Overview:

The objective of this lab was to gain hands-on experience with block ciphers and encryption modes. Through this lab, I became familiar with encryption algorithms and how different encryption modes impact the resulting ciphertext. By the end of the lab, I was able to use OpenSSL and hex editors to encrypt, decrypt, and analyze messages.

Lab Environment:

- **OpenSSL**: I used OpenSSL to perform encryption and decryption tasks. The Kali Linux environment I worked in already had OpenSSL binaries installed. However, if I wanted to use OpenSSL libraries for programming, I would need to install additional components such as header files and manuals.
- **Hex Editor**: For modifying and viewing binary files, I used GHex, a hex editor. This tool allowed me to view data in both hex and ASCII formats. While I worked with GHex, I was informed that other hex editors like Bless might offer additional features. If necessary, I could install them.

Analysis:

Encryption Using Different Ciphers and Modes

For this task, I explored various encryption algorithms and modes using OpenSSL commands. I used the following command format to encrypt and decrypt files:

% openssl enc ciphertype -e -in plain.txt -out cipher.bin -K

00112233445566778889aabbccddeeff -iv 0102030405060708

I tested three different encryption modes: ECB, CBC, OFB and CFB. Below are my results:

ECB:

```
┌──(kali㉿kali)-[~]
└─$ openssl enc -aes-128-ecb -e -in /home/kali/Downloads/plain.txt -out ciphe
r.bin -K 00112233445566778889aabbcddeefff
hex string is too short, padding with zero bytes to length
```

CBC:

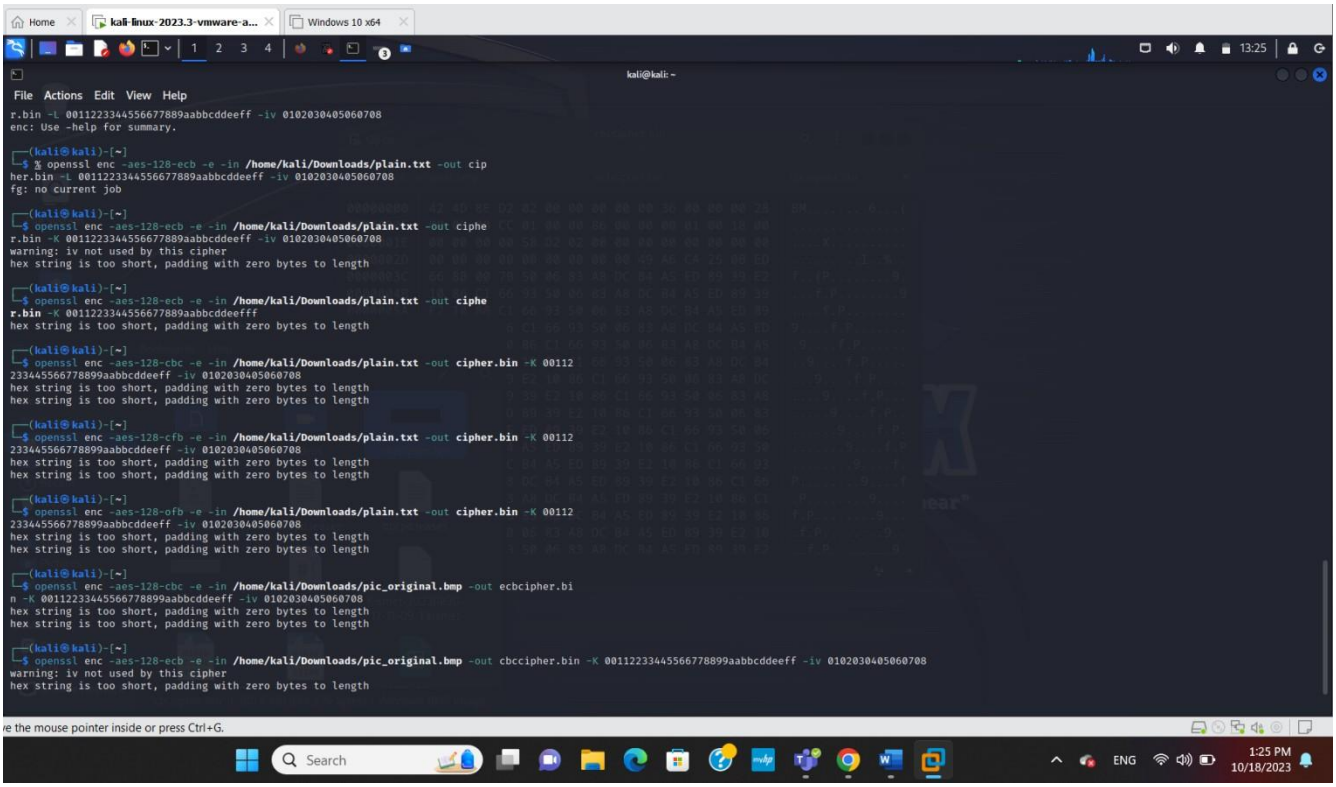
```
┌──(kali㉿kali)-[~]
└─$ openssl enc -aes-128-cbc -e -in /home/kali/Downloads/plain.txt -out cipher.bin -K 00112
233445566778899aabbcddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
```

CFB:

```
┌──(kali㉿kali)-[~]
└─$ openssl enc -aes-128-cfb -e -in /home/kali/Downloads/plain.txt -out cipher.bin -K 00112
233445566778899aabbcddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
```

OFB:

```
┌──(kali㉿kali)-[~]
└─$ openssl enc -aes-128-ofb -e -in /home/kali/Downloads/plain.txt -out cipher.bin -K 00112
233445566778899aabbcddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
```

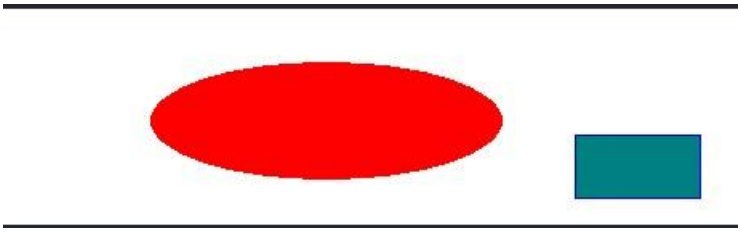


Encryption Mode – ECB vs. CBC

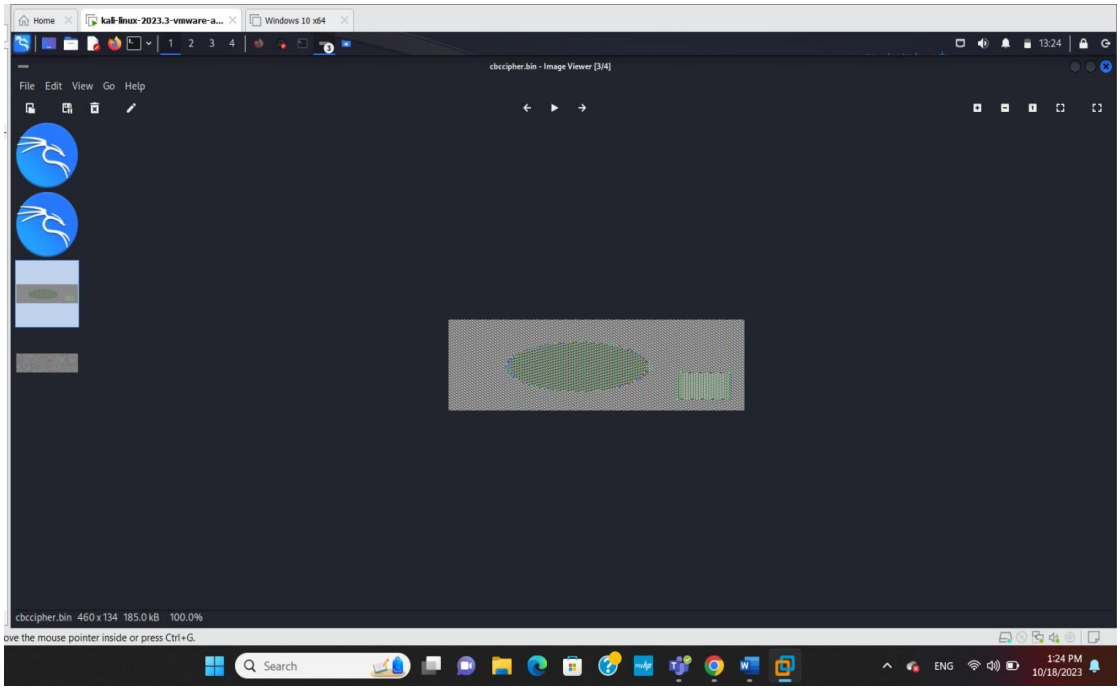
I encrypted an image file using both ECB and CBC modes. After encrypting the image, I replaced the header of the encrypted file with the header from the original image using a hex editor, to ensure the file was viewed correctly as a BMP.

The file pic_original.bmp contains a simple picture.

Original image:

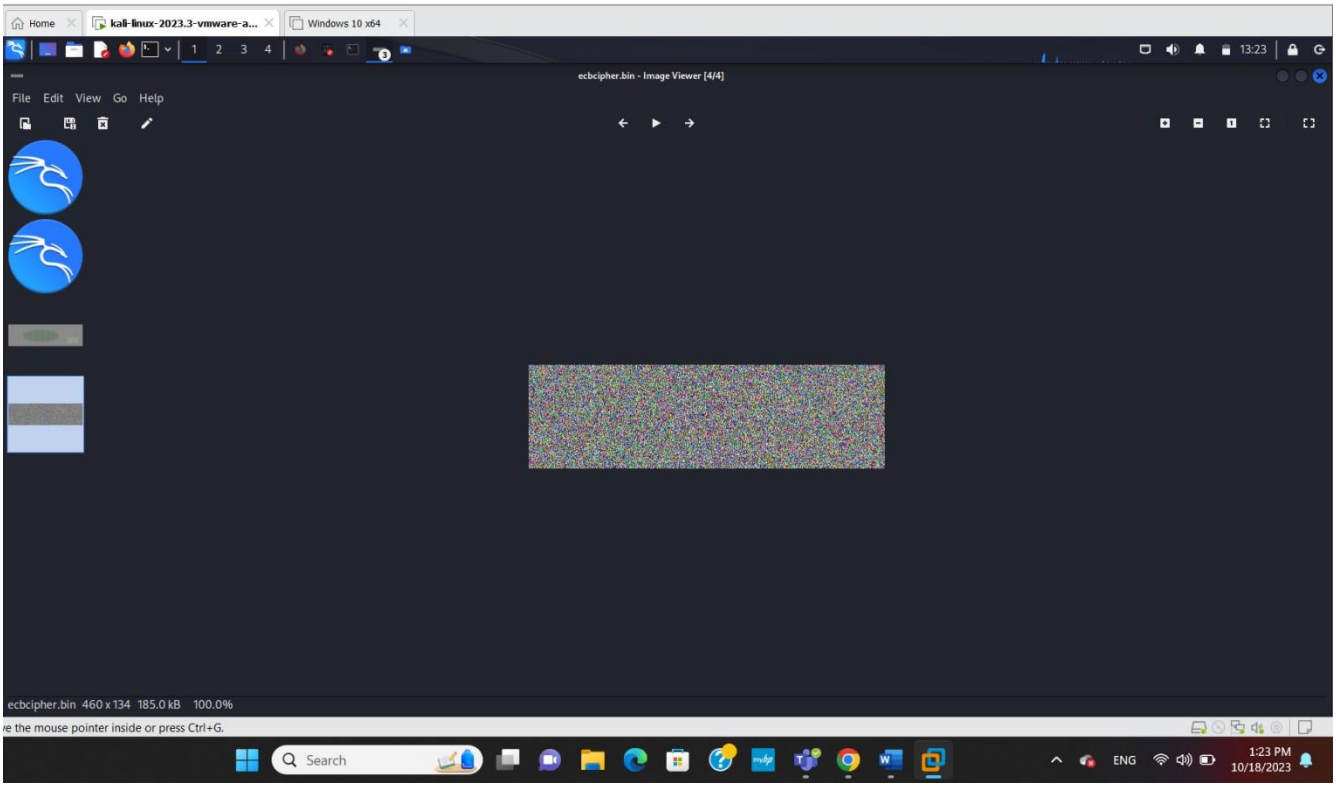


ECB:



With **ECB mode** I observed that even though the image was encrypted, I could still gather some information about the original image. This is because ECB mode encrypts identical plaintext into identical ciphertext, revealing patterns in the encrypted data.

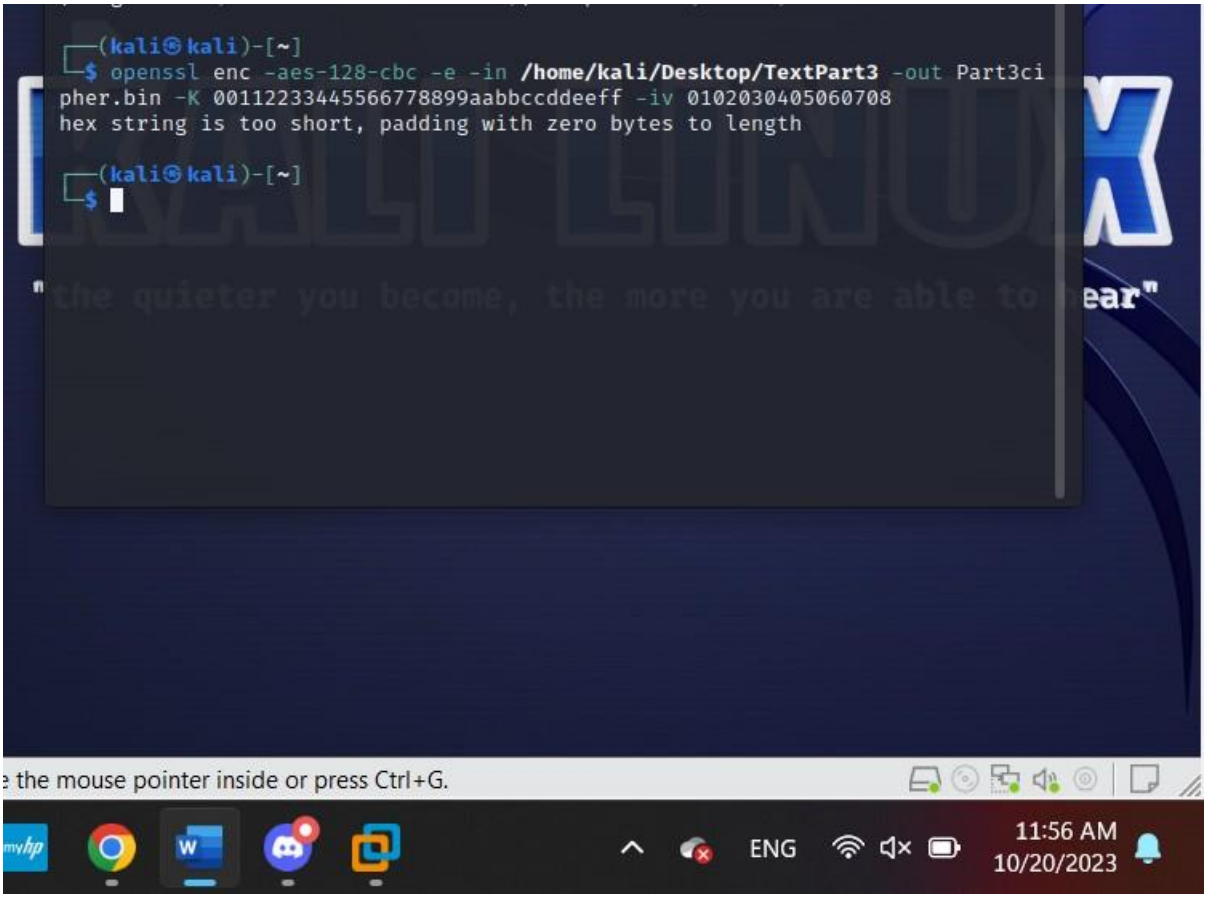
CBC:

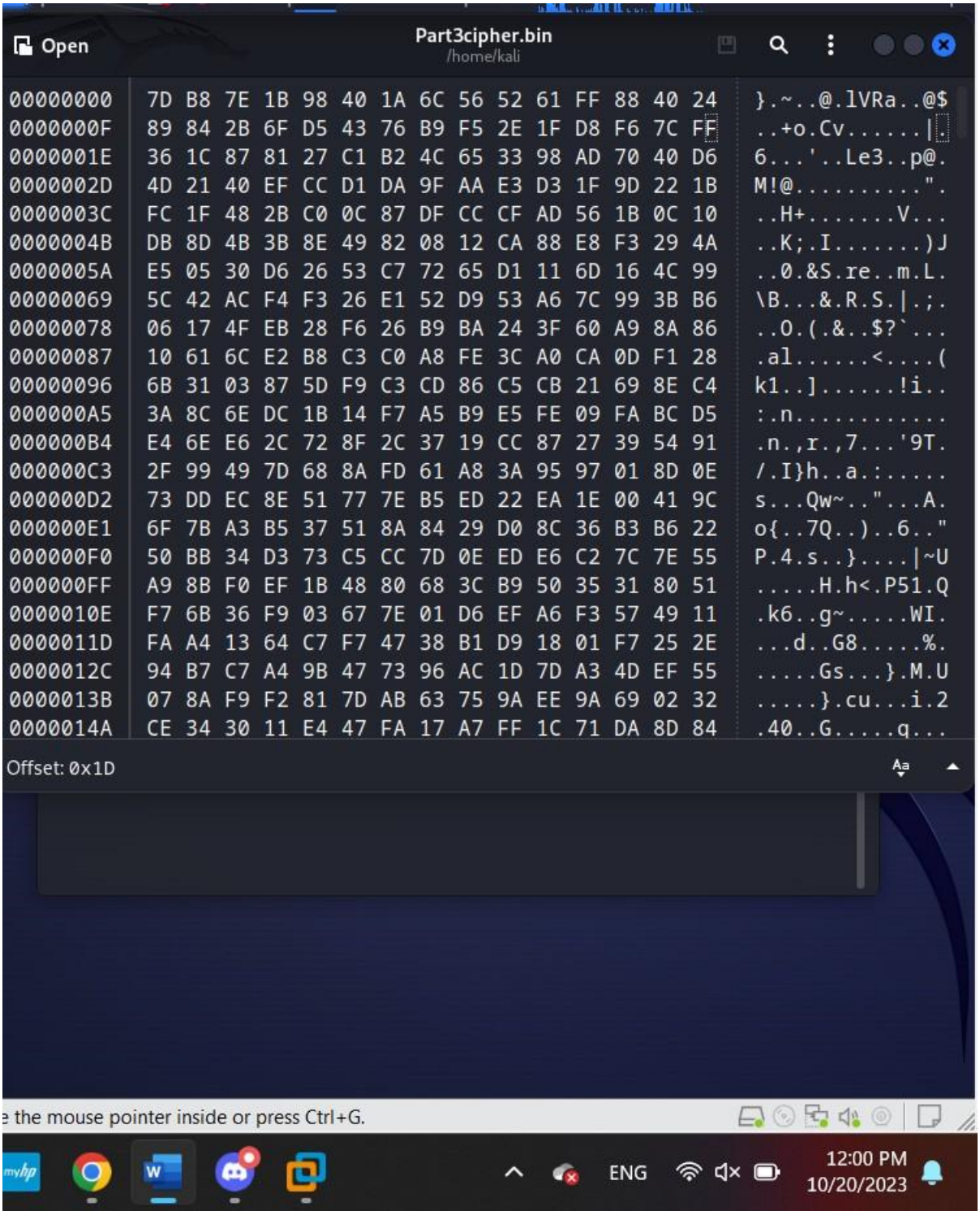


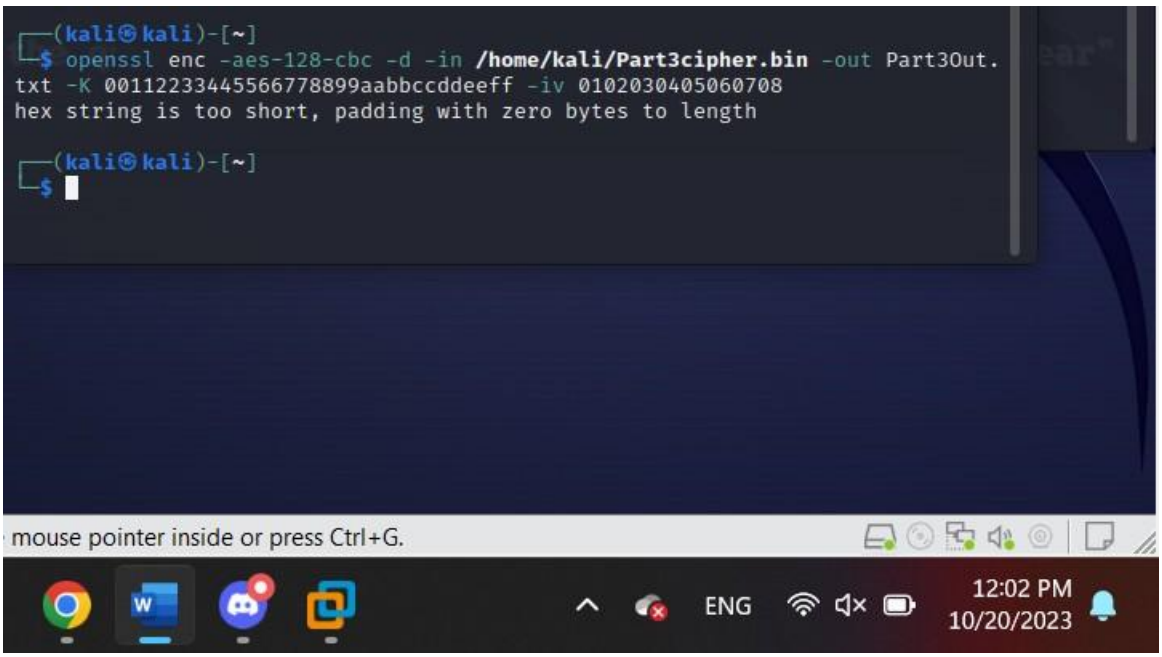
With CBC mode, the result was significantly different. The encryption removed the one-to-one correspondence between plaintext and ciphertext, which made it difficult to derive any useful information about the original image from the encrypted file.

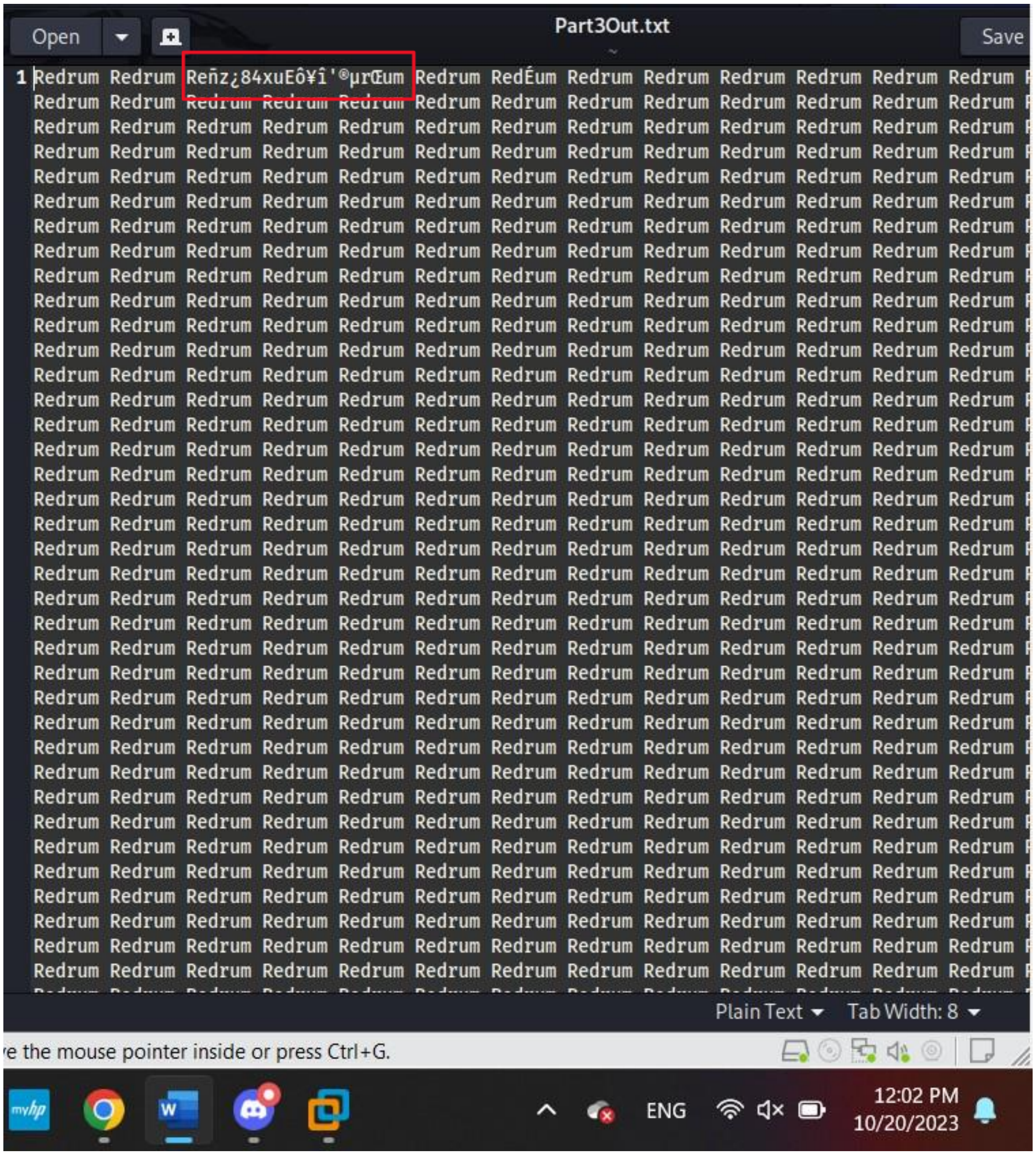
CBC Encryption Mode – Corrupted Ciphertext

For this task, I created a text file of at least 64 bytes and encrypted it using AES-128. Then, I deliberately corrupted a single bit of the 30th byte in the encrypted file using a hex editor. Afterward, I attempted to decrypt the corrupted file.




Open
Part3cipher.bin
/home/kali
00000000 7D B8 7E 1B 98 40 1A 6C 56 52 61 FF 88 40 24 }.~..@.lVRa..@$
0000000F 89 84 2B 6F D5 43 76 B9 F5 2E 1F D8 F6 7C FF ..+o.Cv.....|.
0000001E 36 1C 87 81 27 C1 B2 4C 65 33 98 AD 70 40 D6 6...'..Le3..p@.
0000002D 4D 21 40 EF CC D1 DA 9F AA E3 D3 1F 9D 22 1B M!@.....".
0000003C FC 1F 48 2B C0 0C 87 DF CC CF AD 56 1B 0C 10 ..H+.....V...
0000004B DB 8D 4B 3B 8E 49 82 08 12 CA 88 E8 F3 29 4A ..K;.I.....)J
0000005A E5 05 30 D6 26 53 C7 72 65 D1 11 6D 16 4C 99 ..0.&S.re..m.L.
00000069 5C 42 AC F4 F3 26 E1 52 D9 53 A6 7C 99 3B B6 \B...&.R.S.|.;.
00000078 06 17 4F EB 28 F6 26 B9 BA 24 3F 60 A9 8A 86 ..O.(.&..$?`...
00000087 10 61 6C E2 B8 C3 C0 A8 FE 3C A0 CA 0D F1 28 .al.....<....(
00000096 6B 31 03 87 5D F9 C3 CD 86 C5 CB 21 69 8E C4 k1..]......!i..
000000A5 3A 8C 6E DC 1B 14 F7 A5 B9 E5 FE 09 FA BC D5 :.n.....
000000B4 E4 6E E6 2C 72 8F 2C 37 19 CC 87 27 39 54 91 .n.,r.,7...'9T.
000000C3 2F 99 49 7D 68 8A FD 61 A8 3A 95 97 01 8D 0E /.I}h..a.:.....
000000D2 73 DD EC 8E 51 77 7E B5 ED 22 EA 1E 00 41 9C s...Qw~.."...A.
000000E1 6F 7B A3 B5 37 51 8A 84 29 D0 8C 36 B3 B6 22 o{..7Q..)..6.."
000000F0 50 BB 34 D3 73 C5 CC 7D 0E ED E6 C2 7C 7E 55 P.4.s..}....|~U
000000FF A9 8B F0 EF 1B 48 80 68 3C B9 50 35 31 80 51 .....H.h<.P51.Q
0000010E F7 6B 36 F9 03 67 7E 01 D6 EF A6 F3 57 49 11 .k6..g~.....WI.
0000011D FA A4 13 64 C7 F7 47 38 B1 D9 18 01 F7 25 2E ...d..G8.....%.
0000012C 94 B7 C7 A4 9B 47 73 96 AC 1D 7D A3 4D EF 55 .....Gs...}.M.U
0000013B 07 8A F9 F2 81 7D AB 63 75 9A EE 9A 69 02 32 .....}.cu...i.2
0000014A CE 34 30 11 E4 47 FA 17 A7 FF 1C 71 DA 8D 84 .40..G.....q...
Offset: 0x1D
e the mouse pointer inside or press Ctrl+G.
ENG
12:00 PM
10/20/2023


(kali㉿kali)-[~]
$ openssl enc -aes-128-cbc -d -in /home/kali/Part3cipher.bin -out Part3Out.
txt -K 00112233445566778899aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
(kali㉿kali)-[~]
$
mouse pointer inside or press Ctrl+G.
ENG
12:02 PM
10/20/2023



Looking at the output, I could clearly see the corrupted text at the top. This occurred because I changed some of the bits in the encrypted file using GHex.