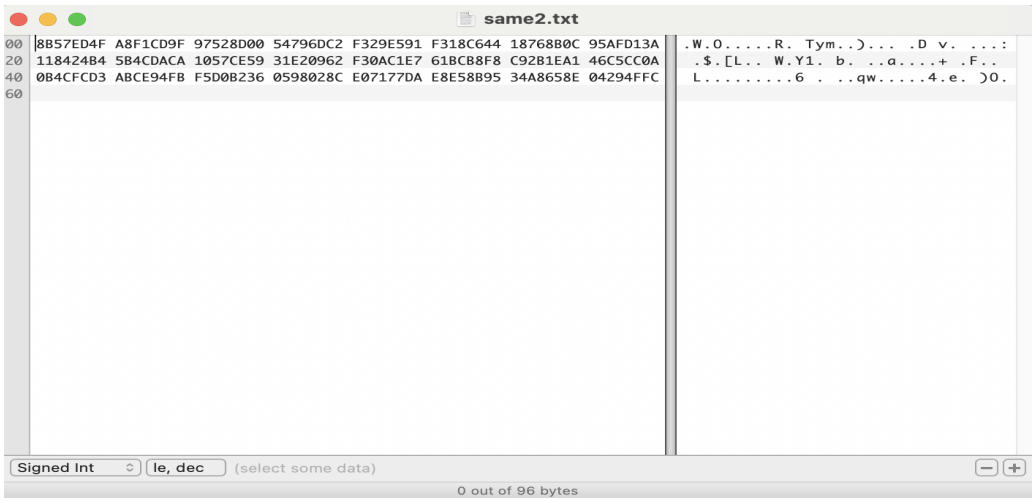



**Output files:**

same1.txt




same2.txt




**Observation:** The ciphertext generated for the same plain text is exactly the same when the same IV is used.

### 6.2. Common mistakes: Use the same IV

Here the goal is to find how much of P2 can be revealed if P1, C1 and C2 are known.

We know,

Plaintext (P1): This is a known message!
Ciphertext (C1): a469b1c502c1cab966965e50425438e1bb1b5f9037a4c15913
Plaintext (P2): (unknown to you)
Ciphertext (C2): bf73bcd3509299d566c35b5d450337e1bb175f903fafc15913

Let us convert P1 into hexadecimal format:

P1 : 54 68 69 73 20 69 73 20 61 20 6B 6E 6F 77 6E 20 6D 65 73 73 61 67 65 21 0A

**In OFB Mode, P1 (XOR) C1 = P2 (XOR) C2**

**So, P2 = P1 (XOR) C1 (XOR) C2**

On calculation P2 = 4F726465723A204C61756E63682061206D697373696C65210A

On conversion to ASCII, text generated = “Order: Launch a missile!”

**Observation:**

In OFB mode, if an attacker has access to a plaintext (P1) and its corresponding ciphertext (C1) and the IV is always the same, they can compute the keystream used to encrypt P1 by XORing P1 with C1. Once they have the keystream, they can XOR it with C2 to obtain the plaintext P2, even if P2 is unknown.

In CFB mode, if the same IV is used, a similar vulnerability exists. An attacker who knows P1 and C1 can compute the keystream in the same way as in OFB mode and use it to decrypt C2 and reveal the entire content of P2.

In summary, the primary difference in the context of the known-plaintext attack is that using the same IV in OFB mode does not reveal information about other encrypted messages, while in CFB mode, it can potentially reveal information about other blocks encrypted with the same IV, depending on the IV reuse patterns and the encryption algorithm used.

### 6.3. Common mistakes: Use a predictable IV

We know,

Encryption method: 128-bit AES with CBC mode.
Key (in hex): 00112233445566778899aabbccddeeff (known only to Bob)
Ciphertext (C1): bef65565572ccee2a9f9553154ed9498 (known to both)

IV used on P1 (known to both)
(in ascii): 1234567890123456
(in hex) : 31323334353637383930313233343536
Next IV (known to both)
(in ascii): 1234567890123457
(in hex) : 31323334353637383930313233343537

Bob sent an encrypted message and Eve knows its content is either ‘Yes’ or ‘No’.

Our goal is to construct a message P2 and ask Bob to encrypt it and give you the ciphertext. And use this opportunity to figure out whether the actual content of P1 is Yes or No .

We can find out P2 by using:
P1 = IV1 (XOR) IV2 (XOR) P2

Let us assume that P2 = ‘Yes’

P2 (in hex): 59 65 73
P2 (with padding): 59657305050505050505050505050505

After XOR operations,

P1 = 59657305050505050505050505050504
On conversion to ASCII, text generated = “Yes”

**Observation:**
In conclusion, through a chosen-plaintext attack in the context of AES-CBC encryption, it is possible to make an educated guess about the content of P1. By carefully crafting a new plaintext message P2 and observing the changes in the ciphertext when P2 is encrypted, one can infer the likely content of P1 based on whether the ciphertext changes significantly or not.

**Task 7: Programming using the Crypto library [Extra Credit]**

**Goal:** The goal of this task is to write a program to find the key that is used for the encryption when plaintext and ciphertext are given.

- AES-128-CBC cipher is used for encryption.
- The key used to encrypt this plaintext is an English word shorter than 16 characters; the word can be found from a typical English dictionary. Since the word has less than 16 characters (i.e. 128 bits), pound signs (#: hexadecimal value is 0x23 ) are appended to the end of the word to form a key of 128 bits.

The plaintext, ciphertext, and IV are listed in the following:

- Plaintext (total 21 characters): This is a secret tool
- Ciphertext (in hex format): ece6753e938f8f903cabbbe12d395bf5f7eae38ad918a2d3e1c3a832476d5c7a
- IV (in hex format): 0102030405060708090a0b0c0d0e0f

**Steps Performed:**

- Download the words.txt file from the internet.
- Create a .txt for storing the plaintext called plaintext.txt of 21 characters with given data.
- Program for finding the key:
- Run using the below command by including the lcrypto library.
  gcc -o findKey findKey.c -lcrypto
- After that we will verify if the key generated is correct using the openssl command by encrypting the given plaintext with the generated key and given key.
- The above command will generate a ciphertext file and using hex fiend we can compare the contents of the generated file with the given ciphertext in hex format to verify if the key is correct.

**Program: [findKey.c]**

**Git link:** https://github.com/anushasonte/CS458_InformationSecurity/blob/main/Lab2/findKey.c

```
#include <stdio.h>
#include <string.h>
#include <openssl/evp.h>

void hex_string_to_bytes(const char *hex_string, unsigned char *bytes, size_t hex_string_len) {
    for (size_t i = 0; i < hex_string_len / 2; i++) {
        sscanf(&hex_string[2 * i], "%2hhx", &bytes[i]);
    }
}

int main() {
    // Define the provided plaintext, ciphertext, and IV
    const char *plaintext = "This is a secret tool";
    const char *ciphertext_hex
 = "ece6753e938f8f903cabbbe12d395bf5f7eae38ad918a2d3e1c3a832476d5c7a";
    const char *iv_hex = "0102030405060708090a0b0c0d0e0f";

    // Load the English word list from a file (words.txt)
    FILE *wordlist = fopen("words.txt", "r");
    if (wordlist == NULL) {
        perror("Failed to open wordlist file");
        return 1;
    }

    char word[16];
    while (fgets(word, sizeof(word), wordlist)) {
        // Remove newline characters from the word
        word[strcspn(word, "\n")] = '\0';
```

```
// Initialize OpenSSL's EVP context
EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
EVP_CIPHER_CTX_init(ctx);

// Convert IV and ciphertext from hex strings to bytes
unsigned char iv[16];
unsigned char ciphertext[128];
hex_string_to_bytes(iv_hex, iv, 32);
hex_string_to_bytes(ciphertext_hex, ciphertext, 128);

// Pad the word with pound signs (#) to form a 128-bit key
unsigned char key[16];
memset(key, '#', sizeof(key));

// Copy the word into the key, ensuring it doesn't exceed the key size
size_t word_len = strlen(word);
if (word_len > 16) {
    word_len = 16;
}
memcpy(key, word, word_len);

// Set up the decryption parameters
EVP_DecryptInit_ex(ctx, EVP_aes_128_cbc(), NULL, key, iv);

// Decrypt the ciphertext
unsigned char decrypted[128];
int decrypted_len;
EVP_DecryptUpdate(ctx, decrypted, &decrypted_len, ciphertext, 128);
EVP_DecryptFinal_ex(ctx, decrypted + decrypted_len, &decrypted_len);
```

```
        // Check if the decrypted plaintext matches the given plaintext
        if (strncmp((char *)decrypted, plaintext, strlen(plaintext)) == 0) {
            printf("Found the key: %s\n", word);
            break;
        }

        EVP_CIPHER_CTX_free(ctx);
    }

    fclose(wordlist);
    return 0;
}
```

**Output:**

```
anushasp@Anushas-MacBook-Air Task7 % gcc -o findKey findKey.c $LDFLAGS $CPPFLAGS -lcrypto
ld: warning: dylib (/opt/homebrew/opt/openssl@1.1/lib/libcrypto.dylib) was built for newer macOS version (14.0) than being linked (13.3)
anushasp@Anushas-MacBook-Air Task7 % ./findKey
Found the key: safety
anushasp@Anushas-MacBook-Air Task7 %
```

The Key found = ‘safefy’

**Steps to run the program:**

- Download the program ‘findKey.c’ using below gitlink
  https://github.com/anushasonte/CS458_InformationSecurity/blob/main/Lab2/findKey.c
- Download the words.txt file using the below gitlink
  https://github.com/anushasonte/CS458_InformationSecurity/blob/main/Lab2/words.txt
- Put both the above files in the same folder.
- Use below commands:
  gcc -o findKey findKey.c -lcrypto
  ./findKey

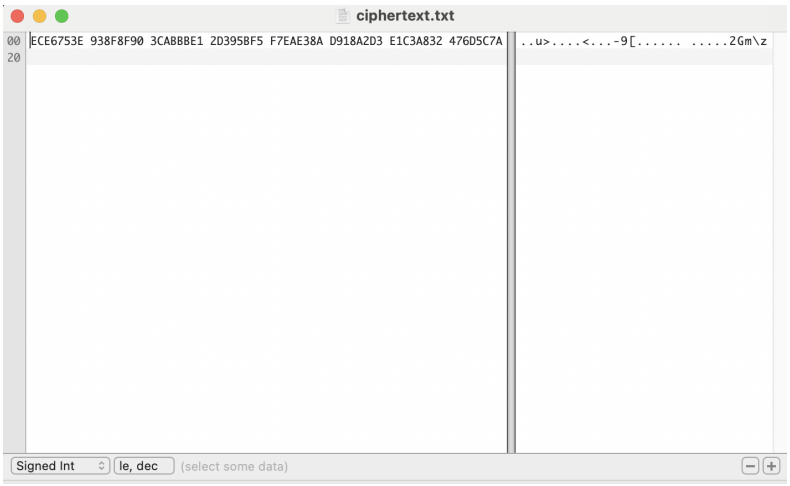
**Verification:**

- Let us try to encrypt the given plain text using the key found and see if the ciphertext generated would be the same as given in the input.

  Key (in hex format): 736166657479232323232323232323
  IV (in hex format): 0102030405060708090000a0b0c0d0e0f

```
anushasp@Anushas-MacBook-Air Task7 % openssl enc -e -aes-128-cbc -in plaintext.txt -out ciphertext.txt -K 73616665747923232323232323232323 -iv 0102030405060708
809000a0b0c0d0e0f
anushasp@Anushas-MacBook-Air Task7 %
```

Opening the generated ciphertext.txt using hexfiend.




We can see that the hexadecimal format of the ciphertext is exactly the same as that of the key! Hence, the key generated by the program is correct and

Key = safety
(in hex): 736166657479232323232323232323

All the files generated as part of this lab for all tasks can be found below:
https://github.com/anushasonte/CS458_InformationSecurity/blob/main/Lab2/Lab2.zip