Deniz Aytemiz

ECE 5560- Symmetric-Key & Block Mode Lab

TASK 1.

Plaintext: This is a plaintext

In Figure 1, there is a screenshot of usage of the nano command. Here I fill the plain.txt document with a message. I used this command after using the cat command to create the plain.txt file.

Figure 1: nano command for creating plain text

In Figure 2, aes-128-ecb is used as the mode of operation and the cipher is used. We can observe the hex values of the ciphertext stored. We check whether it worked correctly by comparing the original plaintext and the decrypted plaintext from the cipher. The decrypted plaintext is stored in a different file in the directory named plain_dcrpy.txt

```
  System load:  0.01              Processes:               213
  Usage of /:   46.2% of 11.57GB  Users logged in:         1
  Memory usage: 42%               IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                IPv4 address for ens5:    172.31.2.147

 * Ubuntu Pro delivers the most comprehensive open source security and
   compliance features.

   https://ubuntu.com/aws/pro

216 updates can be installed immediately.
4 of these updates are security updates.
To see these additional updates run: apt list --upgradable


3 updates could not be installed automatically. For more details,
see /var/log/unattended-upgrades/unattended-upgrades.log

Last login: Tue Oct 19 19:08:07 2021 from 18.206.107.24
ubuntu@ip-172-31-2-147:~$ cat plain.txt
This is a plaintext

ubuntu@ip-172-31-2-147:~$ openssl enc -aes-128-ecb -e -in plain.txt -out cipher.bin -K 1234445555
hex string is too short, padding with zero bytes to length
ubuntu@ip-172-31-2-147:~$ cat cipher.bin
0Hrxf8S{e>a'hl?
ubuntu@ip-172-31-2-147:~$ hexdump cipher.bin
0000000 30c7 48ac 7872 a866 a038 5389 7bc1 651a
0000010 063e 0490 2761 a8c1 6c9c cb92 03fc 0af3
0000020
ubuntu@ip-172-31-2-147:~$ openssl enc -aes-128-ecb -d -in cipher.bin -out plain_dcrp.txt -K 1234445555
hex string is too short, padding with zero bytes to length
ubuntu@ip-172-31-2-147:~$ cat plain_dcrp.txt
This is a plaintext

ubuntu@ip-172-31-2-147:~$
```

Figure 2: aes-128-ecb used for encryption and decryption

In Figure 3, aes-128-cbc is used as the mode of operation and the cipher is used. We can observe the hex values of the ciphertext stored. We check whether it worked correctly by comparing the original plaintext and the decrypted plaintext from the cipher. The decrypted plaintext is stored in a different file in the directory named plain_CBC.txt

```
Last login: Tue Oct 19 19:08:29 2021 from 18.206.107.25
ubuntu@ip-172-31-2-147:~$ openssl enc -aes-128-cbc -e -in plain.txt -out cipherCBC.bin -K 1234445555 -iv 11223344556677889900
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
ubuntu@ip-172-31-2-147:~$ hexdump cipherCBC.bin
0000000 f27b 5a3d e1c1 6d81 223e e0c8 0775 0abf
0000010 3eb4 c373 68ea 730a 7a1a 5ba2 80c4 6c16
0000020
ubuntu@ip-172-31-2-147:~$ openssl enc -aes-128-cbc -d -in cipherCBC.bin -out plain_CBC.txt -K 1234445555
-iv 11223344556677889900
hex string is too short, padding with zero bytes to length
non-hex digit
invalid hex iv value
ubuntu@ip-172-31-2-147:~$ openssl enc -aes-128-cbc -d -in cipherCBC.bin -out plain_CBC.txt -K 1234445555 -iv 11223344556677889900
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
ubuntu@ip-172-31-2-147:~$
ubuntu@ip-172-31-2-147:~$
ubuntu@ip-172-31-2-147:~$ openssl enc -aes-128-cbc -d -in cipherCBC.bin -out plain_CBC.txt -K 1234445555 -iv 11223344556677889900
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
ubuntu@ip-172-31-2-147:~$
ubuntu@ip-172-31-2-147:~$ cat plain_CBC.txt
This is a plaintext
```

Figure 3: aes-128-cbc used for encryption and decryption

In Figure 4, the same process is done for bf-cbc.

```
ubuntu@ip-172-31-2-147:~$ openssl enc -bf-cbc -e -in plain.txt -out cipherbf.bin -K 1234445555 -iv 11223344556677889900
hex string is too long, ignoring excess
hex string is too short, padding with zero bytes to length
ubuntu@ip-172-31-2-147:~$ hexdump cipherbf.bin
0000000 5224 4c91 56b2 a1bb 1f97 a732 d1d4 1725
0000010 fe6b f0c2 9d12 b3f0
0000018
ubuntu@ip-172-31-2-147:~$ openssl enc -bf-cbc -d -in cipherbf.bin -out plainbf.bin -K 1234445555 -iv 11223344556677889900
hex string is too long, ignoring excess
hex string is too short, padding with zero bytes to length
ubuntu@ip-172-31-2-147:~$ cat plainbf
cat: plainbf: No such file or directory
ubuntu@ip-172-31-2-147:~$ cat plainbf.
cat: plainbf.: No such file or directory
ubuntu@ip-172-31-2-147:~$ openssl enc -bf-cbc -d -in cipherbf.bin -out plainbf.txt -K 1234445555 -iv 11223344556677889900
hex string is too long, ignoring excess
hex string is too short, padding with zero bytes to length
ubuntu@ip-172-31-2-147:~$ cat plainbf.txt
This is a plaintext

ubuntu@ip-172-31-2-147:~$
```

Figure 4: bf-cbc used for encryption and decryption

TASK 2.

Code:

```
denizaytemiz@Denizs-MacBook-Air Desktop % openssl enc -aes-128-ecb -e -in pic_original.bmp -out myencryptedphoto_ECB.bin -K 123444555511 -iv 11222333444422
denizaytemiz@Denizs-MacBook-Air Desktop % head -c 54 pic_original.bmp > header
denizaytemiz@Denizs-MacBook-Air Desktop % tail -c +55 myencryptedphoto_ECB.bin > body
denizaytemiz@Denizs-MacBook-Air Desktop % cat header body > mynewphoto_from_encrypted_ECB.bmp
denizaytemiz@Denizs-MacBook-Air Desktop % open mynewphoto_from_encrypted_ECB.bmp
denizaytemiz@Denizs-MacBook-Air Desktop % openssl enc -aes-128-cbc -e -in pic_original.bmp -out myencryptedphoto_CBC.bin -K 123444555511 -iv 11222333444422
denizaytemiz@Denizs-MacBook-Air Desktop % head -c 54 pic_original.bmp > header
denizaytemiz@Denizs-MacBook-Air Desktop % tail -c +55 myencryptedphoto_CBC.bin > body
denizaytemiz@Denizs-MacBook-Air Desktop % cat header body > mynewphoto_from_encrypted_CBC.bmp
denizaytemiz@Denizs-MacBook-Air Desktop % open mynewphoto_from_encrypted_CBC.bmp
denizaytemiz@Denizs-MacBook-Air Desktop %
```

Figure 5: Encryption of picture with ECB and CBC modes

Results:

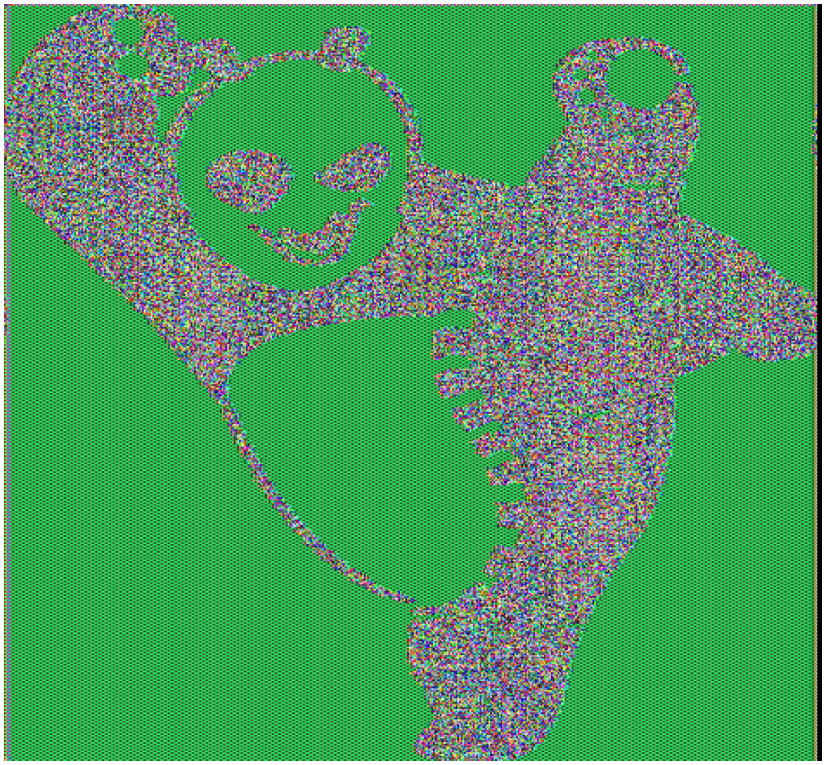

Figure 6: ECB encryption

Figure 7: CBC encryption

Observations: In ECB mode, the plaintexts are encrypted in parallel hence there is no connection between the encrypted blocks. Therefore similar areas in the picture are giving similar ciphertexts which makes it possible to detect the shape of the panda. However in CBC mode, the ciphertext of one encryption affects the next encryption. Hence the resulting image looks more random, there is no clear pattern. It is better encrypted.

TASK 3.

Uploaded as a separate file.

TASK 4.

1.

ECB mode:

```
izs-MacBook-Air Desktop % echo -n  'This is some text for testing'  > ECBfile.txt
izs-MacBook-Air Desktop % openssl enc -aes-128-ecb -e -in ECBfile.txt -out ECB_cipher.bin -K 123444555511 -iv 11222333444422
izs-MacBook-Air Desktop % openssl enc -aes-128-ecb -d -in ECB_cipher.bin -out ECB_out.txt -K 123444555511 -iv 11222333444422 -nopad
izs-MacBook-Air Desktop % hexdump -C ECB_out.txt
69 73 20 69 73 20  73 6f 6d 65 20 74 65 78  |This is some tex|
66 6f 72 20 74 65  73 74 69 6e 67 03 03 03  |t for testing...|

izs-MacBook-Air Desktop %
```

Figure 8: Padding in ECB mode

ECB mode is a block cipher mode, hence it uses the padding. It can be seen the input text is equally divided into equal sized blocks. The second block is filled with dots as padding.

CBC mode:

```
izaytemiz@Denizs-MacBook-Air Desktop % openssl enc -aes-128-cbc -e -in ECBfile.txt -out CBC_cipher.bin -K 123444555511 -iv 11222333444422
izaytemiz@Denizs-MacBook-Air Desktop % openssl enc -aes-128-cbc -d -in CBC_cipher.bin -out CBC_out.txt -K 123444555511 -iv 11222333444422 -nopad
izaytemiz@Denizs-MacBook-Air Desktop % hexdump -C CBC_out.txt
00000  54 68 69 73 20 69 73 20  73 6f 6d 65 20 74 65 78  |This is some tex|
00010  74 20 66 6f 72 20 74 65  73 74 69 6e 67 03 03 03  |t for testing...|
00020
izaytemiz@Denizs-MacBook-Air Desktop %
```

Figure 9: Padding in CBC mode

CBC mode is block cipher mode, hence it uses the padding. We can see the input text is equally divided into equal sized blocks. The second block is filled with dots as padding.

CBF mode:

```
izaytemiz@Denizs-MacBook-Air Desktop % openssl enc -aes-128-cfb -e -in ECBfile.txt -out CFB_cipher.bin -K 123444555511 -iv 11222333444422
izaytemiz@Denizs-MacBook-Air Desktop % openssl enc -aes-128-cfb -d -in CFB_cipher.bin -out CFB_file.txt -K 123444555511 -iv 11222333444422 -nopad
izaytemiz@Denizs-MacBook-Air Desktop % hexdump -C CFB_file.txt
00000  54 68 69 73 20 69 73 20  73 6f 6d 65 20 74 65 78  |This is some tex|
00010  74 20 66 6f 72 20 74 65  73 74 69 6e 67           |t for testing|
0001d
izaytemiz@Denizs-MacBook-Air Desktop %
```

Figure 10: Padding in CBF mode

CFB turns block ciphers into stream ciphers hence does not require any padding since the cipher and plaintext will be of the same length.

OFB mode:

```
izaytemiz@Denizs-MacBook-Air Desktop % openssl enc -aes-128-ofb -e -in ECBfile.txt -out OFB_cipher.bin -K 123444555511 -iv 11222333444422
izaytemiz@Denizs-MacBook-Air Desktop % openssl enc -aes-128-ofb -d -in OFB_cipher.bin -out OFB_out.txt -K 123444555511 -iv 11222333444422 -nopad
izaytemiz@Denizs-MacBook-Air Desktop % hexdump -C OFB_out.txt
00000  54 68 69 73 20 69 73 20  73 6f 6d 65 20 74 65 78  |This is some tex|
00010  74 20 66 6f 72 20 74 65  73 74 69 6e 67           |t for testing|
0001d
izaytemiz@Denizs-MacBook-Air Desktop %
```

Figure 11: Padding in OFB mode

OFB turns block ciphers into stream ciphers hence does not require any padding since the cipher and plaintext will be of the same length.

2.

TASK 5:

We create an empty file with 1 KB with the following command to terminal:

fallocate -l 1KB task5.txt

1. **CBC:**

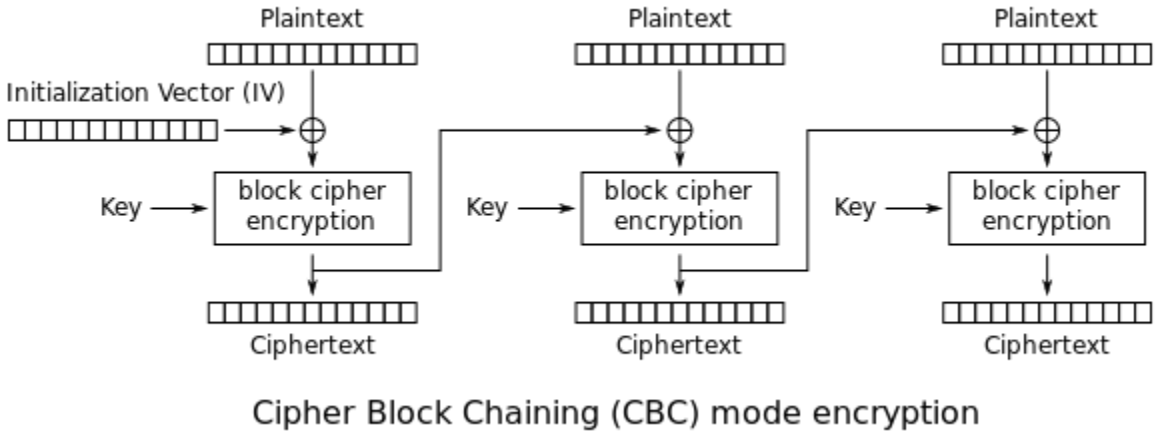

Figure 12: Cipher Block Chaining schematic

As it can be seen from the schematic in Figure 12, in CBC mode error in the byte of a ciphertext would cause the next block of ciphertext to be wrong too. Since previous ciphertext are used for the upcoming ones. So in this mode, I assume the error would propagate.

Opening the file in hexed.it, the initial empty file task5.txt is Figure 13.

Figure 13: Task 5 in hex coded form

After encrypting task5.txt file AES 128, 42th byte is manually changed as in Figure 14.

Figure 14: CBC mode encrypted file changed manually.

In Figure 15, the errored CBC encrypted file is decrypted and the resulting file task5_CBC_error.txt is compared with task5.txt. The files are different, meaning obtaining wrong output.

```
zaytemiz@Denizs-MacBook-Air Desktop % openssl enc -aes-128-cbc -d -in task5_CBC_error.bin -out task5_CBC_error_dec.txt -K 123444555511 -iv 11222333444422
zaytemiz@Denizs-MacBook-Air Desktop % diff task5.txt task5_CBC_error_dec.txt
ry files task5.txt and task5_CBC_error_dec.txt differ
zaytemiz@Denizs-MacBook-Air Desktop %
```

Figure 15: Code for decryption

In Figure 16, task5_CBC_error.txt is displayed in the hexadecimal form. As it is expected the whole cipher block is errored. Therefore error is propagated to 16 bytes.

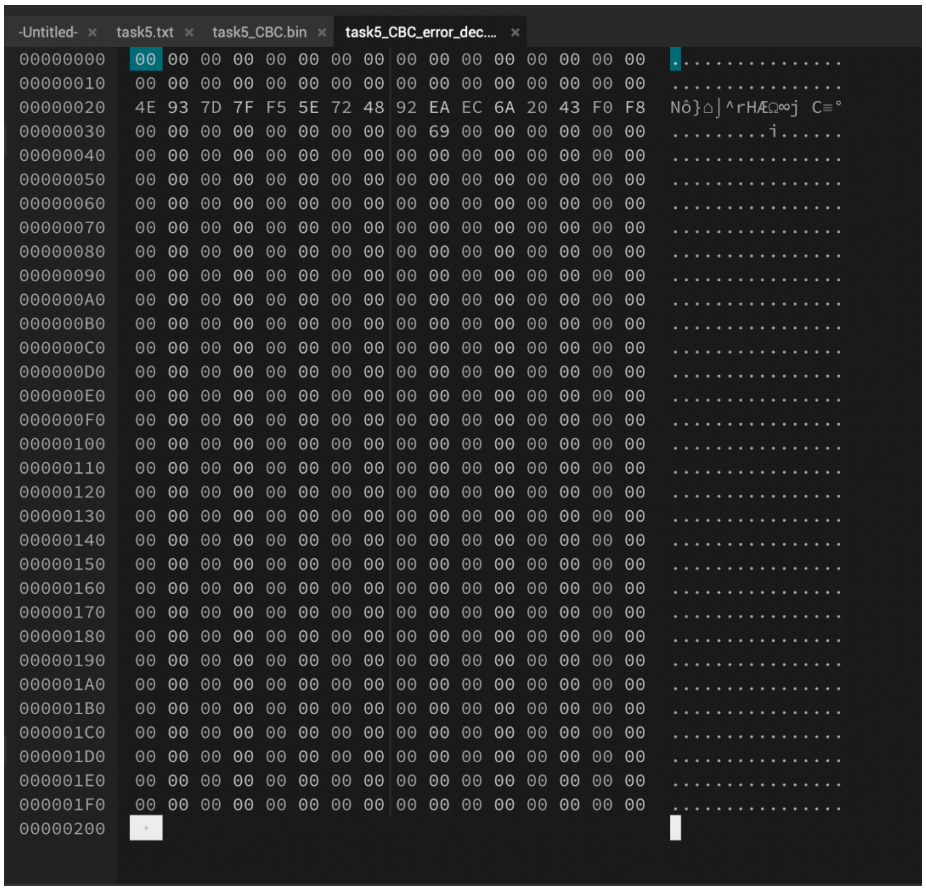

Figure 16: Error propagation for CBC mode

2. **ECB:**

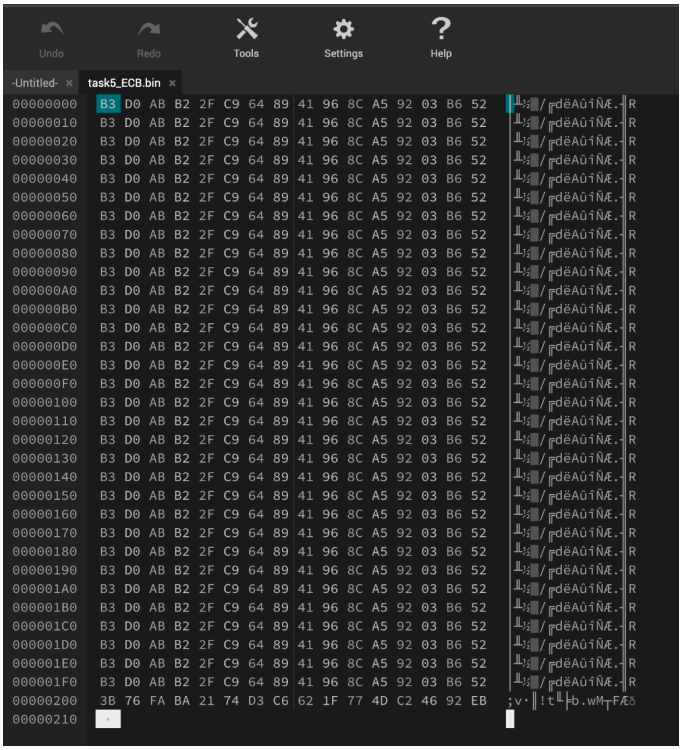

Figure 17: Correct cipher

We change the 42th bit:

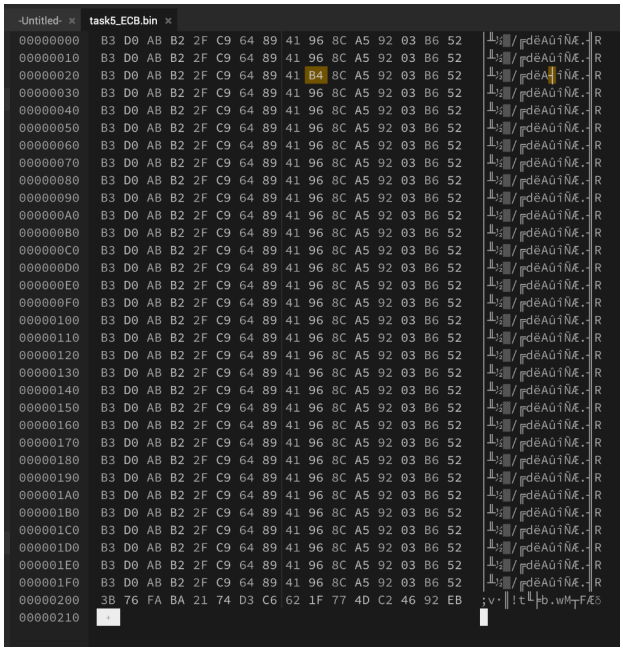

Figure 18: Error in cipher

The output can be observed, error did propagate for the cipher part: Since we gave the wrong icipher as input the resulting 16 byte block came out wrong.

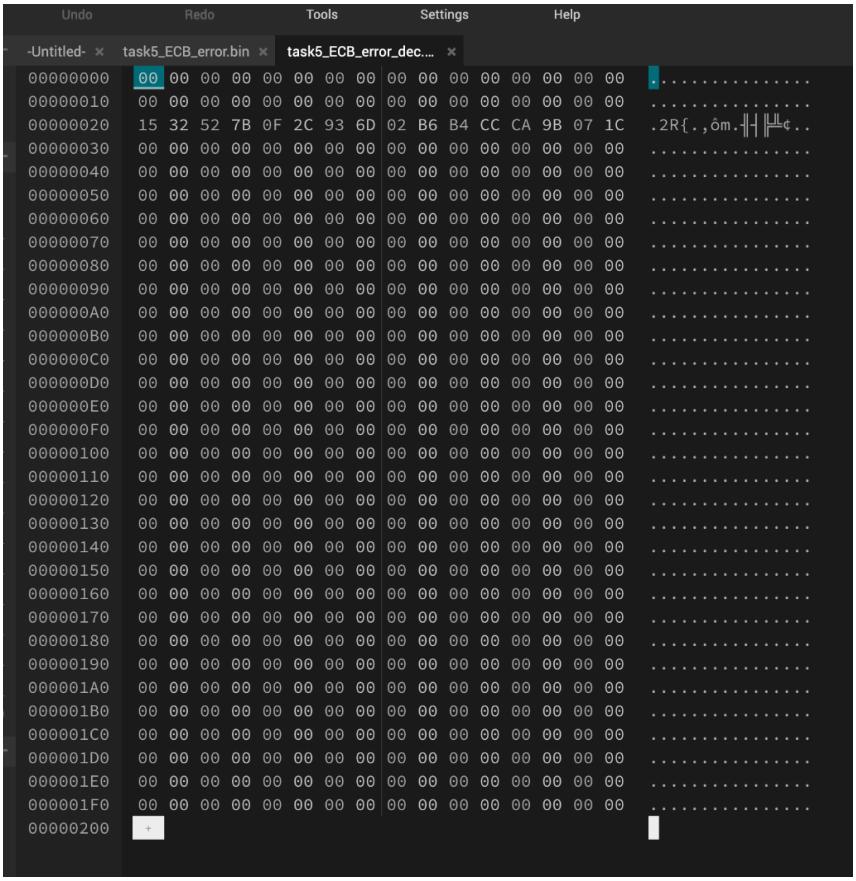

Figure 18: Error propagated in the ECB mode

3. **OFB:**

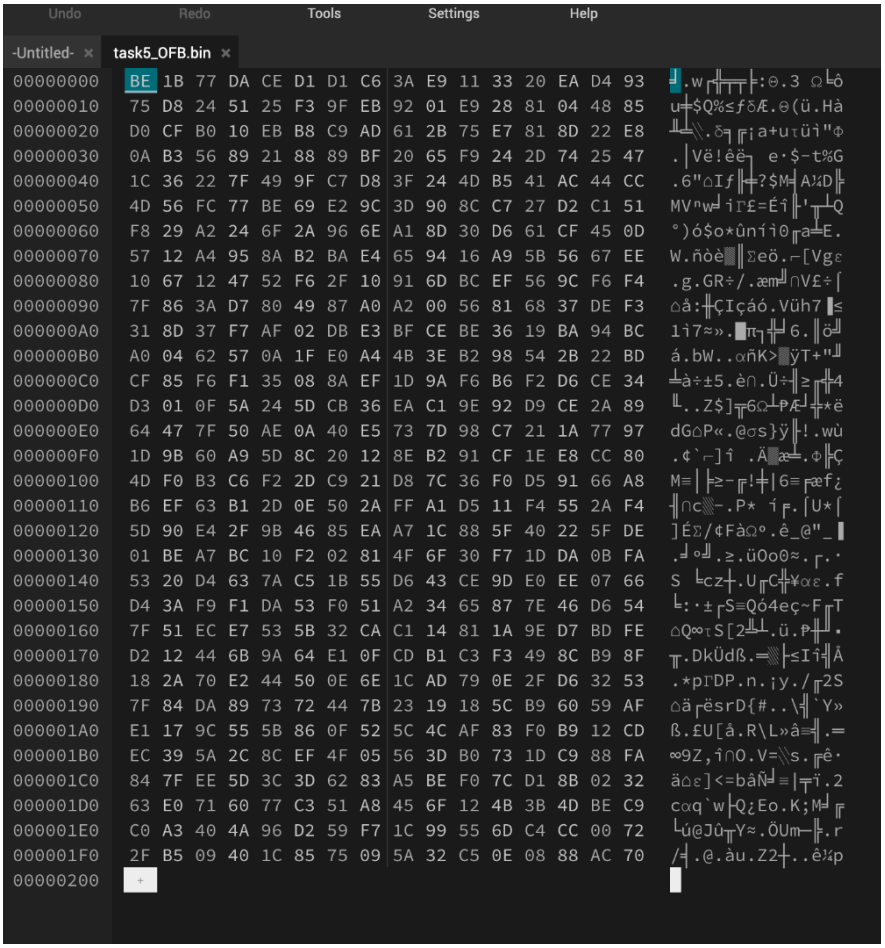

Figure 19: Encrypted file with no error:

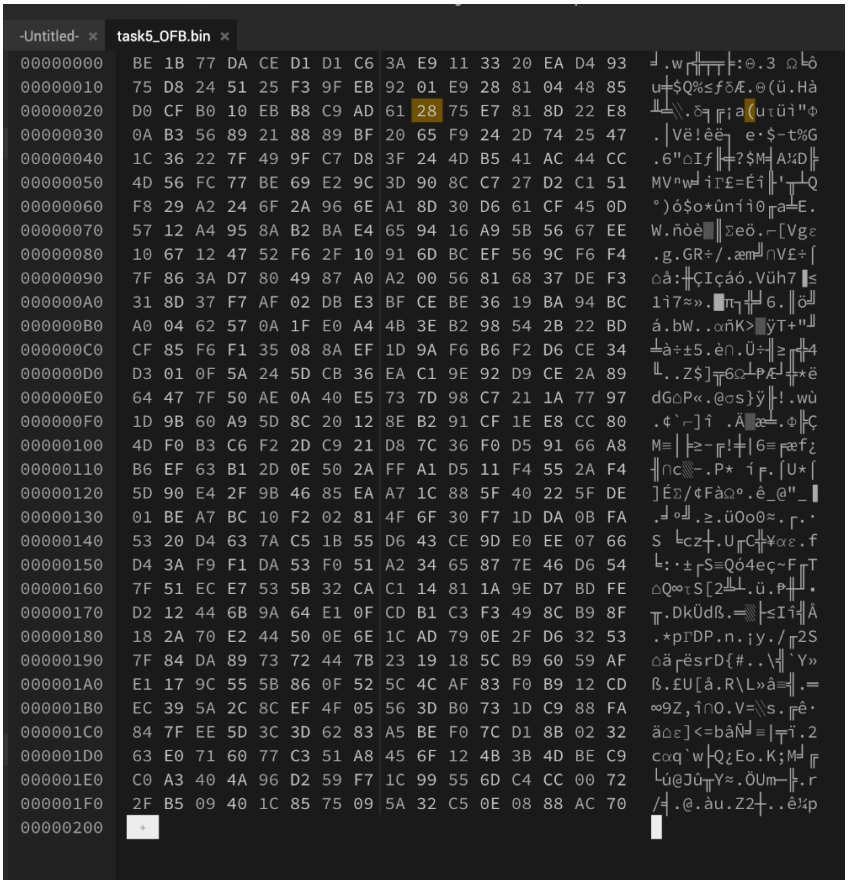

Figure 20: Encrypted file with error

Decrypting:

```
izaytemiz@Denizs-MacBook-Air Desktop % openssl enc -aes-128-ofb -e -in task5.txt -out task5_OFB.bin -K 123444555511 -iv 11222333444422
izaytemiz@Denizs-MacBook-Air Desktop % openssl enc -aes-128-ofb -d -in task5_OFB_error.bin -out task5_OFB_error_dec.txt -K 123444555511 -iv 11222333444422
izaytemiz@Denizs-MacBook-Air Desktop %
```

Figure 21: The terminal code

Error did not propagate as shown in Figure 22. Only 1 byte of information is wrong.

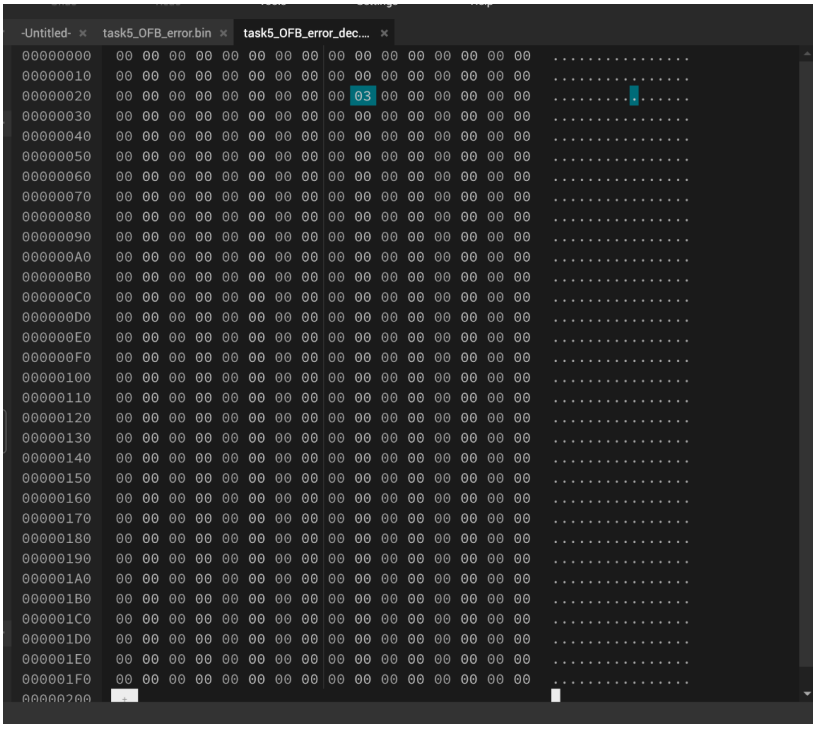

Figure 22: The decrypted file

4. **CFB**:

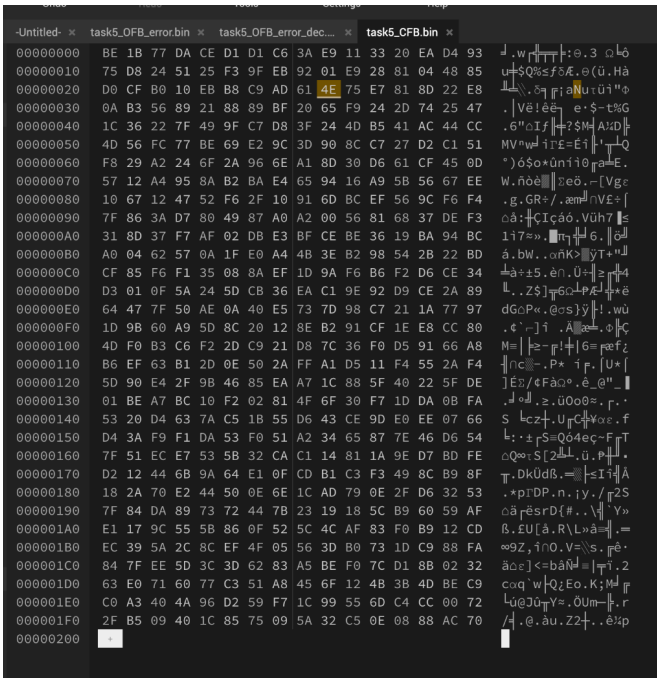

Figure 23: The errored encrypted file

You can see the error did not propagate in Figure 24. Only one byte is changed.

Figure 24: The error propagation in CFB mode.

TASK 6:

6.1

When the same iv is used under the same key resulting in the same encryption scheme for the same plaintext. Hence it is not very safe. However when the iv is unique the ciphertext is encrypted completely differently.

```
key:  b'\x01#Eg\x89\xab\xcd\xef\x01#Eg\x89\xab\xcd\xef\x01#Eg\x89\xab\xcd\xef\x01#Eg\x89\xab\xcd\xef'
plaintext b'This is a Test code'
iv is b'\x11\x11\x11\x11""""33333333'
Encrypted text:  b'\xde\\\x04\x8f\x93\xec\xe1*\xce5\xc6\xca\x7f\xba\xcdh\xca\xe4\xc8;\xb4\x7f\xed\xea\xb1\xa3`\xe2\xcb\x82|V'
Encrypted:    de5c048f93ece12ace35c6ca7fbacd68cae4c83bb47fedeab1a360e2cb827c56
```

Figure 25: Same iv is used under same key

```
key:  b'\x01#Eg\x89\xab\xcd\xef\x01#Eg\x89\xab\xcd\xef\x01#Eg\x89\xab\xcd\xef\x01#Eg\x89\xab\xcd\xef'
plaintext b'This is a Test code'
iv is b'\x11\x11\x11\x11"""""3"#01234'
Encrypted text:  b'M\x01\x9a\x1f6\xaf\xdbq\x83M\xeaQ\x80\x9d\x13\xfd8G:\xd5mJ_\x8bf>\x1d\x89"?9Z'
Encrypted:    4d019a1f36afdb71834dea51809d13fd38473ad56d4a5f8b663e1d89223f395a
```

Figure 26: Different iv is used under same key

6.2

If Iv is known in OFB mode the scheme remains very insecure. With the known plaintext-ciphertext pairs, one knows the output of the encryption which will be input to the next encryption block along with the key. Therefore for the next block the only unknown remains is the key. If an attacker has enough pairs of ciphertexts and plaintext pairs then it is possible for the attacker to deduce the key too. Hence the plaintext 2 can be fully recovered.

For CFB mode the case is similar. If IV is known and it is repeatedly used, the attacker can recover the next plaintext blocks. If the same IV is used for the same scheme one will know the value of encrypted iv which is Ek(IV).

C1 = Ek(Ek(iv) xor Plaintext0) xor Plaintext1

In the expression above, Ek(iv), plaintext0 and C1 are known. Therefore one can detect the C1 too. Hence both modes become vulnerable with the known and repeated ivs.

6.3