Secret-Key Encryption Lab Report

Emre Can Tüzer

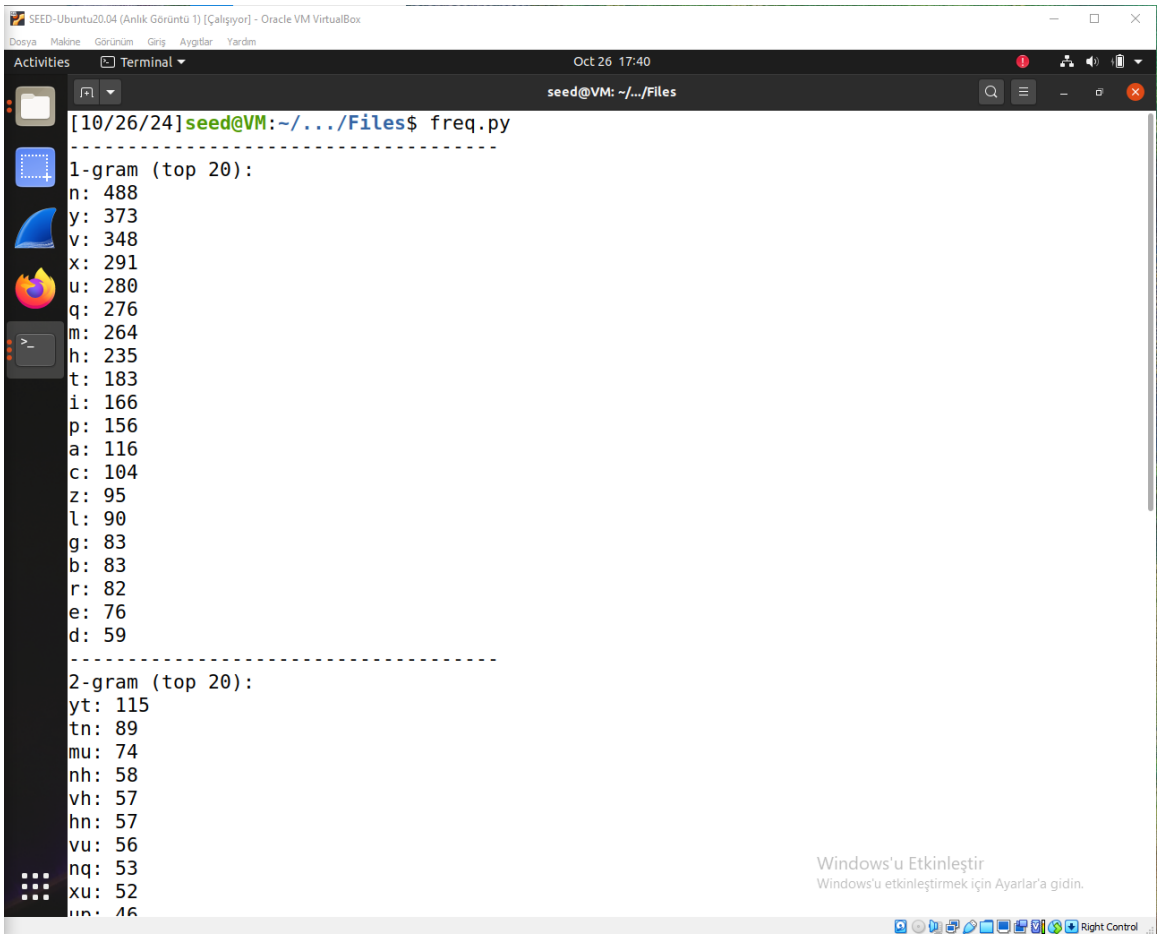
Date: November 4, 2024

BGK-503

Kemal Bıçakcı

Task 1: Frequency Analysis

Utilized a Python script to analyze the frequency of n-grams in a given ciphertext.



Starting with common letters, the letter "ytn" is a good choice for the English word "THE" because it appears frequently.

ytn - THE

ytnu - THEu

ytnd - THEd

ytnhn - THEhE (h would be R)

hnden yürüdüm

lnhn - lERE - (l would be W)

ytvy - THvT - (v would be A)

vhn - vRE - ARE

hnp - REp - RED

vup = ANp (p would be d)

ytnmh - THEmh - THEmR - (m would be i)

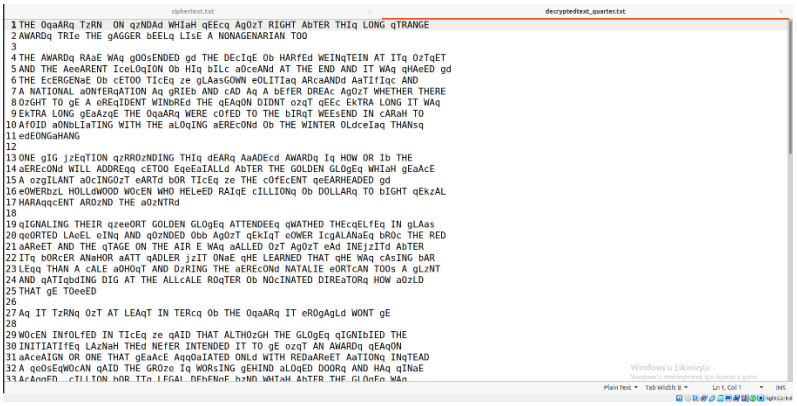
vuxytnh - vuxTHEh - AuxTHER - (ANOTHER) -- (u=N , x=O)

ivupmur - iANDINr - Maybe LANDING (i = L, r =g)

ivup - LAND

Since I was unable to locate any more, I will decipher these words and look over the text as a whole.

tr 'ytnhlvmuxpirszkqb' 'THERWAINODLGKUXSF' < decryptedtext_quarter.txt > decryptedtext_half.txt



WEEsEND - s is K

AROzND - z is U

EkTRA - k is X

qTAGE - q is S

AbTER - b is F

tr 'ytnhlvmuxpirszkqb' 'THERWAINODLGKUXSF' < decryptedtext_quarter.txt > decryptedtext_half.txt

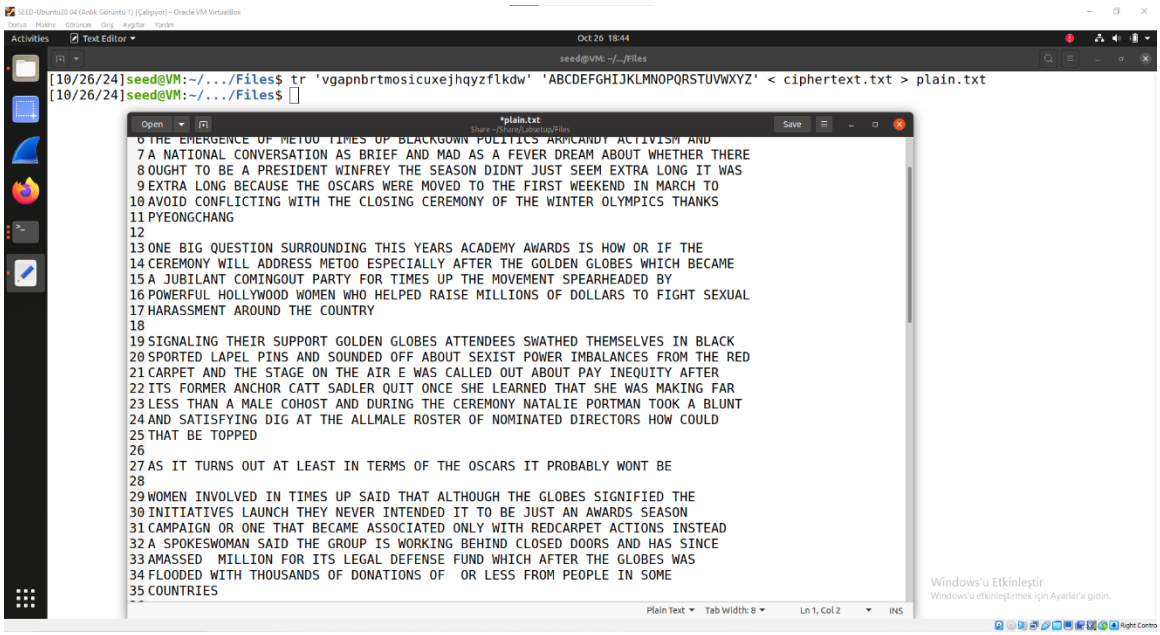
aOUNTRd - aOUNTRd - a is C, d is Y

aOULD - Could - a is C

jUIT - Quit - j is Q

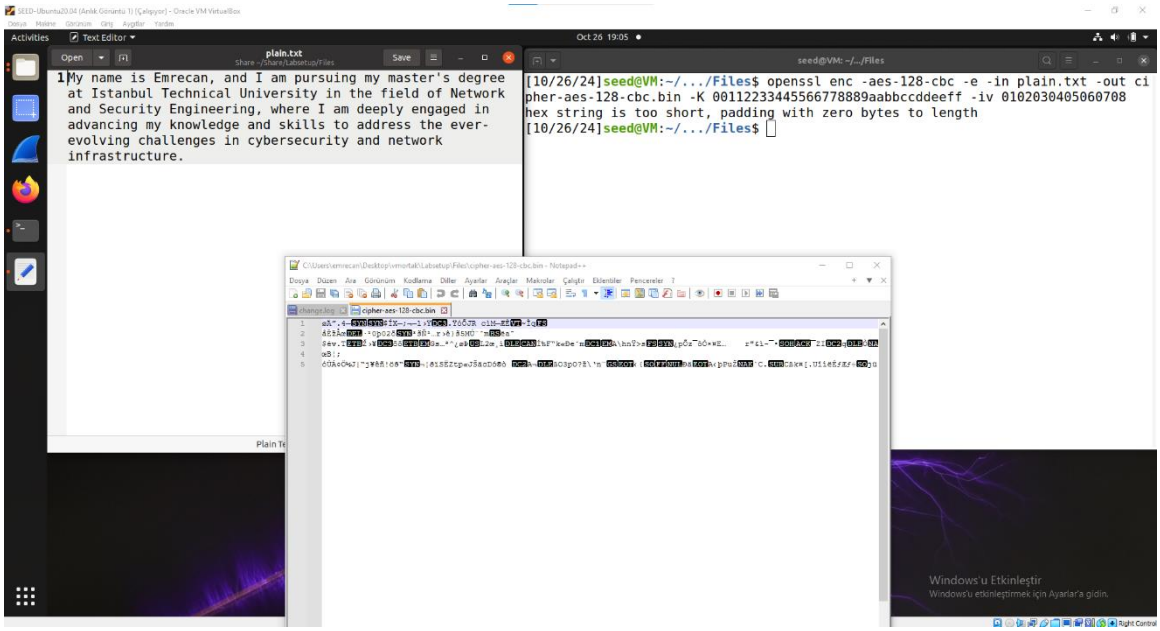
The last command to fully decode the text is:

tr 'vgapnbrtmosicuxejhqyzflkdw' 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' < ciphertext.txt > plain.txt

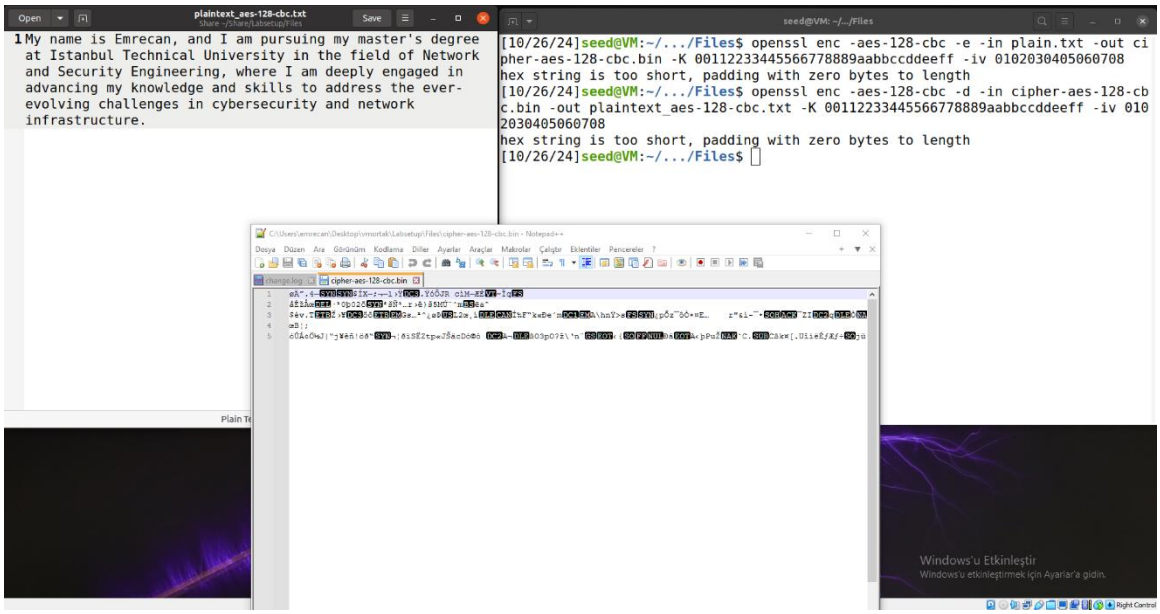


Task 2: Encryption using Different Ciphers and Modes

Encryption AES-128-CBC



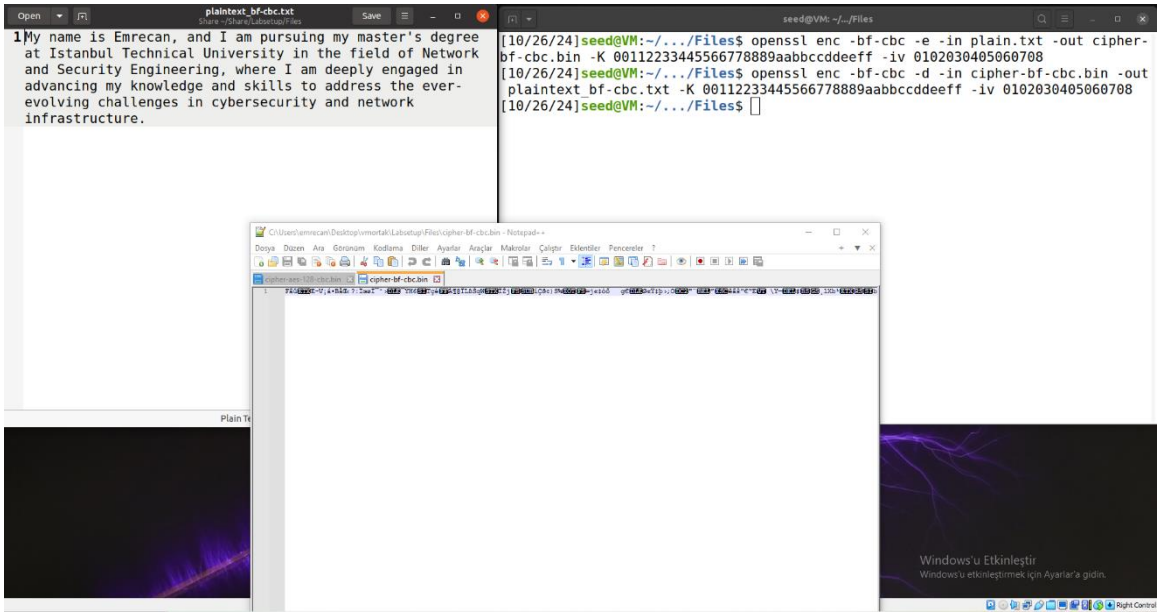
openssl enc -aes-128-cbc -e -in plain.txt -out cipher-aes-128-cbc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708



openssl enc -aes-128-cbc -d -in cipher-aes-128-cbc.bin -out plaintext_aes-128-cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

Encryption BF-CBC

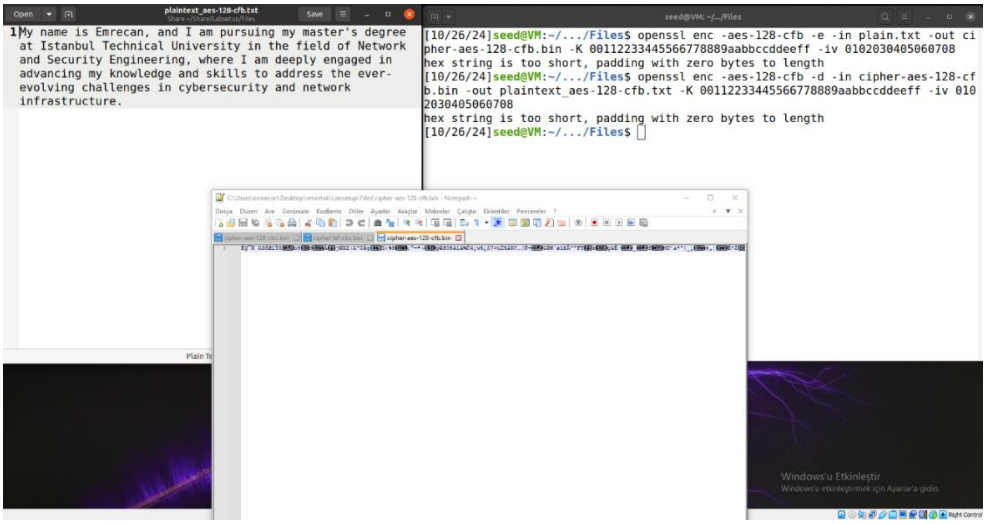
openssl enc -bf-cbc -e -in plain.txt -out cipher-bf-cbc.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708



openssl enc -bf-cbc -d -in cipher-bf-cbc.bin -out plaintext_bf-cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

Encryption AES-128-CFB

openssl enc -aes-128-cfb -e -in plain.txt -out cipher-aes-128-cfb.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708



openssl enc -aes-128-cfb -d -in cipher-aes-128-cfb.bin -out plaintext_aes-128-cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

AES-128-CBC and AES-256-CBC

Every block is encrypted using the data from the previous encrypted block in a chain-like fashion. This mode creates a connection between data chunks and consistently generates distinct output. High error propagation results from a mistake in one block that impacts the decryption of all blocks that follow. This setting removes patterns, improving data privacy.

AES-128-CFB and AES-256-CFB

It is independent of block size and is a stream cipher mode. Real-time data stream encryption is made possible by its ability to function at the byte or bit level rather than data blocks. only impacts the corrupted bit or byte's decryption.

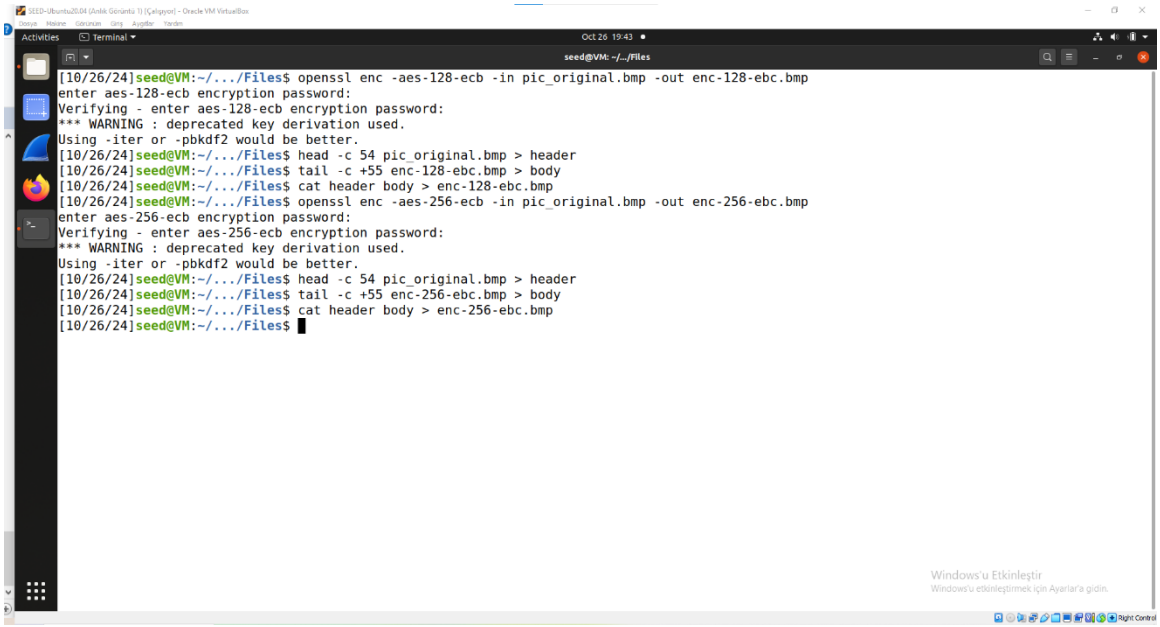
BF-CBC-128 and BF-CBC-256

An alternative to AES, it is an earlier block cipher algorithm. Despite being flexible and safe, it can run more slowly, particularly when dealing with big datasets.

Task 3: Encryption Mode – ECB vs. CBC

Original Picture:

ECB Mode Analysis:



Commands:

openssl enc -aes-128-ecb -in pic_original.bmp -out enc-128-ebc.bmp

head -c 54 pic_original.bmp > header

tail -c +55 enc-128-ebc.bmp > body

cat header body > enc-128-ebc.bmp

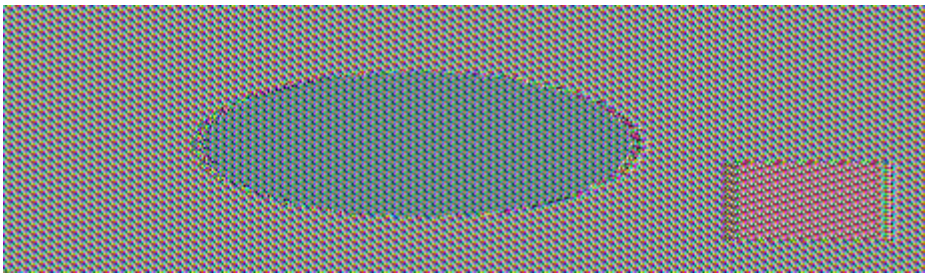
openssl enc -aes-256-ecb -in pic_original.bmp -out enc-256-ebc.bmp

head -c 54 pic_original.bmp > header

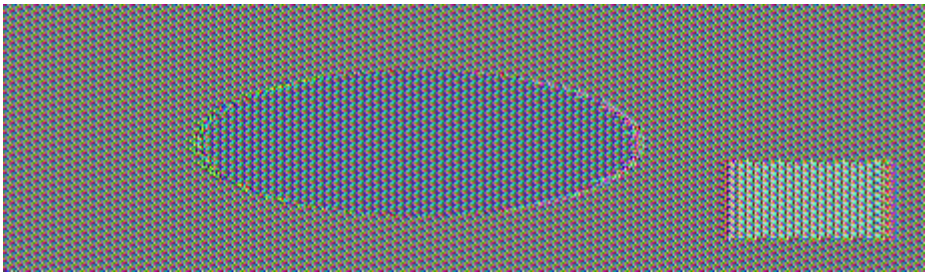
tail -c +55 enc-256-ebc.bmp > body

cat header body > enc-256-ebc.bmp

Output:



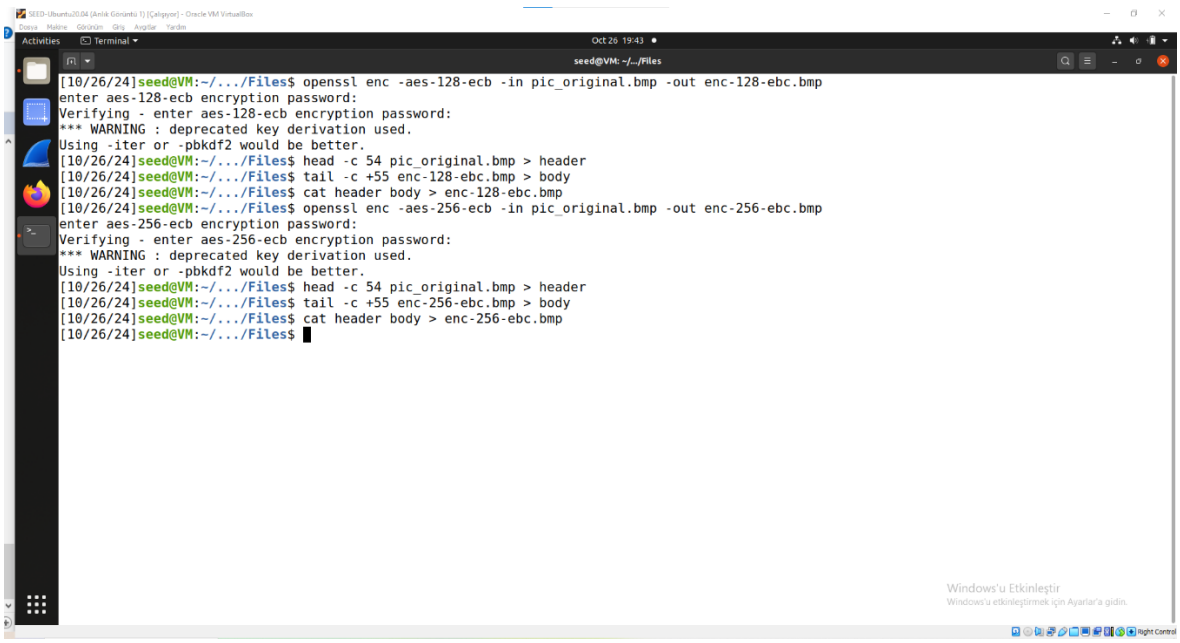
ECB-128



ECB-256

There are visible variations between AES-128 and AES-256. Because the method uses a bigger key space, more diverse color changes and tones can be seen when encrypting the same data.

CBC Mode Analysis:



Commands:

```
openssl enc -aes-128-cbc -in pic_original.bmp -out enc-128-cbc.bmp

head -c 54 pic_original.bmp > header

tail -c +55 enc-128-cbc.bmp > body

cat header body > enc-128-cbc.bmp

openssl enc -aes-256-cbc -in pic_original.bmp -out enc-256-cbc.bmp

head -c 54 pic_original.bmp > header

tail -c +55 enc-256-cbc.bmp > body

cat header body > enc-256-cbc.bmp
```

Output:

CBC-128

CBC-256

We can see visible changes between AES-128 and AES-256. Even identical plaintext blocks change into distinct ciphertext blocks in CBC mode because each block is encrypted in a chain with the one before it. AES-256 differs from AES-128 primarily in that it uses a longer key to boost security. There isn't a discernible visual difference in the result, though. In conclusion, the ECB mode is more susceptible to ciphertext analysis due to the patterns it preserves. On the other hand, because data blocks are interdependent, the CBC mode offers a more secure result by hiding patterns and encrypting each block in a unique way. Whereas in CBC mode, errors propagate to succeeding blocks, increasing the level of corruption in the decryption process, in ECB mode, an error in one block only affects that block.

Task 4: Padding

I intended to figure out the messages on my own while using this task. Start by creating 3 files which contain 4 bytes, 8 bytes, and 16 bytes, respectively. Data is encrypted using block cipher techniques in numerous block sizes. The remaining space is filled with padding if the data is not a full block.

echo -n "emre" > plain-1.txt == 4 byte

echo -n "emrecant" > plain-2.txt == 8 byte

echo -n "emrecantuzeremre" > plain-3.txt == 16 byte

Padding AES-128-CBC AES-256-CBC

openssl enc -aes-128-cbc -e -in plain-1.txt -out enc-cipher-1-cbc.bin

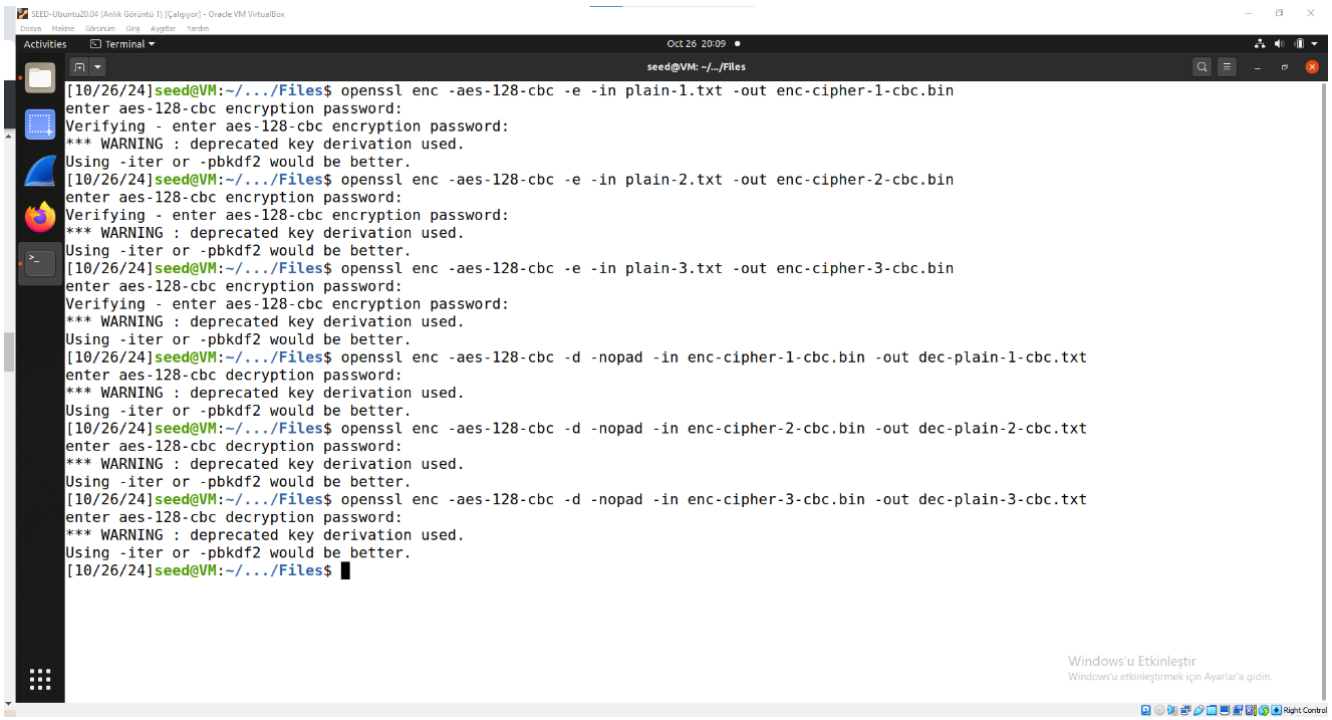
openssl enc -aes-128-cbc -e -in plain-2.txt -out enc-cipher-2-cbc.bin

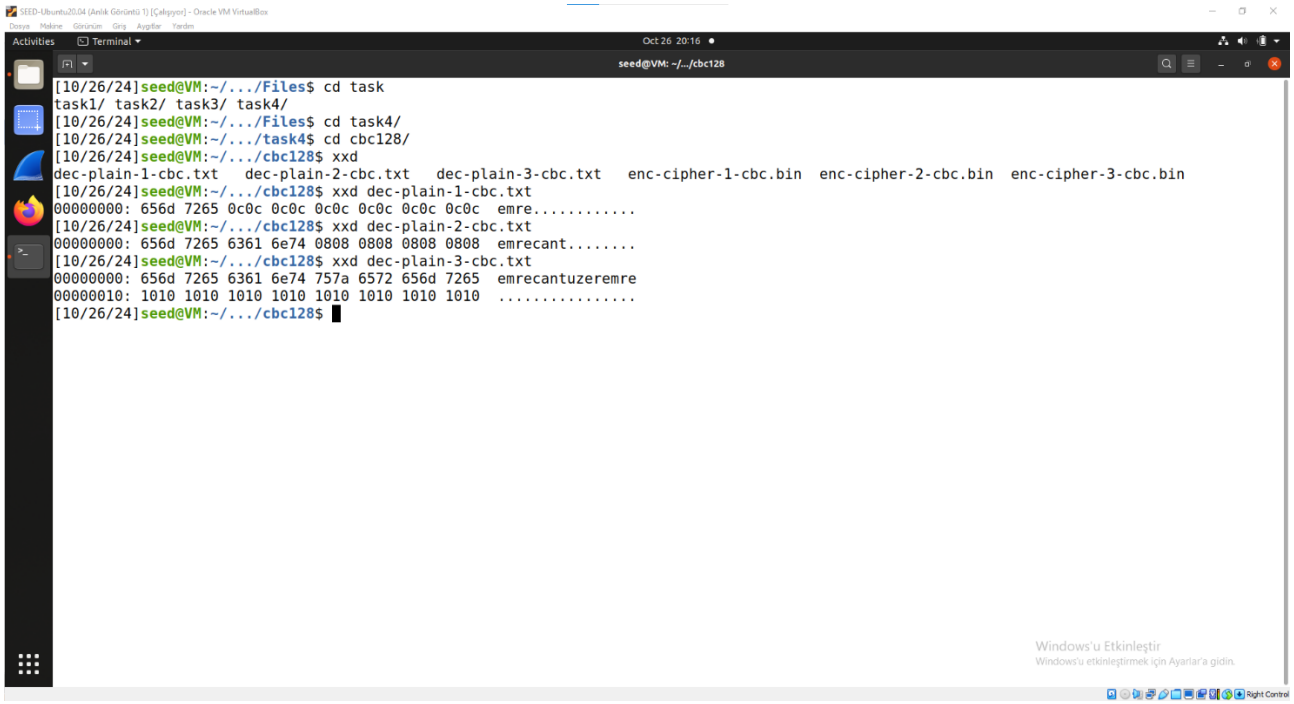
openssl enc -aes-128-cbc -e -in plain-3.txt -out enc-cipher-3-cbc.bin

openssl enc -aes-128-cbc -d -nopad -in enc-cipher-1-cbc.bin -out dec-plain-1-cbc.txt

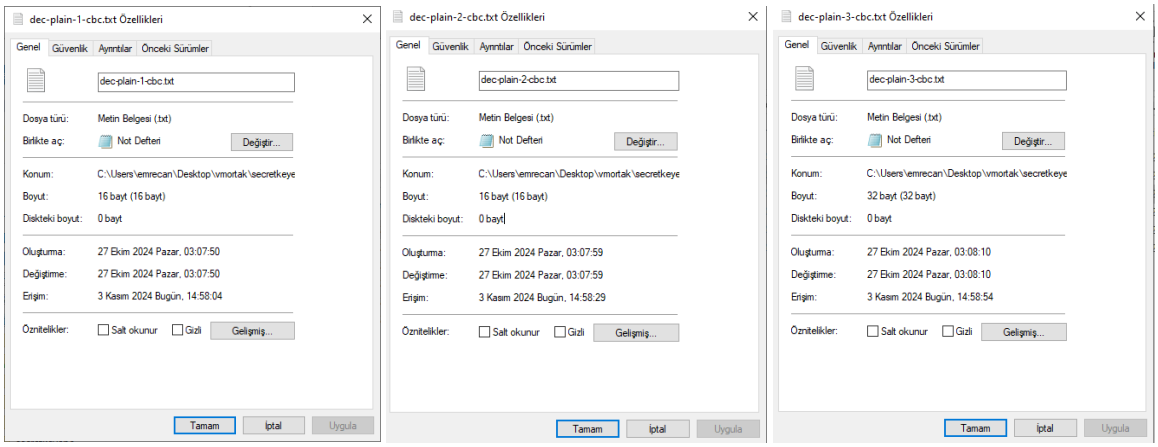
openssl enc -aes-128-cbc -d -nopad -in enc-cipher-2-cbc.bin -out dec-plain-2-cbc.txt

openssl enc -aes-128-cbc -d -nopad -in enc-cipher-3-cbc.bin -out dec-plain-3-cbc.txt

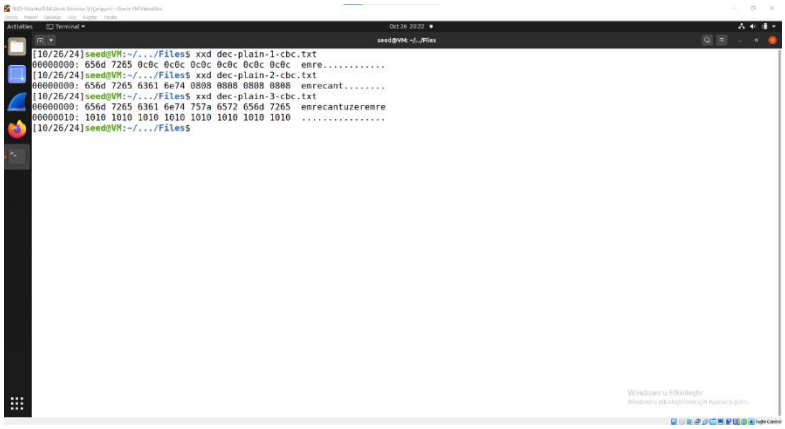




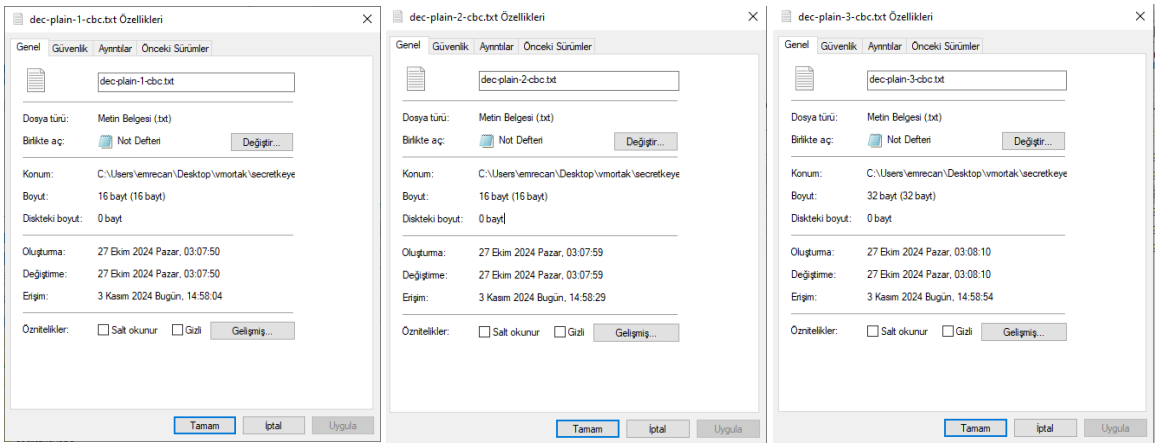
File Sizes:



Prior to encryption, the file size was 4,8,16 bytes. The data is padded with 12,8,16 bytes to fill the block size. File sizes rise to 16,16,32 bytes after encryption. The padding is eliminated during decryption, restoring the original size. The file size is 16 bytes, or the entire block size, prior to encryption. The file size grows to 32 bytes as a full block is filled, adding an additional 16 bytes of padding.



File Sizes:



Prior to encryption, the file size was 4,8,16 bytes. The data is padded with 12,8,16 bytes to fill the block size. File sizes rise to 16,16,32 bytes after encryption. The padding is eliminated during decryption, restoring the original size. The file size is 16 bytes, or the entire block size, prior to encryption. The file size grows to 32 bytes as a full block is filled, adding an additional 16 bytes of padding.

Padding AES-128-ECB AES-256-ECB

openssl enc -aes-128-ecb -e -in plain-1.txt -out enc-cipher-1-ecb.bin

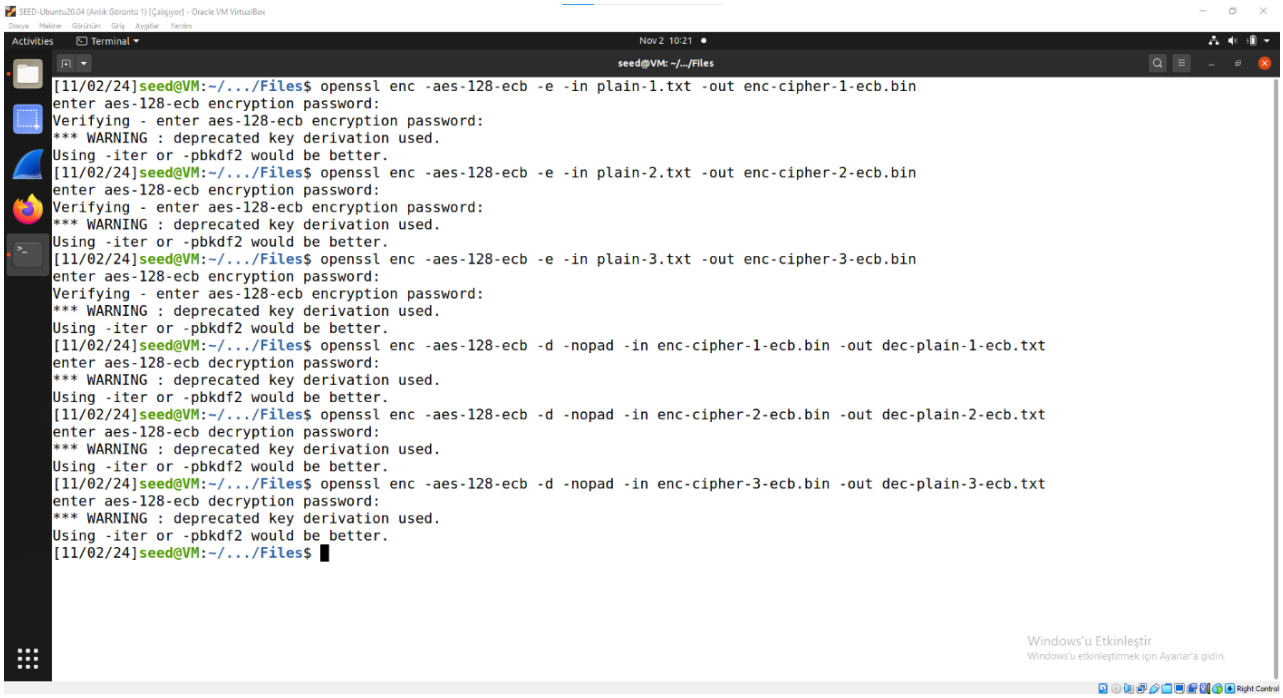
openssl enc -aes-128-ecb -e -in plain-2.txt -out enc-cipher-2-ecb.bin

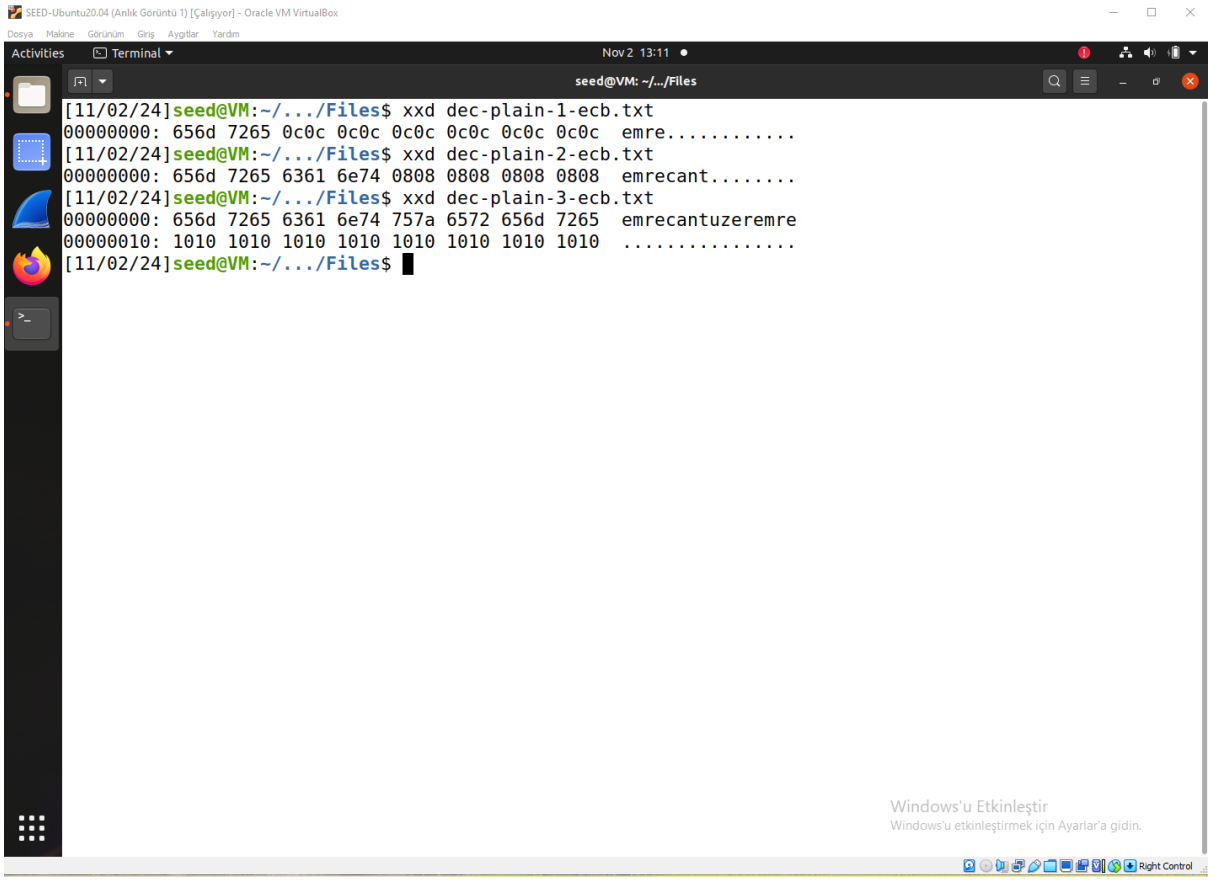
openssl enc -aes-128-ecb -e -in plain-3.txt -out enc-cipher-3-ecb.bin

openssl enc -aes-128-ecb -d -nopad -in enc-cipher-1-ecb.bin -out dec-plain-1-ecb.txt

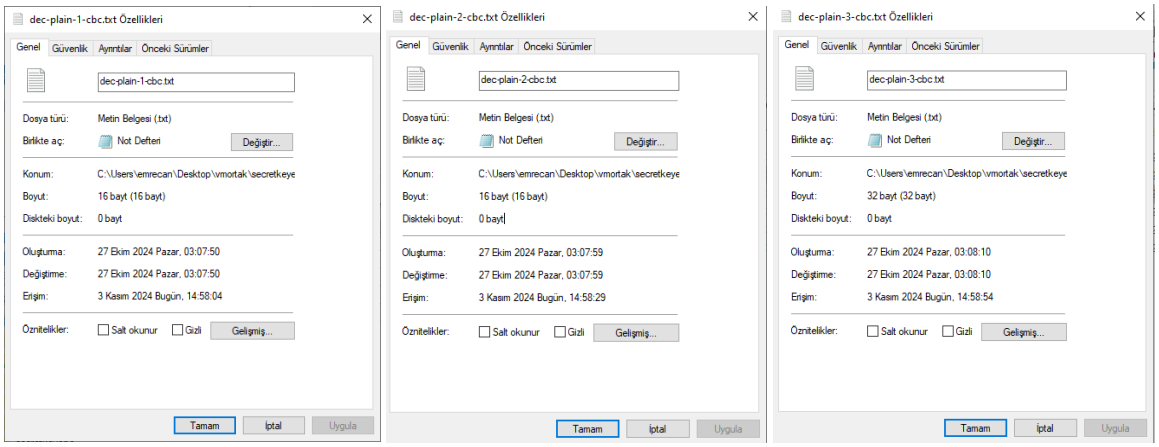
openssl enc -aes-128-ecb -d -nopad -in enc-cipher-2-ecb.bin -out dec-plain-2-ecb.txt

openssl enc -aes-128-ecb -d -nopad -in enc-cipher-3-ecb.bin -out dec-plain-3-ecb.txt





File Sizes:



Prior to encryption, the file size was 4,8,16 bytes. The data is padded with 12,8,16 bytes to fill the block size. File sizes rise to 16,16,32 bytes after encryption. The padding is eliminated during decryption, restoring the original size. The file size is 16 bytes, or the entire block size, prior to encryption. The file size grows to 32 bytes as a full block is filled, adding an additional 16 bytes of padding.

openssl enc -aes-256-ecb -e -in plain-1.txt -out enc-cipher-1-ecb.bin

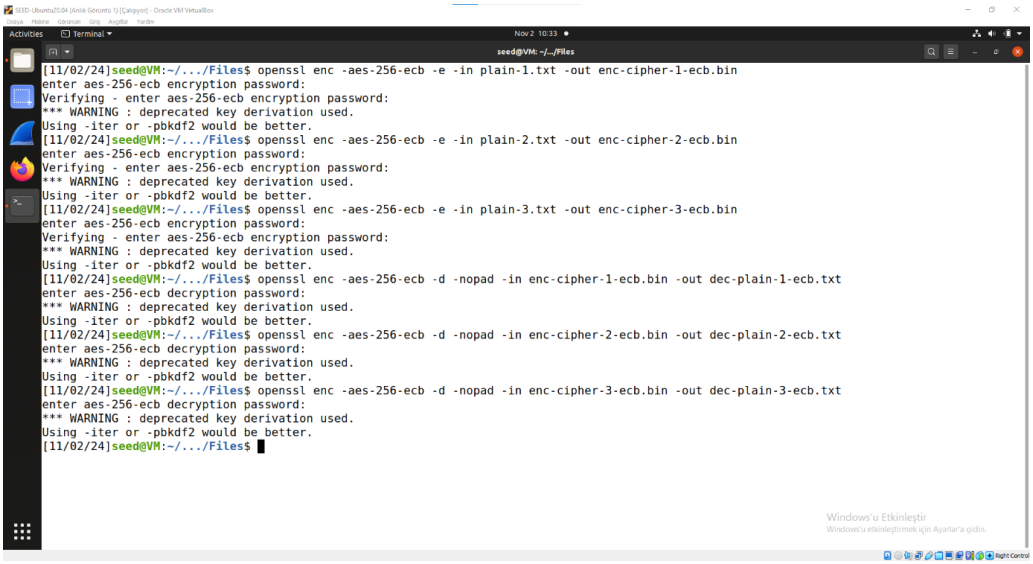
openssl enc -aes-256-ecb -e -in plain-2.txt -out enc-cipher-2-ecb.bin

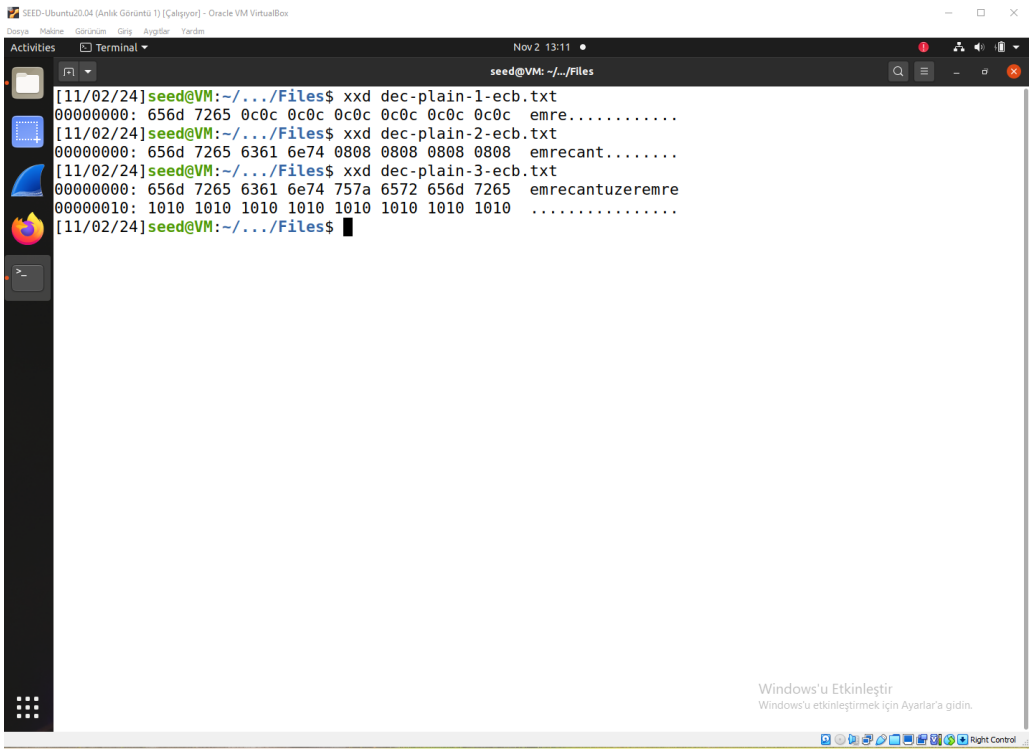
openssl enc -aes-256-ecb -e -in plain-3.txt -out enc-cipher-3-ecb.bin

openssl enc -aes-256-ecb -d -nopad -in enc-cipher-1-ecb.bin -out dec-plain-1-ecb.txt

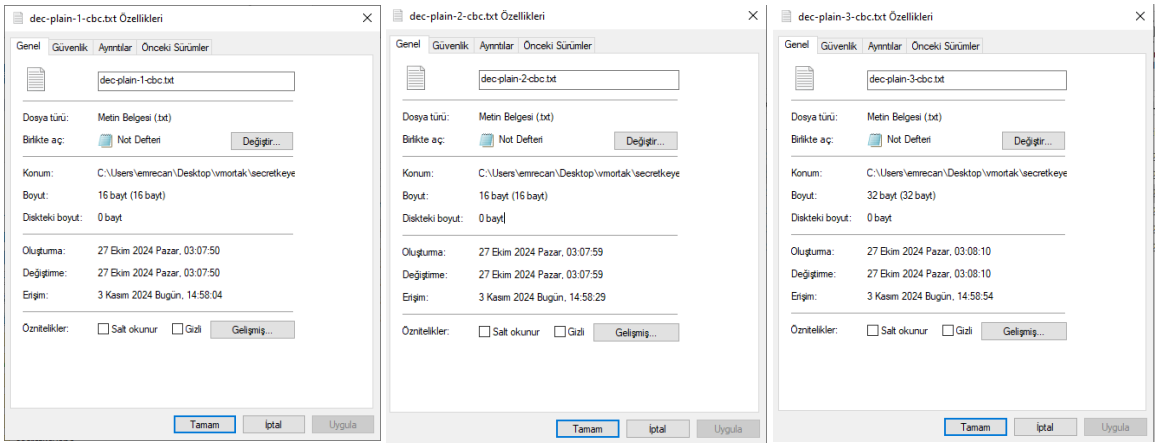
openssl enc -aes-256-ecb -d -nopad -in enc-cipher-2-ecb.bin -out dec-plain-2-ecb.txt

openssl enc -aes-256-ecb -d -nopad -in enc-cipher-3-ecb.bin -out dec-plain-3-ecb.txt





File Sizes:



Prior to encryption, the file size was 4,8,16 bytes. The data is padded with 12,8,16 bytes to fill the block size. File sizes rise to 16,16,32 bytes after encryption. The padding is eliminated during decryption, restoring the original size. The file size is 16 bytes, or the entire block size, prior to encryption. The file size grows to 32 bytes as a full block is filled, adding an additional 16 bytes of padding.

Padding AES-128-CFB AES-256-CFB

openssl enc -aes-128-cfb -e -in plain-1.txt -out enc-cipher-1-cfb.bin

openssl enc -aes-128-cfb -e -in plain-2.txt -out enc-cipher-2-cfb.bin

openssl enc -aes-128-cfb -e -in plain-3.txt -out enc-cipher-3-cfb.bin

openssl enc -aes-128-cfb -d -nopad -in enc-cipher-1-cfb.bin -out dec-plain-1-cfb.txt

openssl enc -aes-128-cfb -d -nopad -in enc-cipher-2-cfb.bin -out dec-plain-2-cfb.txt

openssl enc -aes-128-cfb -d -nopad -in enc-cipher-3-cfb.bin -out dec-plain-3-cfb.txt

openssl enc -aes-256-cfb -e -in plain-1.txt -out enc-cipher-1-cfb.bin

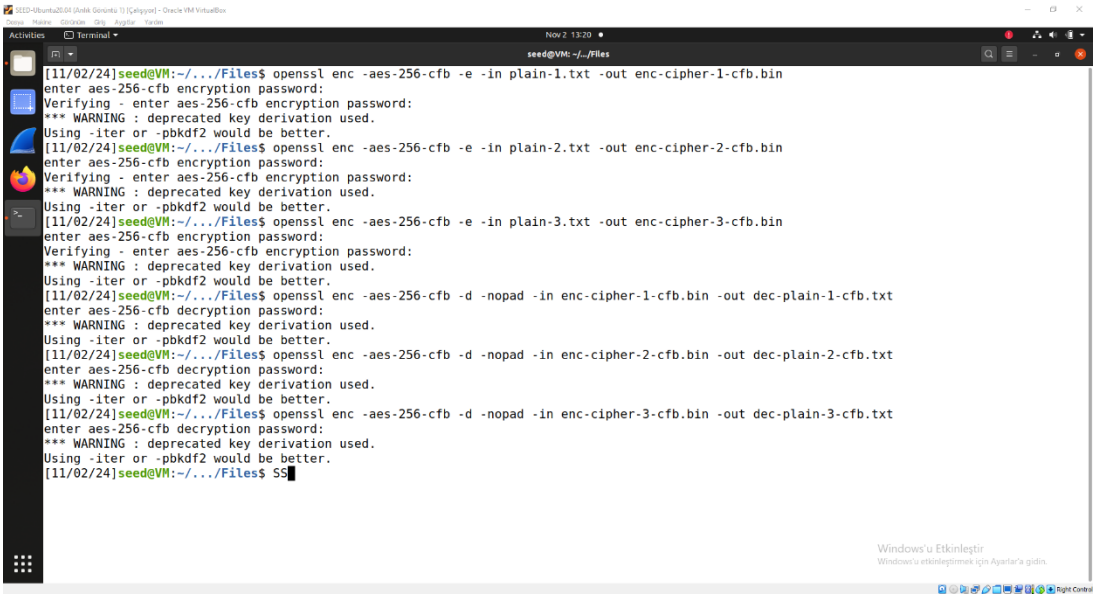
openssl enc -aes-256-cfb -e -in plain-2.txt -out enc-cipher-2-cfb.bin

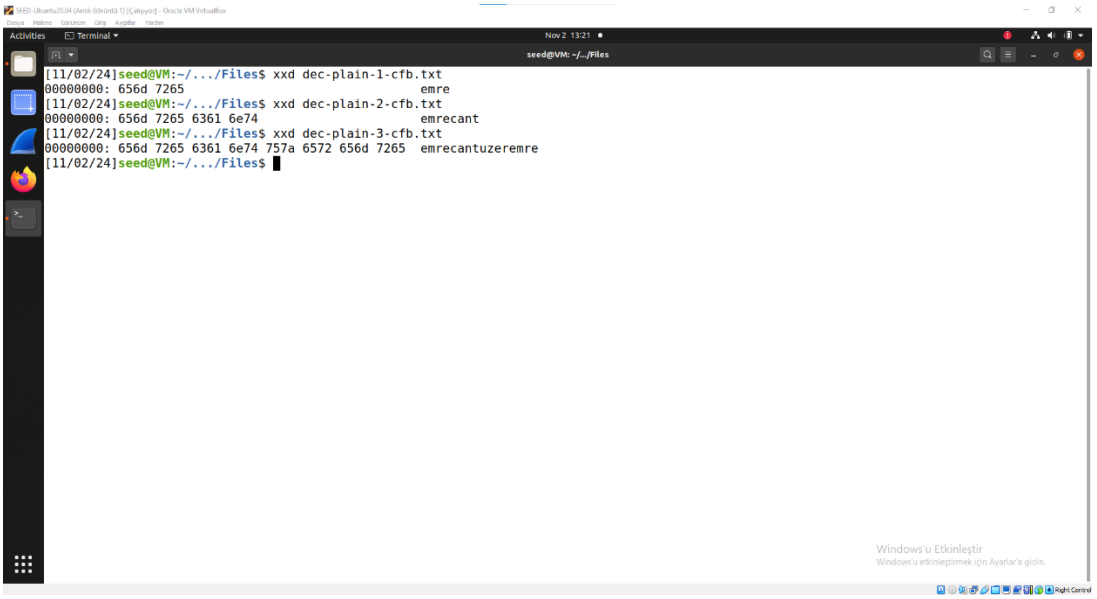
openssl enc -aes-256-cfb -e -in plain-3.txt -out enc-cipher-3-cfb.bin

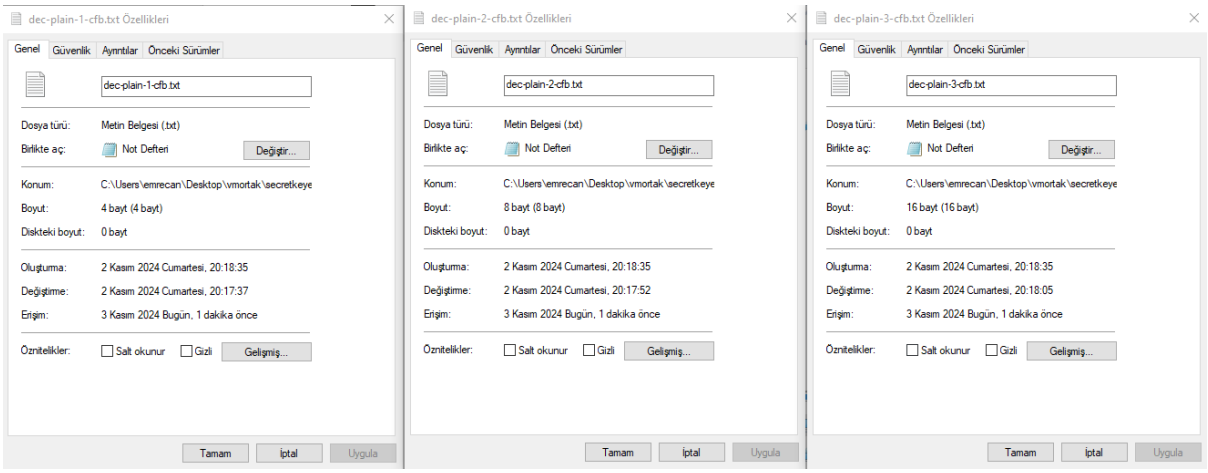
openssl enc -aes-256-cfb -d -nopad -in enc-cipher-1-cfb.bin -out dec-plain-1-cfb.txt

openssl enc -aes-256-cfb -d -nopad -in enc-cipher-2-cfb.bin -out dec-plain-2-cfb.txt

openssl enc -aes-256-cfb -d -nopad -in enc-cipher-3-cfb.bin -out dec-plain-3-cfb.txt







In streaming modes, the file size doesn't change while encryption and decryption are underway. Before and after encryption and decryption, the files' sizes don't change. These modes function independently of the data stream's block sizes.

Padding AES-128-CFB AES-256-CFB

openssl enc -aes-128-ofb -e -in plain-1.txt -out enc-cipher-1-ofb.bin

openssl enc -aes-128-ofb -e -in plain-2.txt -out enc-cipher-2-ofb.bin

openssl enc -aes-128-ofb -e -in plain-3.txt -out enc-cipher-3-ofb.bin

openssl enc -aes-128-ofb -d -nopad -in enc-cipher-1-ofb.bin -out dec-plain-1-cfb.txt

openssl enc -aes-128-ofb -d -nopad -in enc-cipher-2-ofb.bin -out dec-plain-2-cfb.txt

openssl enc -aes-128-ofb -d -nopad -in enc-cipher-3-ofb.bin -out dec-plain-3-cfb.txt

openssl enc -aes-256-ofb -e -in plain-1.txt -out enc-cipher-1-cfb.bin

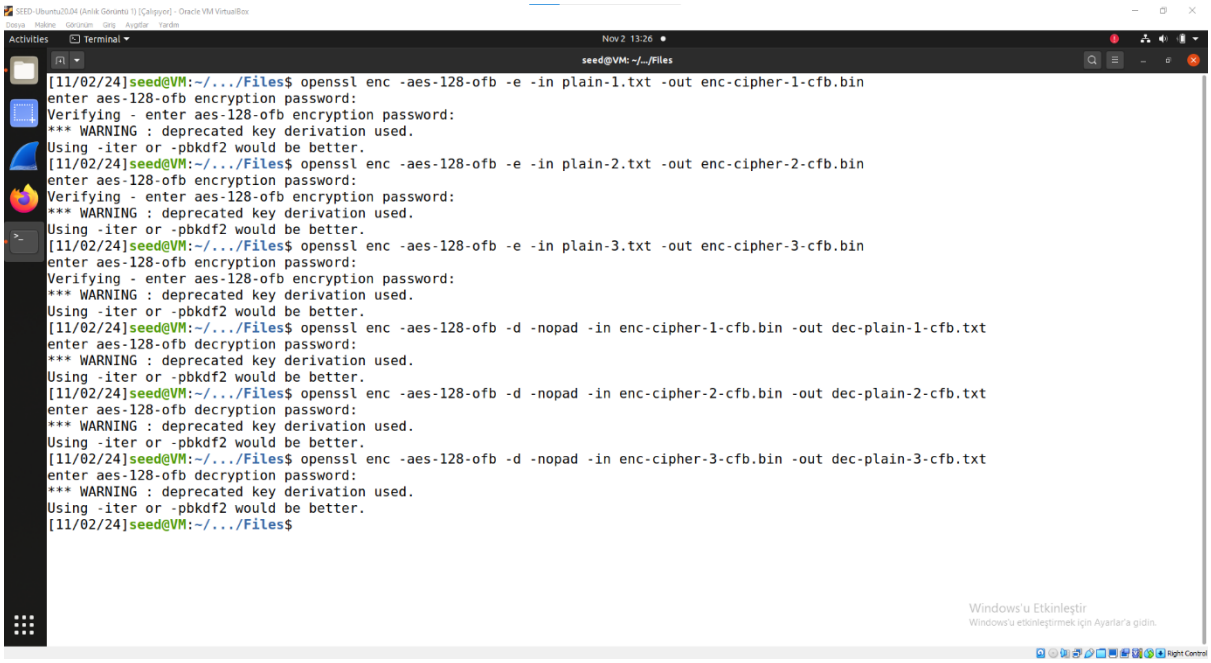
openssl enc -aes-256-ofb -e -in plain-2.txt -out enc-cipher-2-cfb.bin

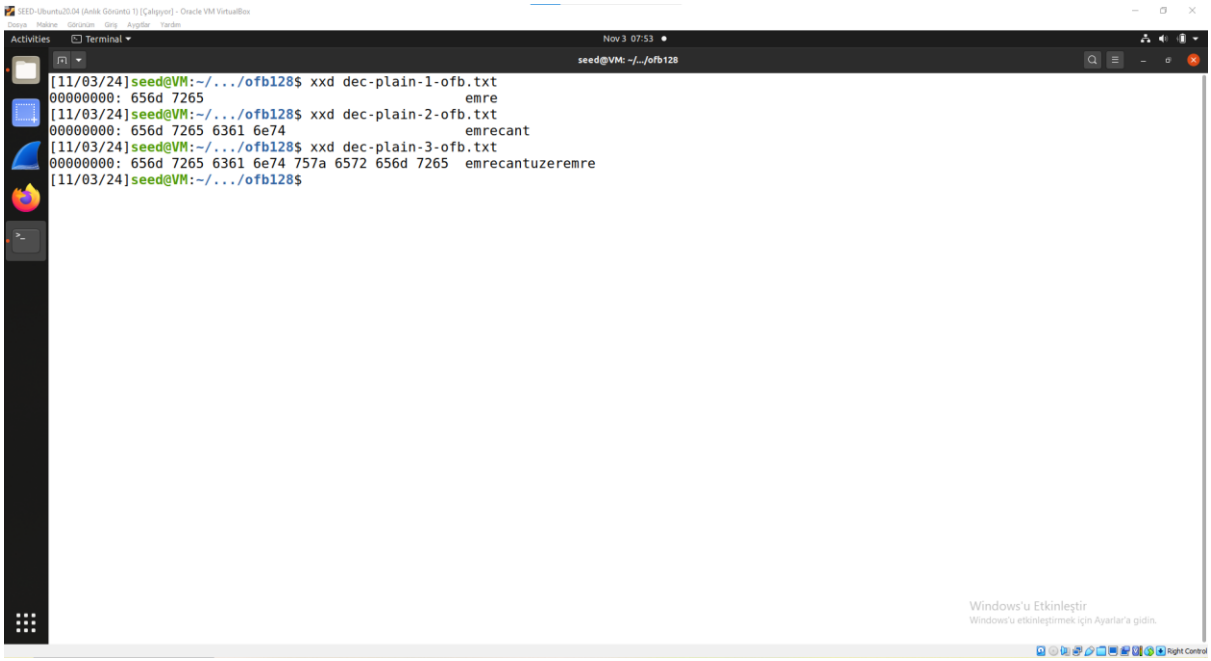
openssl enc -aes-256-ofb -e -in plain-3.txt -out enc-cipher-3-cfb.bin

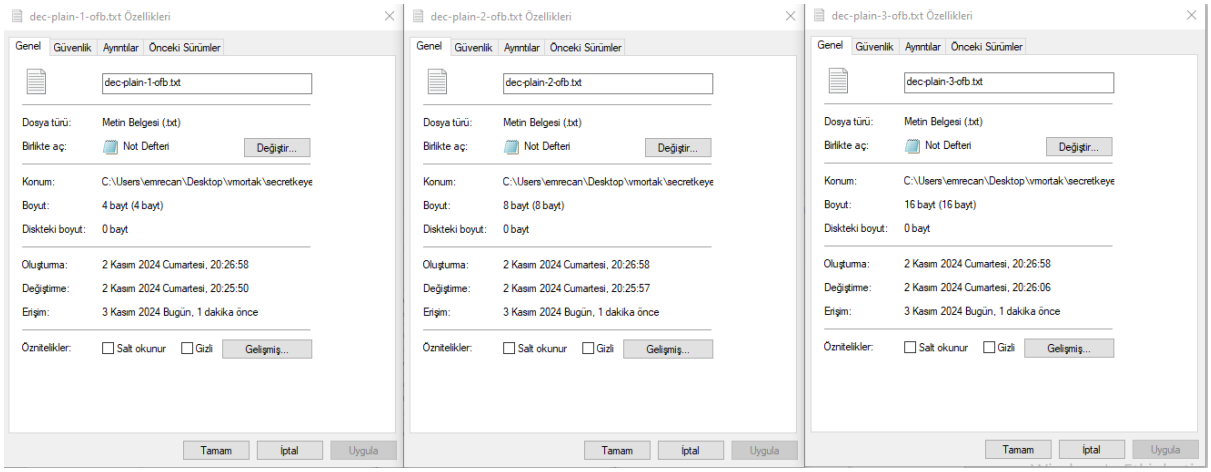
openssl enc -aes-256-ofb -d -nopad -in enc-cipher-1-ofb.bin -out dec-plain-1-cfb.txt

openssl enc -aes-256-ofb -d -nopad -in enc-cipher-2-ofb.bin -out dec-plain-2-cfb.txt

openssl enc -aes-256-ofb -d -nopad -in enc-cipher-3-ofb.bin -out dec-plain-3-cfb.txt



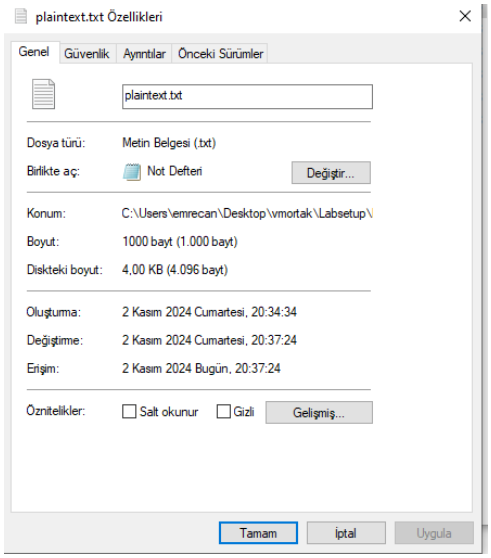




In streaming modes, the file size doesn't change while encryption and decryption are underway. Before and after encryption and decryption, the files' sizes don't change. These modes function independently of the data stream's block sizes.

Task 5: Error Propagation – Corrupted Cipher Text

Making a new text file with 1000 bytes is the first stage in this project.



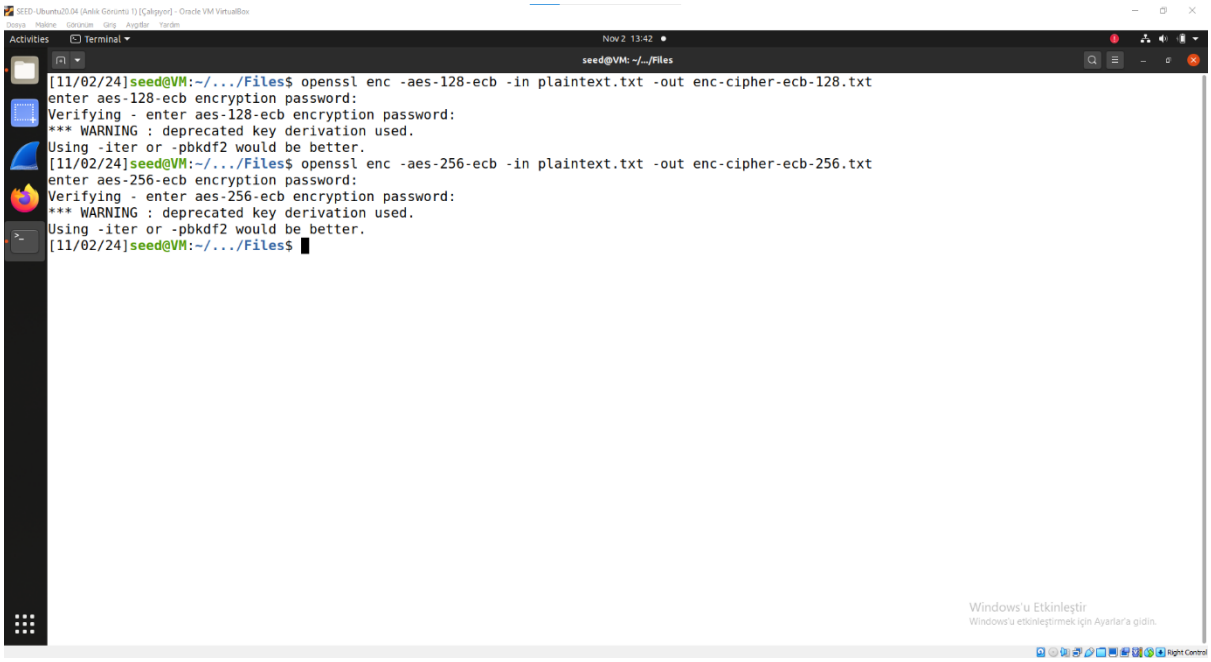
ECB Mode Analysis:

In ECB mode, each block is encrypted separately, therefore a mistake in one block will only result in that block's incorrect decryption. This makes it possible to decrypt blocks other than corrupted ones.

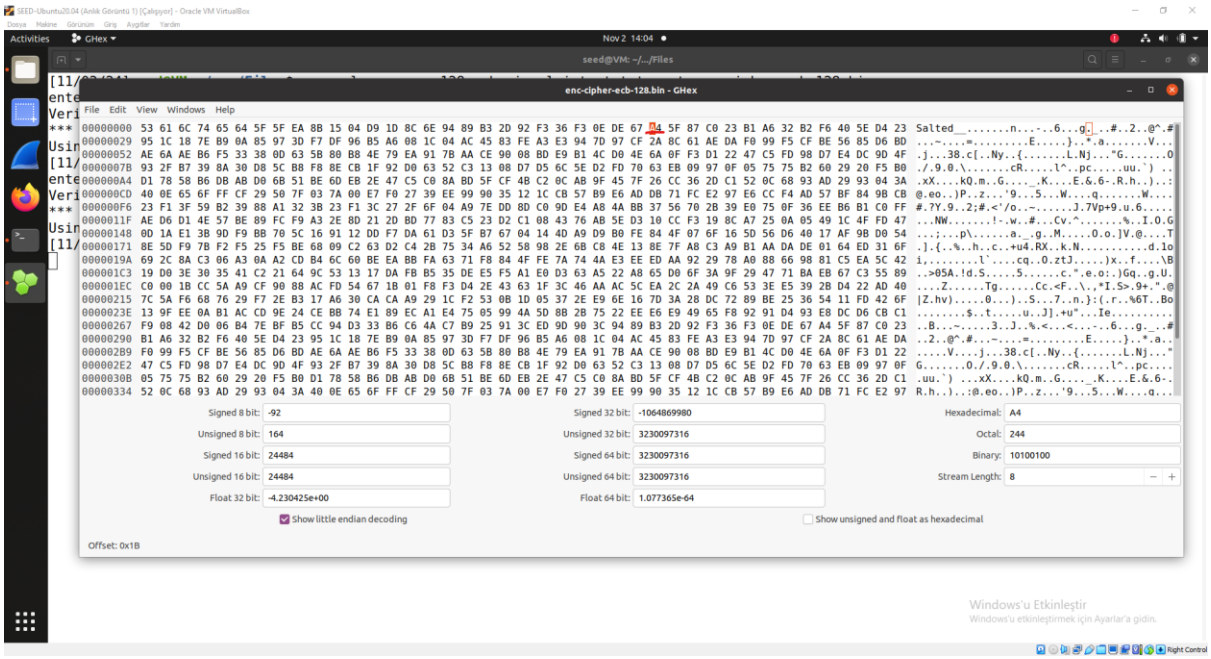
Encryption:

openssl enc -aes-128-ecb -in plaintext.txt -out enc-cipher-ecb-128.bin

openssl enc -aes-256-ecb -in plaintext.txt -out enc-cipher-ecb-256.bin



At the *55th* Byte, switched from B4 to B5:



Decryption:

openssl enc -aes-128-ecb -d -in enc-cipher-ecb-128.bin -out dec-plain-128-ecb.txt

openssl enc -aes-128-ecb -d -in enc-cipher-ecb-256.bin -out dec-plain-256-ecb.txt

Output:

The remaining blocks were correctly decoded, with the exception of the faulty block.

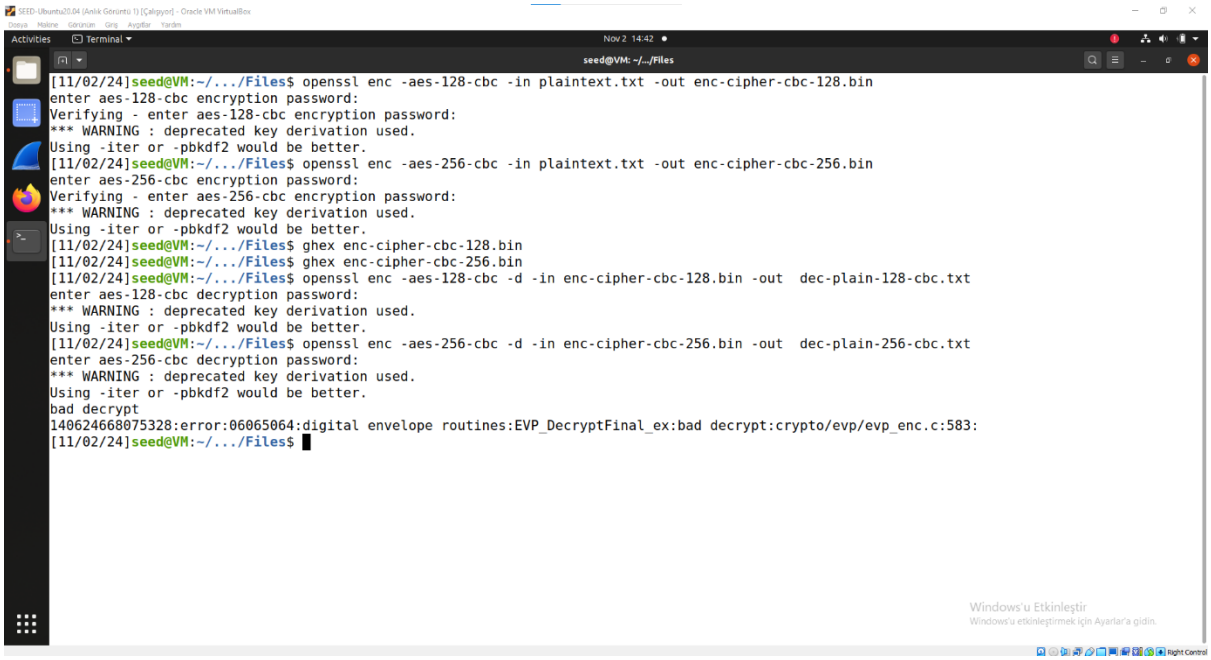
CBC Mode Analysis:

When a block in CBC mode becomes corrupted, the decryption of that block fails, and the corrupted block is impacted along with one consecutive block. Because the first block following the bad block is XORed with the output of the bad block that was erroneously decoded, it is inaccurate.

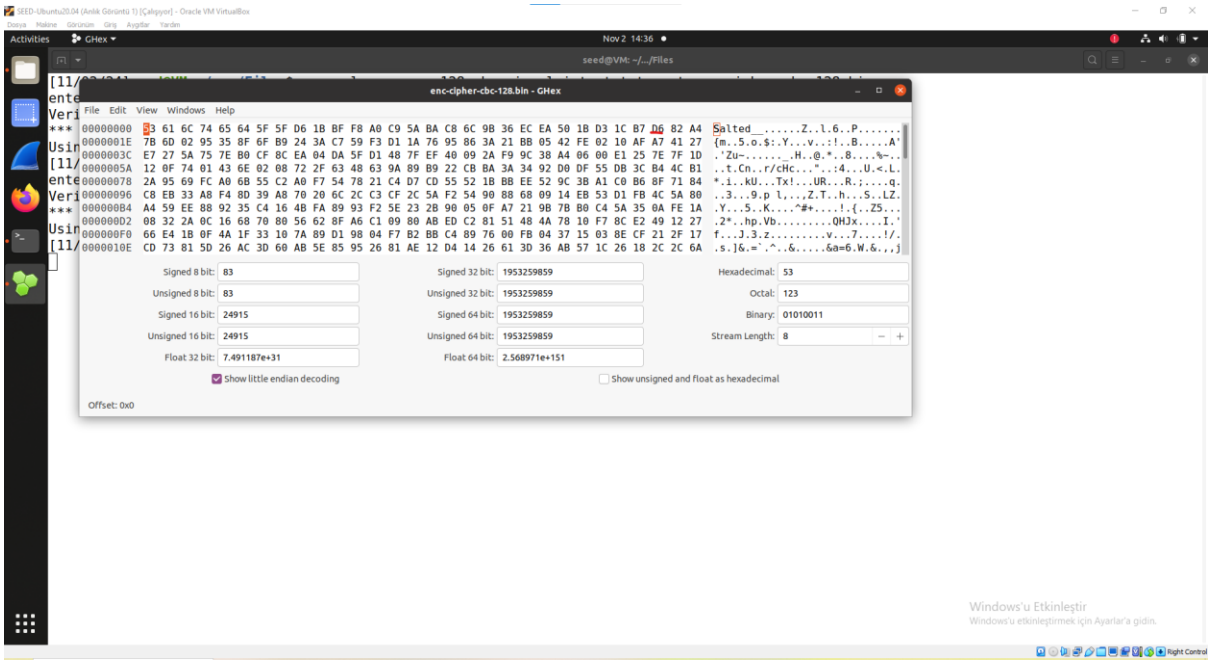
Encryption:

openssl enc -aes-128-cbc -in plaintext.txt -out enc-cipher-cbc-128.bin

openssl enc -aes-256-cbc -in plaintext.txt -out enc-cipher-cbc-256.bin



At the *55th* Byte, switched from D6 to D7:

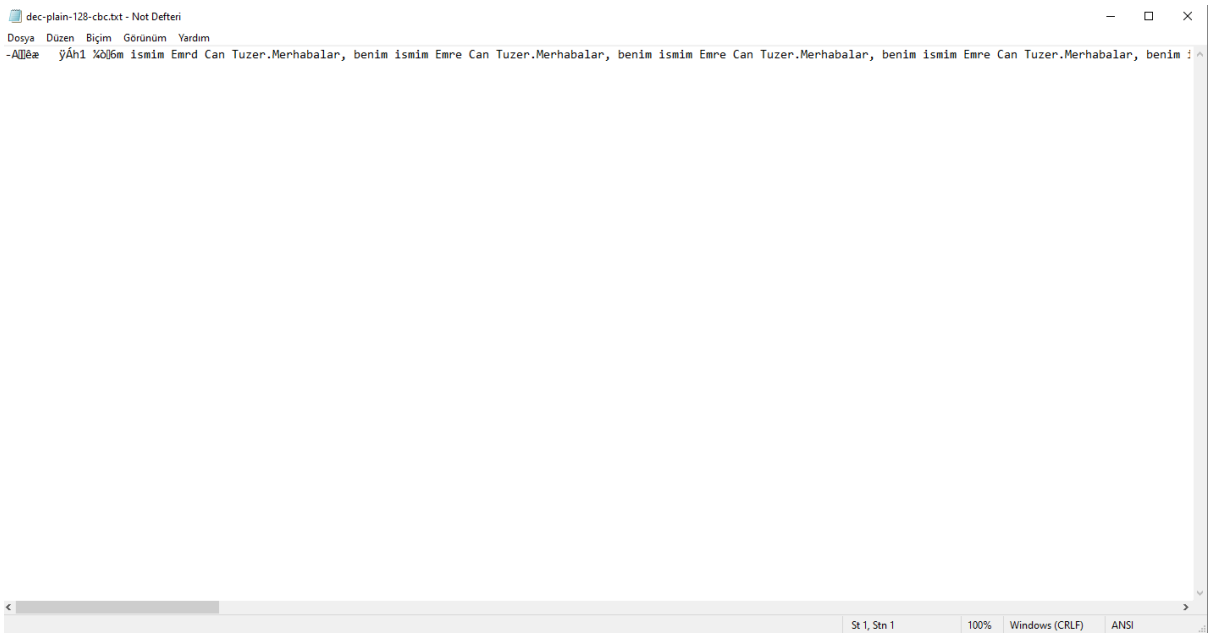


Decryption:

openssl enc -aes-128-cbc -d -in enc-cipher-cbc-128.bin -out dec-plain-128-cbc.txt

openssl enc -aes-256-cbc -d -in enc-cipher-cbc-256.bin -out dec-plain-256-cbc.txt

Output:



While subsequent blocks were decoded successfully, the block with the 55th byte and the block immediately after it were deciphered wrongly.

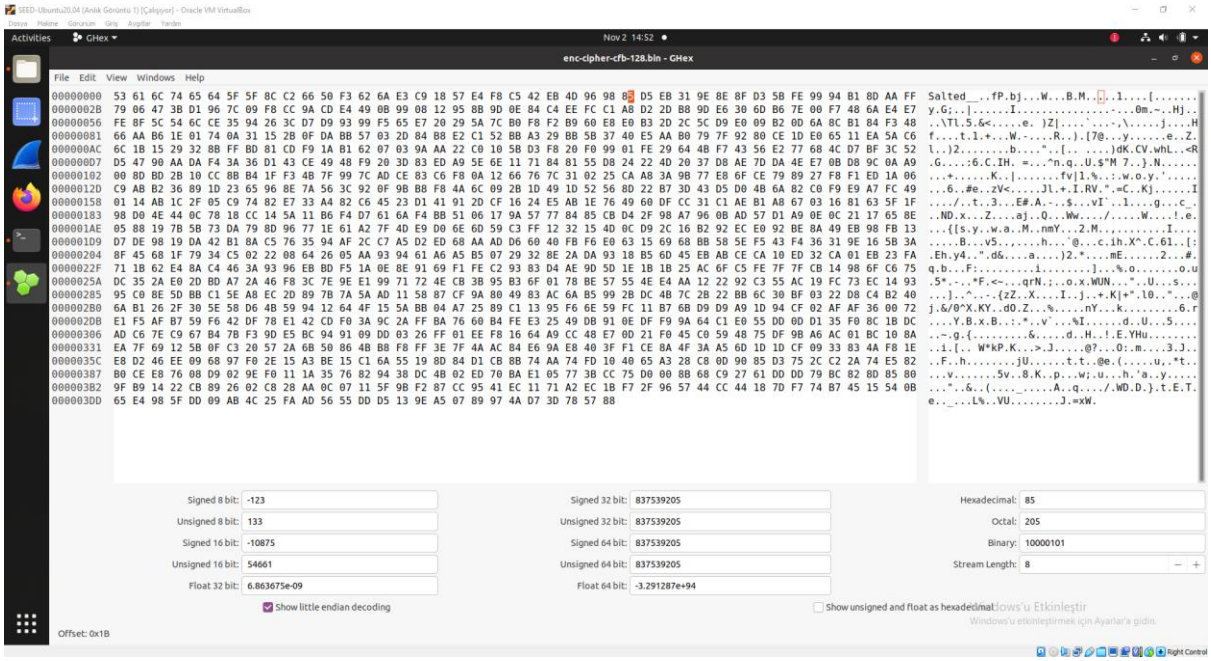
CFB Mode Analysis:

Encryption:

openssl enc -aes-128-cfb -in plaintext.txt -out enc-cipher-cfb-128.bin

openssl enc -aes-256-cfb -in plaintext.txt -out enc-cipher-cfb-256.bin

At the *55th* Byte, switched from 85 to 86:

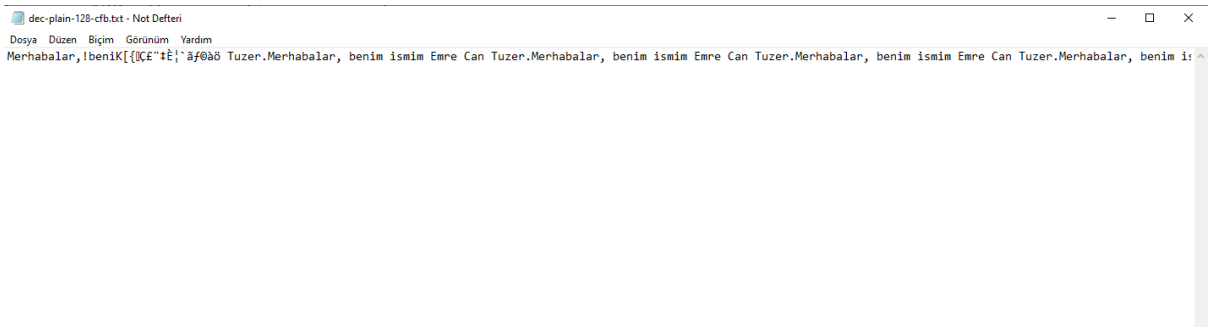


Decryption:

openssl enc -aes-128-cfb -d -in enc-cipher-cfb-128.bin -out dec-plain-128-cfb.txt

openssl enc -aes-256-cfb -d -in enc-cipher-cfb-256.bin -out dec-plain-256-cfb.txt

Output:



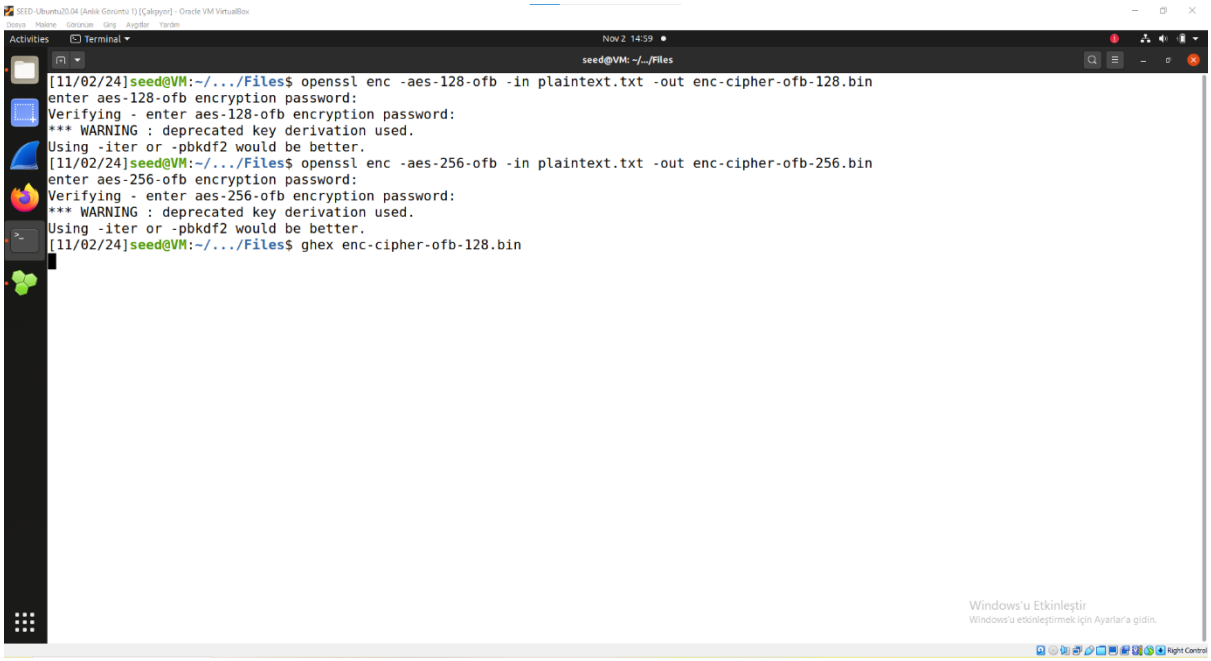
OFB Mode Analysis:

In OFB mode, a string from the encryption stream is separately XORed with each block. Because of this, OFB mode is less likely to propagate errors. Only that block is impacted when it becomes corrupted.

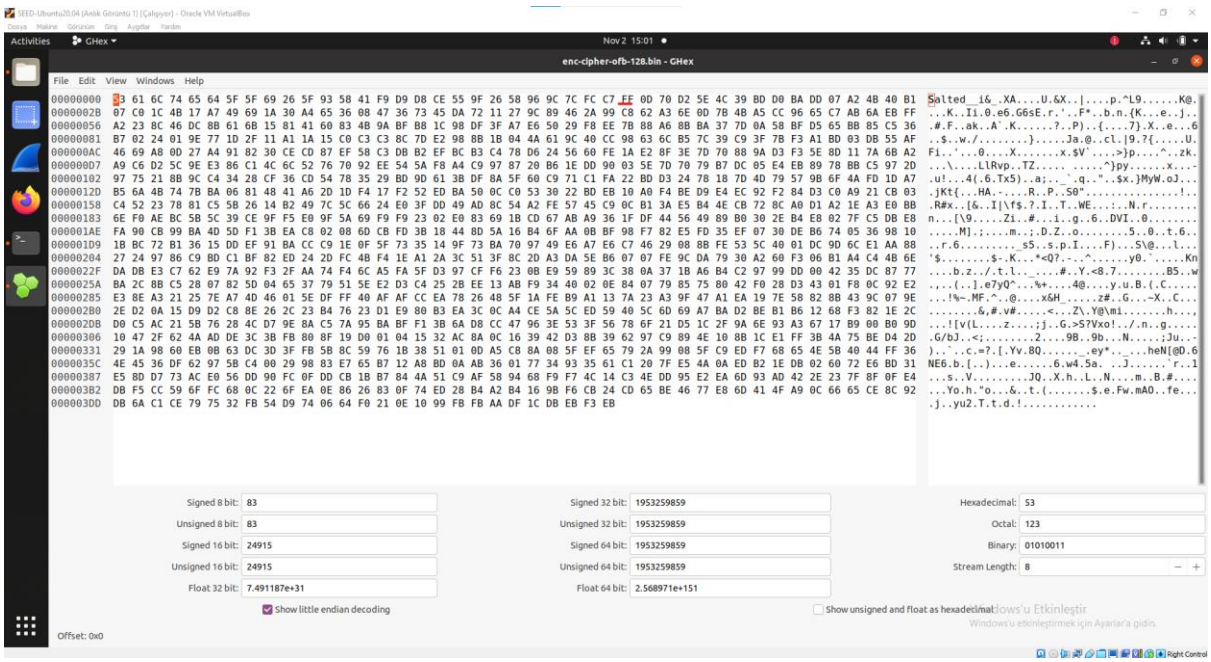
Encryption:

openssl enc -aes-128-ofb -in plaintext.txt -out enc-cipher-ofb-128.bin

openssl enc -aes-256-ofb -in plaintext.txt -out enc-cipher-ofb-256.bin



At the *55th* Byte, switched from FE to FF:

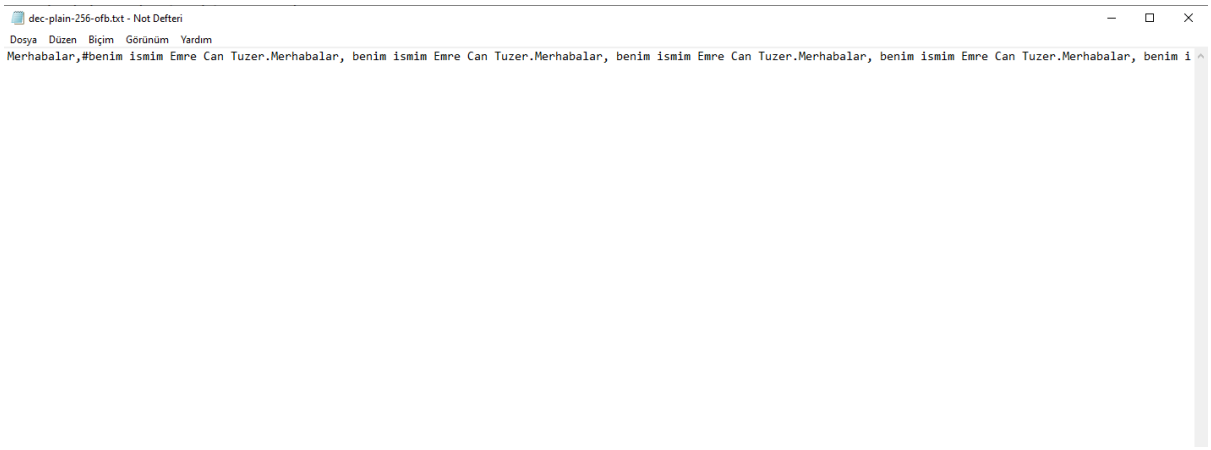


Decryption:

openssl enc -aes-128-cfb -d -in enc-cipher-cfb-128.bin -out dec-plain-128-cfb.txt

openssl enc -aes-256-cfb -d -in enc-cipher-cfb-256.bin -out dec-plain-256-cfb.txt

Output:



The other blocks were all proper, with the exception of the block that included the 55th byte.

Finally, it has been noted that OFB isolates faults and limits error propagation to a minimum, CBC and CFB cause flaws to spread by upsetting the blockchain and impacting data integrity, while ECB limits errors on a block basis and can be decrypted independently.