

# 資訊安全期末書面報告

## CBC 位元翻轉攻擊

B062040027 鄭乃心

B062040020 張詠晴

B064011007 徐筱媛

## 目錄

1、前言-----	3
2、CBC 模式背景-----	3
3、CBC 加密、解密-----	4
4、CBC 位元翻轉攻擊-----	7
5、攻擊實作一-----	8
6、攻擊實作二-----	11
7、攻擊防範-----	16
8、心得-----	19

## 1、前言

在課堂的內容中，CBC 模式有被提到過兩次，第一次是第七章介紹的 CBC 簡介，與第二次是在介紹第十二章 Data Authentication Algorithm 的時候，該的演算法邏輯和 CBC 模式相同，再加上 DAA 很常用於 MAC 位址，電腦網路課堂上也有提過，這就讓我們更加想理解 CBC 模式的實際應用以及更多的背後原理。

在本文中，我們會先介紹 CBC 模式的背景以及加解密的原理，再介紹 CBC 位元翻轉攻擊，並且說明我們完成的兩個實作，藉由簡單的 python 程式碼更容易了解整個 CBC 位元翻轉攻擊的原理，再將此攻擊實際應用在網頁攻擊上。最後介紹一篇論文，該論文加入了 Hash 的觀念，改善了 CBC 模式。

## 2、CBC 模式背景

目前我們先不討論 CBC 加密模式本身的安全性，而是先來討論為什麼不直接使用分組加密如 AES 或 3DES 等加密模式來對資料進行加密。以 AES 為例，長序列加密時，首先分組成 128 位元，再分別用給定的 K 進行加密。對於一般資料而言，若兩個分組恰巧數值相同，那麼加密結果也必然相同。相反地，若數值不同，加密結果就會有很大的可能是不相同的。具體可以看下方圖片一、二的前後對比，圖一為加密前的明文，圖二為透過 AES 的 ECB 模式下加密後的模樣。



圖一



圖二



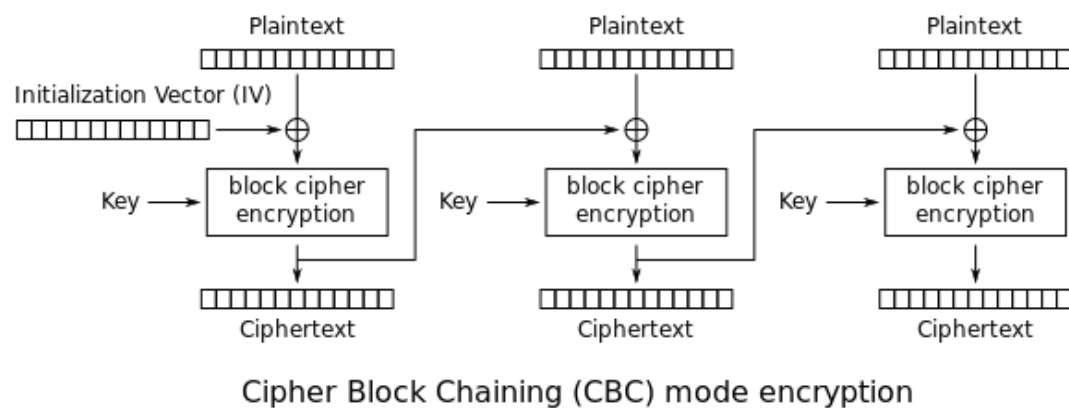
圖三

很顯然，圖片直接反映出這種加密模式透露了某種程度上的訊息。圖片加密之後仍然能看出企鵝的輪廓，可以分辨企鵝的腳、肚子、身體等區域，非常直觀的可以觀察出該加密模式的不安全。而之所以會某種程度上透露出了訊息，是因為沒有利用計數器以及隨機值，這種確定性加密在面對選擇明文攻擊時是不安全的。也因此，CBC 加密模式才會誕生。圖三是透過 CBC 模式加密後的結果，肉眼是看不出資訊的，但不能以肉眼看到的隨機性來判斷他的安全性，有許多不安全的加密方法也有可能產生其隨機輸出。我們在下一個章節更深入的介紹 CBC 模式。

### 3、 CBC 加密、解密

1976 年，IBM 發明了 CBC(Cipher-block chaining)模式。在 CBC 模式中，每個明文分組先與前一個密文分組進行 XOR 運算，再進行加密，通常使用 DES 或 AES 這兩種分組密碼演算法。在這種方法中，每個密文分組都依賴於他前面的所有明文。且為了保整每條訊息的唯一性，在第一個分組中會使用

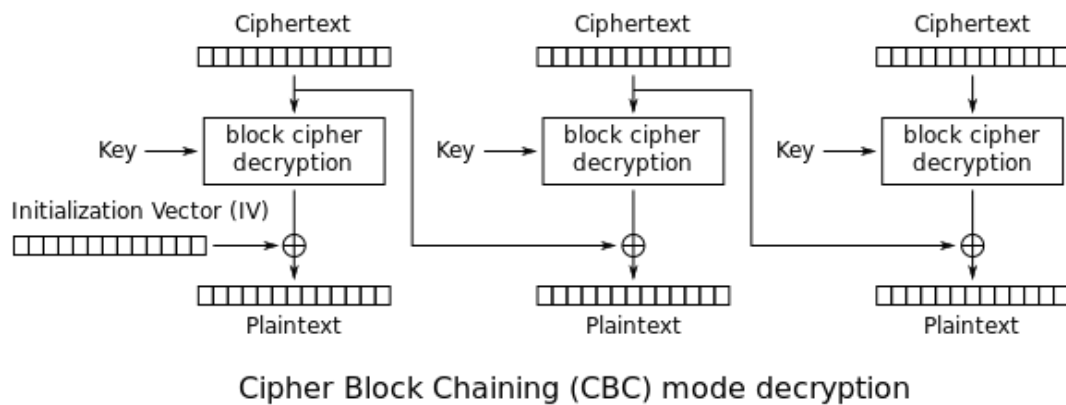
到初始化向量 IV，用來隨機化加密位元分組，保證即使加密多次相同的明文，也能得到不同的密文。我們透過下面這張圖來更清楚地了解到 CBC 模式的加密是如何進行。



首先，先對明文進行分組，使每組長度相同。通常為 8 或 16 位元組，長度取決於加密演算法。若有長度不足的分組，則需要進行填補。接著隨機生成一個初始化向量 (IV)，將第一個明文分組與 IV 做 XOR 運算。再將剛剛的結果進行加密，得到第一個明文分組的密文。

從第二個明文分組開始，先將明文分組與上一組的密文做 XOR 運算，再將結果進行加密，得到該分組的密文。最後，將 IV 和這 n 個密文分組連在一起便得到了明文用 CBC 模式加密後的密文。

說明完了 CBC 模式下的加密，我們接著介紹解密的情況。CBC 模式底下解密的流程如下圖所示：



首先先照一定長度將密文分組，其中密文的第一組是初始的 IV 值，第二組密文對應第一組明文.....以此類推。

分好組後，從第二組密文密文開始一次用算法進行解密運算得到 n 組中間值，這時候得到的值並不是明文，要想得到明文還要做一次 XOR 運算。第一個中間值與初始 IV 值作 XOR 運算得到明文，第二個中間值與前一組的密文 XOR 運算得到第二組明文，以此類推最後一組中間值與倒數第二組密文進行 XOR 運算便可得到最後一組明文，將所有明文連在一起便是最終的明文。

我們摘錄了 Dan Boneh 講義中的表格，說明了 CBC 模式下，加密多少資料以內可抵禦選擇密文攻擊：

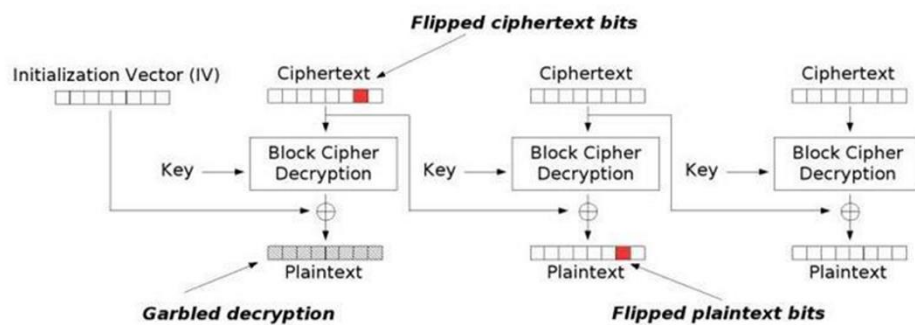
	CBC
uses	PRP
parallel processing	No
Security of rand. enc.	$q^2 L^2 \ll  X $
dummy padding block	Yes
1 byte msgs (nonce-based)	16x expansion

$|X|$  是輸入分組的大小，L 是單次輸入最長長度，q 是有幾則訊息。

我們用 3DES 和 AES 來試算一下：

3DES 的 $|X|$ 是 64 bit，若希望  $q^2/|X|=1/2^{32}$ ，也就是說攻擊者區分真亂數和 CBC/CTR 的機率是  $1/2^{32}$ 。則  $q^2 = 2^{32} \rightarrow q = 2^{16}$ ，加密 65536 個訊息後要換金鑰。另外，AES 的 $|X|$ 是 128 bit， $q^2/|X|=1/2^{32}$ ， $q^2 = 2^{96} \rightarrow q = 2^{48}$ ，這個大小夠用非常非常久。如此我們可以看出 CBC 的安全性，但在此安全性下還是有漏洞，下一節我們就來介紹 CBC 下可實現的攻擊。

#### 4、CBC 位元翻轉攻擊



CBC 模式的加密過程為：

$$C_i = E_k(P_i \oplus C_{i-1})$$
$$C_0 = IV$$

其解密過程為：

$$P_i = D_k(C_i) \oplus C_{i-1}$$
$$C_0 = IV$$

也就是說，CBC 進行翻轉攻擊的原理如上圖所示，Ciphertext-N-1 是用來產生下一塊明文，這就是位元翻轉攻擊發揮作用的地方。如果我們改變

Ciphertext-N-1 的一個位元組，然後與下一個解密後的分組做 XOR 運算，我們就可以得到不同的明文。

CBC 在解密時，將密文進行解密運算之後，要與 IV 作 XOR 運算，我們無法得知解密和 XOR 運算過後的結果 Cipher[i]，但可以藉由修改密文 Cipher[i-1]來控制明文 Plain[i]，達到想要的行為。

但我們可以從理論中得知，該攻擊必須能夠獲得每一次的密文與解密完的結果。並且只能從最後一組密文開始向前修改，且每次改完一組，都需要重新計算解密後的資料，再藉由解密後的資料來修改前一組密文。故此攻擊在實現上有一定的先決條件，雖然實作容易並非如此容易完成此攻擊。

此攻擊透過破壞秘聞的位元組來改變明文位元組，藉此繞過過濾器，或者更改使用者許可權提升至管理員，又或者可以改變應用程式預期之明文以完成攻擊者想達成的事情。

## 5、 攻擊實作一

首先，我們先用 Python 實作前面章節說明到的觀念。

```
//定義 AES 加密方法
```



```
def encrypt(iv,plaintext):
    if len(plaintext)%16 != 0:
        print("plaintext length is invalid")
        return
    if len(iv) != 16:
        print("IV length is invalid")
        return
    key="1234567890123456"
    aes_encrypt = AES.new(key,AES.MODE_CBC,IV=iv)
    return b2a_hex(aes_encrypt.encrypt(plaintext))
```

//定義 AES 解密方法

```
def decrypt(iv,cipher):
    if len(iv) != 16:
        print("IV length is invalid")
        return
    key="1234567890123456"
    aes_decrypt = AES.new(key,AES.MODE_CBC,IV=iv)
    return b2a_hex(aes_decrypt.decrypt(a2b_hex(cipher)))
```

最開始先定義加密與解密方法，這裡用到的是 AES 加解密。在加解密函式中，我們定義了金鑰的數值，是因為調用加解密函式的人是不會知道金鑰是少少的。接下來就是簡單地定義一個測試函式：

```
def test():
    iv=b'\x83Z\x8dn\xc3s\xcb\xfe\xe11cP\x00\xb1\xfcg'
    plaintext="0123456789ABCDEFhelllocbcflipping"
    print("plaintext: ",plaintext)
    cipher=encrypt(iv, plaintext)
    print("cipher: ",cipher)
    de_cipher = decrypt(iv, cipher)
    print("de_cipher: ",de_cipher)
    print("a2b_hex(de_cipher): ",a2b_hex(de_cipher))
```

在測試函式中，使用者可以自行定義 IV 值、明文，並且得到密文。再將結果直接印在螢幕上。我們會得到：

```
plaintext: 0123456789ABCDEFhelllocbcflipping
cipher: b'0c3ae64147d5f80fdebfbec04e493548986f6e090f1ab73902373b20110f250f'
de_cipher: b'3031323334353637383941424344454668656c6c6f636263666c697070696e67'
a2b_hex(de_cipher): b'0123456789ABCDEFhelllocbcflipping'
```

接下來我們正式來實現 CBC 位元翻轉攻擊的概念。我們對一開始的測試函式做了一些修改。在第一個修改中，我們利用 CBC 位元翻轉攻擊，藉由修改密

文位元組來使明文的最後一個字符 g 變為大寫。也就是將密文的最後一個位元

組與 'g' 還有 'G' 做 XOR 運算。對更改過後的密文去做解密。

```
def test():
    iv=b'\x83Z\x8dn\xc3s\xcb\xfe\xe11cP\x00\xb1\xfcg'
    plaintext="0123456789ABCDEFhellobcflipping"
    print("plaintext: ",plaintext)
    cipher=encrypt(iv, plaintext)
    print("cipher: ",cipher)
    de_cipher = decrypt(iv, cipher)
    print("de_cipher: ",de_cipher)
    print("a2b_hex(de_cipher): ",a2b_hex(de_cipher))
    print()
    #修改1
    bin_cipher = bytearray(a2b_hex(cipher))
    bin_cipher[15] = bin_cipher[15] ^ ord('g') ^ ord('G')
    de_cipher = decrypt(iv,b2a_hex(bin_cipher))
    print('----把最後的g改成G----')
    print("de_cipher2: ",de_cipher)
    print("a2b_hex(de_cipher): ",a2b_hex(de_cipher))
```

執行的結果，我們可以得到：

```
plaintext: 0123456789ABCDEFhellobcflipping
cipher: b'0c3ae64147d5f80fdebfbec04e493548986f6e090f1ab73902373b20110f250f'
de_cipher: b'3031323334353637383941424344454668656c6c6f636263666c697070696e67'
a2b_hex(de_cipher): b'0123456789ABCDEFhellobcflipping'
----把最後的g改成G----
de_cipher2: b'f8720f243415cf625674924312bc477868656c6c6f636263666c697070696e47'
a2b_hex(de_cipher): b'\xf8r\x0f$4\x15\xcfbVt\x92C\x12\xbcGxhellobcflippinG'
```

我們可以觀察出最後一個位元組 g 已經成功改寫為大寫 G，但由於更改了

密文的最後一個位元組，導致前面的資料也被更動。解密後的明文前半段變成

了亂碼，接著我們進一步把前半段的亂碼改成我們指定的字串。也就是我們必

須透過修改 IV 的值來控制第一組密文解密出的結果。我們透過 IV 與剛剛第一

次修改密文後得到的解密結果值，還有字元 'X' 做 XOR 運算，得出新的 IV

值。

```

def test():
    iv=b'\x83Z\x8dn\xc3s\xcb\xfe\xe1lcP\x00\xb1\xfcg'
    plaintext="0123456789ABCDEFhellocbclipping"
    print('----明文----')
    print("plaintext: ",plaintext)
    cipher=encrypt(iv, plaintext)
    print('----密文----')
    print("cipher: ",cipher)
    de_cipher = decrypt(iv, cipher)
    print('----解密----')
    print("de_cipher: ",de_cipher)
    print("a2b_hex(de_cipher): ",a2b_hex(de_cipher))
    print()
    #修改1
    bin_cipher = bytearray(a2b_hex(cipher))
    bin_cipher[15] = bin_cipher[15] ^ ord('g') ^ ord('G')
    de_cipher = decrypt(iv,b2a_hex(bin_cipher))
    print('----把最後的g改成G----')
    print("de_cipher2: ",de_cipher)
    print("a2b_hex(de_cipher): ",a2b_hex(de_cipher))
    print()
    #修改2
    bin_decipher = bytearray(a2b_hex(de_cipher))
    bin_iv = bytearray(iv)
    for i in range(len(iv)):
        bin_iv[i] = bin_iv[i] ^ bin_decipher[i] ^ ord('X')
    de_cipher = decrypt(bytes(bin_iv),b2a_hex(bin_cipher))
    print('----把第一組的字節改成X----')
    print("de_cipher3:",de_cipher)
    print("a2b_hex(de_cipher3): ",a2b_hex(de_cipher))

```

最後我們利用求出的新 IV 值去解密，也就成功地得到了這個結果：

```

de_cipher2:  b'f8720f243415cf625674924312bc477868656c6c6f636263666c697070696e47'
a2b_hex(de_cipher):  b'\xf8r\x0f$4\x15\xcfbVt\x92C\x12\xbcGxhellocbclipping'
----把第一組的字節改成X----
de_cipher3:  b'585858585858585858585858585858585868656c6c6f636263666c697070696e47'
a2b_hex(de_cipher3):  b'XXXXXXXXXXXXXXXXXhellocbclipping'

```

在這個實作中，我們可以知道使用者能夠在不知道金鑰的情況下，通過修改密文與 IV 值來控制明文，將其修改成自己想要的內容。那接下來得階段，我們將這個 CBC 概念實作在網路攻擊上面，詳細請看實作二。

## 6、 攻擊實作二

在瀏覽 CBC 位元組翻轉攻擊相關 CTF 範例後，我們選擇其中流程相對完整的《Login Form》一題，作為第二份實作。完整程式碼一共五個檔案，其中

四個為網頁 ( HTML 、 PHP ) 檔案、一個為 Python Jupiter NoteBook

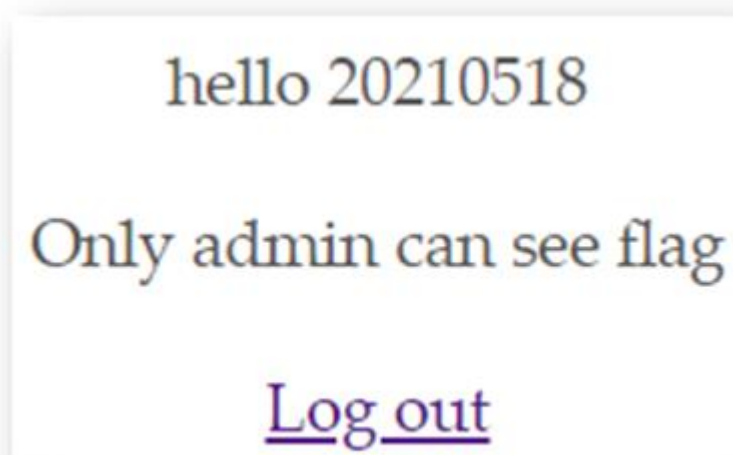
( ipynb ) 檔案。

在使用 xampp 開始 Apache 模組後，進入 Login.php，此時我們會看到一個簡單的登入界面。由於對其尚未了解，先隨便嘗試了一組輸入：

20210518 及 123，點擊登入後獲得回應："hello 20210518 Only admin can see flag"。



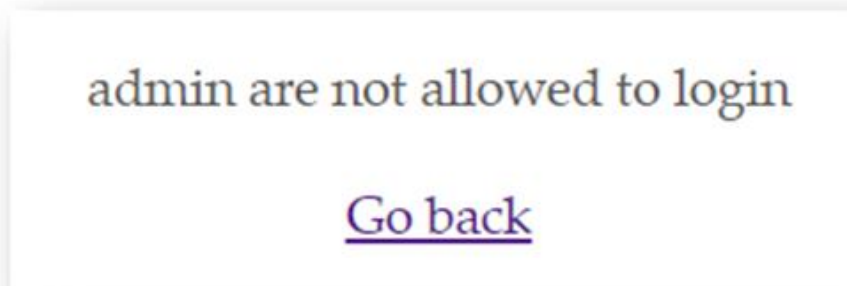
A screenshot of a web page titled "Login Form". Below the title is a subtitle: "Fill out the form below to login to my super awesome imaginary control panel." There are two input fields: "username:" with the value "20210518" and "password:" with the value "\*\*\*". Below the password field is a "Login" button.



A screenshot of the response after logging in. It displays the text "hello 20210518" and "Only admin can see flag" in a serif font. At the bottom, there is a link labeled "Log out" in a purple, underlined font.

因此，第二步中，使用輸入 admin、123 嘗試登入，卻又得到"admin are not allowed to login"之回應，意料之中，題目並沒有如此簡單。至此，我們

得到了兩個關鍵資訊——若 username 是 admin 則無法登入、順利登入後若 username 不是 admin 則無法獲取 flag，即，無法解題。由此，我們便可應用本報告之主題——CBC 位元組反轉攻擊。



首先，確定目標：使用非 admin 之 username 登入，並透過修改 cookie 將 username 之解密結果更改為 admin，以獲取 flag。詳細流程如下：

A. 決定輸入對為 ( admix, 123 )

此時，若徑直登入，獲得之回應自然是“ hello admix Only admin can see flag” ，而序列化輸入的結果與自 cookie 獲取之 IV、密文分別為：

*a:2:{s:8:"userna*

*me";s:5:"admix";*

*s:8:"password";s*

*:3:"123";}*

*E Aj9PlCFRdfH57JPmzVH1Q==*

*vdbLhWgHudsUmNkmse0J4nlqDW/adXu+f6/yGg2aboyyr+UaCuPzlsRaR*

*2X1J1HYQrz76DCzGw/HVVtQBc+bDA==*

B. 透過位元組翻轉公式，將 *me";s:5:"admi**x**";* 修改為 *me";s:5:"admin**n**";*

修改後的密文及解密後的明文為：

*3ewoCAbMjAoZbRicERidJW1lljtzOjU6ImFkbWluljtzOjg6lnBhc3N3b3J*

*kljtzOjM6IjEyMyI7fQ==*

*me";s:5:"admin";*

*s:8:"password";s*

*:3:"123";}*

*:3:"123";}*

C. 透過位元組翻轉公式修改 IV，修正受到位元組翻轉而變成亂碼的部分

修改後的 IV 為：

*rN7nDC068+XkqN+g71+0kQ==*

D. 將翻轉後的密文和 IV 存入 cookie，並重新整理，即成功獲得 flag

```
newCipher: vdbLhwgHudsUmtkmsfsJ4nIqDW/adXu%2Bf6/yGg2aboyyr%2BUaCuPzIsRaR2X1J1HYQrz76DCzGw/HVVtQBc%2BbDA%3D%3D  
newiv: rN7nDC068%2BXkqN%2Bg71%2B0kQ%3D%3D
```

```
setcookie("iv", base64_encode($iv));  
setcookie("cipher", base64_encode($cipher));
```

Hello admin

Flag is ctf{123cbcchange}

Log out

## 7、 攻擊防範

對於 CBC 模式下的加密，顯然存在著巨大的漏洞，那該如何使這種加密模式更加安全呢？

首先，對於前面的理解，對於 CBC 加密的攻擊方式，或者說是 CBC 加密的破解方式，最重要的核心在於，攻擊者只需要知道其中一個 IV 值，就可以控制前面的所有明文。也就是說，假設有  $L$  個區塊，當明文為  $M = M[1] \dots M[L]$ ，經過一個加密函數  $F$ ，我們得到  $F(K, M) = C[1] \dots C[L]$ 。

當你想要計算出  $C[i]$ ，你得知道  $M[1] \dots M[i] \Rightarrow C[i]$ 。由於 CBC 的計算方式，你想得到  $C[i]$ ，不需要知道明文  $M[i+1], \dots, M[L]$ 。這種加密演算法函數  $F$  我們就稱為 on-line cipher。

一個被大眾認同足夠安全的概念為偽隨機排列(PRP)，但這種概念在實際計算上是不可行的，不然我們將會看到加密第  $i$  個明文只會輸出第  $i$  個密文的情況。因此，對於 on-line cipher，我們必須放棄讓他滿足 PRP 的安全屬性，我們使用一個適當的安全替代概念。我們只要求在密文為 on-line cipher 的情況下能「盡量隨機」，這個概念我們稱為 on-line-PRP。

那 CBC 加密法能夠滿足這個概念以達到更高的安全性嗎？「On-Line Ciphers and the Hash-CBC Construction」論文中提供了一種叫做 Hash-CBC (HCBC) 的結構

$$\mathcal{H}: \{0,1\}^{hk} * \{0,1\}^n \rightarrow \{0,1\}^n$$



我們將介紹論文中 HCBC 並證明其對選擇明文的安全性。這種結構是類似於 CBC 加密模式。唯一的區別是每個輸出塊在被 XOR 之前通過一個金鑰雜湊函數(hash function)到下一個輸入塊。雜湊函數的密鑰是保密的。首先先定義:

$$n, d \in \mathbb{N}, n, d \geq 1$$

$$E: \{0,1\}^{ek} * \{0,1\}^n \rightarrow \{0,1\}^n : \text{分組加密演算法}$$

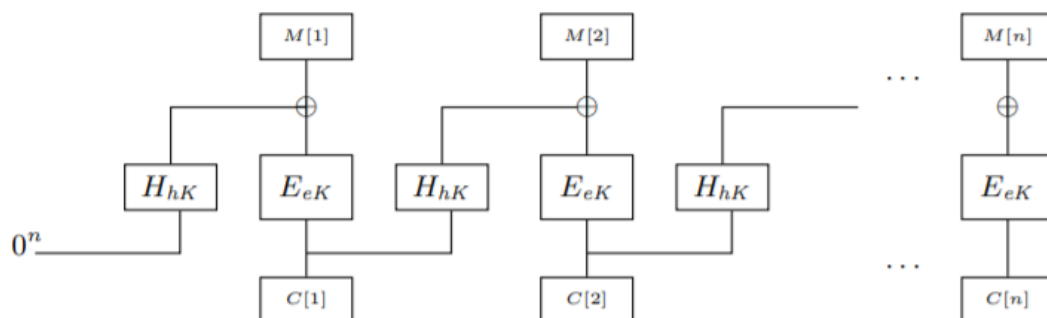
$$\mathcal{H}: \{0,1\}^{hk} * \{0,1\}^n \rightarrow \{0,1\}^n : \text{family of hash functions}$$

將兩者合併定義為一個新的加密模式:

$$\mathcal{HCBC}: \{0,1\}^{ek+hk} * D_{d,n} \rightarrow D_{d,n}$$

一對金鑰  $ek$  與  $hk$  與之對應， $ek$  是  $E$  的金鑰和  $hk$  是  $H$  的金鑰。

密文和他的 inverse 分別為  $M, C \in D_{d,n}$ ，下圖為 HCBC 的加密流程:



<pre> HCBC(<math>eK \  hK, M</math>) Parse <math>M</math> as <math>M[1] \dots M[l]</math> with <math>l \geq 1</math> <math>C[0] \leftarrow 0^n</math> For <math>i = 1, \dots, l</math> do     <math>P[i] \leftarrow H(hK, C[i-1]) \oplus M[i]</math>     <math>C[i] \leftarrow E(eK, P[i])</math> EndFor Return <math>C[1] \dots C[l]</math> </pre>	<pre> <math>\text{HCBC}^{-1}(eK \  hK, C)</math> Parse <math>C</math> as <math>C[1] \dots C[l]</math> with <math>l \geq 1</math> <math>C[0] \leftarrow 0^n</math> For <math>i = 1, \dots, l</math> do     <math>P[i] \leftarrow E^{-1}(eK, C[i])</math>     <math>M[i] \leftarrow H(hK, C[i-1]) \oplus P[i]</math> EndFor Return <math>M[1] \dots M[l]</math> ■ </pre>
---	--

下面的定理告訴我們，假如  $E$  為一個 PRP，要防止選擇明文攻擊，以及  $H$  為一個 AXU 類別的湊碼函數為一個 AXU 類別的湊雜函數，那麼 HCBC，那麼 HCBC 就是一個能夠防止選擇明文攻擊的 OPRP。

**Theorem 1.** *Let  $E: \{0, 1\}^{ek} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$  be a block cipher, and let  $H: \{0, 1\}^{hk} \times \{0, 1\}^n \rightarrow \{0, 1\}^n$  be a family of hash functions. Let HCBC be the  $n$ -on-line cipher associated to them as per Construction 1. Then, for any integers  $t, q_e, \mu_e \geq 0$  such that  $\mu_e/n \leq 2^{n-1}$ , we have*

$$\text{Adv}_{\text{HCBC}}^{\text{oprpcpa}}(t, q_e, \mu_e) \leq \text{Adv}_E^{\text{prpcpa}}(t, \mu_e/n, \mu_e) + \left( \frac{\mu_e^2 - n\mu_e}{n^2} \right) \cdot \text{Adv}_H^{\text{axu}} + \frac{\mu_e^2 + 2n(q_e + 1)\mu_e}{n^2 \cdot 2^n}.$$

((此定理的證明先略過))

依照前面定義過的架構， $\text{HCBC}(\pi(hK, \cdot))$  中我們分別使用  $\pi$  and  $\pi^{-1}$  來代替  $E$  和他的 inverse。以下的證明將 on-line cipher 看做是一個  $2^n$  位元的 tree 位元的 tree 在  $\{0, 1\}^n$  上做排列，之後再經過一連串的混合論證(一連串的不同 Game 讓我們從不同的 Game 讓我們從  $\text{OPermd}_n$  到 HCBC。

最後得到函數  $\pi$  的任何兩個輸入相等的機率極小，這調表著假如沒有特定情況，密文塊的值不是全然由前面的密文塊決定，是隨機以及獨特的。

更詳細的證明內容我們在這裡並不介紹，但我們可以知道 HCBC 將每個輸出塊先經由雜湊函數做 XOR 運算到下一個輸入塊。在原本的 CBC 位元翻轉攻擊中，我們可以利用修改密文塊來控制明文，但在這種情況下，我即使獲得了密文，但也無法藉由直接控制密文來得到想要的明文。因為此時明文已經不是由前面的密文所控制，而是經過雜湊函數所推得。即使我控制了密文，也無法控制雜湊函數的運算。故若要防範 CBC 位元翻轉攻擊，我們可以加上雜湊函數的運算來保證其安全性。

## 8、心得

如同前言所述，我們對 CBC 模式的初次認識是發生在課堂中，因此在討論本次報告主題時，有了相當的共識，並且快速地展開了準備。自訂好題目至完成報告初稿，我們查閱了一些文獻，以決定報告的架構、實作的方式。

繳交初稿後，教授、助教也給予了一些建議，讓我們能即時修改，在上台報告時有更好的表現和結果，能夠獲得老師的嘉許，我們都非常開心，也更有動力在期末報告時分享我們所學習到的新知。

本次報告中，事實上也有非常多的嘗試，因為疫情的緣故必須採取線上報告，對部分組員是第一次，對網路、設備的穩定都有一定程度的要求。還記得我們原先可能是第一週報告的最後一組，組員前一天都戰戰兢兢地準備，結果

隔天，正準備報告時——學校網路崩潰了，非常曲折，不過，我們也利用多出來的一週時間，再加入一些細節，使報告更加流暢。

此外，組員們依據自己的特長各司其職，同時也互相幫助，其中特別讓人印象深刻的部分，莫過於每當在討論群組裡詢問「是否有熟悉某一語言（例如：PHP）、某一領域（例如：數論）的人？」、「是否能主導該部分？」時，有人願意站出來，讓報告的流程能如同預期完成，雖然這本就是在分組報告時應有的基礎態度，但大學的四年生涯走到現在，我們都有許多在其他課堂的分組報告中被「荼毒」過的經歷，便顯得這次報告體驗有多好了。

本學期的資訊安全課程及報告，給我們留下的不僅僅是對新知更加深入的了解，也是非常良好的報告體驗與經驗，而部分組員的畢業專題也與資訊安全相關，期許我們未來能夠對這一領域的知識與技術有更好的理解、掌握。

## 9、 參考資料

區塊加密法工作模式

<https://zh.wikipedia.org/wiki/分组密码工作模式>

CBC 比特翻轉攻擊詳解

<https://www.quarkay.com/security/45/CBC-bit-reversal-attack-analysis>

Online Cryptography Course -- Dan Boneh

<https://crypto.stanford.edu/~dabo/courses/OnlineCrypto/slides/04-using-block-v2-annotated.pdf>

On-Line Ciphers and the Hash-CBC Construction

<https://www.iacr.org/archive/crypto2001/21390291.pdf>

白帽子講 Web 安全

<http://resources.infosecinstitute.com/cbc-byte-flipping-attack-101-approach/>

[CTF]AES-CBC 位元組翻轉攻擊

<https://blog.csdn.net/V1040375575/article/details/111773524>