Assignment 2 Report
Block Cipher Modes

Wee JunJie
U1540273K

1 ECB Mode

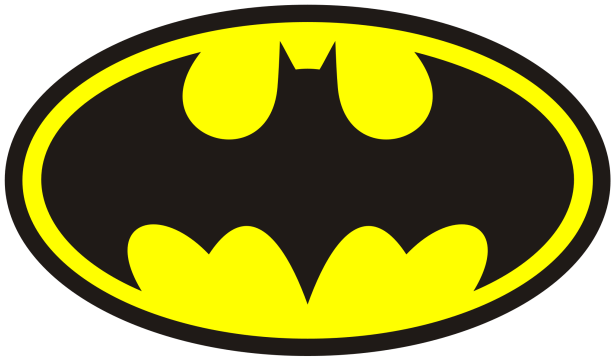
1.1 Implementation

The solution code (version 1) for Assignment 1 was being used and modified for Task 2 and Task 3 as well. In the `main.cpp`, ECB Mode can be selected by the user as Mode 1. In this report, all the encryptions and decryptions uses the key: `0f0e0d0c0b0a0908 0706050403020100`.

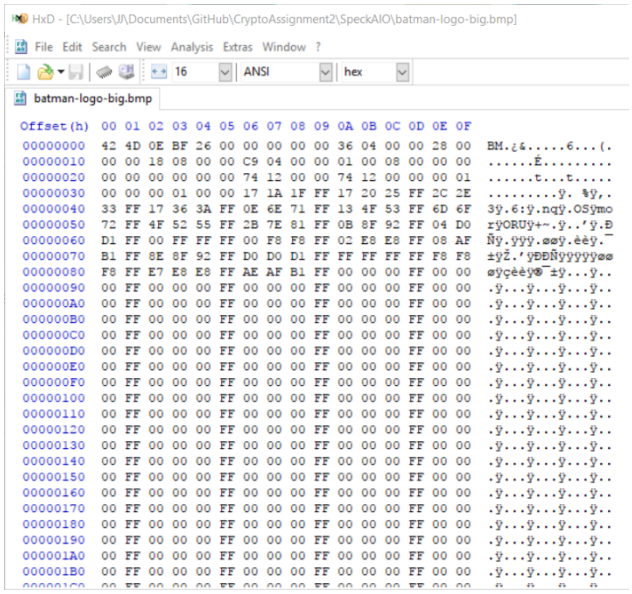
1.2 Encryption of Bitmap File

1.2.1 Bitmap File Header

The Bitmap File Format contains the BMP File Header and the DIB Info Header. The BMP File Header is of 14 bytes and DIB Info Header is of 40 bytes. The first 4 bytes of the BMP File Header represents the file signature of Bitmap. i.e. `42 4D` The next 4 bytes shows the size of the respective Bitmap file. In this case, it shows `0E BF 26 00` in little endian hence representing 2,539,278 bytes in decimal notation. Following this, the subsequent 4 bytes are reserved for the application that created the Bitmap. The last 4 bytes of BMP File Header shows the starting address of the Bitmap Image Data. In this case, the starting address is at `36` as shown at byte address `0A` in Figure 1. Hence, it can be seen starting at byte address `36` with a `17` as well. The DIB Info Header in general consists of the image information and the pixel format of the BMP. In total, 54 bytes needs to be copied in order to open in BMP File Format.



(a) Original Image

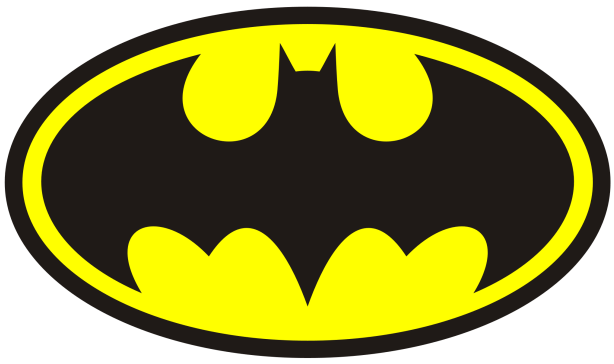


(b) Original Hex Data

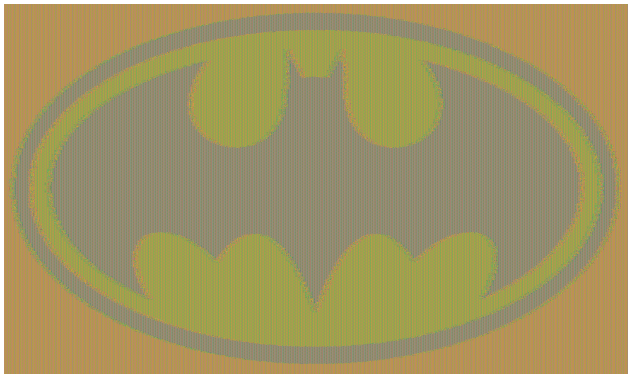
Figure 1: Encryption of Bitmap in Speck 128-128 ECB Mode

1.2.2 Weakness of ECB Mode in Bitmap

Upon encrypting the image and replacing the first encrypted 54 bytes with the BMP File Header and DIB Info Header, the weakness of ECB Mode can be seen in Figure 2(b) as there is original image data being visible after the encryption.



(a) Before Encryption



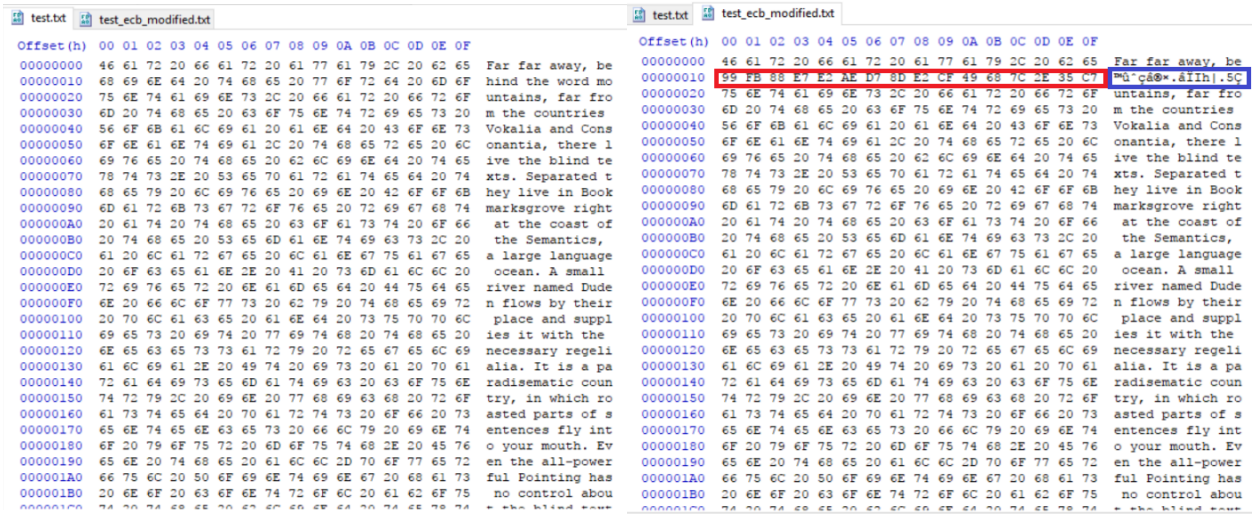
(b) After Encryption

Figure 2: Encryption of Bitmap in Speck 128-128 ECB Mode

1.3 Encryption of Text File

Using the text shown in Appendix, the text file test.txt was encrypted in ECB mode and the 18th byte was then replaced with `0A`. The modified encrypted text file was then decrypted to test_ecb_modified.txt. Under the ECB Mode, since every ciphertext is decrypted to its plaintext without any block cipher chaining, hence only the respective plaintext would be

affected by the 18th byte. In this case, the 2nd message block cannot be decrypted back correctly due to the 18th byte being passed through the Speck 128-128 which affected only the rest of the bytes in that message block. Thus, it can be seen in Figure 3(b) that the hex for the 2nd row is still scrambled.



(a) Original Text

(b) Decrypted Text after modifying 18th byte

Figure 3: Decryption of Modified Text File using Speck 128-128 ECB Mode

2 CBC Mode

2.1 Implementation

2.1.1 Initialisation Vector

The CBC Mode is implemented by setting up the Initialisation Vector (IV) using a cryptographically secure random number generator. i.e. /dev/random. Note that the Cygwin Environment was used. Hence, the /dev/random in Cygwin is similar to the CryptGenRandom under the Windows API. Note that in this assignment, the Initialisation Vector was then written into the encrypted file as the first message block. The Code for generating one `unsigned long long` is shown below.

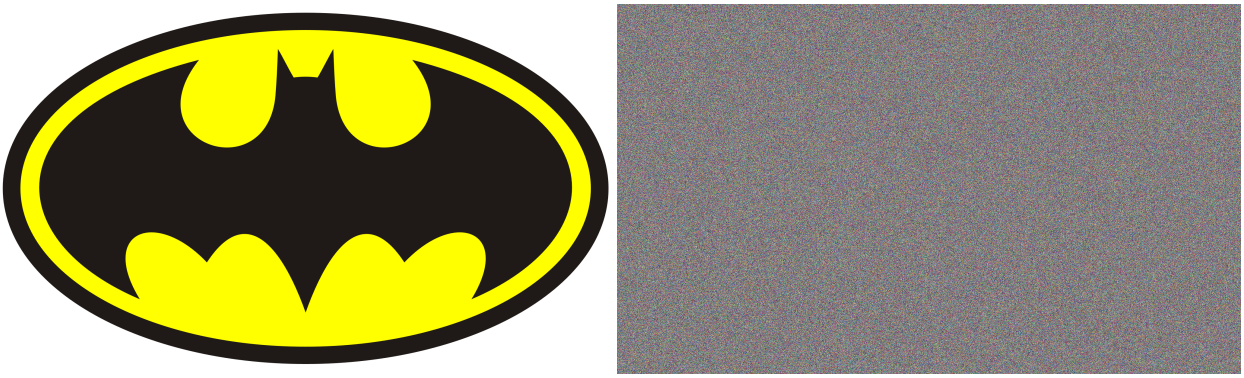
```
unsigned long long InitialVec()
{
    int myFile = open("/dev/random", O_RDONLY);
    unsigned long long rand;
    unsigned long long randomNum = read(myFile, &rand, sizeof(rand)) ;
    close(myFile);
    return rand;
}
```

2.1.2 CBC Encryption and Decryption

The encryption in CBC mode goes as follows: The initialisation vector first XORs with the plaintext before passing through the Speck 128-128. The encrypted output which is the ciphertext will also XOR with the next plaintext. Since the encryption takes place after the XOR, padding is required for the plaintext with regards to the last message block. The same padding method from the ECB mode is also applied here. The decryption is also similar. The ciphertext, being also the IV for the next ciphertext, passes through the Speck 128-128 Decryption and XORs with the IV to retrieve back its own plaintext.

2.2 Encryption of Bitmap File

Similarly, we placed back the Bitmap File Format into the encrypted hex data. Note that the first message block contains the IV. Hence, we need to remove the IV first before replacing back the 54 bytes. Compared to the ECB Mode, block cipher chaining actually is stronger as now we cannot see any part of the original image visible in the encrypted Bitmap.

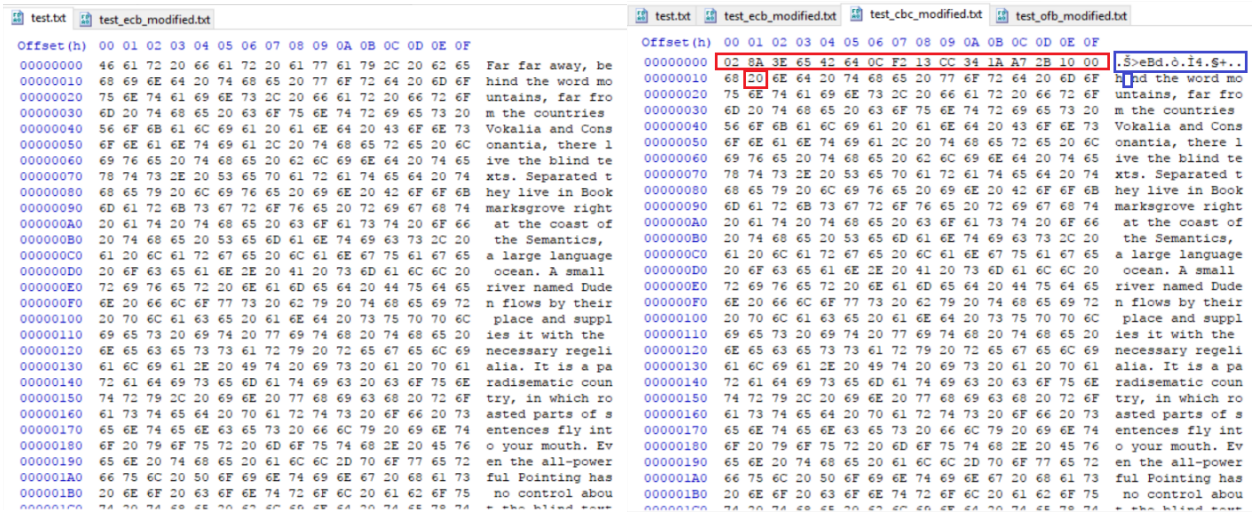


(a) Before Encryption (b) After Encryption

Figure 4: Encryption of Bitmap in Speck 128-128 CBC Mode

2.3 Encryption of Text File

Using the same text as the one in Section 1.3, the text file test.txt was encrypted in CBC mode and the 18th byte was then replaced with `0A`. The modified encrypted text file was then decrypted to test_cbc_modified.txt. Under the CBC Mode, since there is block cipher chaining, hence the 18th byte which belongs to the 1st message block would affect the 1st and 2nd message block of the decrypted plaintext. In this case, the 1st message block cannot be decrypted back correctly due to the 18th byte being passed through the Speck 128-128 Decryption which affected the rest of the bytes in that message block. Other than the 1st message block, the 18th byte of the 2nd message block of the plaintext is affected and not the rest of it as this is only due to the XOR operation. Thus, it can be seen in Figure 5(b) that the hex for the 1st row and the 18th byte is different.



(a) Original Text (b) Decrypted Text after modifying 18th byte

Figure 5: Decryption of Modified Text File using Speck 128-128 CBC Mode

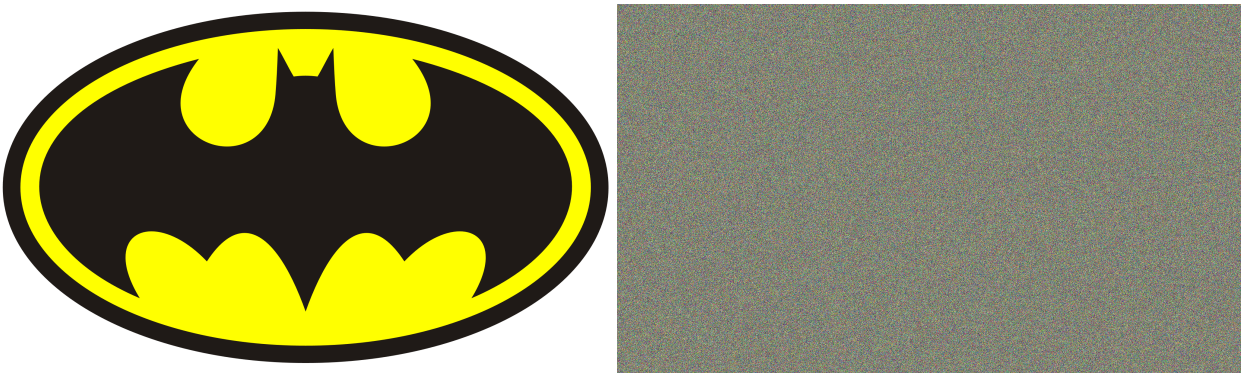
3 OFB Mode

3.1 Implementation

The OFB also uses the same method of generating IV as mentioned in Section 2.1.1. In OFB Mode, since the Speck 128-128 Encryption is used for the IV and not the plaintext, hence there is no padding required as we could just XOR the plaintext with the encrypted IV to obtain the ciphertext.

3.2 Encryption of Bitmap File

Similar to CBC Mode, the original image data is not visible in the encrypted output. However, we note the slight differences in RGB pixels as compared to the one in CBC mode.

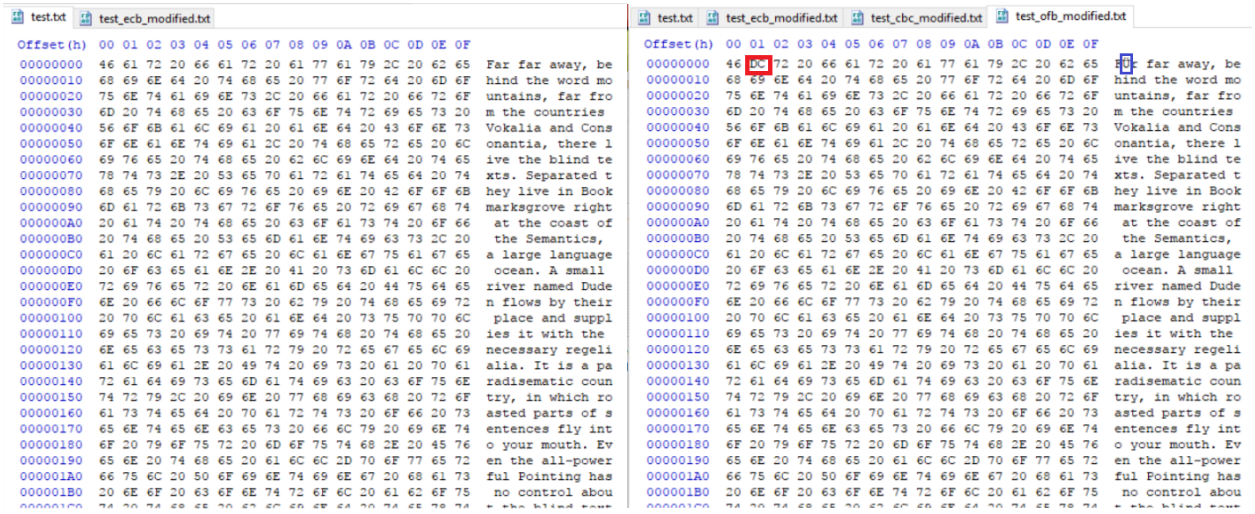


(a) Before Encryption (b) After Encryption

Figure 6: Encryption of Bitmap in Speck 128-128 OFB Mode

3.3 Encryption of Text File

Using the same text as the one in Section 1.3, the text file test.txt was encrypted in OFB mode and the 18th byte was then replaced with `0A`. The modified encrypted text file was then decrypted to test_ofb_modified.txt. Under the OFB Mode, only the 18th byte will be decrypted wrongly since the ciphertext does not pass through the Block Cipher and only XORs with the encrypted IV. Hence, we can see in Figure 7 that the hex data only differ by the 18th byte.



(a) Original Text (b) Decrypted Text after modifying 18th byte

Figure 7: Decryption of Modified Text File using Speck 128-128 OFB Mode