Lab2-part3

7/12/2023

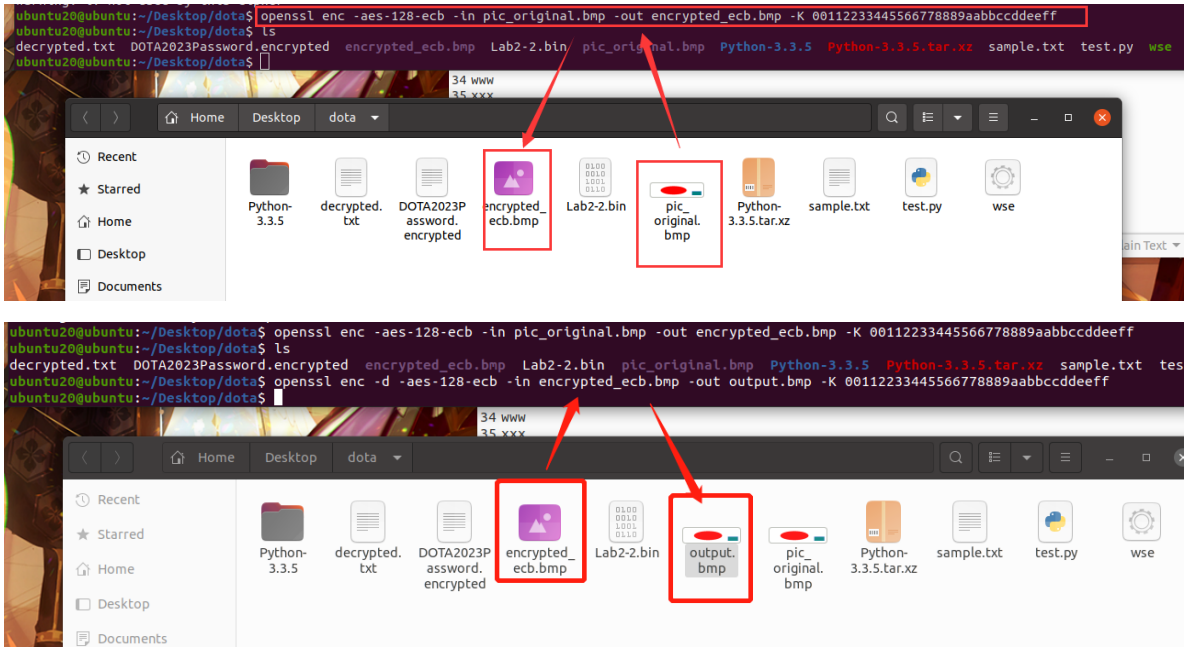
I write this write up using by Markdown. Because some pictures cannot be displayed, you can visit my github homepage to see the complete writeup. (which has picture)

3.1

picture：[pic_original.bmp (460×134) (nus.edu.sg)]

First, ECBencryption. And there's no problem to perform decryption at this time, which proves that the encryption and decryption are normal.

```
openssl enc -aes-128-ecb -in pic_original.bmp -out encrypted_ecb.bmp -K
00112233445566778889aabbccddeeff
openssl enc -d -aes-128-ecb -in encrypted_ecb.bmp -out output.bmp -K
00112233445566778889aabbccddeeff
```

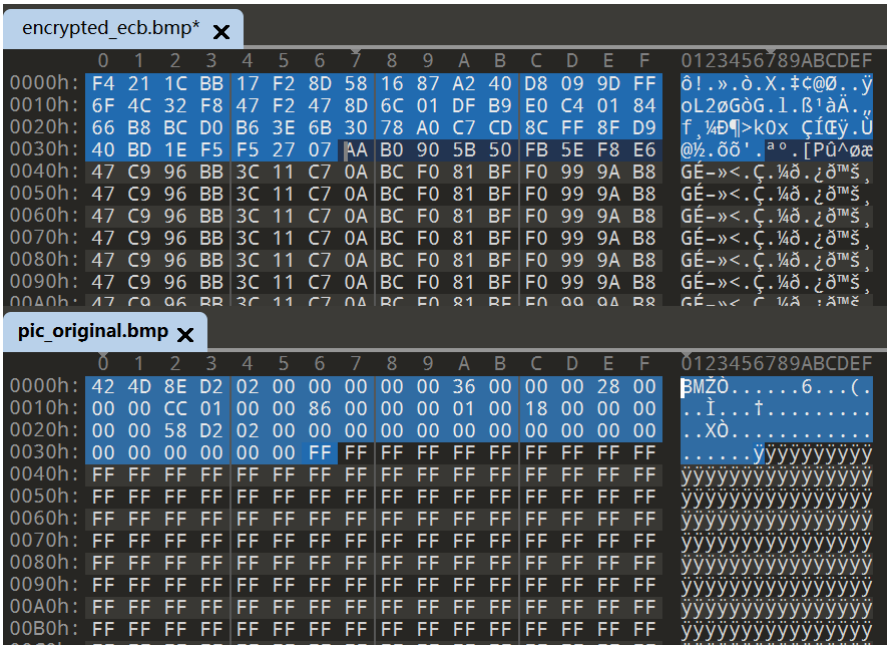


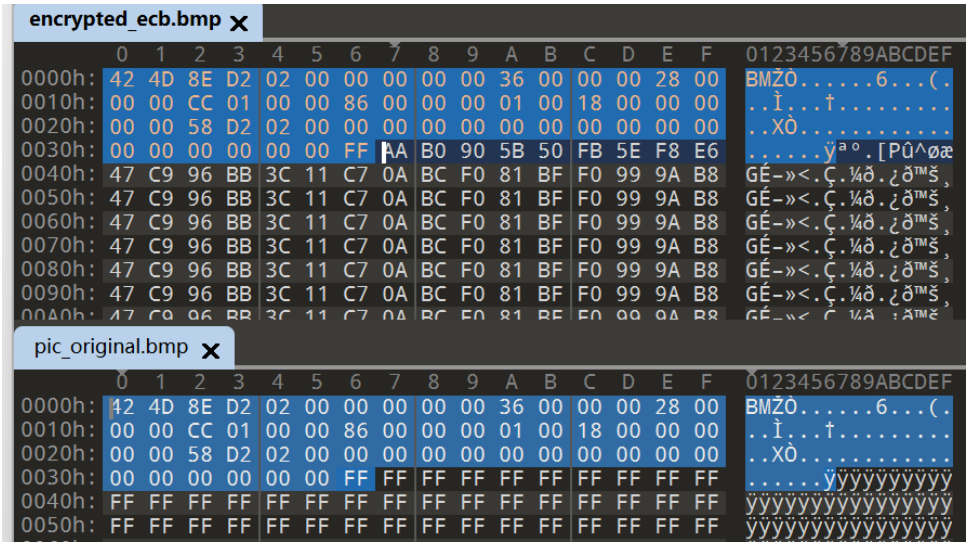
However, the encrypted picture cannot be recognized and cannot be opened normally. And I choose one binary tools. here I use 010editor.

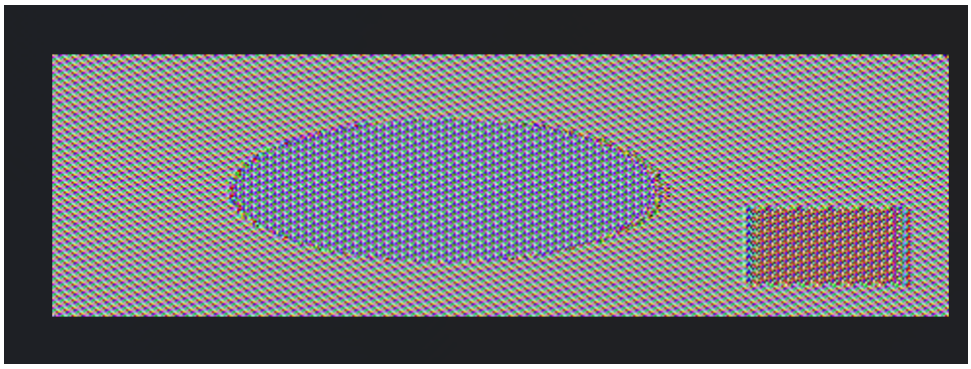
We know that for the .bmp file, the first 54 bytes contain the header information about the picture, we have to set it correctly, so the encrypted file can be treated as a legitimate .bmp file.

The first 54bytes of the encrypted image are `ô!□»□ò`, while the original image is `BMŽÒ□` (the binary in the binary tool cannot be copied completely, so the copy result here is a bit problematic). Then, use 010editor to modify the header part, copy the header content of the original picture, and find that the encrypted picture can be recognized and opened.

The encrypted picture obtained in ECB mode can still see the outline and color of the original picture, but it is disturbed to a certain extent.

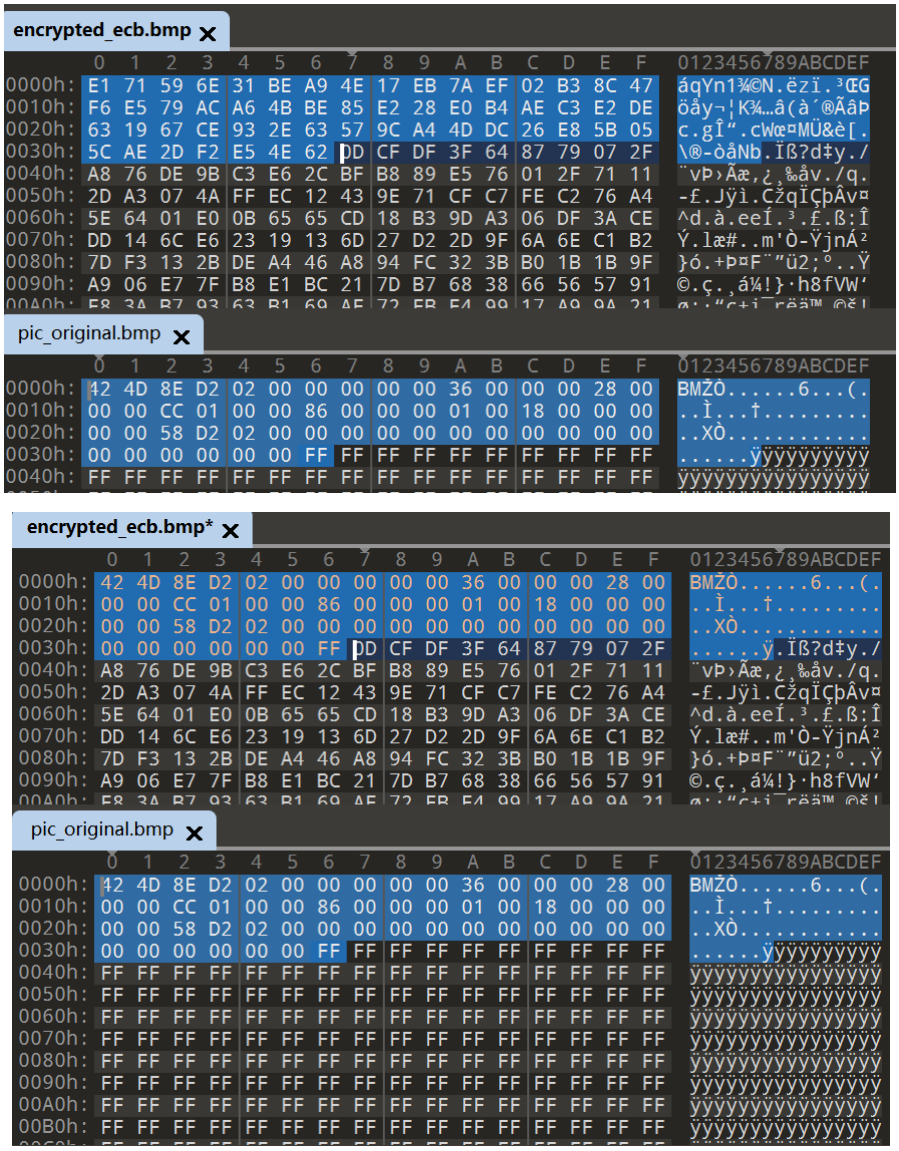






Then try CBC mode. The operation steps are similar. But after getting the encrypted picture, this time it is completely impossible to see what the original picture looks like, only a piece of mottled color.

```
openssl enc -d -aes-128-ecb -in encrypted_ecb.bmp -out output.bmp -K
00112233445566778889aabbccddeeff -iv 0102030405060708
```



The differences between these two mode are because:

ECB mode divides the entire image into fixed-size blocks and encrypts each block independently with the same key. This means that the same input block will always produce the same output block. For example, if there are repeating patterns in the image, after encryption, these patterns may repeat in the same way, so that the encrypted image retains some recognizable characteristics.

Howerver, in the CBC mode, during the encryption process, the output of the previous encrypted block is XORed with the current plaintext block, and then encrypted. The encryption of each block depends on the result of the encryption of the previous block. This makes it more difficult for an image encrypted in CBC mode to preserve the characteristics of the original image,

because even if two blocks of input are very similar, they may produce completely different outputs.

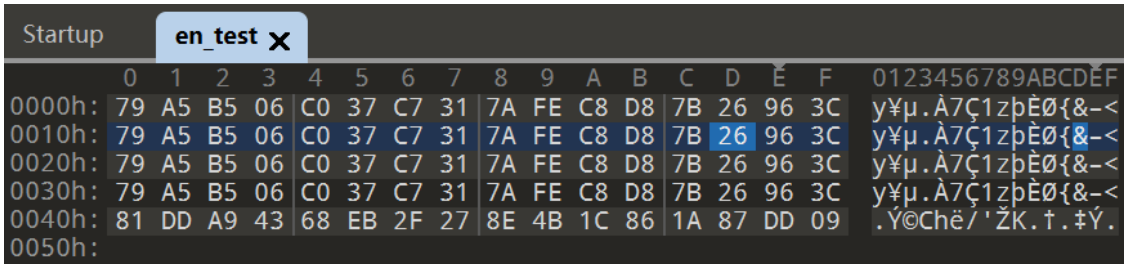
3.2

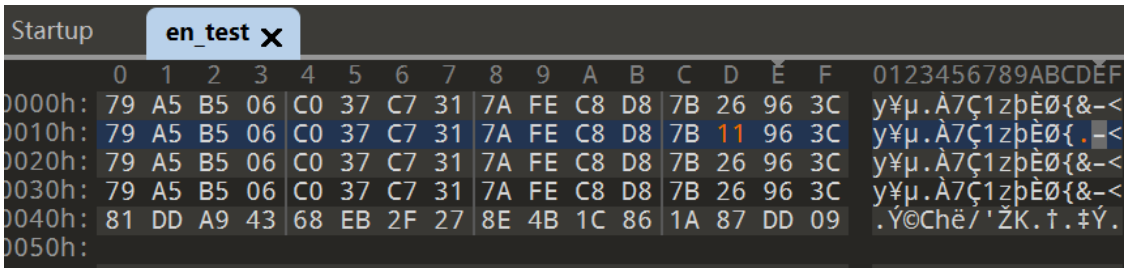
——operating phase——

- ECB mode: first encrypt and decrypt

  ```
  openssl enc -aes-128-ecb -in test -out en_test -K
  00112233445566778889aabbccddeeff
  openssl enc -d -aes-128-ecb -in en_test -out de_test -K
  00112233445566778889aabbccddeeff
  ```

  Now a 80bytes file is generated, and then use 010editor to modify 30 bits. Originally, 30 bits should be 26. I changed it to 11.

  The original content is `1111111111111111111111111111111111111111111111111111111111111111`, after decrypting the modified file, it becomes `1111111111111116%ï¿½Yï¿½Wï¿½ï¿½tï¿½ï¿½ï¿½111111111 1111111111111111111111`. There are 16 bytes corrupted (just one block).

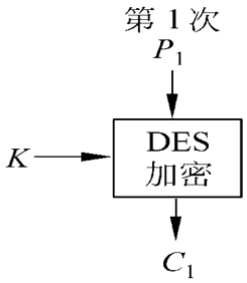
- CBC mode: Experiments in other modes are carried out in the same way. In CBCmode, the plaintext is the same, and the ciphertext becomes `HTï¿½kï¿½rï¿½|ï¿½ï¿½1111111111111ï¿½11111111111111111111`, indicating that all modified blocks which are after this block are affected, not only the current block.
- CFB mode: In the CFB mode test, modifying one byte of ciphertext becomes `11111111111111111111111111v11eXï¿½ï¿½Bwï¿½ï¿½ONï¿½ï¿½\811111111111111111`, which only affects the current block and does not affect other blocks
- ODB mode: In odb mode, the other block isn't affected, and other bit in the same block isn't affected either. Only the corresponding bit is affected. The decrypted plaintext is `1111111111111111111111111111i11111111111111111111111111111111` , Only one bit of the ciphertext changed from "1" to "i"

——explain——

- ECB: In ECB mode, each block of plaintext is encrypted independently. Therefore, if a single bit in the ciphertext gets corrupted, only the corresponding block of plaintext will be affected. The rest of the blocks will remain unaffected. For example, when processing the i-th block, use Pi, K to get Ci, and will not use the feedback result of the previous or

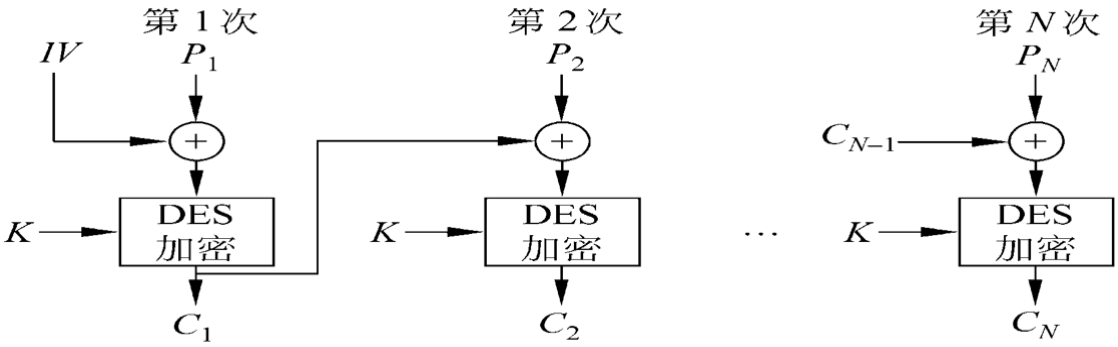
subsequent block.

`Pi、K --DES--> Ci`

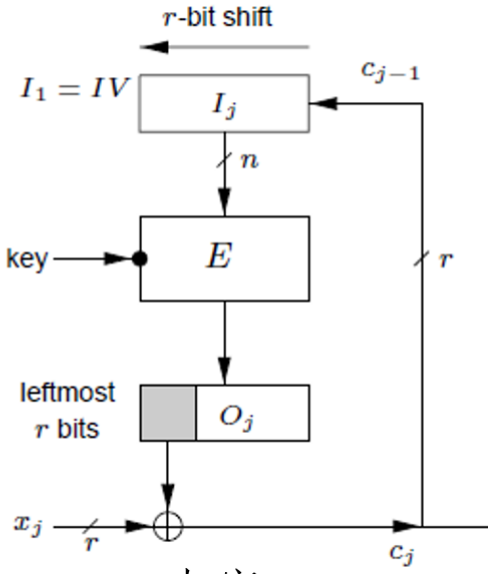


- CBC：In this mode, the plaintext block is XORed with the last ciphertext block first, and then encrypted. If a single bit in the ciphertext gets corrupted, it will affect the corresponding block of plaintext as well as the subsequent blocks due to the chaining effect.

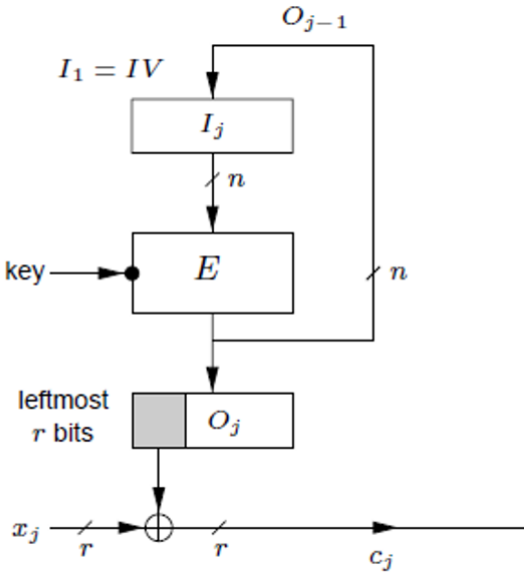
`Pi、IV、K、Ci-1 --DES--> Ci`



- CFB：The encryption and XOR operations are performed for each block separately. If a single bit in the ciphertext gets corrupted, it will only affect the corresponding block of plaintext and will not propagate further. One or more bit errors appearing in any r-bit ciphertext group will affect the decryption of this group and the subsequent [n/r] ciphertext groups



- OFB: OFB mode is structurally similar to CFB mode, but the feedback content is the output of DES instead of ciphertext. So if a single bit in the ciphertext gets corrupted, it will only affect the corresponding bit in the keystream generation process.


O_{j-1}
$I_1 = IV$
I_j
n
key
E
n
leftmost
r bits
O_j
x_j
r
r
c_j