Istanbul Technical University

Informatics Institute

Cybersecurity Engineering and Cryptography

Secret-Key Encryption Lab

Detailed Report

Name: Halis Taha Şahin
ID: 707201006
E-Mail: sahinh20@itu.edu.tr
Semester: Fall 2020/2021
Fachsemester: 01

Date: December 8, 2020

Contents

1 Task 1: Frequency Analysis **1**

2 Task 2: Encryption using Different Ciphers and Modes **3**

3 Task 3: Encryption Mode – ECB vs. CBC **4**

4 Task 4: Padding **5**

5 Task 5: Error Propagation – Corrupted Cipher Text **8**

6 Task 6: Initial Vector (IV) and Common Mistakes **12**

6.1 Task 6.1 12

6.2 Task 6.2. Common Mistake: Use the Same IV 13

6.3 Task 6.3. Common Mistake: Use a Predictable IV 14

7 Task 7: Programming using the Crypto Library **15**

1 Task 1: Frequency Analysis

First, I downloaded the cipher text and performed frequency analysis.

$ wget https://seedsecuritylabs.org/Labs_16.04/Crypto/Crypto_Encryption/files/ciphertext.txt

Cipher text:

```
halis@VM:/home/seed/crypto/task1$ head -n 2 ciphertext.txt
ytn xqavhq yzhu  xu qzupvd ltmat qnncq vgxzy hmrty vbynh ytmq ixur qyhvurn
vlvhpq yhme ytn gvrrnh bnniq imsn v uxuvrnuvhmvu yxx
halis@VM:/home/seed/crypto/task1$
```

Figure 1: Cipher Text

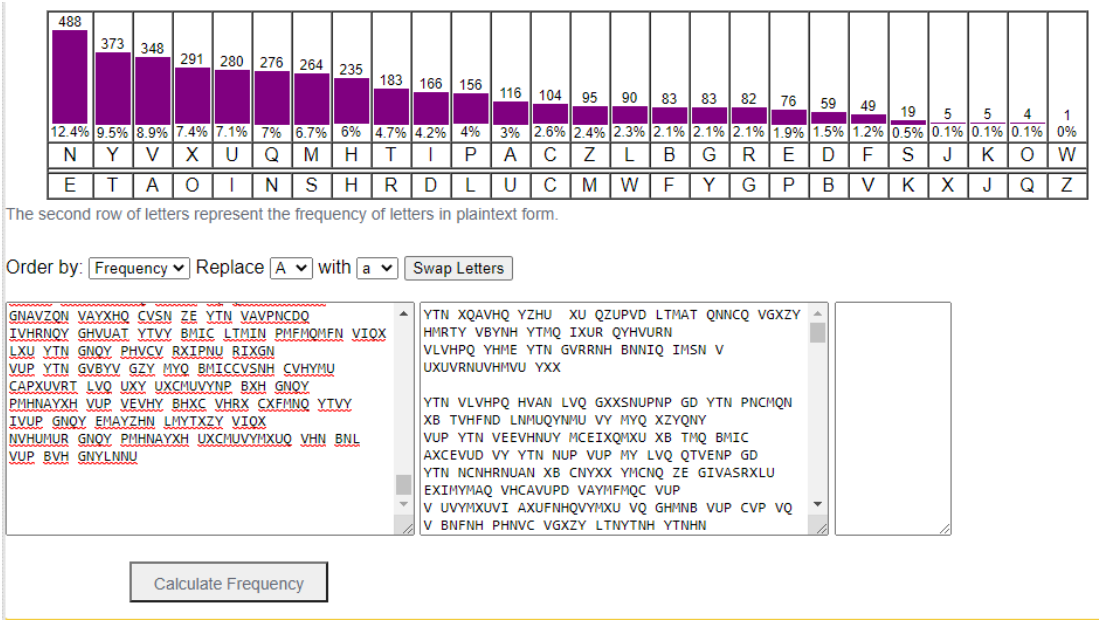

Figure 2: Frequency Analysis

Then, using the frequency analysis we obtained, I changed the letters here with the most frequently used letters in English.

```
halis@VM:/home/seed/crypto/task1$ tr 'nyvxuqmhtipaczlgbredfskjow' 'etaoinshrdlucmwfy
gpbvkxjqz' < ciphertext.txt > decipher.txt
halis@VM:/home/seed/crypto/task1$ head -n 2 decipher.txt
tre onuahn tmhi  oi nmilab wrsur neecn afomt hsgrt ayteh trsn doig nthaige
awahln thsp tre faggeh yeedn dske a ioiageiahsai too
halis@VM:/home/seed/crypto/task1$
```

After making the first edit, I will make changes step by step to find the English words. The first word seems to be "the". Fourth word seeems to be "on".

```
halis@VM:/home/seed/crypto/task1$ tr 'nyvxuqmhtipaczlgbredfskjow' 'etaonisrhdlucmwfy
gpbvkxjqz' < ciphertext.txt > decipher.txt
halis@VM:/home/seed/crypto/task1$ head -n 2 decipher.txt
the oiuari tmrn  on imnlab whsuh ieeci afomt rsght ayter thsi dong itrange
awarli trsp the fagger yeedi dske a nonagenarsan too
halis@VM:/home/seed/crypto/task1$
```

```
halis@VM:/home/seed/crypto/task1$ tr 'nyvxuqmhtipaczlgbredfskjow' 'etaonisrhdlucmwyf
gpbvkxqjz' < ciphertext.txt > decipher.txt
halis@VM:/home/seed/crypto/task1$ head -n 2 decipher.txt
the oiuari tmrn  on imnlab whsuh ieeci ayomt rsght after thsi dong itrange
awarli trsp the yagger feedi dske a nonagenarsan too
halis@VM:/home/seed/crypto/task1$ tr 'nyvxuqmhtipaczlgbredfskjow' 'etaonsirhdlucmwyf
gpbvkxqjz' < ciphertext.txt > decipher.txt
halis@VM:/home/seed/crypto/task1$ head -n 2 decipher.txt
the osuars tmrn  on smnlab whiuh seecs ayomt right after this dong strange
awarls trip the yagger feeds dike a nonagenarian too
halis@VM:/home/seed/crypto/task1$
```

```
halis@VM:/home/seed/crypto/task1$ tr 'nyvxuqmhtipaczlgbredfskjow' 'etaonsirhldcmuwbf
gpyvkxqjz' < ciphertext.txt > decipher.txt
halis@VM:/home/seed/crypto/task1$ head -n 2 decipher.txt
the oscars turn  on sunday which seems about right after this long strange
awards trip the bagger feels like a nonagenarian too
halis@VM:/home/seed/crypto/task1$
```

I was able to get plain text at the end of the edits.

Final command must be like this.

$ tr 'nyvxuqmhtipaczlgbredfskjow' 'etaonsirhldcmuwbfgpyvkxqjz' < ciphertext.txt > decipher.txt

We can find key after this stage.

$ tr 'abcdefghijklmnopqrstuvwxyz' 'cfmypvbrlqxwiejdsgkhnazotu' < ciphertext.txt > decipher.txt

Key: "cfmypvbrlqxwiejdsgkhnazotu"

```
the oscars turn  on sunday which seems about right after this long strange
awards trip the bagger feels like a nonagenarian too

the awards race was bookended by the demise of harvey weinstein at its outset
and the apparent implosion of his film company at the end and it was shaped by
the emergence of metoo times up blackgown politics armcandy activism and
a national conversation as brief and mad as a fever dream about whether there
ought to be a president winfrey the season didnt just seem extra long it was
extra long because the oscars were moved to the first weekend in march to
avoid conflicting with the closing ceremony of the winter olympics thanks
pyeongchang

one big question surrounding this years academy awards is how or if the
ceremony will address metoo especially after the golden globes which became
a jubilant comingout party for times up the movement spearheaded by
powerful hollywood women who helped raise millions of dollars to fight sexual
harassment around the country

signaling their support golden globes attendees swathed themselves in black
sported lapel pins and sounded off about sexist power imbalances from the red
carpet and the stage on the air e was called out about pay inequity after
its former anchor catt sadler quit once she learned that she was making far
less than a male cohost and during the ceremony natalie portman took a blunt
and satisfying dig at the allmale roster of nominated directors how could
that be topped
```

Figure 3: Final Plaintext

2 Task 2: Encryption using Different Ciphers and Modes

I created an executable file for this task and performed encryption and decryption with the help of this file. Also I used the "words.txt" at the lab web site. The encryption modes I use are as follows:

- aes-128-cbc
- aes-128-ecb
- aes-128-ofb

Commands:

```
openssl enc -aes-128-cbc -e -in words.txt -out cipher1.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-ecb -e -in words.txt -out cipher2.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-256-ofb -e -in words.txt -out cipher3.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

3 Task 3: Encryption Mode – ECB vs. CBC

In this task, we will encrypt the "bmp" extension image with two different cipher (ecb, cbc) and look at the difference between them. First I downloaded the picture.

$ wget https://seedsecuritylabs.org/Labs_16.04/Crypto_Encryption/files/pic_original.bmp

Then I encrypted the picture using the password "123456".

```
openssl enc -aes-128-ecb -e -in pic_original.bmp -out pic_ecb.bmp -k 123654
openssl enc -aes-128-cbc -e -in pic_original.bmp -out pic_cbc.bmp -k 123654
```

Then I had to view the pictures I encrypted. For this, I corrected the header parts as follows.

```
// Get original header from ".bmp"
head -c 54 pic_original.bmp  > header
// Get bodies from encrypted images
tail -c +55 pic_ecb.bmp > ecb_body
tail -c +55 pic_cbc.bmp > cbc_body
// Merge header and bodies
cat header ecb_body > new_ecb.bmp
cat header cbc_body > new_cbc.bmp
```

After performing these operations I viewed the pictures. The differences can be seen clearly in the pictures. It is obvious that there is a confidentiality breach in the ECB encryption method. Therefore, the use of ECB encryption will be against the confidentiality policy of cyber security. In the CBC encryption method, no information about the original image is available.

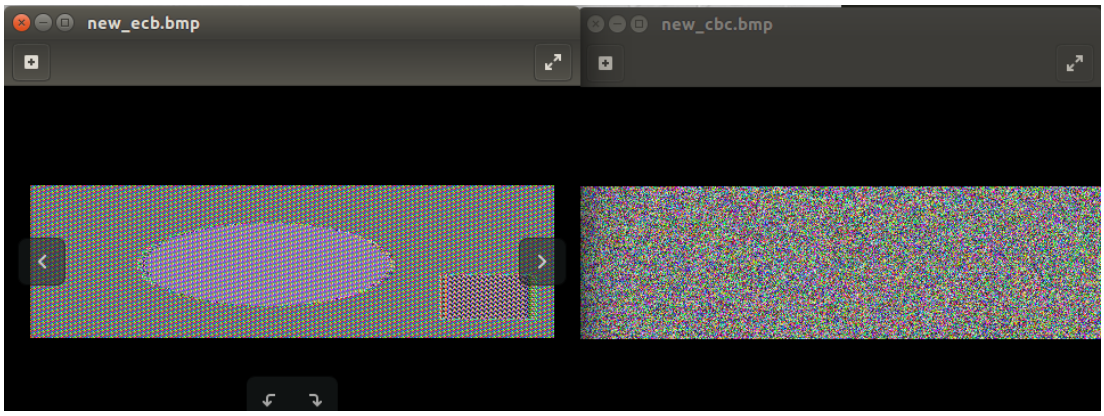

Figure 4: Encrypted images.

4 Task 4: Padding

In this task we will evaluate the padding in encryption modes. First I created three files with sizes 5, 10 and 16 bytes.

```
echo -n "12345" > f1.txt
echo -n "1234567891" > f2.txt
echo -n "1234567891234567" > f3.txt
```

Then I encrypted and decrypted all three files using ECB, CBC, CFB and OFB.

```
openssl enc -aes-128-cbc -e -in f1.txt -out enc_cbc_f1.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-cbc -e -in f2.txt -out enc_cbc_f2.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-cbc -e -in f3.txt -out enc_cbc_f3.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc -aes-128-cbc -d -in enc_cbc_f1.bin -out dec_cbc_1.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
openssl enc -aes-128-cbc -d -in enc_cbc_f2.bin -out dec_cbc_2.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
openssl enc -aes-128-cbc -d -in enc_cbc_f3.bin -out dec_cbc_3.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

openssl enc -aes-128-ecb -e -in f1.txt -out enc_ecb_f1.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-ecb -e -in f2.txt -out enc_ecb_f2.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-ecb -e -in f3.txt -out enc_ecb_f3.bin \
```

```
-K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc -aes-128-ecb -d -in enc_ecb_f1.bin -out dec_ecb_1.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
openssl enc -aes-128-ecb -d -in enc_ecb_f2.bin -out dec_ecb_2.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
openssl enc -aes-128-ecb -d -in enc_ecb_f3.bin -out dec_ecb_3.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

openssl enc -aes-128-cfb -e -in f1.txt -out enc_cfb_f1.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-cfb -e -in f2.txt -out enc_cfb_f2.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-cfb -e -in f3.txt -out enc_cfb_f3.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc -aes-128-cfb -d -in enc_cfb_f1.bin -out dec_cfb_1.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
openssl enc -aes-128-cfb -d -in enc_cfb_f2.bin -out dec_cfb_2.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
openssl enc -aes-128-cfb -d -in enc_cfb_f3.bin -out dec_cfb_3.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

openssl enc -aes-128-ofb -e -in f1.txt -out enc_ofb_f1.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-ofb -e -in f2.txt -out enc_ofb_f2.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
openssl enc -aes-128-ofb -e -in f3.txt -out enc_ofb_f3.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc -aes-128-ofb -d -in enc_ofb_f1.bin -out dec_ofb_1.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
openssl enc -aes-128-ofb -d -in enc_ofb_f2.bin -out dec_ofb_2.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
openssl enc -aes-128-ofb -d -in enc_ofb_f3.bin -out dec_ofb_3.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad
```

As a result of this process, I saw that there is no padding in ECB and CBC encryption modes. When I researched, I learned that padding is mandatory for these encryption modes. Because to perform encryption with a block cipher in ECB or CBC mode the length of the input to be encrypted must be an exact multiple of the block length B in bytes. If we decrypt it with the "-nopad" option. Files of 5, 10 and 16 bytes will have values of 16, 16 and 32 bytes.

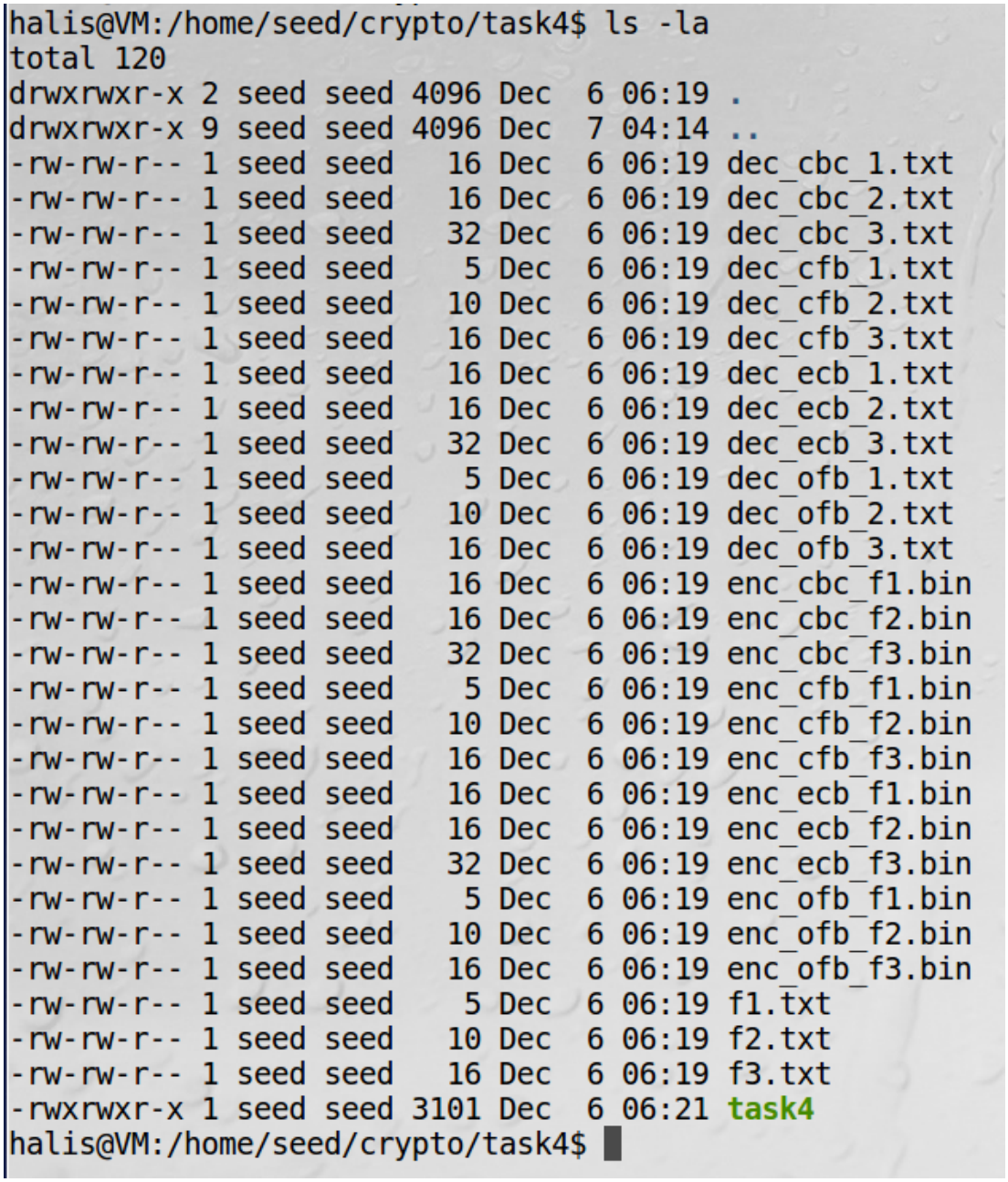

Figure 5: Encrypted files.

For other modes of encryption, OFB and CFB, padding is not required. In these cases the ciphertext is always the same length as the plaintext, and a padding method is not applicable.

With the "-nopad" option, we can easily see the padding in decrypted files. (0x0b, 0x06, 0x10 are used for padding.)

```
halis@VM:/home/seed/crypto/task4$ hexdump -C dec_cbc_1.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345...........|
00000010
halis@VM:/home/seed/crypto/task4$ hexdump -C dec_cbc_2.txt
00000000  31 32 33 34 35 36 37 38  39 31 06 06 06 06 06 06  |1234567891......|
00000010
halis@VM:/home/seed/crypto/task4$ hexdump -C dec_cbc_3.txt
00000000  31 32 33 34 35 36 37 38  39 31 32 33 34 35 36 37  |1234567891234567|
00000010  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  |................|
00000020
halis@VM:/home/seed/crypto/task4$
```

Figure 6: Padding - CBC Mode

```
halis@VM:/home/seed/crypto/task4$ hexdump -C dec_ecb_1.txt
00000000  31 32 33 34 35 0b 0b 0b  0b 0b 0b 0b 0b 0b 0b 0b  |12345...........|
00000010
halis@VM:/home/seed/crypto/task4$ hexdump -C dec_ecb_2.txt
00000000  31 32 33 34 35 36 37 38  39 31 06 06 06 06 06 06  |1234567891......|
00000010
halis@VM:/home/seed/crypto/task4$ hexdump -C dec_ecb_3.txt
00000000  31 32 33 34 35 36 37 38  39 31 32 33 34 35 36 37  |1234567891234567|
00000010  10 10 10 10 10 10 10 10  10 10 10 10 10 10 10 10  |................|
00000020
halis@VM:/home/seed/crypto/task4$
```

Figure 7: Padding - ECB Mode

5 Task 5: Error Propagation – Corrupted Cipher Text

In this task we will examine the error propagation in encryption modes. First of all, I created a file with a size of 1000 bytes.

$ python -c "print '1234567890'*99 +'123456789'" > file.txt

```
halis@VM:/home/seed/crypto/task5$ python -c "print '1234567890'*99 +'123456789'"
 > file.txt
halis@VM:/home/seed/crypto/task5$ ls -la
total 28
drwxrwxr-x 2 seed seed 4096 Dec  8 01:59 .
drwxrwxr-x 9 seed seed 4096 Dec  7 04:14 ..
-rw-rw-rw- 1 seed seed 1000 Dec  6 06:43 dec.txt
-rw-rw-rw- 1 seed seed  261 Dec  6 06:38 difference.py
-rw-rw-rw- 1 seed seed 1000 Dec  8 02:00 file.txt
-rw-rw-rw- 1 seed seed 1008 Dec  6 06:43 output.bin
-rwxrwxrwx 1 seed seed  289 Dec  6 06:42 task5
halis@VM:/home/seed/crypto/task5$
```

Figure 8: Make 1000 byte file.

First, I encrypted the file we created with ECB, CBC, CFB and OFB.

```
openssl enc  -aes-128-ecb -e -in file.txt -out output_ecb.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc  -aes-128-cbc -e -in file.txt -out output_cbc.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc  -aes-128-cfb -e -in file.txt -out output_cfb.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc  -aes-128-ofb -e -in file.txt -out output_ofb.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

Later, I changed the 55th bytes in four encrypted files.

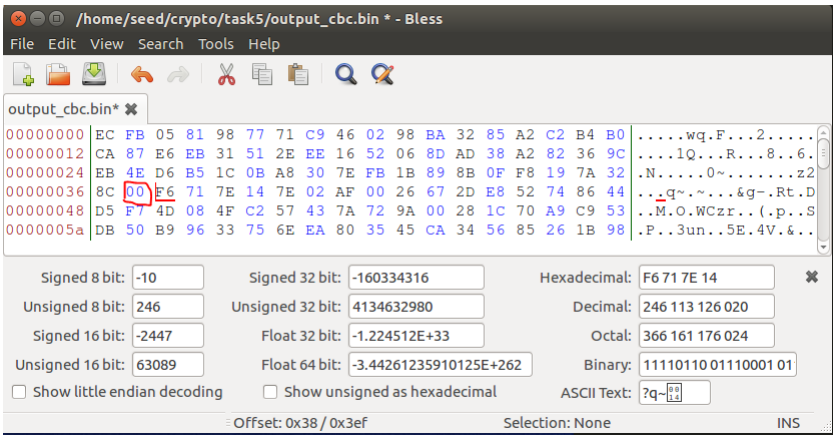

Figure 9: output_cbc.bin

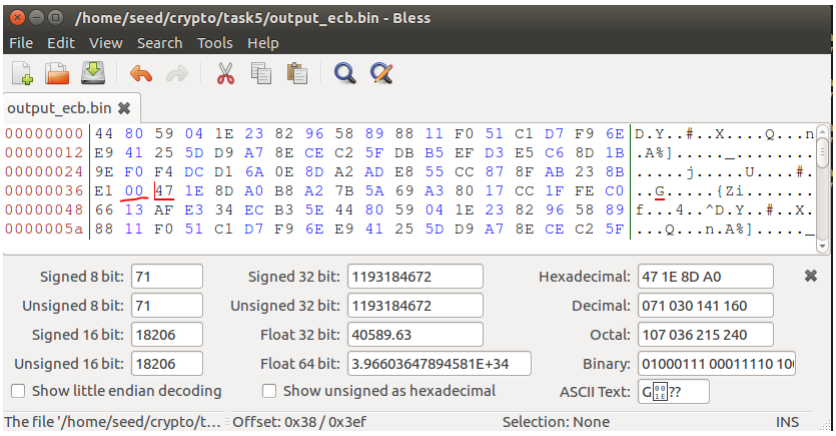

Figure 10: output_ecb.bin

Then I decrypted the files I made changes and calculated the error propagations.

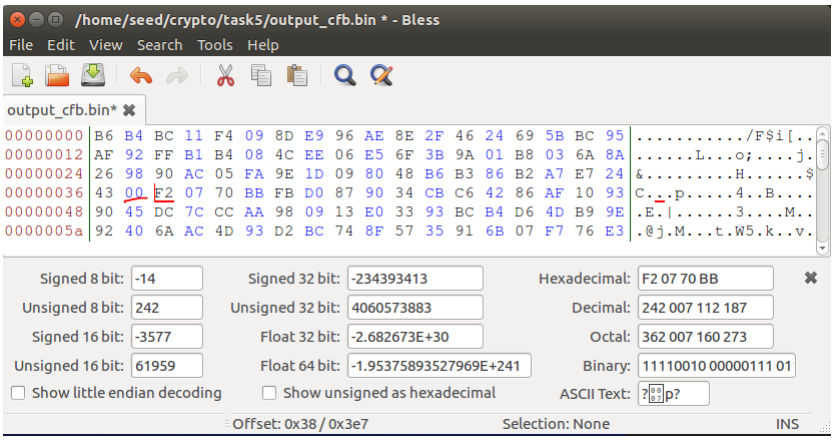

Figure 11: output_cfb.bin

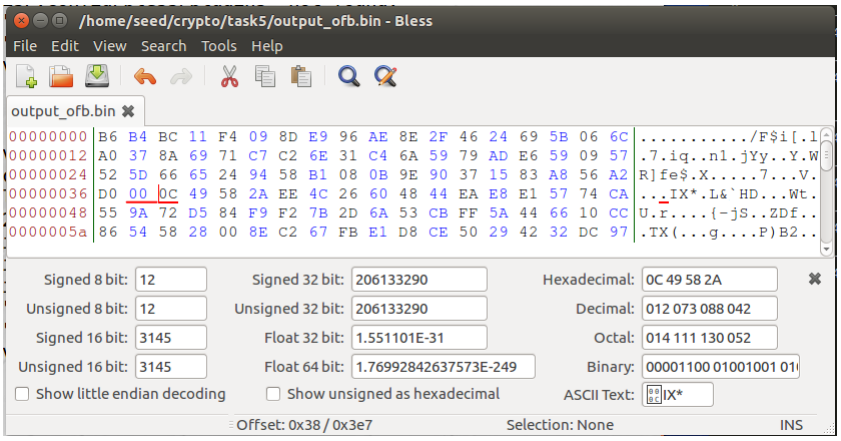

Figure 12: output_ofb.bin

```
openssl enc  -aes-128-ecb -d -in output_ecb.bin -out dec_ecb.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc  -aes-128-cbc -d -in output_cbc.bin -out dec_cbc.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc  -aes-128-cfb -d -in output_cfb.bin -out dec_cfb.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc  -aes-128-ofb -d -in output_ofb.bin -out dec_ofb.txt \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708
```

I wrote a code like this to calculate the error propagation.

```
#!/usr/bin/python3
with open('file.txt', 'rb') as f:
    f1 = f.read()
with open('dec_cbc.txt', 'rb') as f:
    f2 = f.read()
with open('dec_ecb.txt', 'rb') as f:
    f3 = f.read()
with open('dec_cfb.txt', 'rb') as f:
    f4 = f.read()
with open('dec_ofb.txt', 'rb') as f:
    f5 = f.read()

res = 0
for i in range(min(len(f1), len(f2))):
    if f1[i] != f2[i]:
        res += 1
print("CBC : "+str(res+abs(len(f1)-len(f2))))

res = 0
for i in range(min(len(f1), len(f3))):
    if f1[i] != f3[i]:
        res += 1
print("ECB : "+str(res+abs(len(f1)-len(f3))))

res = 0
for i in range(min(len(f1), len(f4))):
    if f1[i] != f4[i]:
        res += 1
print("CFB : "+str(res+abs(len(f1)-len(f4))))

res = 0
for i in range(min(len(f1), len(f5))):
    if f1[i] != f5[i]:
        res += 1
print("OFB : "+str(res+abs(len(f1)-len(f5))))
```

The output of the code was as follows.

Cipher Mode	Error Propagation
OFB	1
ECB	16
CBC	17
CFB	17

Table 1: Error propagation values.

```
halis@VM:/home/seed/crypto/task5$ sudo python difference.py
CBC : 17
ECB : 16
CFB : 17
OFB : 1
halis@VM:/home/seed/crypto/task5$
```

Figure 13: output of difference.py

Error propagation rates can vary according to the methods of encryption methods.

6 Task 6: Initial Vector (IV) and Common Mistakes

In this task, we will see how the initial vector selection affects the encryption results and what kind of vulnerabilities it causes.

6.1 Task 6.1

I have created a text file for this task. I first encrypted this file using the CBC encryption method. Then I encrypted it using the same initial vector and encrypted it using a different initial vector. The results were as follows.

```
openssl enc  -aes-128-cbc -e -in task6.txt -out output.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc  -aes-128-cbc -e -in task6.txt -out output2.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060708

openssl enc  -aes-128-cbc -e -in task6.txt -out output3.bin \
-K 00112233445566778889aabbccddeeff -iv 0102030405060709
```

If we use the same initial vector, we will get the same result. However, if we change the initial vector, we will get different results.

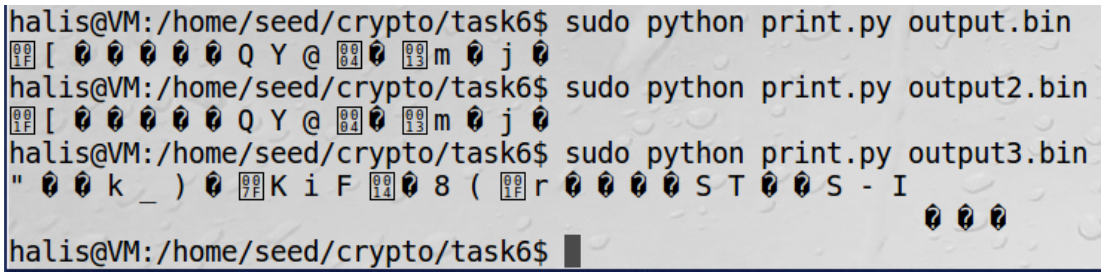

Figure 14: Different initial vectors.

6.2 Task 6.2. Common Mistake: Use the Same IV

For OFB mode, If the key and IV keep unchanged, known-plaintext attack is feasible. We define a method to obtain P2 as follows.

$$P2 = P1 \oplus C1 \oplus C2 \tag{1}$$

We can obtain the situation given in Equation 1 as follows.

```
from sys import argv

p1 = bytearray("This is a known message!",encoding='utf-8')
c1 = bytearray.fromhex("a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159")
c2 = bytearray.fromhex("bf73bcd3509299d566c35b5d450337e1bb175f903fafc159")

p2 = bytearray(x ^ y ^ z for x, y, z in zip(p1, c1, c2))

print(p2.decode('utf-8'))
```

If we run the code, the output will be as follows.

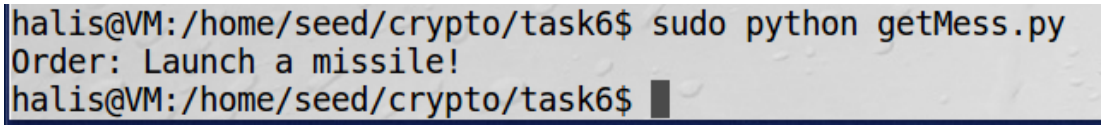

Figure 15: Output of getMess.py

P2 = "Order: Launch a missile!"

For CFB mode, it is the same situation for the initial block.(Apply the XOR process) However, if the key remains secret, the following parts of ciphertext will not be revealed.

6.3 Task 6.3. Common Mistake: Use a Predictable IV

From the previous tasks, we now know that IVs cannot repeat. Another important requirement on IV is that IVs need to be unpredictable for many schemes, i.e., IVs need to be randomly generated.

$$P2 = P1 \oplus IV \oplus IVNext \tag{2}$$

This task is similar to the previous task. However, what we're trying to do here is a little different. We know that Bob's message is "Yes" or "No". We also know the predictable initial vector values.

For this process, I initially guess the P1 message to be "Yes".

P1 = "Yes"

```
from sys import argv

p1 = bytearray("Yes", encoding='utf-8')
padding = 16 - len(p1) \% 16  # padding to match the block size as 128 bit
p1.extend([padding]*padding)
IV = bytearray.fromhex("31323334353637383930313233343536")
IV_NEXT = bytearray.fromhex("31323334353637383930313233343537")
p2 = bytearray(x ^ y ^ z for x, y, z in zip(p1, IV, IV_NEXT))
print(p2.decode('utf-8'), end='')
```

The output of the program is as follows.

```
halis@VM:/home/seed/crypto/task6$ sudo python3 getP.py > p2
halis@VM:/home/seed/crypto/task6$ cat p2
Yes
halis@VM:/home/seed/crypto/task6$
```

Figure 16: Output of getP.py

At this stage we will encrypt P2 and compare the result C2 with C1.

```
openssl enc  -aes-128-cbc  -e -in p2 -out c2 \
-K 00112233445566778899aabbccddeeff -iv 31323334353637383930313233343537
```

```
halis@VM:/home/seed/crypto/task6$ xxd -p c2
bef65565572ccee2a9f9553154ed94983402de3f0dd16ce789e5475779ac
a405
halis@VM:/home/seed/crypto/task6$ xxd -p c1
bef65565572ccee2a9f9553154ed9498
halis@VM:/home/seed/crypto/task6$
```

Figure 17: Comparison between C1 and C2

We found P2 by guessing P1 "Yes". Then we found C2 by encrypting P2. Since the first 16 bytes of C1 and C2 are the same, we get the result that we guessed correctly. If we guessed "No." C1 and C2 would be different and we would realize that we had guessed wrong.

P1 = "Yes" , true hypothesis.

7 Task 7: Programming using the Crypto Library

In the last task, we are asked to find the key by doing brute force on the cipher text. For this, I will use the "words.txt" given on the site. Firstly download the text file.

$ wget https://seedsecuritylabs.org/Labs_16.04/Crypto_Encryption/files/words.txt

Then I tried to get the cipher text given in the PDF file by reading all lines in the "words.txt" file one by one. I specified my work in "myenc.c" file.

The general operation of the program is as follows:

- Read the "words.txt" line by line
- Complete the key with character
- Control the key is true or not
- If key is true, exit the program, else read next line from file.

```
// Initialize the libraries
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <openssl/ssl.h>
#include <string.h>
#include <stdio.h>
#include <string.h>

// Initialize the plain text.
unsigned char *plain_text = "This is a top secret.";
// Initialize the cipher text.
unsigned char cipher_text[] = {0x76,0x4a,0xa2,0x6b,0x55,0xa4,0xda,0x65\
,0x4d,0xf6,0xb1,0x9e,0x4b,0xce,0x00,0xf4,0xed,0x05,0xe0,0x93,0x46,0xfb\
,0x0e,0x76,0x25,0x83,0xcb,0x7d,0xa2,0xac,0x93,0xa2};

// Initialize the methods
int isSame(unsigned char *cipher, int cipher_len, unsigned char *buff, int buff_len);
int isKey(unsigned char *key);

int main(void) {
        unsigned char key[16];        // Initialize the key
        FILE *wordlist = fopen("words.txt", "r");        // Reads words.txt
        while(fgets(key, 16, wordlist) != 0){        // Read file line by line
                key[strlen(key)-1]='\0';        // Remove the new line symbol
                int key_len = strlen(key);        // Get length of key
                if(key_len < 16){// Complete the key size with the #.
                        int len = key_len;
                        while(len < 16){
                                key[len] = 0x23; // 0x23 = #
                                len++;
                        }
                }
                int t = isKey(key);        // Control the the key is true
                if(t == 1){ // If find exit else continue
                        printf("Key is: \%s \n", key);
                        break;
                }
        }
        fclose(wordlist);
        return(0);
}
// Control the key is true or not
int isKey(unsigned char *key){
        int out_size = 1024 +EVP_MAX_BLOCK_LENGTH;
        unsigned char out[out_size];
        int out_len = 0;
```

```
        int plain_text_len = strlen(plain_text);
        int buff_out_len;
        // Initialize the initial vector
        unsigned char iv[] = {0xaa,0xbb, 0xcc, 0xdd,0xee, 0xff, 0x00,\
        0x99,0x88,0x77,0x66,0x55,0x44,0x33,0x22,0x11};
        // Initialize the base methods and control key and iv lengths
        ..
        EVP_CipherFinal_ex(&ctx, out + buff_out_len, &buff_out_len);
        ..
        int compare = isSame(cipher_text, sizeof(cipher_text), out, out_len);
        return compare;
}

// Control byte by byte for original cipher text with different keys.
int isSame(unsigned char *cipher, int cipher_len, unsigned char *buff, int buff_len){
        // Base condition
        if(cipher_len <= 0 || buff_len <= 0 || cipher_len != buff_len) return 0;
        for(int i =0;i < cipher_len; i++)        // Control the every byte
                if(cipher[i] != buff[i])
                        return 0;   // If different return 0
        return 1;   // Return 1 if they are same
}
```

If I run the program, its output will look like this.

```
halis@VM:/home/seed/crypto/task7$ sudo gcc myenc.c -o myenc -lcrypto
halis@VM:/home/seed/crypto/task7$ ./myenc
Key found: Syracuse########
halis@VM:/home/seed/crypto/task7$
```

Figure 18: Comparison between C1 and C2

Key = "Syracuse########"