# Project 5

Purva Naresh Rumde

pr23b

**Secret-Key Encryption Lab**

## Task 2: Encryption using Different Ciphers and Modes

In this task we have just tried three  different ciphers and their modes.

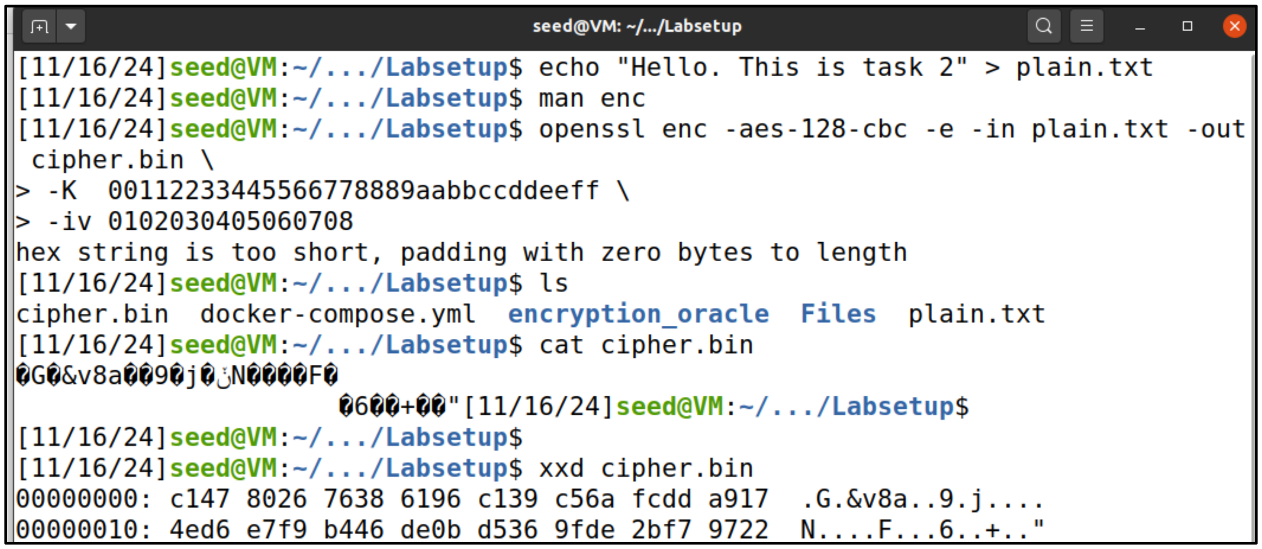
We have used the openssl command to execute these ciphers.

First I wrote the contents in the plaintext file and then used the

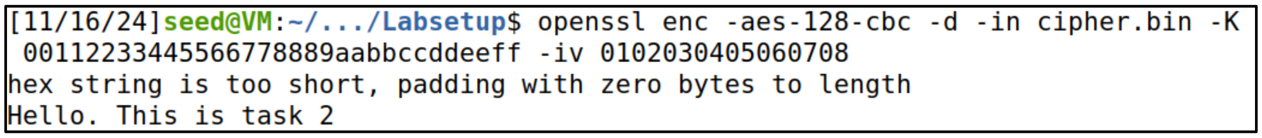
Openssl enc -aes-128-cbc -e -in plain.txt -out.cipher.bin (also used the same command for other 2 ciphers as well

Then to read the cipher.bin file I  just used the xxd cipher.bin command since it's a hex file and to decrypt the bin file I used the -d option and read the file.

**-aes-128-cbc**

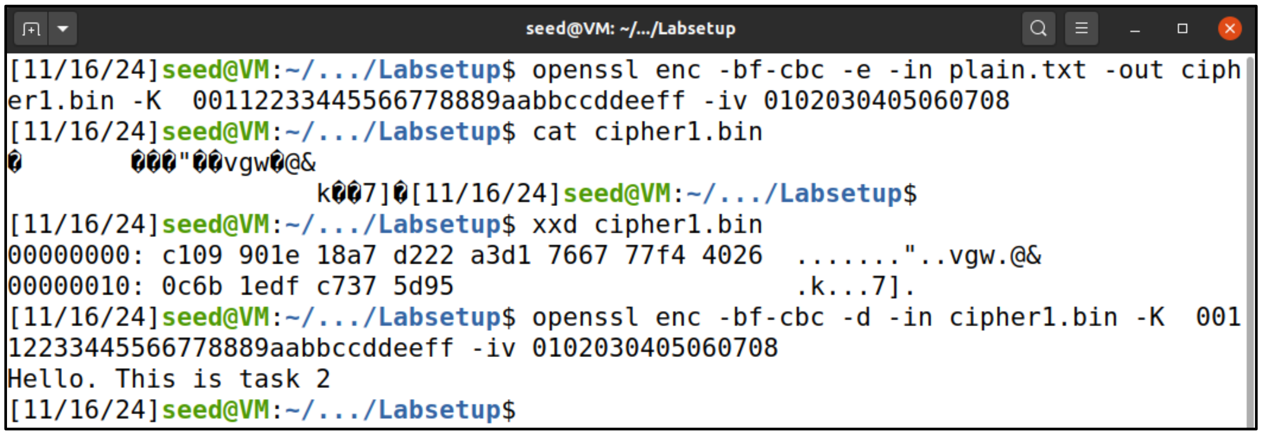


```
[11/16/24]seed@VM:~/.../Labsetup$ echo "Hello. This is task 2" > plain.txt
[11/16/24]seed@VM:~/.../Labsetup$ man enc
[11/16/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -e -in plain.txt -out
 cipher.bin \
> -K  00112233445566778889aabbccddeeff \
> -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../Labsetup$ ls
cipher.bin  docker-compose.yml  encryption_oracle  Files  plain.txt
[11/16/24]seed@VM:~/.../Labsetup$ cat cipher.bin
�G�&v8a��9�j�ݍN����F�
                     �6��+��"[11/16/24]seed@VM:~/.../Labsetup$
[11/16/24]seed@VM:~/.../Labsetup$
[11/16/24]seed@VM:~/.../Labsetup$ xxd cipher.bin
00000000: c147 8026 7638 6196 c139 c56a fcdd a917  .G.&v8a..9.j....
00000010: 4ed6 e7f9 b446 de0b d536 9fde 2bf7 9722  N....F...6..+.."
```



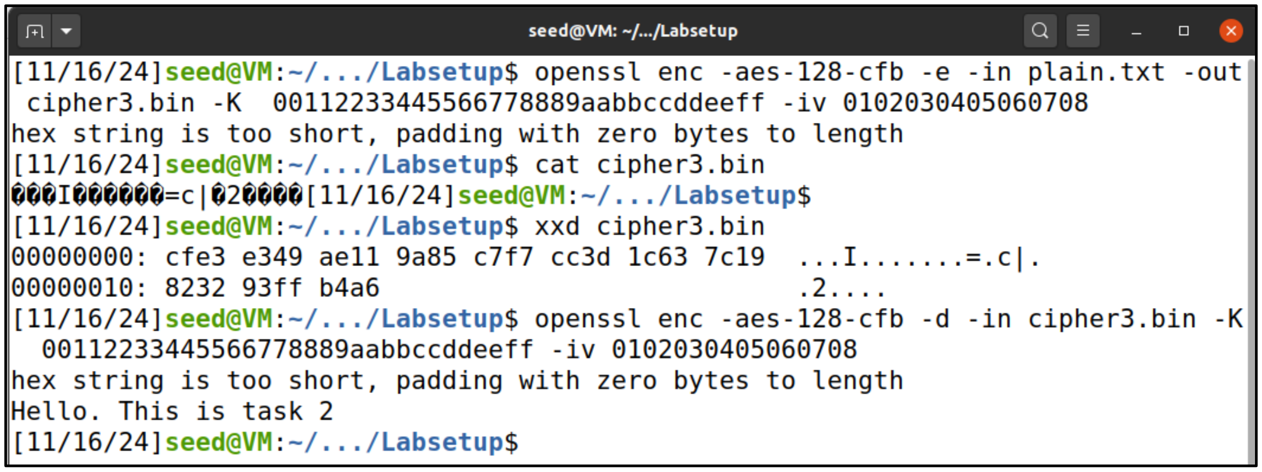
```
[11/16/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cbc -d -in cipher.bin -K
 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
Hello. This is task 2
```

**-bf-cbc**



```
[11/16/24]seed@VM:~/.../Labsetup$ openssl enc -bf-cbc -e -in plain.txt -out ciph
er1.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708
[11/16/24]seed@VM:~/.../Labsetup$ cat cipher1.bin
�       ���"��vgw�@&
                    k��7]�[11/16/24]seed@VM:~/.../Labsetup$
[11/16/24]seed@VM:~/.../Labsetup$ xxd cipher1.bin
00000000: c109 901e 18a7 d222 a3d1 7667 77f4 4026  ......."..vgw.@&
00000010: 0c6b 1edf c737 5d95                      .k...7].
[11/16/24]seed@VM:~/.../Labsetup$ openssl enc -bf-cbc -d -in cipher1.bin -K  001
12233445566778889aabbccddeeff -iv 0102030405060708
Hello. This is task 2
[11/16/24]seed@VM:~/.../Labsetup$
```

**-aes-128-cfb**



```
[11/16/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cfb -e -in plain.txt -out
 cipher3.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../Labsetup$ cat cipher3.bin
���I������=c|�2����[11/16/24]seed@VM:~/.../Labsetup$
[11/16/24]seed@VM:~/.../Labsetup$ xxd cipher3.bin
00000000: cfe3 e349 ae11 9a85 c7f7 cc3d 1c63 7c19  ...I.......=.c|.
00000010: 8232 93ff b4a6                           .2....
[11/16/24]seed@VM:~/.../Labsetup$ openssl enc -aes-128-cfb -d -in cipher3.bin -K
  00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
Hello. This is task 2
[11/16/24]seed@VM:~/.../Labsetup$
```

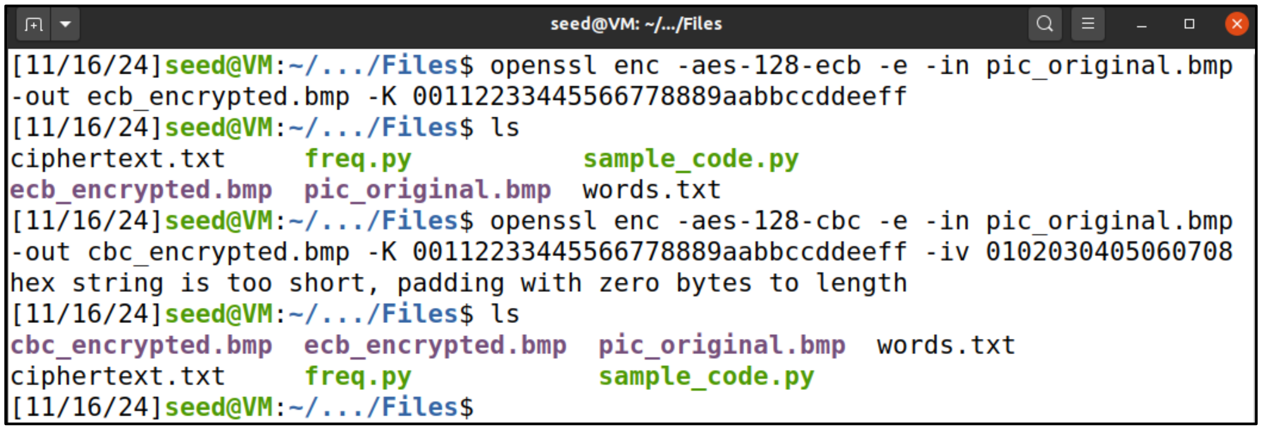
**Task 3: Encryption Mode – ECB vs. CBC**

We used the openssl enc command to encrypt the pic_original.bmp file with both ECB and CBC modes.

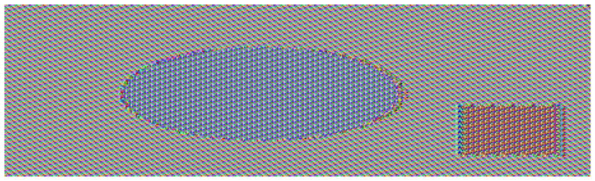
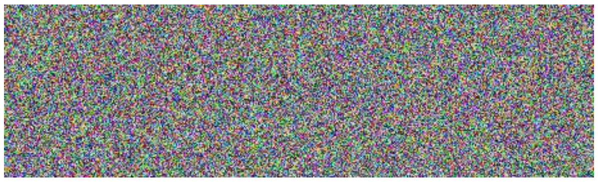
For CBC we needed to add the -iv

Then after encrypting we used the tail, head and header command in order to open the encrypted bmp file. I also tried doing this task with my own image.

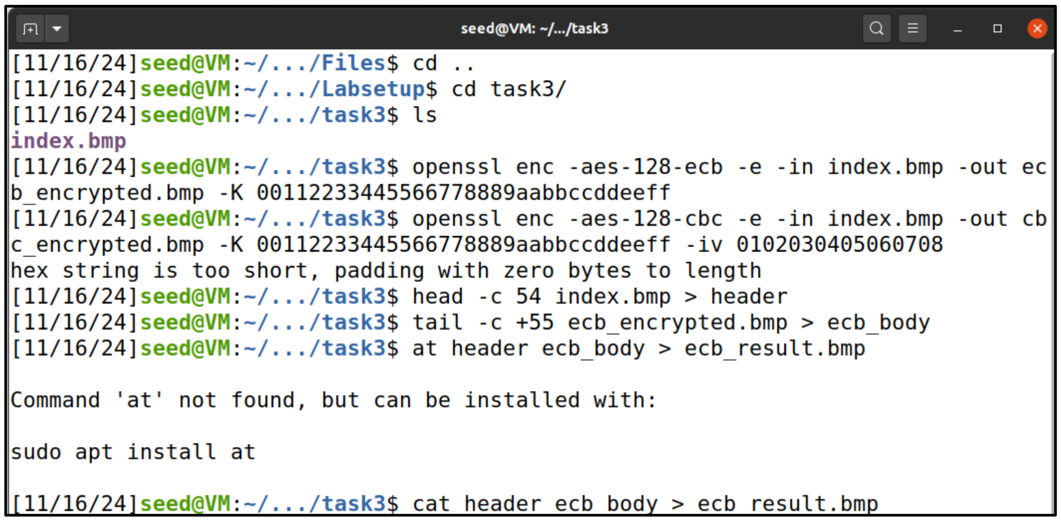
And as you can see that ECB mode encrypts each block independently, leading to visible patterns in the image if blocks have the same content. CBC mode, however, uses chaining, making patterns much harder to recognize.



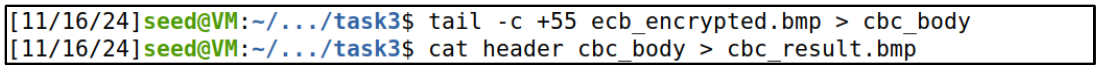
```
[11/16/24]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in pic_original.bmp
-out ecb_encrypted.bmp -K 00112233445566778889aabbccddeeff
[11/16/24]seed@VM:~/.../Files$ ls
ciphertext.txt     freq.py             sample_code.py
ecb_encrypted.bmp  pic_original.bmp  words.txt
[11/16/24]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in pic_original.bmp
-out cbc_encrypted.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../Files$ ls
cbc_encrypted.bmp  ecb_encrypted.bmp  pic_original.bmp  words.txt
ciphertext.txt     freq.py             sample_code.py
[11/16/24]seed@VM:~/.../Files$
```

ECB CBC

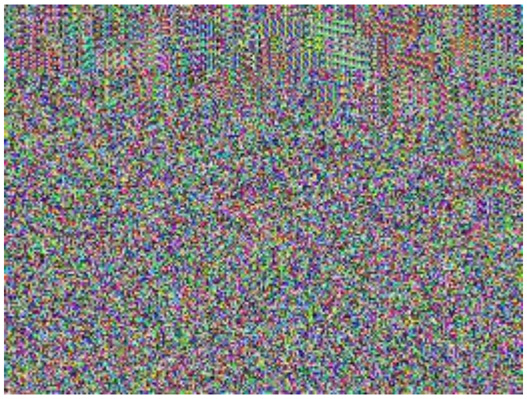


```
[11/16/24]seed@VM:~/.../Files$ cd ..
[11/16/24]seed@VM:~/.../Labsetup$ cd task3/
[11/16/24]seed@VM:~/.../task3$ ls
index.bmp
[11/16/24]seed@VM:~/.../task3$ openssl enc -aes-128-ecb -e -in index.bmp -out ec
b_encrypted.bmp -K 00112233445566778889aabbccddeeff
[11/16/24]seed@VM:~/.../task3$ openssl enc -aes-128-cbc -e -in index.bmp -out cb
c_encrypted.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task3$ head -c 54 index.bmp > header
[11/16/24]seed@VM:~/.../task3$ tail -c +55 ecb_encrypted.bmp > ecb_body
[11/16/24]seed@VM:~/.../task3$ at header ecb_body > ecb_result.bmp

Command 'at' not found, but can be installed with:

sudo apt install at

[11/16/24]seed@VM:~/.../task3$ cat header ecb_body > ecb_result.bmp
```

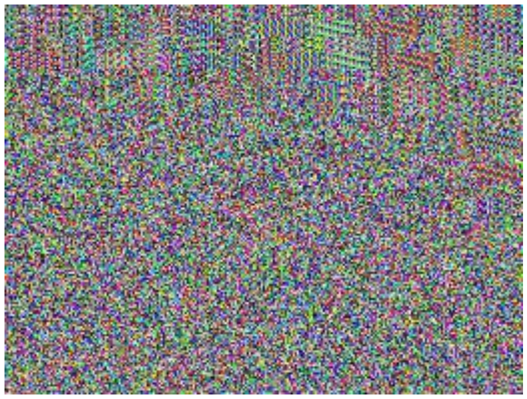


```
[11/16/24]seed@VM:~/.../task3$ tail -c +55 ecb_encrypted.bmp > cbc_body
[11/16/24]seed@VM:~/.../task3$ cat header cbc_body > cbc_result.bmp
```

Here I tried with my image but since my image was too colorful I didn't get to see much difference in both images



ECB



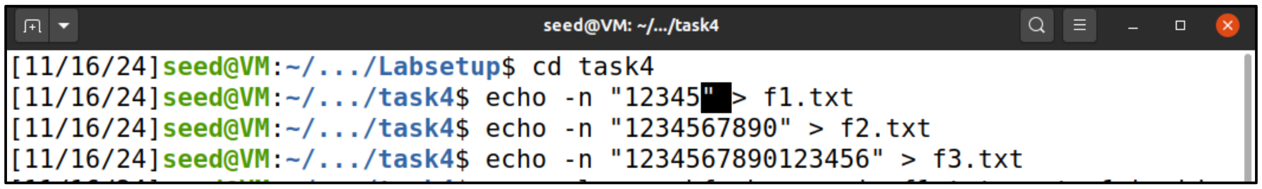
CBC

**Task 4: Padding**

**Identify Padding Behavior in Different Modes:**

Encrypting 3 files with 3 different sizes 5,10,16 with different modes (ECB, CBC, CFB, OFB)

Here I first created 3 files of different sizes f1,f2,f3 and then encrypted and decrypted them using the asked mode.



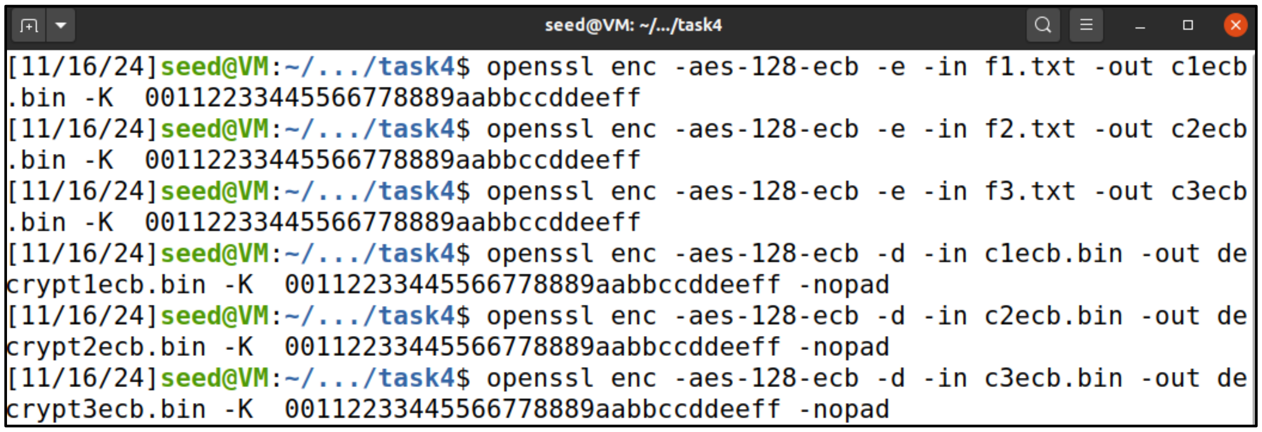
```
[11/16/24]seed@VM:~/.../Labsetup$ cd task4
[11/16/24]seed@VM:~/.../task4$ echo -n "12345" > f1.txt
[11/16/24]seed@VM:~/.../task4$ echo -n "1234567890" > f2.txt
[11/16/24]seed@VM:~/.../task4$ echo -n "1234567890123456" > f3.txt
```

**<u>ecb decryption and padding</u>**

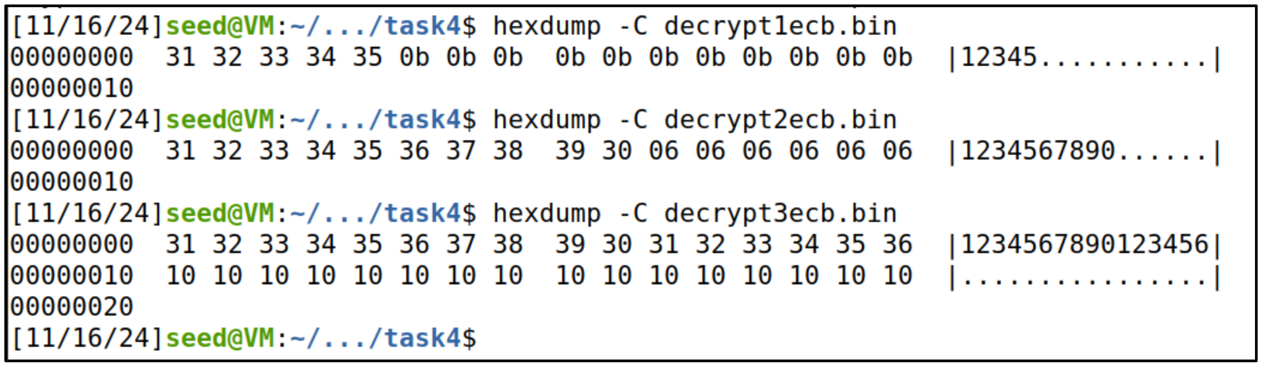
In ecb since the padding is added the sizes of files when encrypted became C1 (F1)= 16 bytes

C2(F2)=16 bytes C3(F3)= 32 bytes.

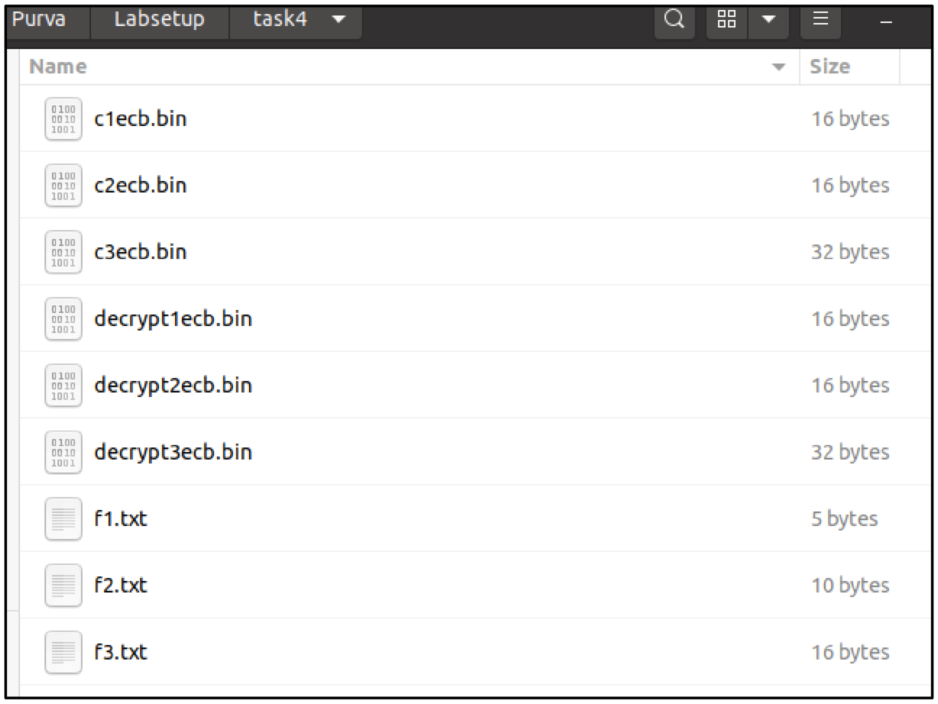
Then I decrypted and to see what padding was added I used the hexdump command.



```
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-ecb -e -in f1.txt -out c1ecb
.bin -K  00112233445566778889aabbccddeeff
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-ecb -e -in f2.txt -out c2ecb
.bin -K  00112233445566778889aabbccddeeff
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-ecb -e -in f3.txt -out c3ecb
.bin -K  00112233445566778889aabbccddeeff
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-ecb -d -in c1ecb.bin -out de
crypt1ecb.bin -K  00112233445566778889aabbccddeeff -nopad
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-ecb -d -in c2ecb.bin -out de
crypt2ecb.bin -K  00112233445566778889aabbccddeeff -nopad
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-ecb -d -in c3ecb.bin -out de
crypt3ecb.bin -K  00112233445566778889aabbccddeeff -nopad
```



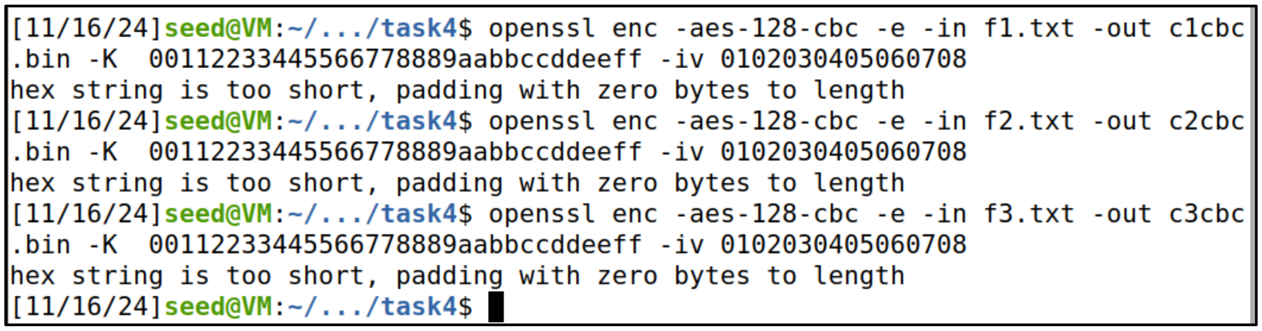
```
[11/16/24]seed@VM:~/.../task4$ hexdump -C decrypt1ecb.bin
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345...........|
00000010
[11/16/24]seed@VM:~/.../task4$ hexdump -C decrypt2ecb.bin
00000000  31 32 33 34 35 36 37 38  39 30 06 06 06 06 06 06  |1234567890......|
00000010
[11/16/24]seed@VM:~/.../task4$ hexdump -C decrypt3ecb.bin
00000000  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36  |1234567890123456|
00000010  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  |................|
00000020
[11/16/24]seed@VM:~/.../task4$
```



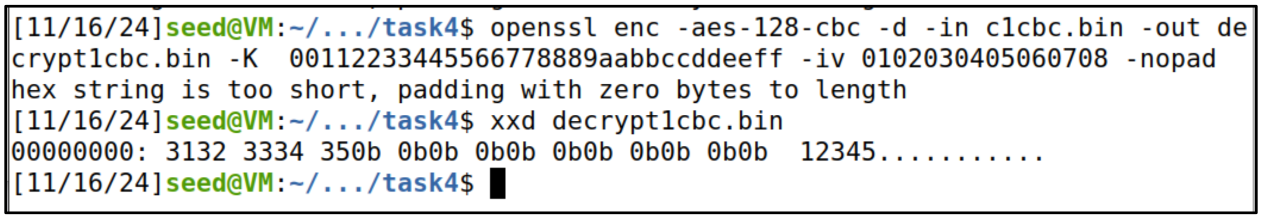
| Name | Size |
|---|---|
| c1ecb.bin | 16 bytes |
| c2ecb.bin | 16 bytes |
| c3ecb.bin | 32 bytes |
| decrypt1ecb.bin | 16 bytes |
| decrypt2ecb.bin | 16 bytes |
| decrypt3ecb.bin | 32 bytes |
| f1.txt | 5 bytes |
| f2.txt | 10 bytes |
| f3.txt | 16 bytes |

**cbc decryption and padding**

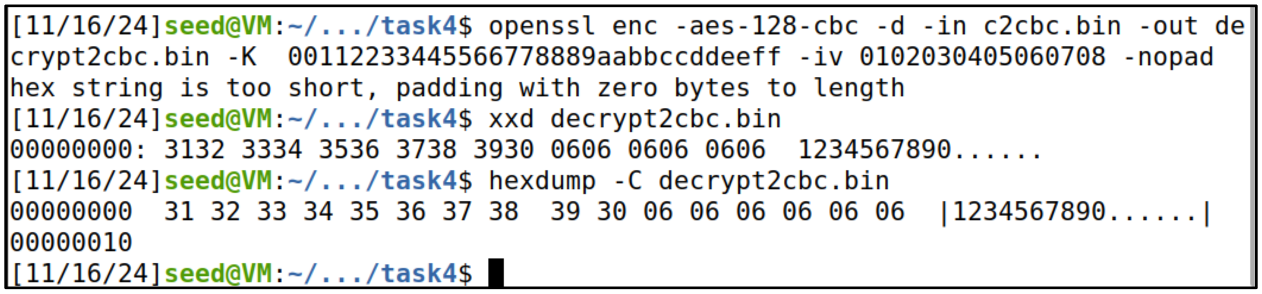
In cbc since the padding is added the sizes of files when encrypted became C1 (F1)= 16 bytes C2(F2)=16 bytes C3(F3)= 32 bytes.



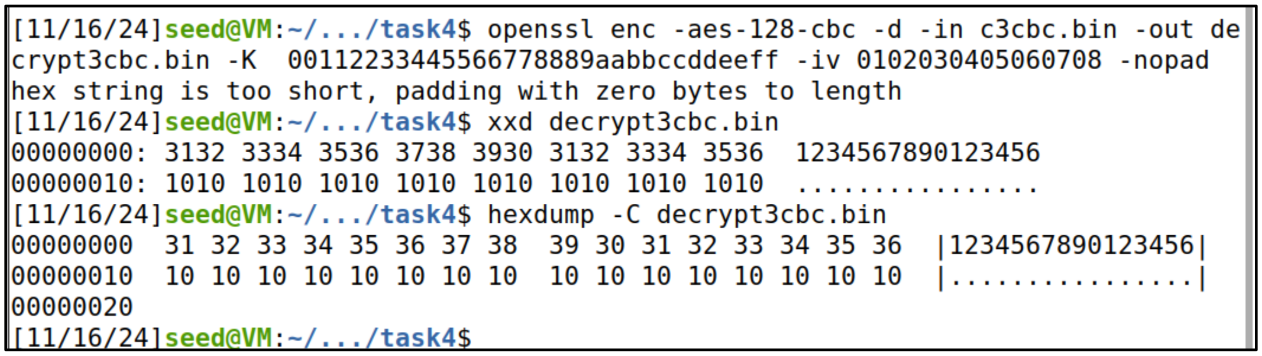
```
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-cbc -e -in f1.txt -out c1cbc
.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-cbc -e -in f2.txt -out c2cbc
.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-cbc -e -in f3.txt -out c3cbc
.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$
```



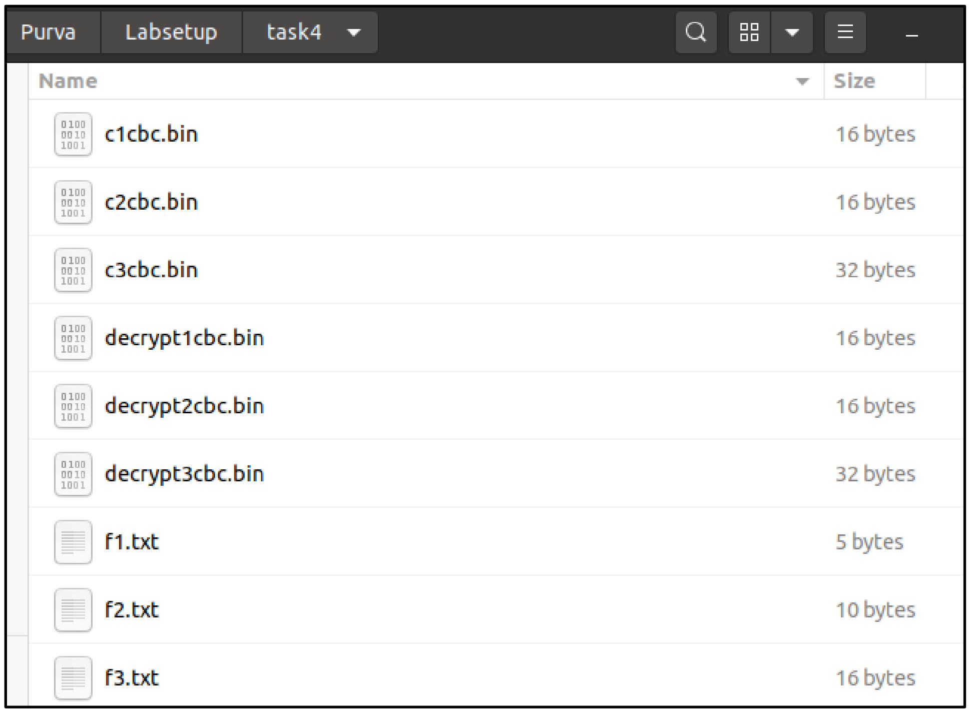
```
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-cbc -d -in c1cbc.bin -out de
crypt1cbc.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ xxd decrypt1cbc.bin
00000000: 3132 3334 350b 0b0b 0b0b 0b0b 0b0b 0b0b  12345...........
[11/16/24]seed@VM:~/.../task4$
```



```
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-cbc -d -in c2cbc.bin -out de
crypt2cbc.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ xxd decrypt2cbc.bin
00000000: 3132 3334 3536 3738 3930 0606 0606 0606  1234567890......
[11/16/24]seed@VM:~/.../task4$ hexdump -C decrypt2cbc.bin
00000000  31 32 33 34 35 36 37 38  39 30 06 06 06 06 06 06  |1234567890......|
00000010
[11/16/24]seed@VM:~/.../task4$
```



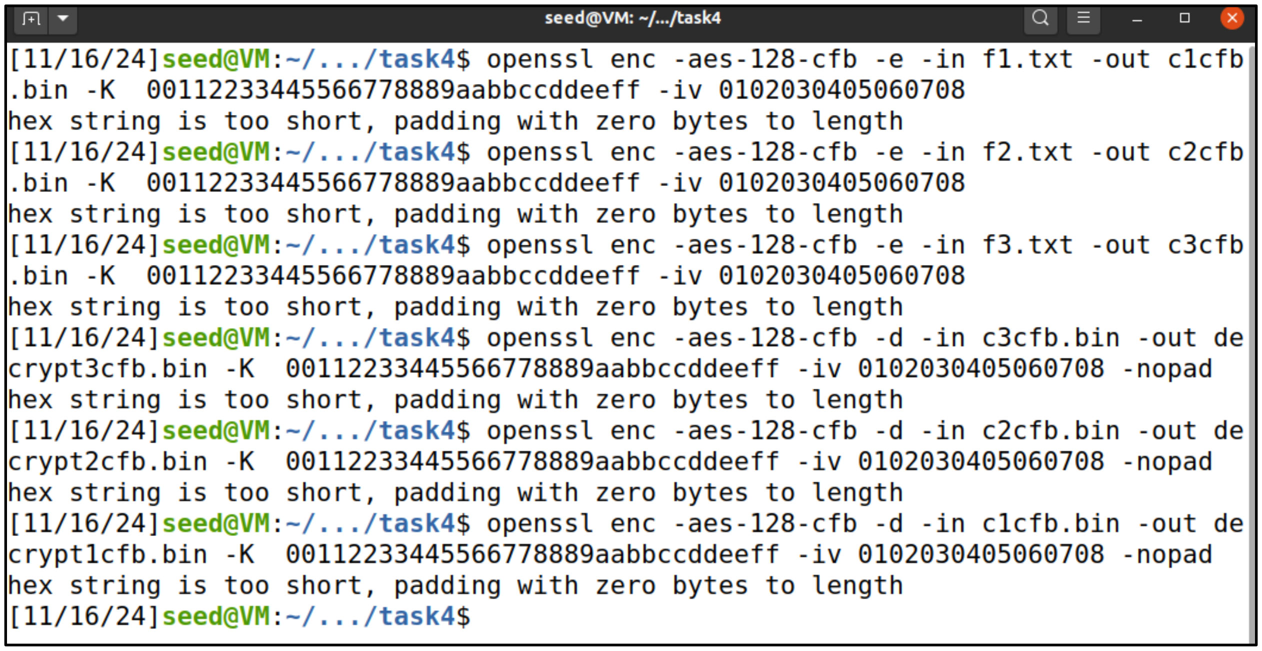
```
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-cbc -d -in c3cbc.bin -out de
crypt3cbc.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ xxd decrypt3cbc.bin
00000000: 3132 3334 3536 3738 3930 3132 3334 3536  1234567890123456
00000010: 1010 1010 1010 1010 1010 1010 1010 1010  ................
[11/16/24]seed@VM:~/.../task4$ hexdump -C decrypt3cbc.bin
00000000  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36  |1234567890123456|
00000010  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  |................|
00000020
[11/16/24]seed@VM:~/.../task4$
```



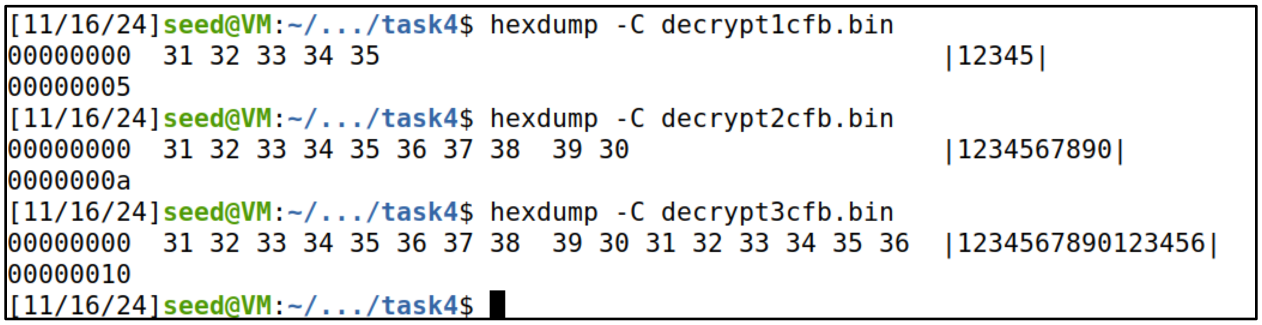
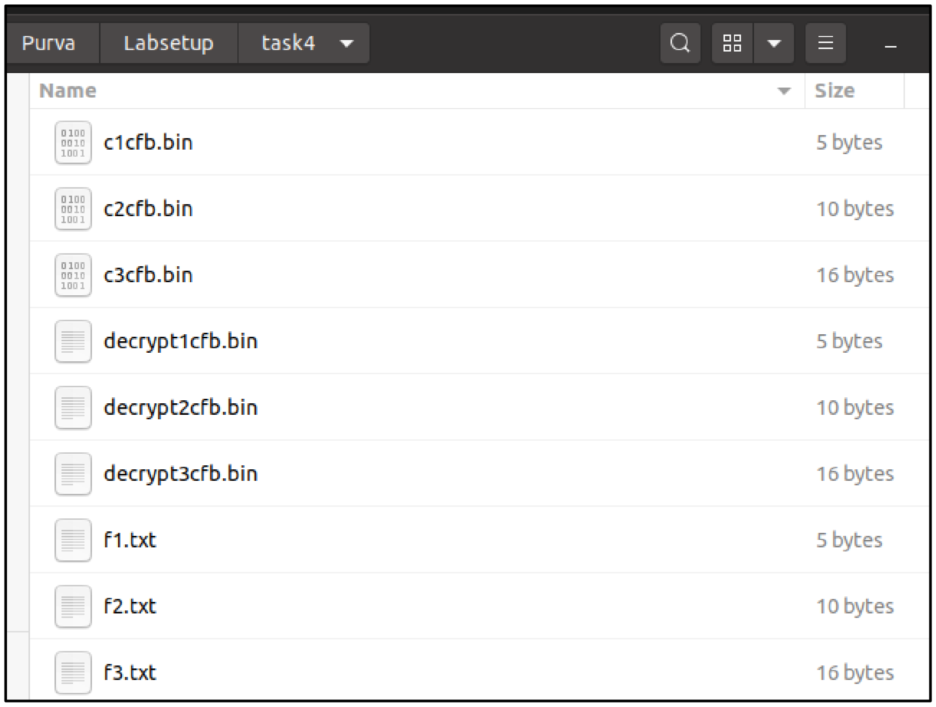
| Name | Size |
|---|---|
| c1cbc.bin | 16 bytes |
| c2cbc.bin | 16 bytes |
| c3cbc.bin | 32 bytes |
| decrypt1cbc.bin | 16 bytes |
| decrypt2cbc.bin | 16 bytes |
| decrypt3cbc.bin | 32 bytes |
| f1.txt | 5 bytes |
| f2.txt | 10 bytes |
| f3.txt | 16 bytes |

## <u>cfb decryption and padding</u>

In cfb since the padding is not added the sizes of files when encrypted remained same so C1 (F1)= 5 bytes C2(F2)=10 bytes C3(F3)= 16 bytes.



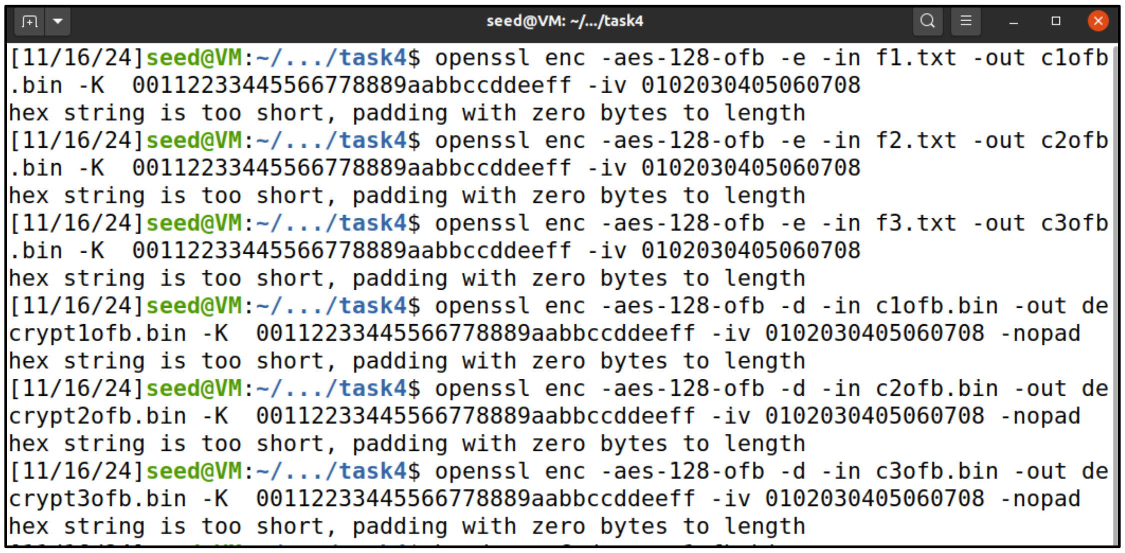
```
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-cfb -e -in f1.txt -out c1cfb
.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-cfb -e -in f2.txt -out c2cfb
.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-cfb -e -in f3.txt -out c3cfb
.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-cfb -d -in c3cfb.bin -out de
crypt3cfb.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-cfb -d -in c2cfb.bin -out de
crypt2cfb.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-cfb -d -in c1cfb.bin -out de
crypt1cfb.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$
```


```
[11/16/24]seed@VM:~/.../task4$ hexdump -C decrypt1cfb.bin
00000000  31 32 33 34 35                                    |12345|
00000005
[11/16/24]seed@VM:~/.../task4$ hexdump -C decrypt2cfb.bin
00000000  31 32 33 34 35 36 37 38  39 30                    |1234567890|
0000000a
[11/16/24]seed@VM:~/.../task4$ hexdump -C decrypt3cfb.bin
00000000  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36  |1234567890123456|
00000010
[11/16/24]seed@VM:~/.../task4$ 
```


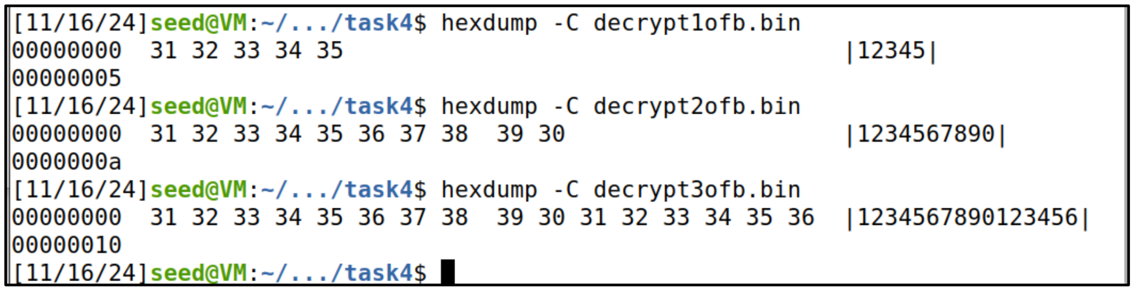
| Name | Size |
|---|---|
| c1cfb.bin | 5 bytes |
| c2cfb.bin | 10 bytes |
| c3cfb.bin | 16 bytes |
| decrypt1cfb.bin | 5 bytes |
| decrypt2cfb.bin | 10 bytes |
| decrypt3cfb.bin | 16 bytes |
| f1.txt | 5 bytes |
| f2.txt | 10 bytes |
| f3.txt | 16 bytes |

**ofb decryption and padding**

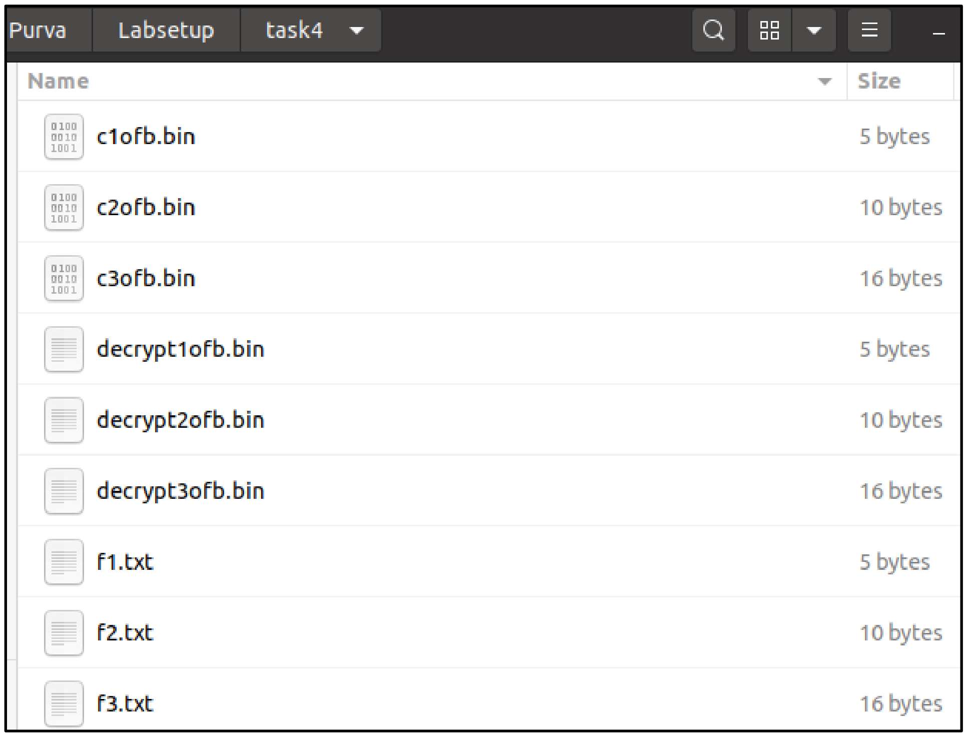
In ofb since the padding is not added the sizes of files when encrypted remained same so  C1 (F1)= 5 bytes C2(F2)=10 bytes  C3(F3)= 16 bytes.



```
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-ofb -e -in f1.txt -out c1ofb
.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-ofb -e -in f2.txt -out c2ofb
.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-ofb -e -in f3.txt -out c3ofb
.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-ofb -d -in c1ofb.bin -out de
crypt1ofb.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-ofb -d -in c2ofb.bin -out de
crypt2ofb.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
[11/16/24]seed@VM:~/.../task4$ openssl enc -aes-128-ofb -d -in c3ofb.bin -out de
crypt3ofb.bin -K  00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
hex string is too short, padding with zero bytes to length
```



```
[11/16/24]seed@VM:~/.../task4$ hexdump -C decrypt1ofb.bin
00000000  31 32 33 34 35                                    |12345|
00000005
[11/16/24]seed@VM:~/.../task4$ hexdump -C decrypt2ofb.bin
00000000  31 32 33 34 35 36 37 38  39 30                    |1234567890|
0000000a
[11/16/24]seed@VM:~/.../task4$ hexdump -C decrypt3ofb.bin
00000000  31 32 33 34 35 36 37 38  39 30 31 32 33 34 35 36  |1234567890123456|
00000010
[11/16/24]seed@VM:~/.../task4$
```



Purva  Labsetup  task4

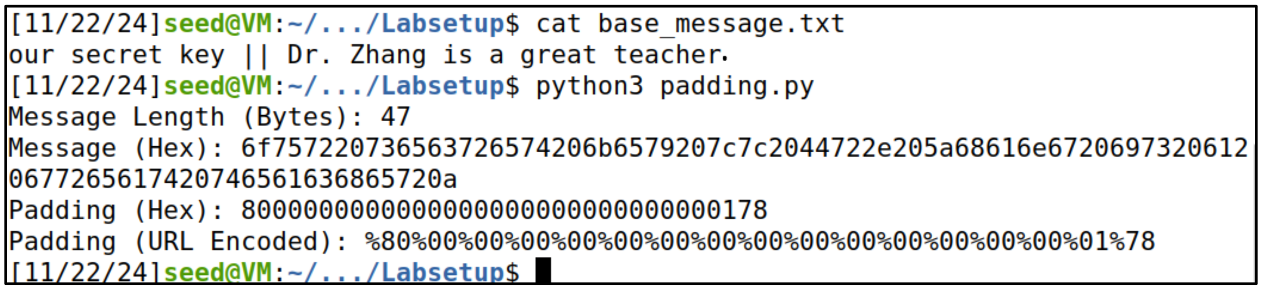
| Name | Size |
|---|---|
| c1ofb.bin | 5 bytes |
| c2ofb.bin | 10 bytes |
| c3ofb.bin | 16 bytes |
| decrypt1ofb.bin | 5 bytes |
| decrypt2ofb.bin | 10 bytes |
| decrypt3ofb.bin | 16 bytes |
| f1.txt | 5 bytes |
| f2.txt | 10 bytes |
| f3.txt | 16 bytes |

# Hash Length Extension Attack

## Task 2

This program is for getting for the padding for the original message:
"our secret key || Dr. Zhang is a great teacher."

```
with open("base_message.txt", "rb") as f:

    message = f.read(

message_length = len(message)

length_in_bits = message_length * 8

padding = b'\x80' + b'\x00' * ((56 - message_length % 64) - 1) + length_in_bits.to_bytes(8, 'big')

url_encoded_padding = ".join(f"%{byte:02x}" for byte in padding)

print("Message Length (Bytes):", message_length)

print("Message (Hex):", message.hex())

print("Padding (Hex):", padding.hex())

print("Padding (URL Encoded):", url_encoded_padding)
```

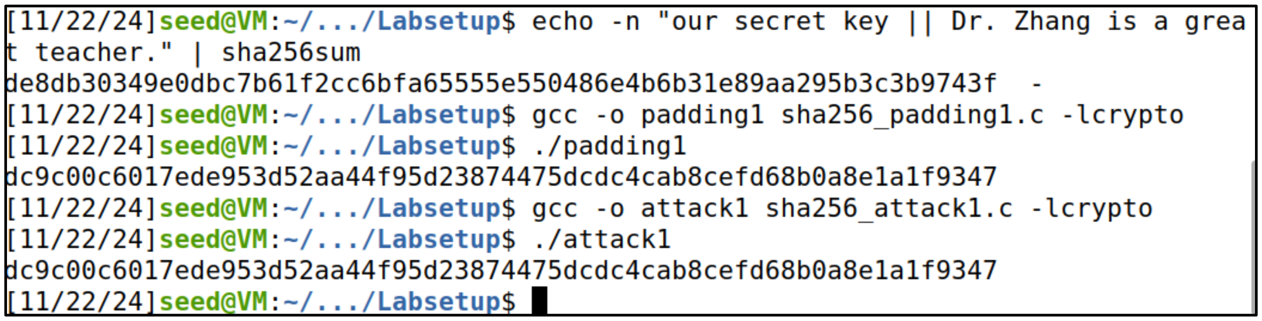


```
[11/22/24]seed@VM:~/.../Labsetup$ cat base_message.txt
our secret key || Dr. Zhang is a great teacher.
[11/22/24]seed@VM:~/.../Labsetup$ python3 padding.py
Message Length (Bytes): 47
Message (Hex): 6f757220736563726574206b6579207c7c2044722e205a68616e6720697320612
0677265617420746561636865720a
Padding (Hex): 8000000000000000000000000000000178
Padding (URL Encoded): %80%00%00%00%00%00%00%00%00%00%00%00%00%00%00%01%78
[11/22/24]seed@VM:~/.../Labsetup$ 
```

This program calculates the hash of the original message + padding + malicious message

<u>sha256_padding1.c</u>

```
#include <stdio.h>

#include <openssl/sha.h>

int main(int argc, const char *argv[])

{

  int i;
  unsigned char buffer[SHA256_DIGEST_LENGTH];
  SHA256_CTX c;
  SHA256_Init(&c);
  SHA256_Update(&c,
  "our secret key || Dr. Zhang is a great teacher."
  "\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00"
  "\x00\x00\x00\x00\x00\x01\x78"
  "Well, maybe not.", 64+16);
  SHA256_Final(buffer, &c);
  for(i = 0; i < 32; i++) {
    printf("%02x", buffer[i]);
  }
  printf("\n");
  return 0;
}
```



```
[11/22/24]seed@VM:~/.../Labsetup$ echo -n "our secret key || Dr. Zhang is a grea
t teacher." | sha256sum
de8db30349e0dbc7b61f2cc6bfa65555e550486e4b6b31e89aa295b3c3b9743f  -
[11/22/24]seed@VM:~/.../Labsetup$ gcc -o padding1 sha256_padding1.c -lcrypto
[11/22/24]seed@VM:~/.../Labsetup$ ./padding1
dc9c00c6017ede953d52aa44f95d23874475dcdc4cab8cefd68b0a8e1a1f9347
[11/22/24]seed@VM:~/.../Labsetup$ gcc -o attack1 sha256_attack1.c -lcrypto
[11/22/24]seed@VM:~/.../Labsetup$ ./attack1
dc9c00c6017ede953d52aa44f95d23874475dcdc4cab8cefd68b0a8e1a1f9347
[11/22/24]seed@VM:~/.../Labsetup$ 
```

**Task 3:**

This program is for the actual hash length extension attack without knowing the secret key.

So first I calculated the hash of "our secret key || Dr. Zhang is a great teacher."

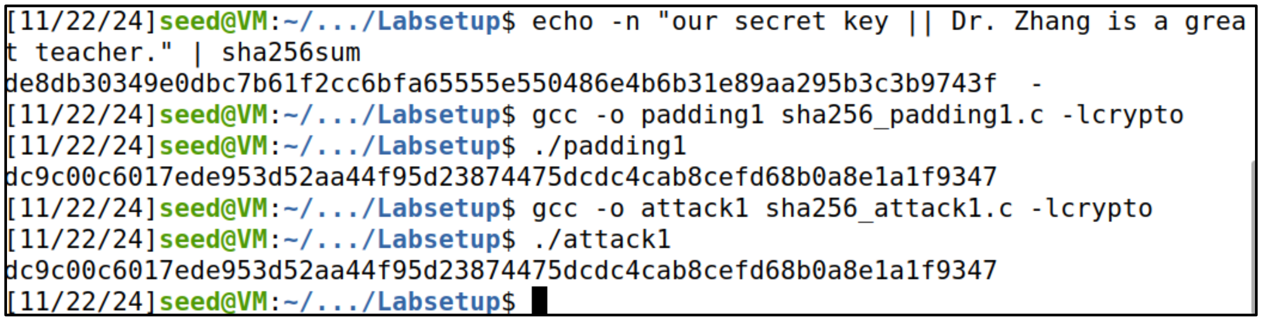
Then put that hash in this below given program.

Then added the malicious message to this hash and that's how I got the same hash as the first program and that's how the attack is successful.

sha256_attack.c

```
#include <stdio.h>

#include <arpa/inet.h>

#include <openssl/sha.h>

int main(int argc, const char *argv[])

{

 int i;

 unsigned char buffer[SHA256_DIGEST_LENGTH];

 SHA256_CTX c;

 SHA256_Init(&c);

 for(i=0; i<64; i++)

  SHA256_Update(&c, "*", 1);

 c.h[0] = htole32(0xde8db303);

 c.h[1] = htole32(0x49e0dbc7);

 c.h[2] = htole32(0xb61f2cc6);

 c.h[3] = htole32(0xbfa65555);

 c.h[4] = htole32(0xe550486e);

 c.h[5] = htole32(0x4b6b31e8);

 c.h[6] = htole32(0x9aa295b3);

 c.h[7] = htole32(0xc3b9743f);

 // Append malicious message

 SHA256_Update(&c, "Well, maybe not.", 16);
```

```
  SHA256_Final(buffer, &c);

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

    printf("%02x", buffer[i]);

  }

  printf("\n");

  return 0;

}
```



```
[11/22/24]seed@VM:~/.../Labsetup$ echo -n "our secret key || Dr. Zhang is a grea
t teacher." | sha256sum
de8db30349e0dbc7b61f2cc6bfa65555e550486e4b6b31e89aa295b3c3b9743f  -
[11/22/24]seed@VM:~/.../Labsetup$ gcc -o padding1 sha256_padding1.c -lcrypto
[11/22/24]seed@VM:~/.../Labsetup$ ./padding1
dc9c00c6017ede953d52aa44f95d23874475dcdc4cab8cefd68b0a8e1a1f9347
[11/22/24]seed@VM:~/.../Labsetup$ gcc -o attack1 sha256_attack1.c -lcrypto
[11/22/24]seed@VM:~/.../Labsetup$ ./attack1
dc9c00c6017ede953d52aa44f95d23874475dcdc4cab8cefd68b0a8e1a1f9347
[11/22/24]seed@VM:~/.../Labsetup$ 
```