Symmetric-key Encryption

By: Killua4564

$ whoami

- Killua4564 / Cheng Yan
- NTUST MIS
- Crypto, Misc, Reverse
- https://killua4564.github.io/

Block Cipher

- 古典密碼學
  - 單表代換加密
  - 多表代換加密
  - 其他類型
- 現代密碼學
  - 對稱加密 / 區塊加密 (Block mode)
    - DES
    - AES
  - 非對稱加密 / 公開金鑰加密
    - RSA
    - Diffie–Hellman
    - ECC
  - 雜湊函數
- 編碼方式

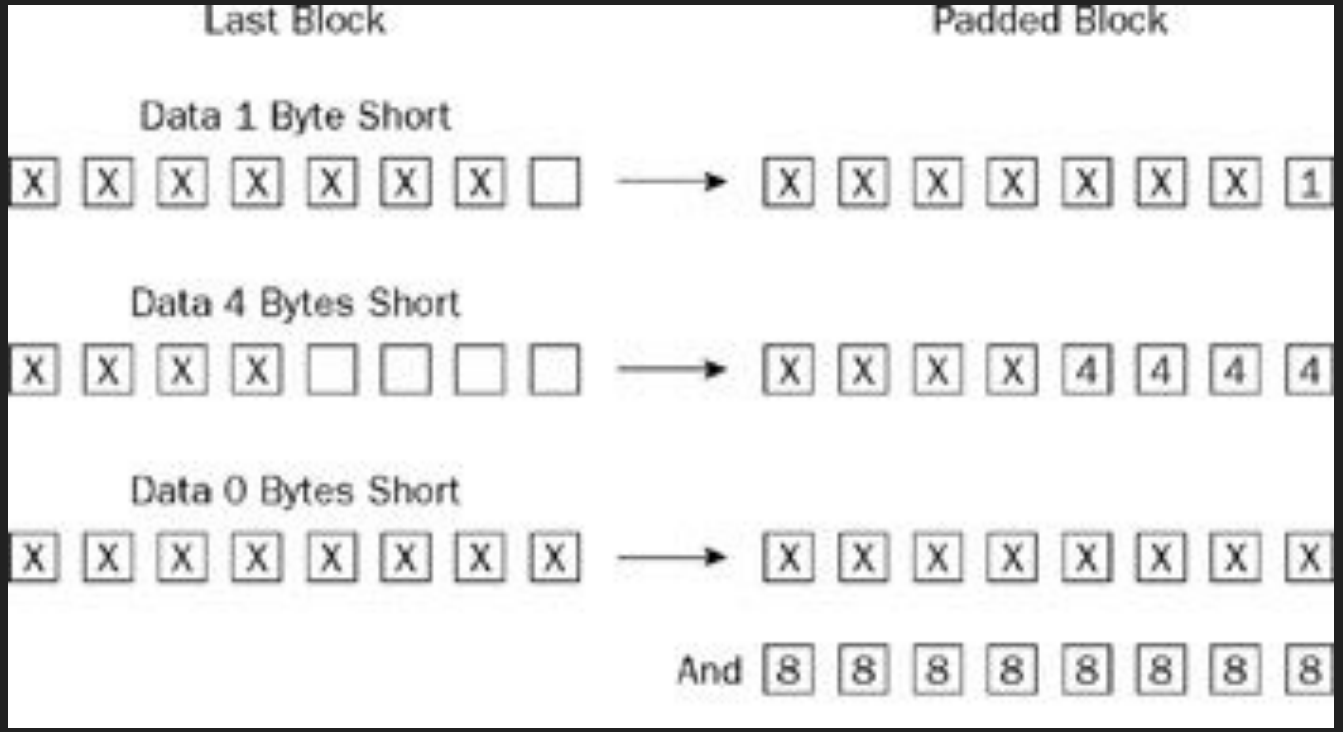
Python Tools

- hashpumpy
- pycrypto / pycryptodome
- pwntools
- sagemath
- xortool

Confusion and Diffusion

- Proposed by Claude Shannon in 1945
- Confusion (混淆)
  - 明文的每個 bit 都應該被 key 的數個部分影響結果
  - 讓兩者之間有複雜的關聯性
  - e.g. s-box、mult
- Diffusion (擴散)
  - 改變明文的一個 bit, 一半以上 bits 的密文應該要被改變
  - 改變密文的一個 bit, 應該要有接近一半 bits 的明文被改變
  - 被改變 bits 的位置要盡量看起來是隨機的
  - e.g. substitution、rotation

PKCS#7 Padding

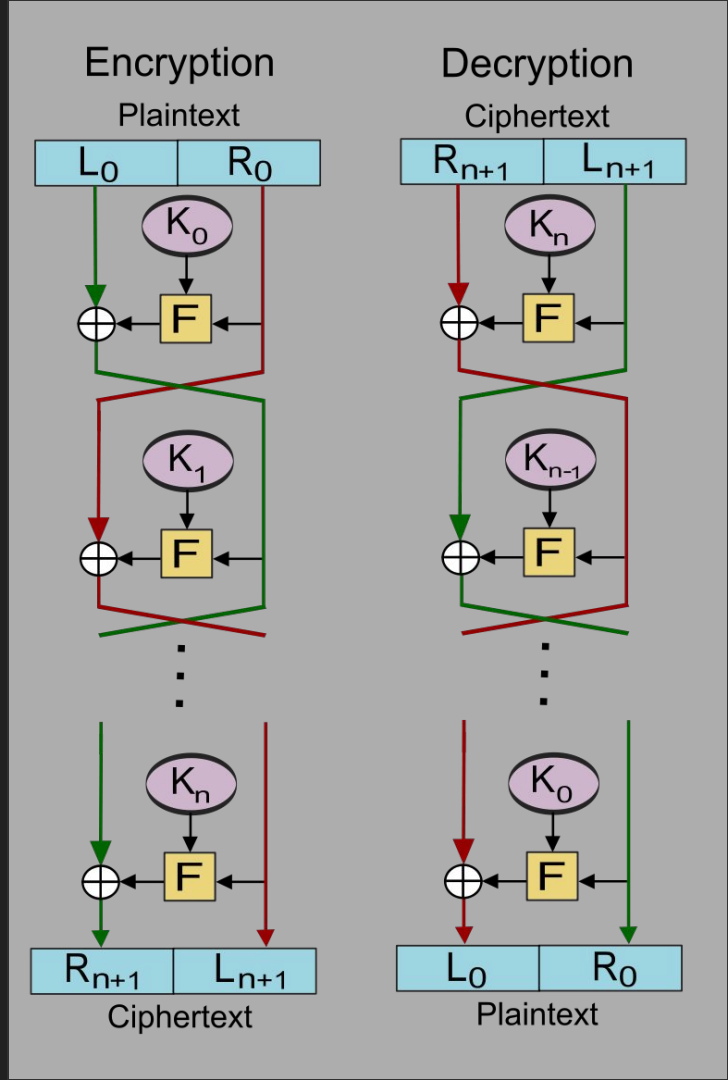


Feistel Cipher

- 加密和解密很相似
- F函式不用是可逆的
- 程式簡單容易寫
- for all i = 1, 2, 3, …, n

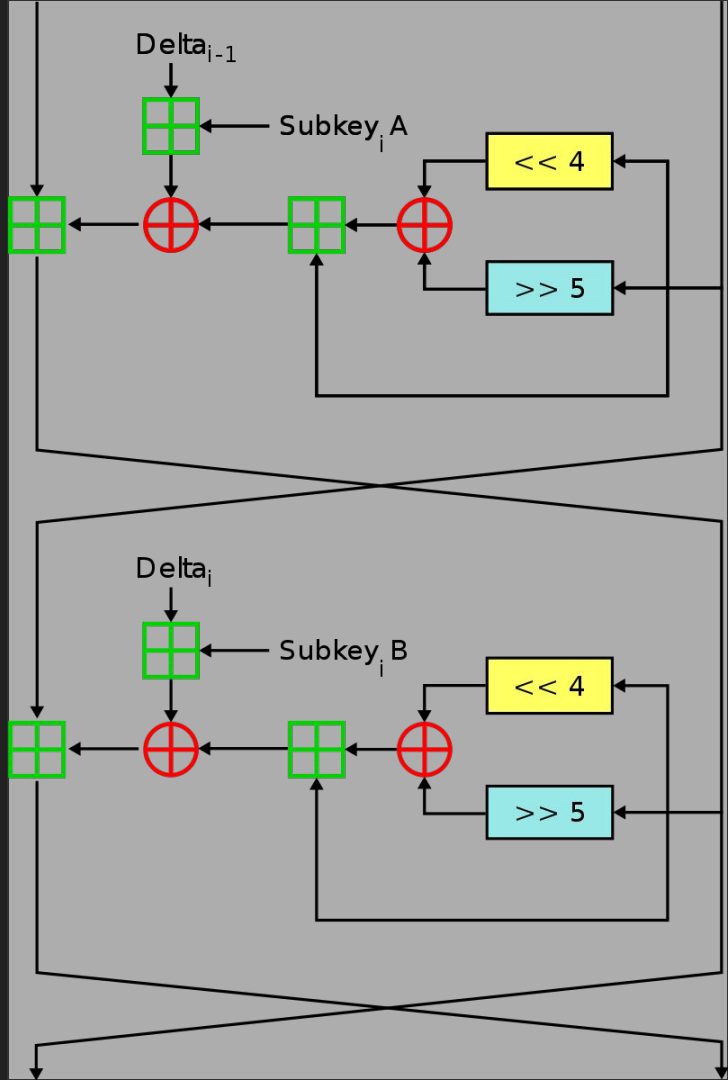
$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus \mathrm{F}(R_i, K_i).$$

e.g. DES、TEA



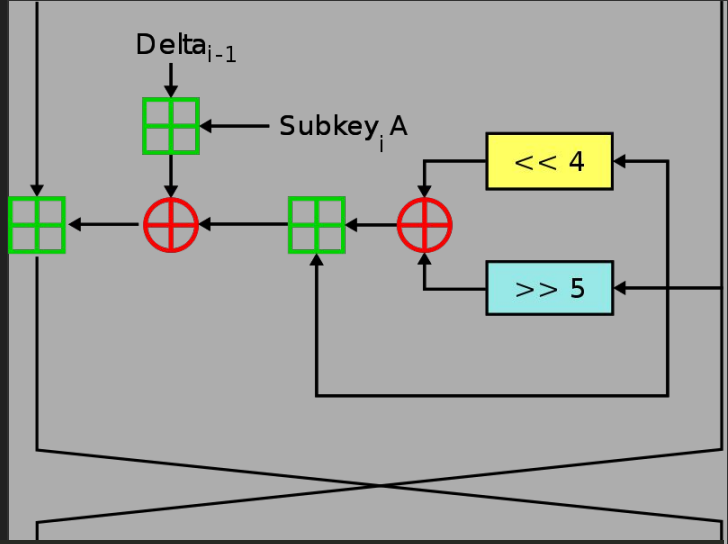
XTEA

- Feistel Cipher
- key size: 128 bits
- block size: 64 bits



XTEA

- Feistel Cipher
- key size: 128 bits
- block size: 64 bits



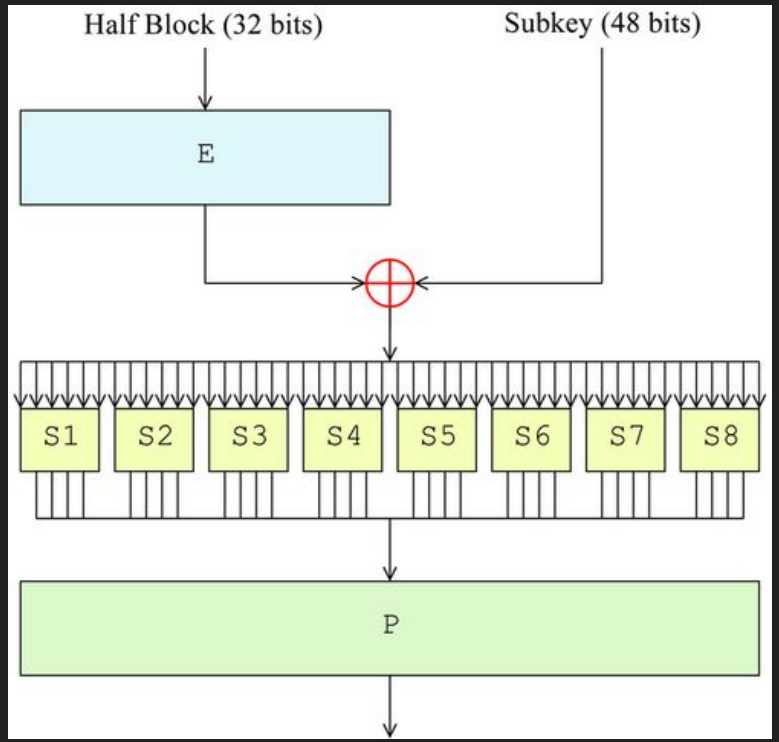

```
def encrypt(self, pt):
    ct = b""
    pt = pad(pt)
    for i in range(0, len(pt), 8):
        pt_block = bytes(pt[i:i+8])
        v0, v1 = struct.unpack("<2L", pt_block)
        sum, delta = 0, 0x9E3779B9
        for _ in range(self.rounds):
            v0 = (v0 + (((v1 << 4 ^ v1 >> 5) + v1) ^ (sum + self.key[sum & 3]))) & 0xFFFFFFFF
            sum = (sum + delta) & 0xFFFFFFFF
            v1 = (v1 + (((v0 << 4 ^ v0 >> 5) + v0) ^ (sum + self.key[sum >> 11 & 3]))) & 0xFFFFFFFF
        ct += struct.pack("<2L", v0, v1)
    return ct
```

DES

- Feistel Cipher
- key size: 56 bits
  - parity check bits
- block size: 64 bits

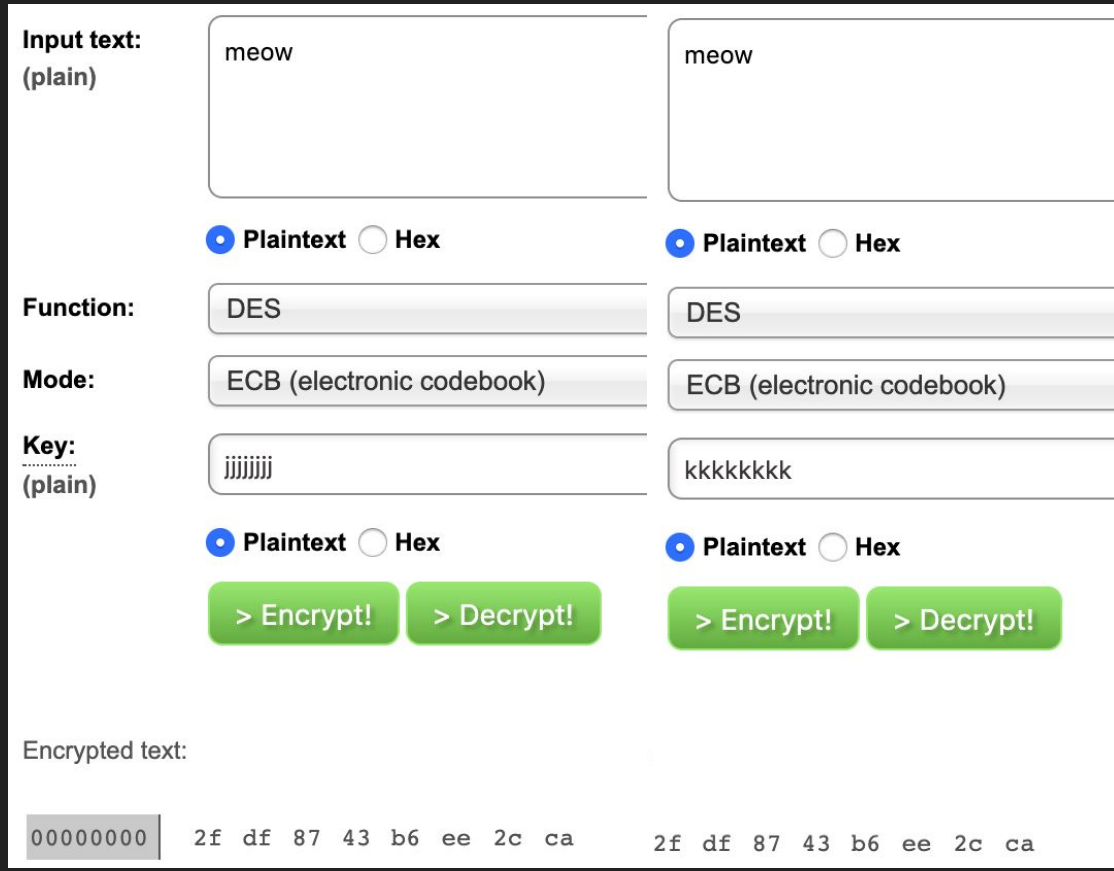
- 3DES
- Enc(Dec(Enc(plaintext, key1), key2), key3)

- 可用差分和線性密碼攻擊



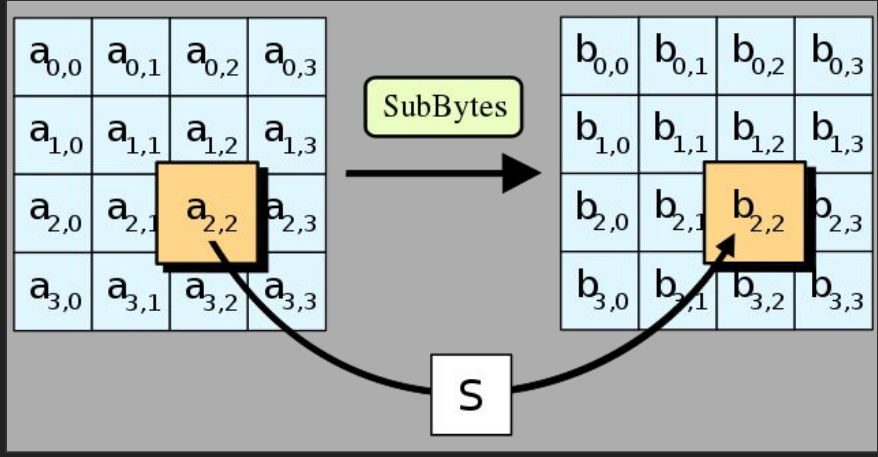
DES crack

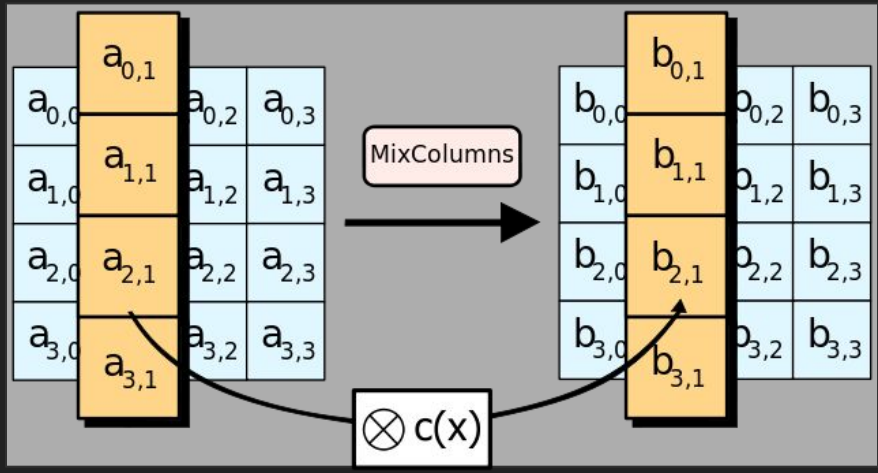
1. Brute force attack parity check bits
2. 二補數特性
   $E_K(P) = C \Leftrightarrow E_{\overline{K}}(\overline{P}) = \overline{C}$
3. 弱金鑰
   $E_K(E_K(P)) = P$
4. 半弱金鑰
   $E_{K_1}(E_{K_2}(P)) = P$



AES

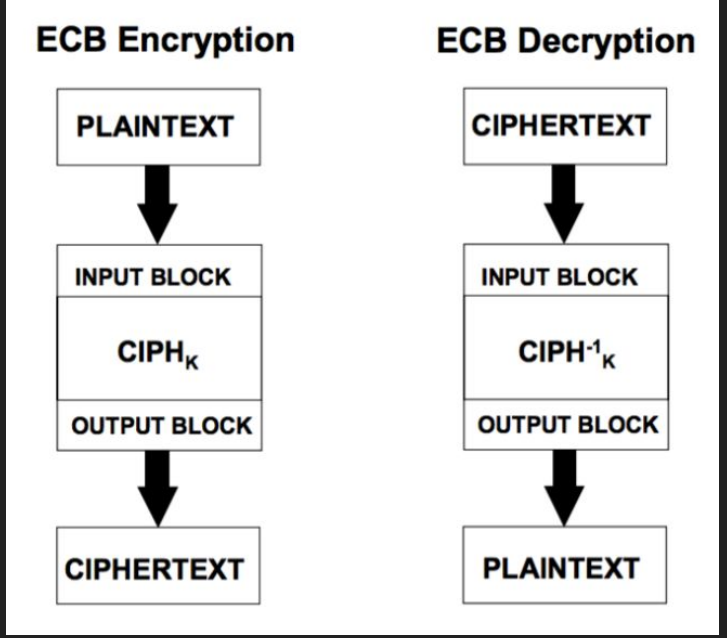
- Block Cipher
- Square State
- key size: 128, 192, 256 bits
- block size: 128 bits
- 步驟:
  - AddRoundKey
  - SubBytes
  - ShiftRows
  - MixColumns
- 曾經被旁道攻擊成功過





ECB Mode

- 電子密碼本模式
- 相同明文 → 相同密文
- 混淆性不夠
- 攻擊: 複製貼上



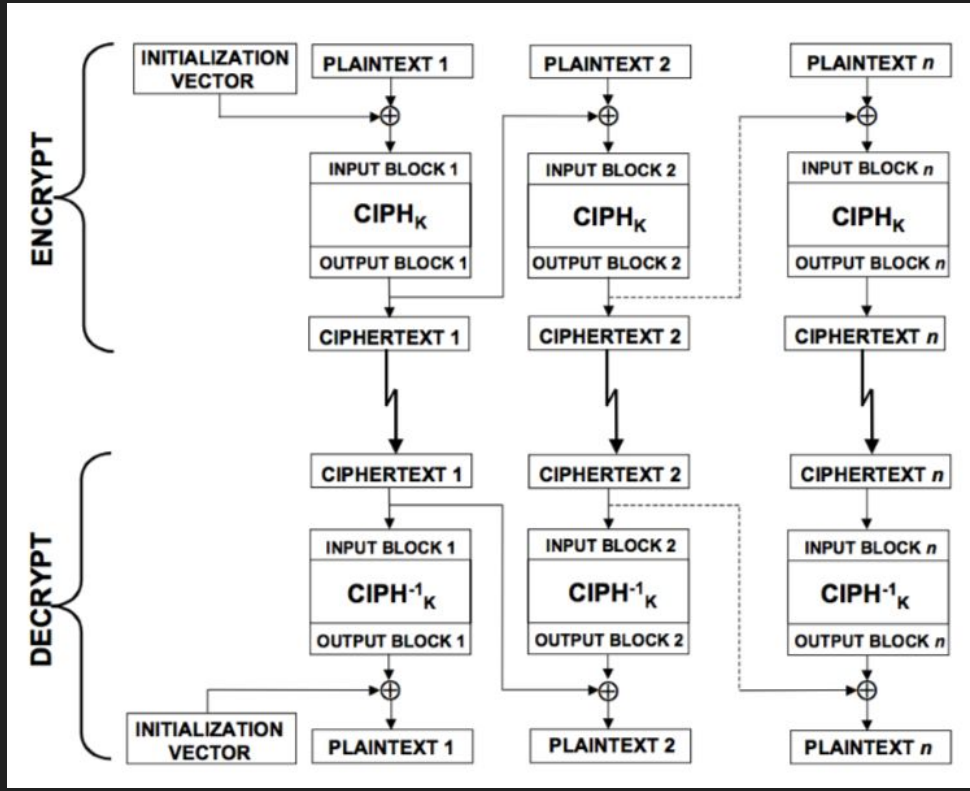

```
|usr=a&pw|=a&root=|N.......| → |A|B|C|

|usr=abcd|Y&pw=aaa|&root=N.| → |D|E|F|


|usr=a&pw|=a&root=|Y&pw=aaa|N.......| → |A|B|E|C|
```

CBC Mode

- 密碼區塊鏈模式
- 相同明文 → 不同密文
- TLS 最常見的模式
- 攻擊:
  Padding Oracle (POODLE)
  BEAST Attack
  EFAIL



CBC Bit-Flipping

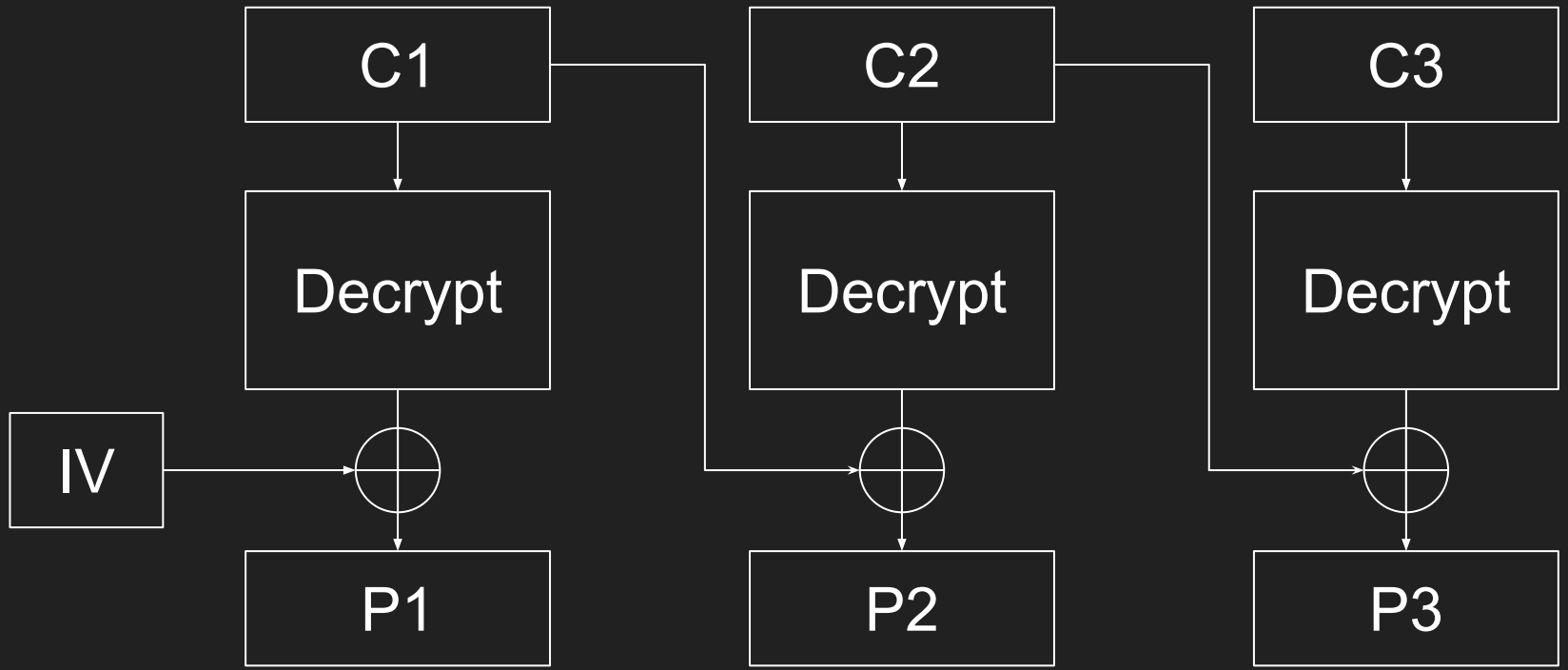
E(P1 || P2 || P3, iv, key) → C1 || C2 || C3

假設 P2 是不重要的資料(損毀也沒關係)
已知 C1, C2, C3, P3 (不知 key) 的情況
想要竄改 P3 為 P4

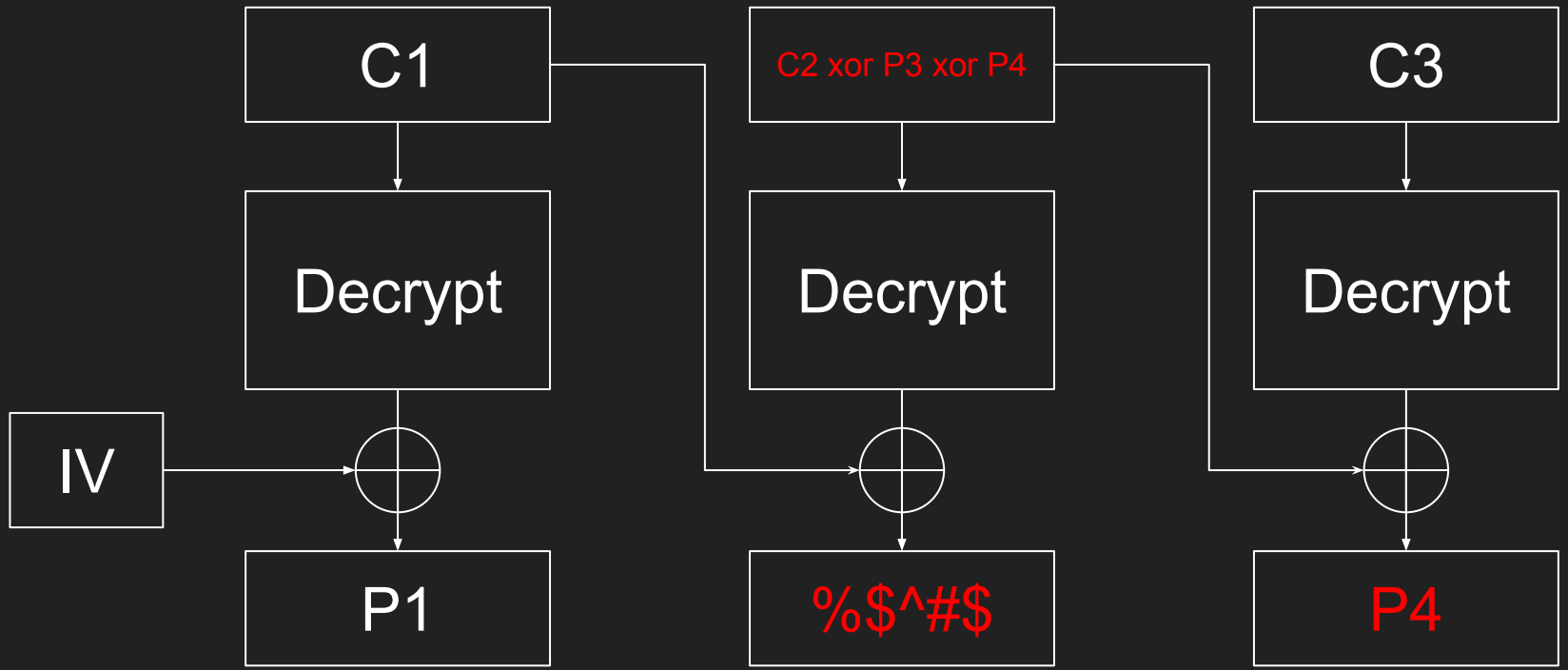
根據 CBC 可以知道 D(C3, key) = P3 xor C2
若將 C2 竄改為 C2 xor P3 xor P4
這樣 C3 被 CBC 解出來就是 P4

D(C1 || (C2 xor P3 xor P4) || C3, iv, key) → P1 || IDC || P4

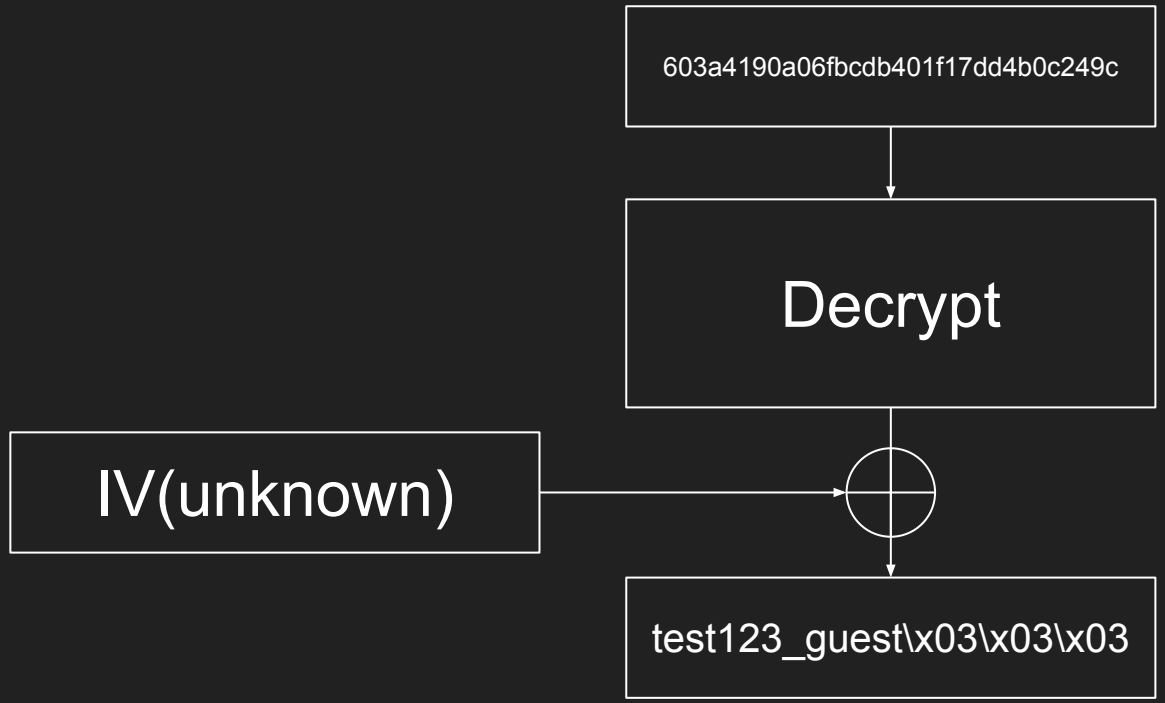
CBC Bit-Flipping



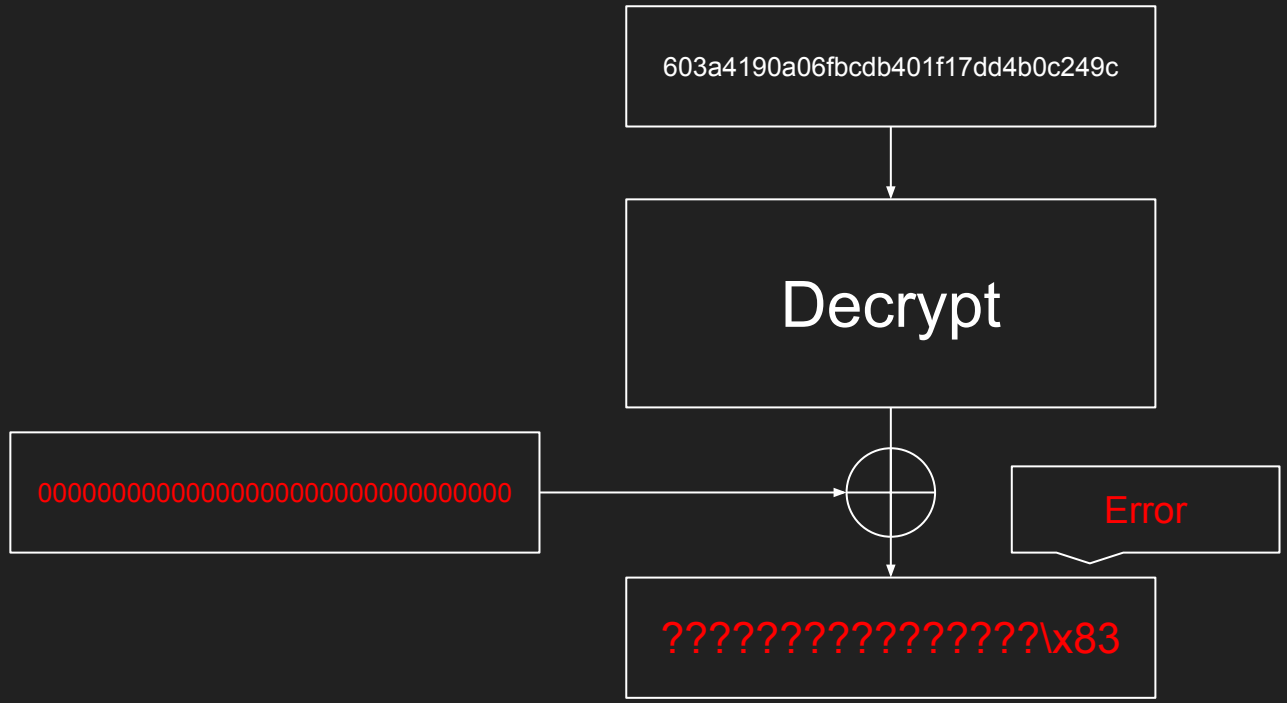
CBC Bit-Flipping



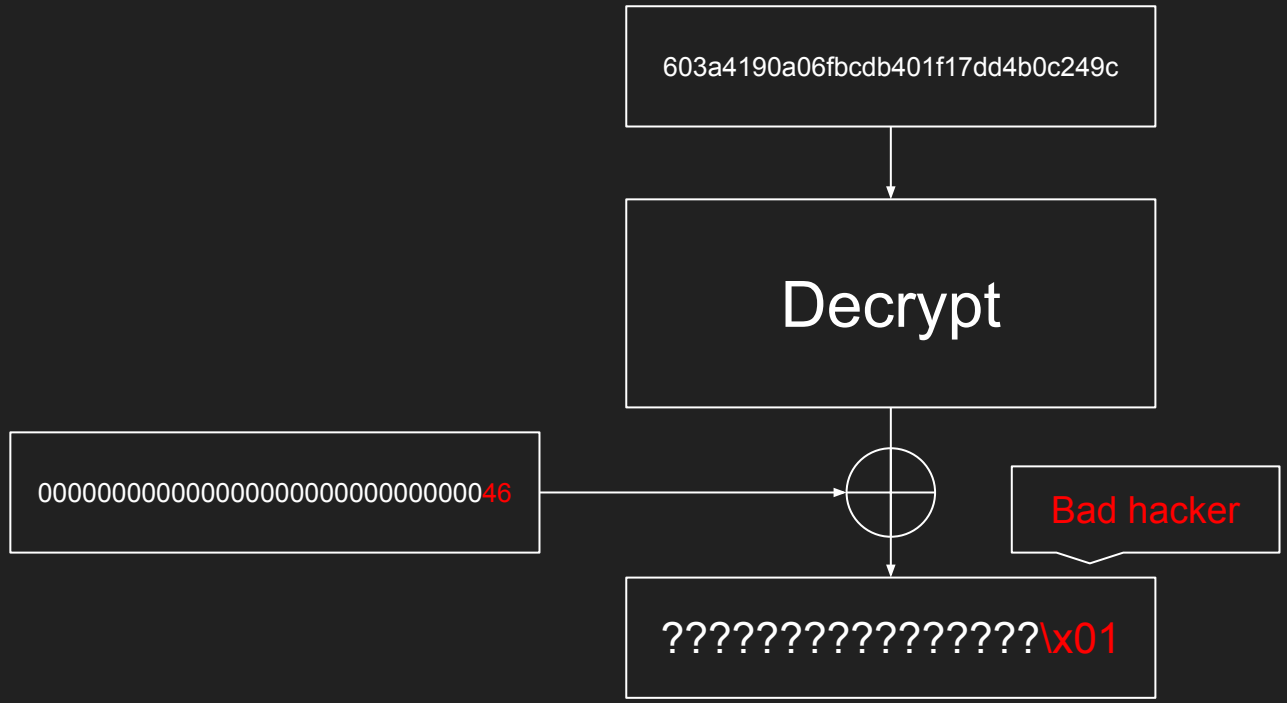
CBC Padding-Oracle



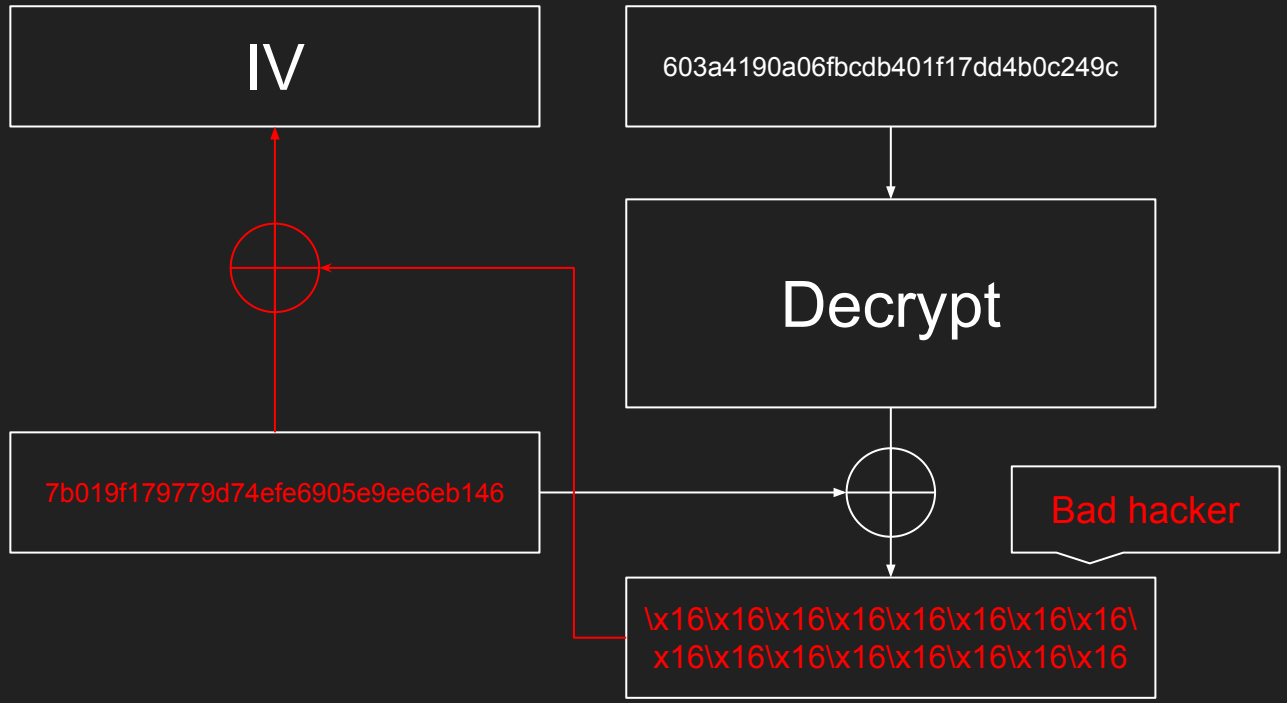
CBC Padding-Oracle



CBC Padding-Oracle



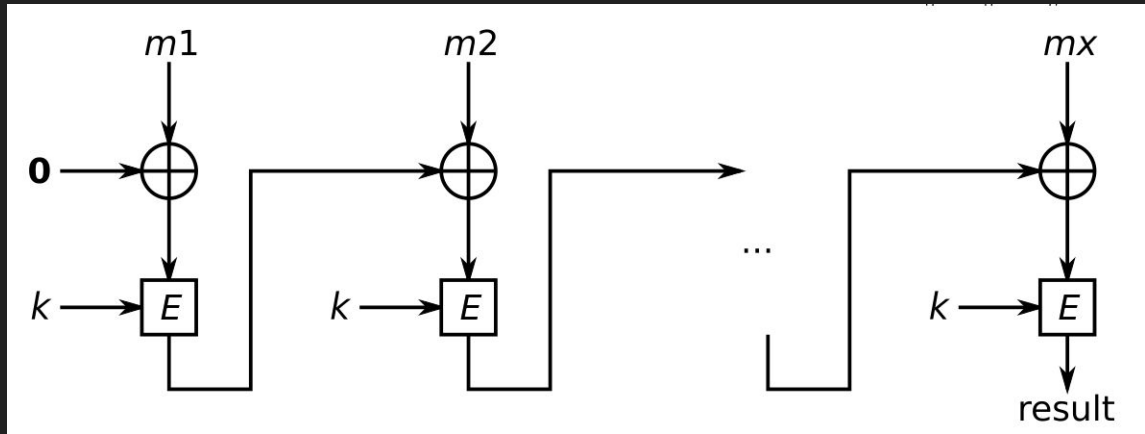
CBC Padding-Oracle



CBC Padding-Oracle

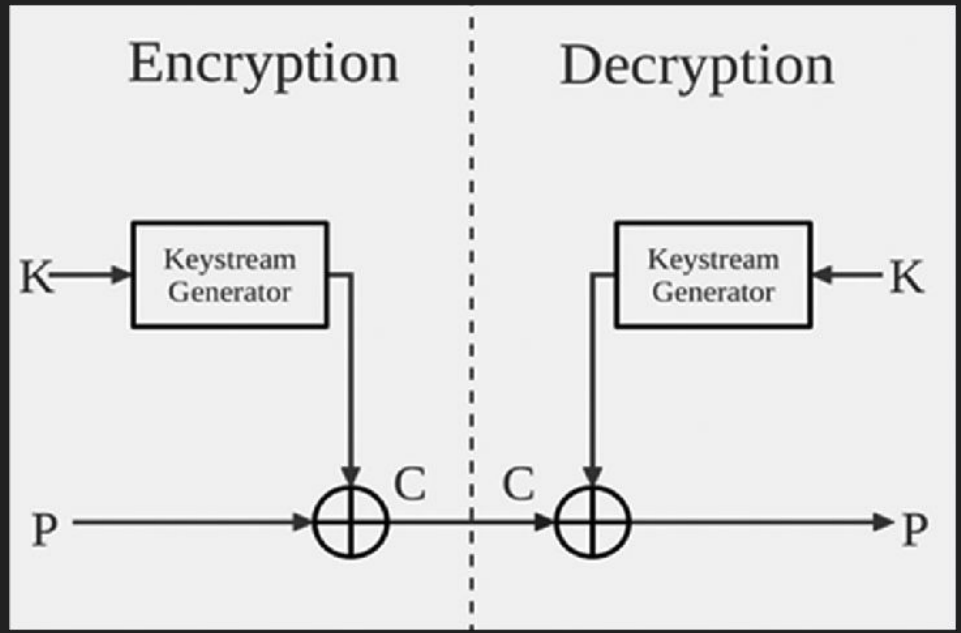
CBC MAC

- 把 message 用 CBC mode 加密
- 將最後一個 block 再用 ECB mode 加密一次
- 其中 CBC 和 ECB mode 的 key 可以是相同的
- 攻擊: 讓兩個不同的 message 產生相同的 MAC
  message1 || null block || MAC(message1)
  message2 || null block || MAC(message2)



Stream Cipher

- 串流加密、資料流加密、流加密
- 通常是 bit by bit xor
- 解決了對稱加密的完善保密性
- e.g. RC4、LFSR

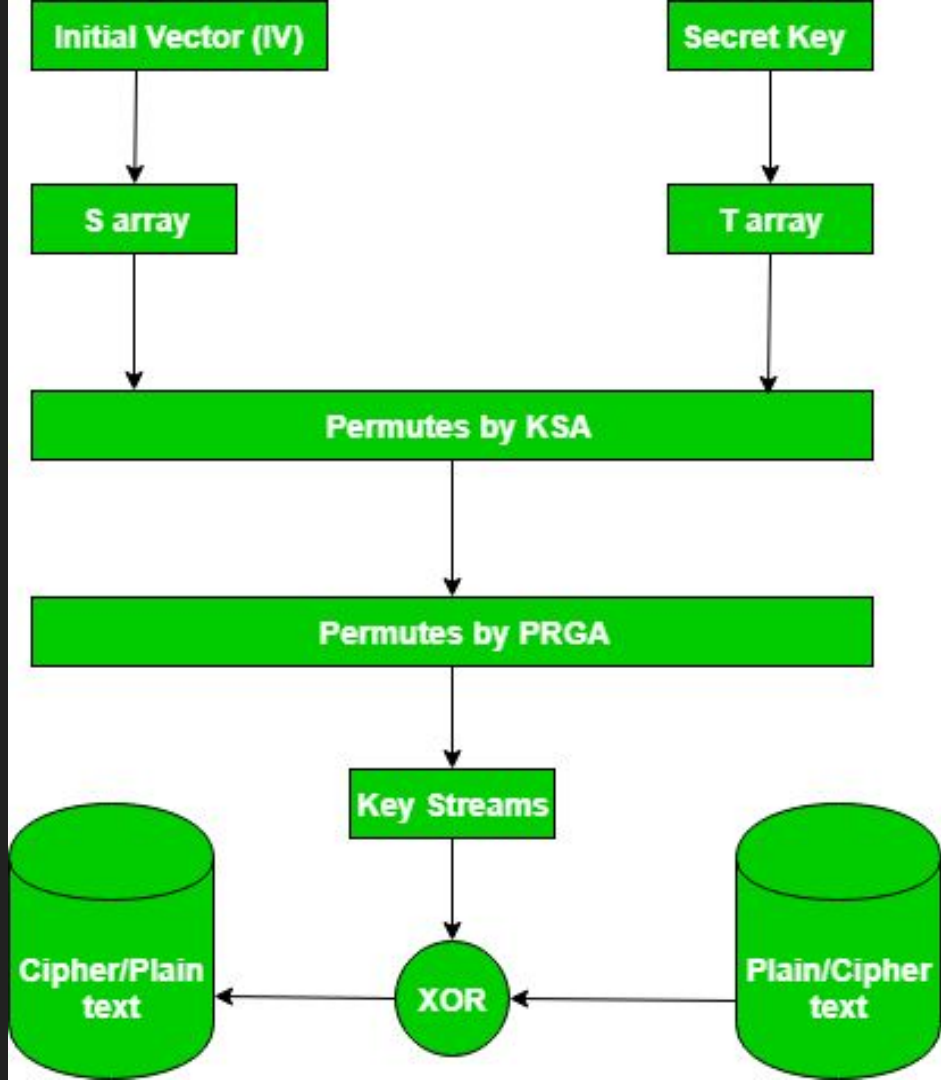


RC4

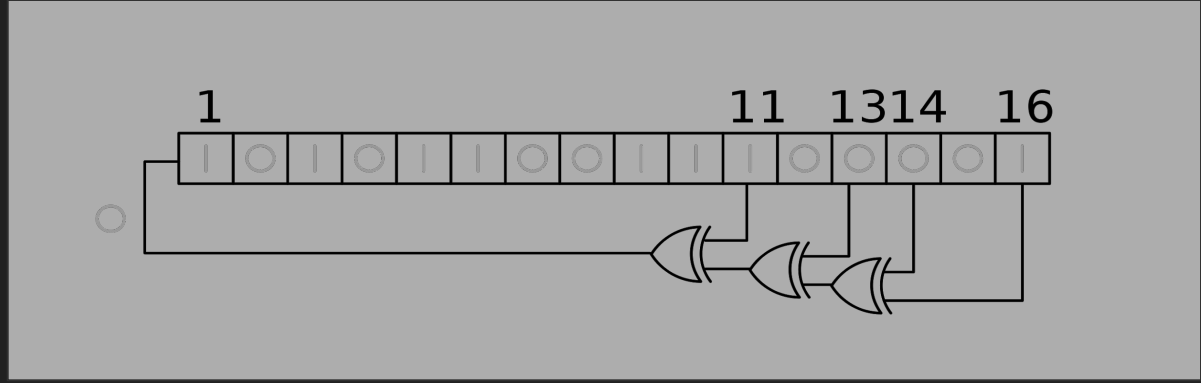
- Stream Cipher
- KSA、PRGA

```python
def KSA(key):
    S, j = list(range(256)), 0
    for i in range(256):
        j = (j + S[i] + key[i % len(key)]) % 256
        S[i], S[j] = S[j], S[i]
    return S

def PRGA(S, size):
    i, j, key = 0, 0, []
    for _ in range(size):
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        key.append(S[(S[i] + S[j]) % 256])
    return bytes(key)
```



LFSR



- 線性反饋移位暫存器
- register 的初始值是 seed
- 一次 output 一個 bit
- 在 GF(2) 底下矩陣乘法
- 若已知 seed 和 taps 的長度容易被破解

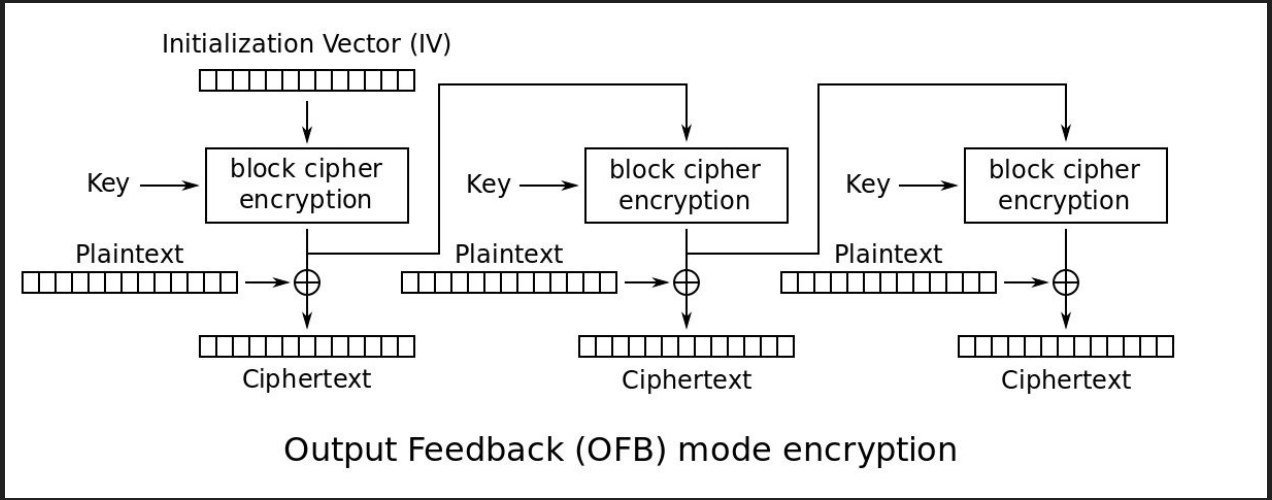
$$\begin{pmatrix} a_k \\ a_{k+1} \\ a_{k+2} \\ \vdots \\ a_{k+n-1} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_0 & c_1 & c_2 & \cdots & c_{n-1} \end{pmatrix} \begin{pmatrix} a_{k-1} \\ a_k \\ a_{k+1} \\ \vdots \\ a_{k+n-2} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_0 & c_1 & c_2 & \cdots & c_{n-1} \end{pmatrix}^k \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

XORShift

- 運算速度快 程式碼簡單
- 定義:
  - message = message xor (message << a)
  - message = message xor (message >> b)
  - message = message xor (message << c)
- 在 GF(2) 底下矩陣乘法
- 工具
  - docker pull sagemath/sagemath
    - sage -pip install pwntools
  - sagemath (https://www.sagemath.org/download.html)
    - sage -f python2
    - sage -i openssl pyopenssl

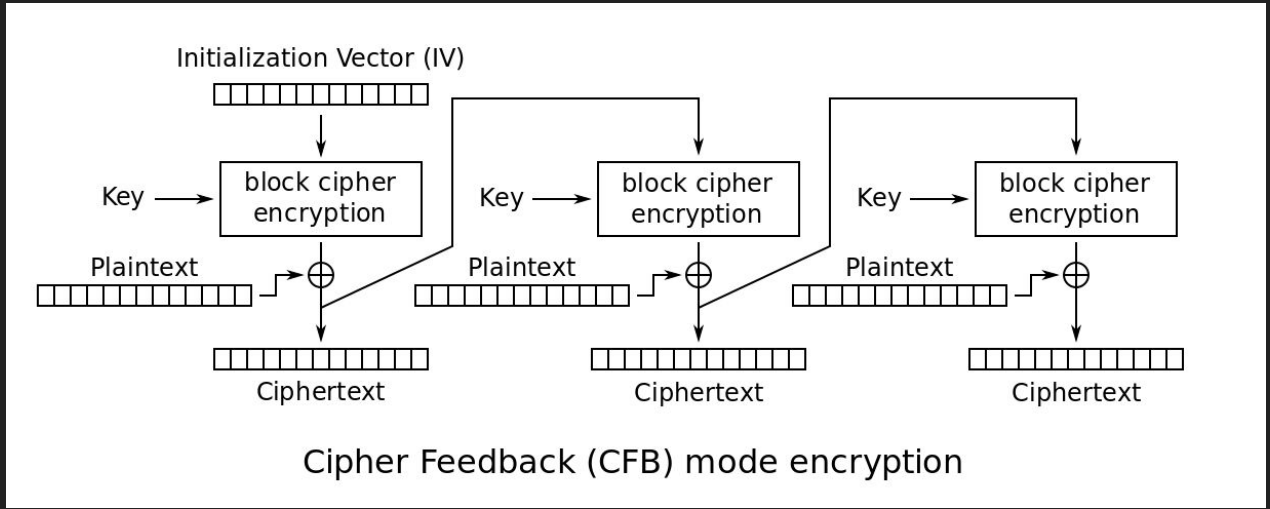
OFB Mode

- 輸出反饋模式
- Synchronous Stream Cipher
- 不受 plaintext 影響下個 block 的 iv
- 若已知明文限制在一定範圍並有大量的密文, 可以實施暴力破解



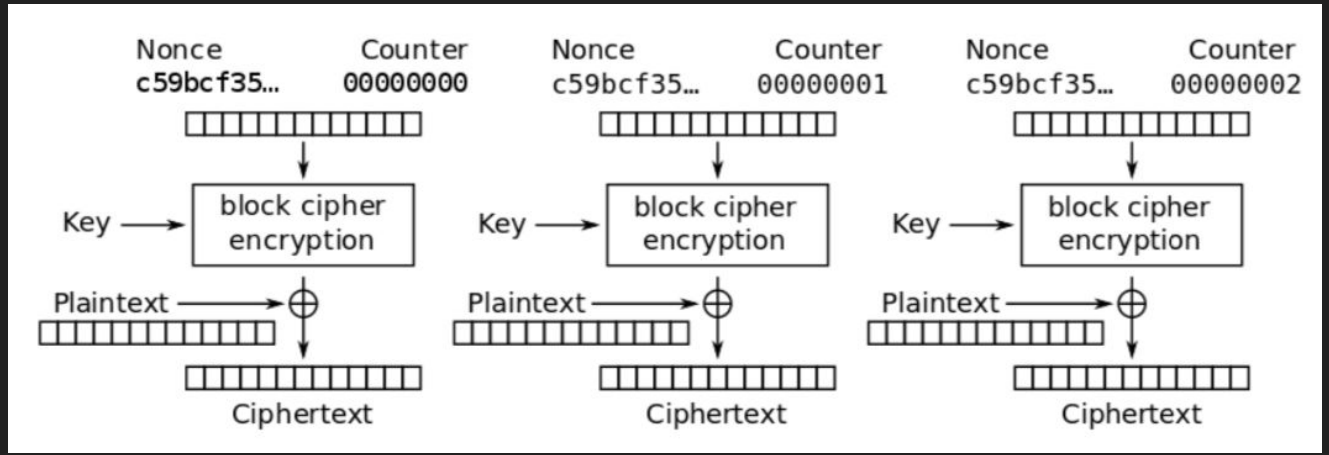
CFB Mode

- 密文反饋模式
- Self-synchronizing Stream Cipher
- 會受 plaintext 影響下個 block 的 iv
- 即使其中一個 block 受損也可以解密出大部分的 block



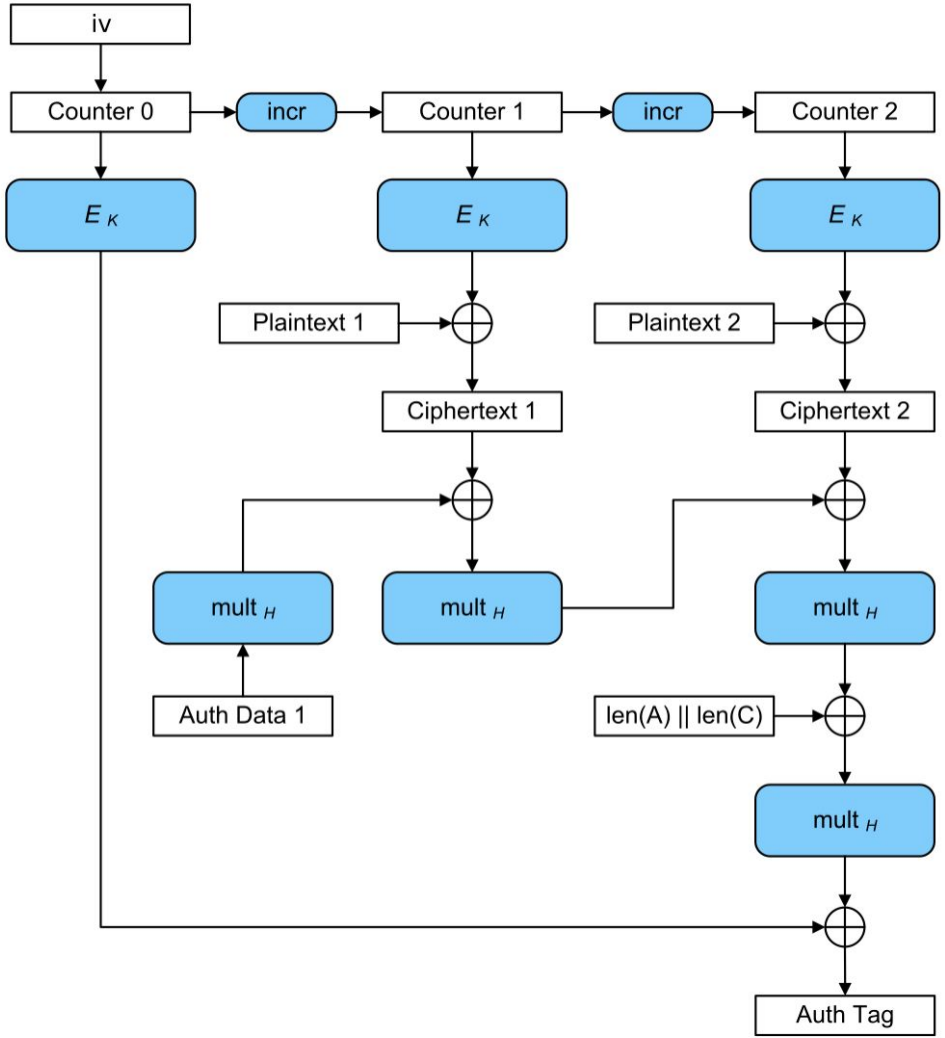
CTR Mode

- 計數器模式
- 由 Nonce || Counter 來產生隨機的 IV
- 確保每次的 IV 皆不同，以達到最大效能的保密性
- 攻擊:
  位元翻轉攻擊 (Bit-Flipping Attack)



GCM Mode

- 伽羅瓦/計數器模式
- 比 CTR mode 多了身份驗證
- 剩下的優缺點大致上沒差
- mult_H 是在 GF(2**128) 做乘法
  - 途中有 bytes 和 ploy 的轉換
- 重複使用 iv 會有 forbidden attack
  - 能拿到任意明文加密結果
  - 就可以偽造任何人的簽章



GCM Mode

假設密文是有兩個區塊, 則簽章可以表示成
Tag = A*H**4 + C1*H**3 + C2*H**2 + L*H + E(J0)
其中未知的只有 H 和 E(J0) 而已

所以如果拿兩個已知明文 m1, m2 去加密, 得到:
T1 = A1*H**4 + C11*H**3 + C12*H**2 + L1*H + E(J0)
T2 = A2*H**4 + C21*H**3 + C22*H**2 + L2*H + E(J0)

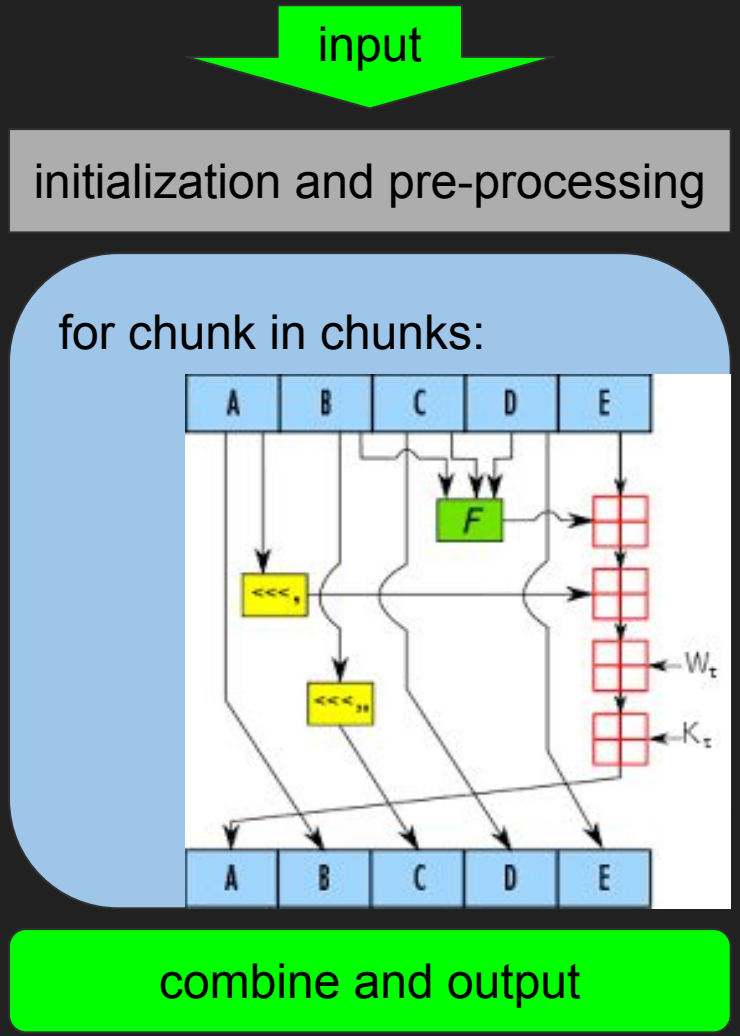
相減後只剩下 H 是未知, 用 sage 求解後帶回原式可得 E(J0)
T1-T2 = (A1-A2)*H**4 + (C11-C21)*H**3 + (C12-C22)*H**2 + (L1-L2)*H
之後就是快樂的偽造時間了~

Hash

雜湊(哈希)函數
md5、sha1、sha256、sha512、sha3
HMAC、PBKDF2、Bcrypt、Scrypt

不可逆特性來安全儲存密碼
製作簽章 憑證來驗證資料的正確性

難免會有被碰撞的一天
目前已有些可以惡意碰撞或偽造



LEA

- 長度擴展攻擊 (Length Extension Attacks)
- 條件
  - 得到 sig = hash(secret + data)
  - 已知 data
  - 已知 len(secret)
- 攻擊
  - sig = hash(secret + data + append)
  - append 可控制 皆可算出 sig
- 工具
  - hashpump (https://github.com/bwall/HashPump)

LEA

將 input message padding 後切成 chunks
初始化外部變數後每輪重新分配給內部變數
每輪內部變數運算結果疊加反饋給外部變數
迴圈結束後將外部變數 append 結果為 hash

假想有個少於 padding 1 chuck 的 message
經過正常的 hash func 得到 digest
如果想要擴充為兩個 chucks
只需將 digest 切分回去給外部變數
並且讓迴圈從第二個 chuck 開始做運算
如此即使不知道原先的 message 是什麼
也可以對其做擴充並直接算出 digest

```
# initialization variables
h0 = 0x67452301
h1 = 0xEFCDAB89
h2 = 0x98BADCFE
h3 = 0x10325476
h4 = 0xC3D2E1F0

# pre-processing
message += b'\x80'
message += b'\x00' * ((56 - len(message) % 64) % 64)
message += message_length.to_bytes(8, byteorder='big')
```

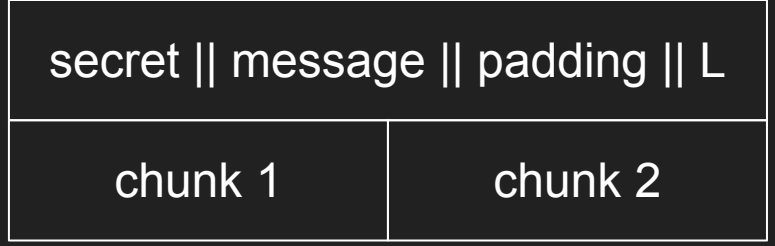
for chunk in chunks:

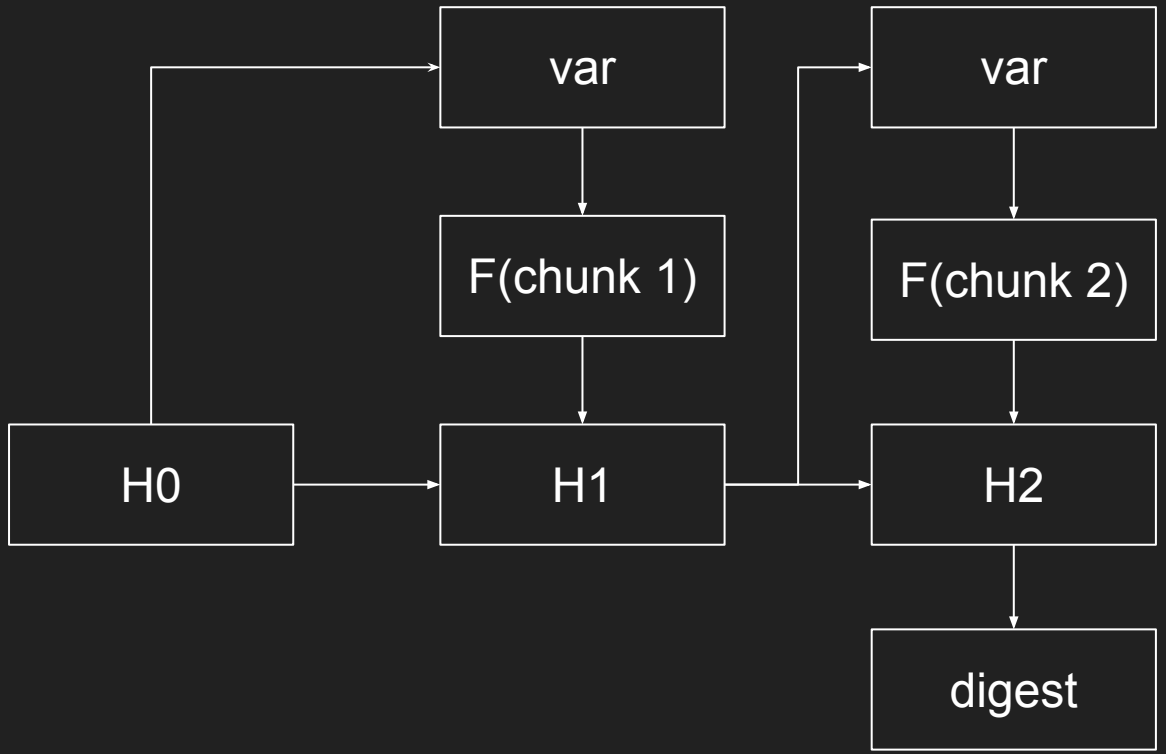
```
# initialize hash value for this chunk
a = h0
b = h1
c = h2
d = h3
e = h4
```

F

```
# add this chunk's hash to result so far
h0 = (h0 + a) & 0xffffffff
h1 = (h1 + b) & 0xffffffff
h2 = (h2 + c) & 0xffffffff
h3 = (h3 + d) & 0xffffffff
h4 = (h4 + e) & 0xffffffff
```

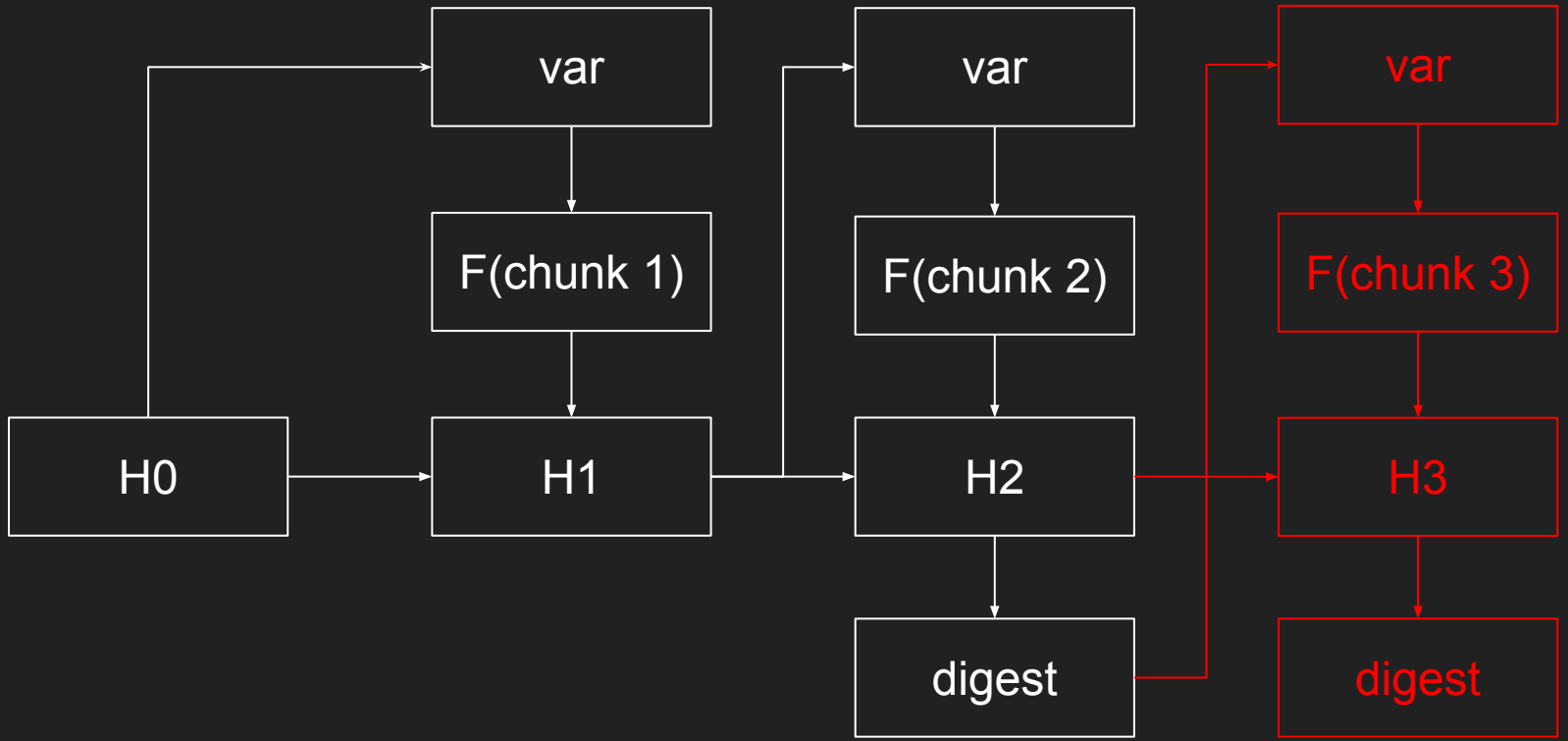
LEA


secret || message || padding || L
chunk 1
chunk 2


var
var
F(chunk 1)
F(chunk 2)
H0
H1
H2
digest

LEA

secret || message || padding || L		append data
chunk 1	chunk 2	chunk 3



HashClash

- 雜湊碰撞
- 在固定 prefix 的情況下，產生兩組 data 使得
  - data1 != data2 and hash(prefix || data1) == hash(prefix || data2)
- 工具
  - hashclash (https://github.com/killua4564/docker-hashclash)

```
[Killua4564:NTUSTISC Killua4564$ md5sum collision1.bin collision2.bin
9c96362cb7fccb439f912f5631681883  collision1.bin
9c96362cb7fccb439f912f5631681883  collision2.bin
```

```
[Killua4564:NTUSTISC Killua4564$ xxd collision1.bin
00000000: 4e54 5553 5449 5343 a525 3162 2392 70fd  NTUSTISC.%1b#.p.
00000010: b4a4 e0c4 bbfc af03 be49 e95a 06da a291  .........I.Z....
00000020: 17cb 5ba6 a96b a537 27d2 4551 de30 0745  ..[..k.7'.EQ.0.E
00000030: 6929 d4f8 4f8a 3d14 91da b14e 9ba8 e14c  i)..O.=....N...L
00000040: 03f4 500b 2410 b56a 3baf fc2c cf39 c2d4  ..P.$..j;..,.9..
00000050: c9b6 a43e 9998 db9a 6b54 abaf 7c1c 3463  ...>....kT..|.4c
00000060: 6826 3b02 f00c ccd3 7a29 c14f 1835 44d4  h&;.....z).O.5D.
00000070: d68b dac0 80db 09e5 aff8 b339 54da 0f2e  ...........9T...
```

```
[Killua4564:NTUSTISC Killua4564$ xxd collision2.bin
00000000: 4e54 5553 5449 5343 a526 3162 2392 70fd  NTUSTISC.&1b#.p.
00000010: b4a4 e0c4 bbfc af03 be49 e95a 06da a291  .........I.Z....
00000020: 17cb 5ba6 a96b a537 27d2 4551 de30 0745  ..[..k.7'.EQ.0.E
00000030: 6929 d4f8 4f8a 3d14 91da b14e 9ba8 e14c  i)..O.=....N...L
00000040: 03f4 500b 2410 b56a 3bae fc2c cf39 c2d4  ..P.$..j;..,.9..
00000050: c9b6 a43e 9998 db9a 6b54 abaf 7c1c 3463  ...>....kT..|.4c
00000060: 6826 3b02 f00c ccd3 7a29 c14f 1835 44d4  h&;.....z).O.5D.
00000070: d68b dac0 80db 09e5 aff8 b339 54da 0f2e  ...........9T...
```

After Lecture…

- ARX (Addition Rotation XOR)
- Simon Block Cipher
- PCBC mode
- Affine Cipher
- Vigenère Cipher
- Rail Fence / Scytale

THE END

Thanks for listening