

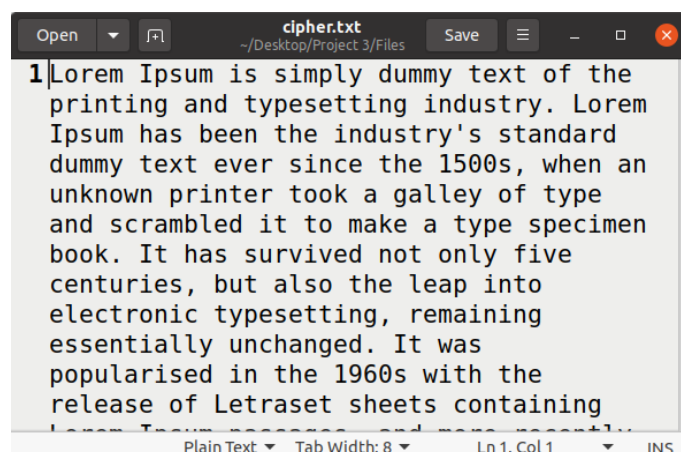
## Task 2: Encryption using Different Ciphers and Modes

```
[11/07/22]seed@VM:~/.../Files$ 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
[11/07/22]seed@VM:~/.../Files$ openssl enc -bf-cbc -e -in plain.txt -out cipher2.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[11/07/22]seed@VM:~/.../Files$ 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
[11/07/22]seed@VM:~/.../Files$
```

**Encryption** of plain.txt into cipher.bin files using CBC, BF-CBC and CFB.

```
cr and _ex_bu de crypt/ crypto/ evp/ evp_ encr.c:300.
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -in cipher.bin -out cipher.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -bf-cbc -d -in cipher2.bin -out cipher2.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -d -in cipher3.bin -out cipher3.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
```

**Decryption** of cipher.bin files into cipher.txt files.



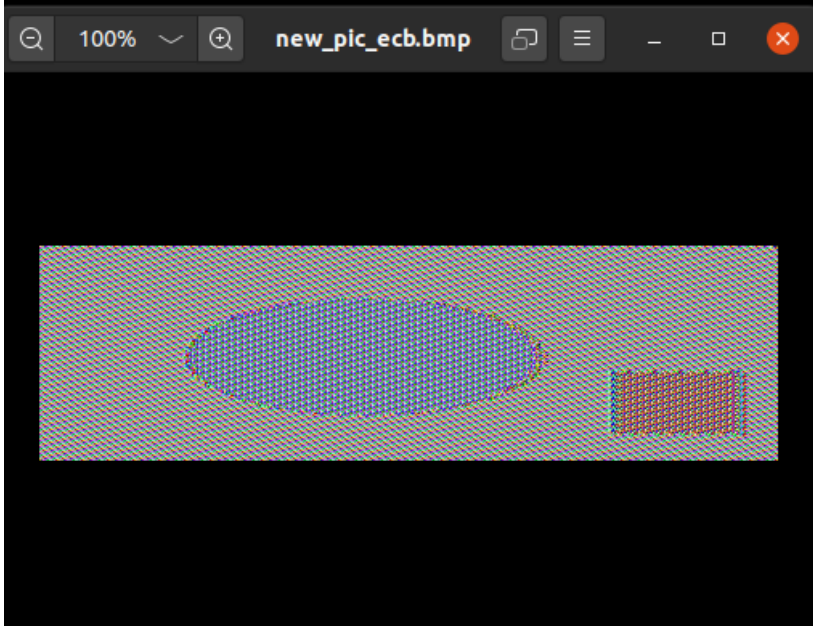
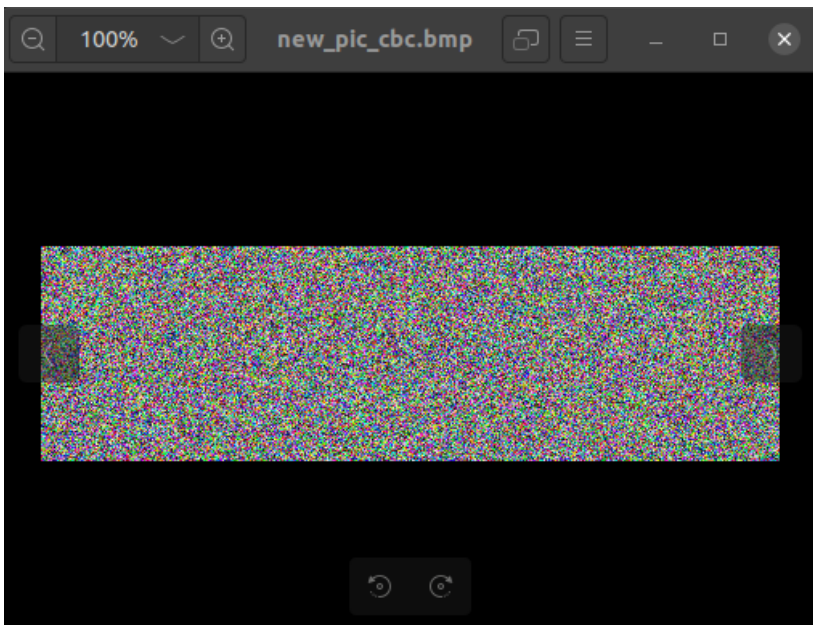
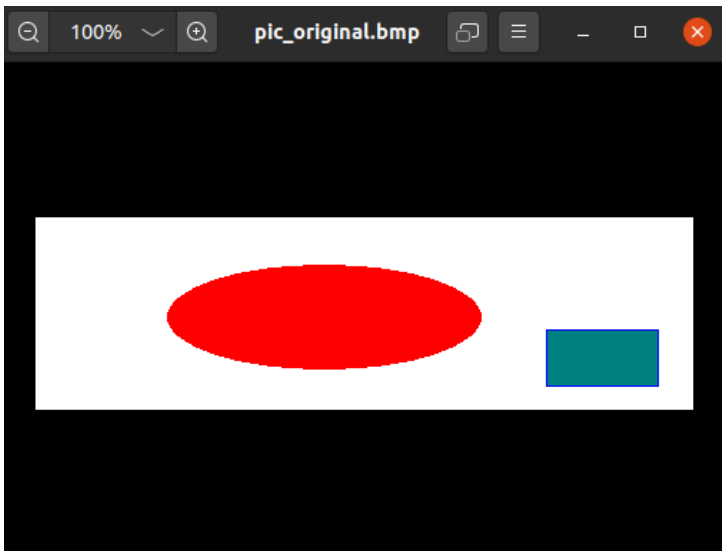
## Successful decryption

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

```
[11/07/22]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in pic_
original.bmp -out pic_ecb.bmp -K 00112233445566778889aabbccddeeff
-iv 0102030405060708
warning: iv not used by this cipher
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in pic
original.bmp -out pic_cbc.bmp -K 00112233445566778889aabbccddeeff
-iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$
```

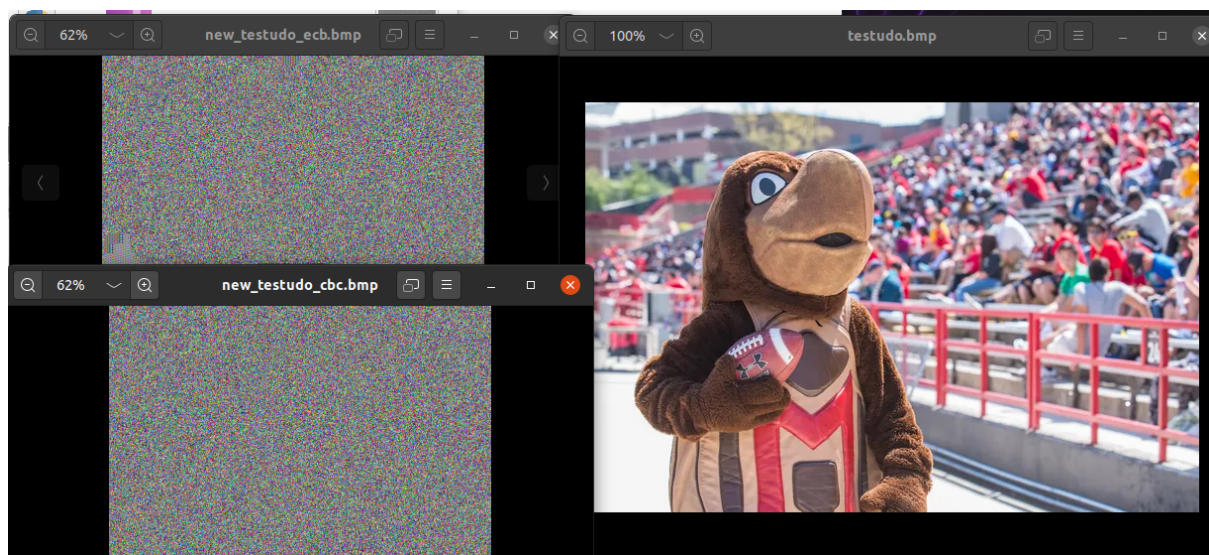
```
[11/13/22]seed@VM:~/.../Files$ head -c 54 pic_original.bmp > header
[11/13/22]seed@VM:~/.../Files$ tail -c +55 pic_ecb.bmp > body
[11/13/22]seed@VM:~/.../Files$ cat header body > new_pic_ecb.bmp
[11/13/22]seed@VM:~/.../Files$ head -c 54 pic_original.bmp > header
[11/13/22]seed@VM:~/.../Files$ tail -c +55 pic_cbc.bmp > body
[11/13/22]seed@VM:~/.../Files$ cat header body > new_pic_cbc.bmp
```

The above 2 images show the process of using ECB and CBC encryptions on the pic\_original.bmp image and the conversion of the header information of the generated files.



As shown in the 3 images above, ECB is the more insecure encryption scheme as the image still resembles the original image. This is because when identical plaintext blocks are encrypted into identical ciphertext blocks, a pattern among the data generated can be observed. In contrast, CBC is more secure as the image is wholly converted into random white noise.

```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in testudo.bmp -out testudo_ecb.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708
warning: iv not used by this cipher
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in testudo.bmp -out testudo_cbc.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ head -c 54 testudo.bmp > header
[11/13/22]seed@VM:~/.../Files$ tail -c +55 testudo_ecb.bmp > body
[11/13/22]seed@VM:~/.../Files$ cat header body > new_testudo_ecb.bmp
[11/13/22]seed@VM:~/.../Files$ head -c 54 testudo.bmp > header
[11/13/22]seed@VM:~/.../Files$ tail -c +55 testudo_cbc.bmp > body
[11/13/22]seed@VM:~/.../Files$ cat header body > new_testudo_cbc.bmp
```



For the 2nd image, I used the above image of Testudo. The observed results here are different from pic\_original.bmp's. Both pictures generated using ECB and CBC show that the original image has been converted to random white noise and is unrecognizable. A plausible explanation is that the testudo.bmp image is more complicated than pic\_original.bmp.



## Task 4: Padding

### 1.

```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plain.txt -out cipher_cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in plain.txt -out cipher_cfb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in plain.txt -out cipher_ofb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in plain.txt -out cipher_ecb.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
warning: iv not used by this cipher
```

```
-rw-rw-r-- 1 seed seed 576 Nov 13 08:17 cipher_cbc.txt
-rw-rw-r-- 1 seed seed 575 Nov 13 08:17 cipher_cfb.txt
-rw-rw-r-- 1 seed seed 576 Nov 13 08:18 cipher_ecb.txt
-rw-rw-r-- 1 seed seed 575 Nov 13 08:17 cipher_ofb.txt
```

CBC and ECB require padding while CFB and OFB do not require padding. CFB and OFB do not require padding as they are stream ciphers.

### 2.

```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f1.txt -out f1_cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f2.txt -out f2_cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in f3.txt -out f3_cbc.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
```

```

-rw-rw-r-- 1 seed seed      16 Nov 13 08:28 f1_cbc.txt
-rw-rw-r-- 1 seed seed       5 Nov 13 02:45 f1.txt
-rw-rw-r-- 1 seed seed      16 Nov 13 08:28 f2_cbc.txt
-rw-rw-r-- 1 seed seed      10 Nov 13 02:44 f2.txt
-rw-rw-r-- 1 seed seed      32 Nov 13 08:28 f3_cbc.txt
-rw-rw-r-- 1 seed seed      16 Nov 13 02:45 f3.txt

```

f1.txt and f2.txt created outputs of 16 bytes while f3.txt generated an output of 32 bytes. It shows the extent of padding in a CBC cipher. CBC uses a block size of 16 bytes - the generated ciphertext has to be in multiples of 16.

Therefore, f1.txt and f2.txt are extended to 16 bytes and f3.txt is extended to 32 bytes. f3.txt is padded even though it is already 16 bytes as it is clearer for the receiver to know whether the file is 16 bytes or 1-15 bytes with padding.

```

[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -nopad -
in f1_cbc.txt -out f1_d.txt -K 00112233445566778889aabbccddeeff -iv
0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -nopad -
in f2_cbc.txt -out f2_d.txt -K 00112233445566778889aabbccddeeff -iv
0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -d -nopad -
in f3_cbc.txt -out f3_d.txt -K 00112233445566778889aabbccddeeff -iv
0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$

```

```

[11/13/22]seed@VM:~/.../Files$ hexdump -C f1_d.txt
00000000 31 32 33 34 35 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b 0b |12345.
.....|
00000010
[11/13/22]seed@VM:~/.../Files$ hexdump -C f2_d.txt
00000000 31 32 33 34 35 36 37 38 39 30 06 06 06 06 06 06 |123456
7890.....|
00000010
[11/13/22]seed@VM:~/.../Files$ hexdump -C f3_d.txt
00000000 31 32 33 34 35 36 37 38 39 30 31 32 33 34 35 36 |123456
7890123456|
00000010 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 |.....
.....|
00000020
[11/13/22]seed@VM:~/.../Files$

```

f1.txt: 0x0b is used to pad the file

f2.txt: 0x06 is used to pad the file

f3.txt: 0x10 is used to pad the file

## Task 5:

```
[11/13/22]seed@VM:~/.../Files$ python3 -c "print('1234567890'*100)"
> textfile1000.txt
-rw-rw-r-- 1 seed seed 1001 Nov 13 08:58 textfile1000.txt
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-ecb -e -in text
file1000.txt -out textfile1000_ecb.txt -K 00112233445566778889aabbcc
ddeeff -iv 0102030405060708
warning: iv not used by this cipher
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in text
file1000.txt -out textfile1000_cbc.txt -K 00112233445566778889aabbcc
ddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cfb -e -in text
file1000.txt -out textfile1000_cfb.txt -K 00112233445566778889aabbcc
ddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-ofb -e -in text
file1000.txt -out textfile1000_ofb.txt -K 00112233445566778889aabbcc
ddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
```

Creating a file and encrypting it using ECB, CBC, CFB and OFB.

CBC:

```
textfile1000_cbc.txt
This file has been changed on disk. You may choose to
ignore the changes but reloading is the only safe option.
Ignore Reload
00000000 EC FB 05 81 98 77 71 C9 46 02 98 BA 32 85 A2 C2 B4 B0 CA .....wq.F...2.....
00000013 87 E6 EB 31 51 2E EE 16 52 06 8D AD 38 A2 82 36 9C EB 4E ...1Q...R...8...6..N
00000026 D6 B5 1C 0B A8 30 7E FB 1B 89 8B 0F F8 19 7A 33 8C 98 F6 .....0.....z3...
00000039 71 7E 14 7E 02 AF 00 26 67 2D EB 52 74 86 44 D5 F7 4D 08 q~...&g-.Rt.D.M.
0000004c 4F C2 57 43 7A 72 9A 00 28 1C 70 A9 C9 53 DB 50 B9 96 33 O.WCzr..(.p.s.P..3
0000005f 75 6E EA 80 35 45 CA 34 56 85 26 1B 98 8C 74 09 AC C0 0C un...5E.4V.&...t...
00000072 CE 99 26 F0 1C 19 59 DF 15 E3 BC 9D F7 07 AA A1 03 61 1A ..&...Y.....a.
00000085 EF 83 8C 98 8F AB B8 B6 C8 86 4D CE A9 2D C0 53 DF 5D D1 .....M...-..S.].
00000098 0F 84 E8 5B D9 EF 59 B1 8E 75 D9 5A 1E 2D 56 35 2F A9 43 ...[...Y..u.Z...V5/.C
```

The 50th byte was corrupted by changing it from 32 to 33.



The screenshot shows a file editor window with the title bar 'textfile1000\_cbc\_d.txt' and the path '~/Desktop/Project 3/Files'. The window contains a text area with the following content: '1789012345678üJ¿°£Pù`E' followed by a byte sequence '0003' in a box, then '7' followed by another byte sequence '0000' in a box, and finally '0056789112345678'. The text is displayed in a monospaced font, and the byte sequences are highlighted with a light blue background.

In CBC, 2 blocks in total were corrupted.

CFB:

```
textfile1000_cbc.txt  textfile1000_cfb.txt
This file has been changed on disk. You may choose to
ignore the changes but reloading is the only safe option.
Ignore  Reload
00000000 B6 B4 BC 11 F4 09 8D E9 96 AE 8E 2F 46 24 69 5B BC 95 AF ...../F$[...
00000013 92 FF B1 B4 08 4C EE 06 E5 6F 3B 9A 01 B8 03 6A 8A 26 98 .....L...o;...j.&
00000026 90 AC 05 FA 9E 1D 09 80 48 B6 B3 86 B2 A7 E7 23 43 2C F2 .....H.....#C,.
00000039 07 70 B8 FB D0 87 90 34 CB C6 42 86 AF 10 93 90 45 DC 7C .p.....4..B.....E_|
0000004c cC AA 98 09 13 E0 33 93 BC B4 D6 4D B9 9E 92 40 6A AC 4D .....3.....M.....@j.M
```

The 50th byte was corrupted by changing it from 22 to 23.

[illegible]

In CFB, 2 blocks in total were corrupted.






ECB:

textfile1000 cbc.txt x textfile1000 cfb.txt x textfile1000 ecb.txt x

This file has been changed on disk. You may choose to ignore the changes but reloading is the only safe option.

```
00000000 44 80 59 04 1E 23 82 96 58 89 88 11 F0 51 C1 D7 F9 6E E9 D.Y..#..X...Q...n.
00000013 41 25 5D D9 A7 8E CE C2 5F DB B5 EF D3 E5 C6 8D 1B 9E F0 A%]....._.....
00000026 F4 D8 D1 6A 0E 8D A2 AD E8 55 C8 87 8F AF C3 8C E1 D1 47 ....j.....U...#...G
00000039 1E 8D A0 B8 A2 7B 5A 69 A3 80 17 C8 1F FE C0 66 13 AF E3 .....{zi.....i...
0000004c 34 EC B3 5E 44 80 59 04 1E 23 82 96 58 89 88 11 F0 51 C1 4...^D.Y..#..X...O.
```


The 50th byte was corrupted by changing it from 8B to 8C.

Open  textfile1000\_ecb\_d.txt Save    
~/Desktop/Project 3/Files  
1123456789012345678901234567890123456789012345678HyS-  
Jfs#ZÃsú  
567890123456789012345678901234567890123456789012345678901234567890

In ECB, 1 block was corrupted.

OFB:

The 50th byte was corrupted by changing it from A3 to A4.



The screenshot shows a code editor window with a dark theme. The title bar at the top indicates the date and time as 'Nov 13 09:24'. The editor's title is 'textfile1000\_ofb\_d.txt' and the file path is '~/Desktop/Project 3/Files'. The main text area contains a repeating sequence of numbers: '67890123456789012345678901232567890123456789012345678901234567'. The sequence is highlighted in yellow, and the cursor is positioned at the end of the first highlighted segment.

In OFB, only the corresponding byte is affected. Therefore, OFB is able to recover the most information as it affects the least number of bytes.

## Task 6:

## 6.1:

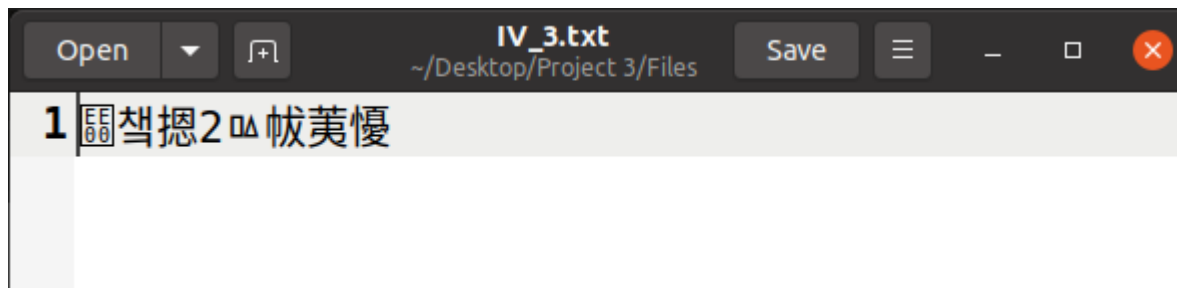
```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plain.txt -out IV_1.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plain.txt -out IV_2.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708
hex string is too short, padding with zero bytes to length
```



IV\_1.txt and IV\_2.txt are encrypted with an iv of 0102030405060708. When the same plaintexts are used with the same IVs, the same ciphertexts are produced, thus after decryption, they produce the same plaintext again.

```
[11/13/22]seed@VM:~/.../Files$ openssl enc -aes-128-cbc -e -in plain.txt -out IV_3.txt -K 00112233445566778889aabbccddeeff -iv 112233445566
hex string is too short, padding with zero bytes to length
```

IV 3.txt is encrypted with a different iv of 112233445566.



This generates a completely different ciphertext.

Using unique IVs is important as it helps defend against known-plaintext attacks. By using a different IV every time, it ensures that an adversary generates a different ciphertext each time, making it harder for the adversary to get clues to decrypt and interpret the original values.

## 6.2

### Using known-plaintext-attack.py

```
1#!/usr/bin/python3
2
3# XOR two bytearrays
4def xor(first, second):
5    return bytearray(x^y for x,y in zip(first, second))
6
7MSG = "This is a known message!"
8HEX_1 = "a469b1c502c1cab966965e50425438e1bb1b5f9037a4c159"
9HEX_2 = "bf73bcd3509299d566c35b5d450337e1bb175f903fafc159"
10# Convert ascii/hex string to bytearray
11D1 = bytes(MSG, "utf-8")
12D2 = bytearray.fromhex(HEX_1)
13D3 = bytearray.fromhex(HEX_2)
14r1 = xor(D1, D2)
15r2 = xor(D2, D3)
16r3 = xor(D2, D2)
17
18r4 = xor(r2,D1)
19print(r4)
20print(r1.hex())
21print(r2.hex())
22print(r3.hex())
23
```

