

# Crypto

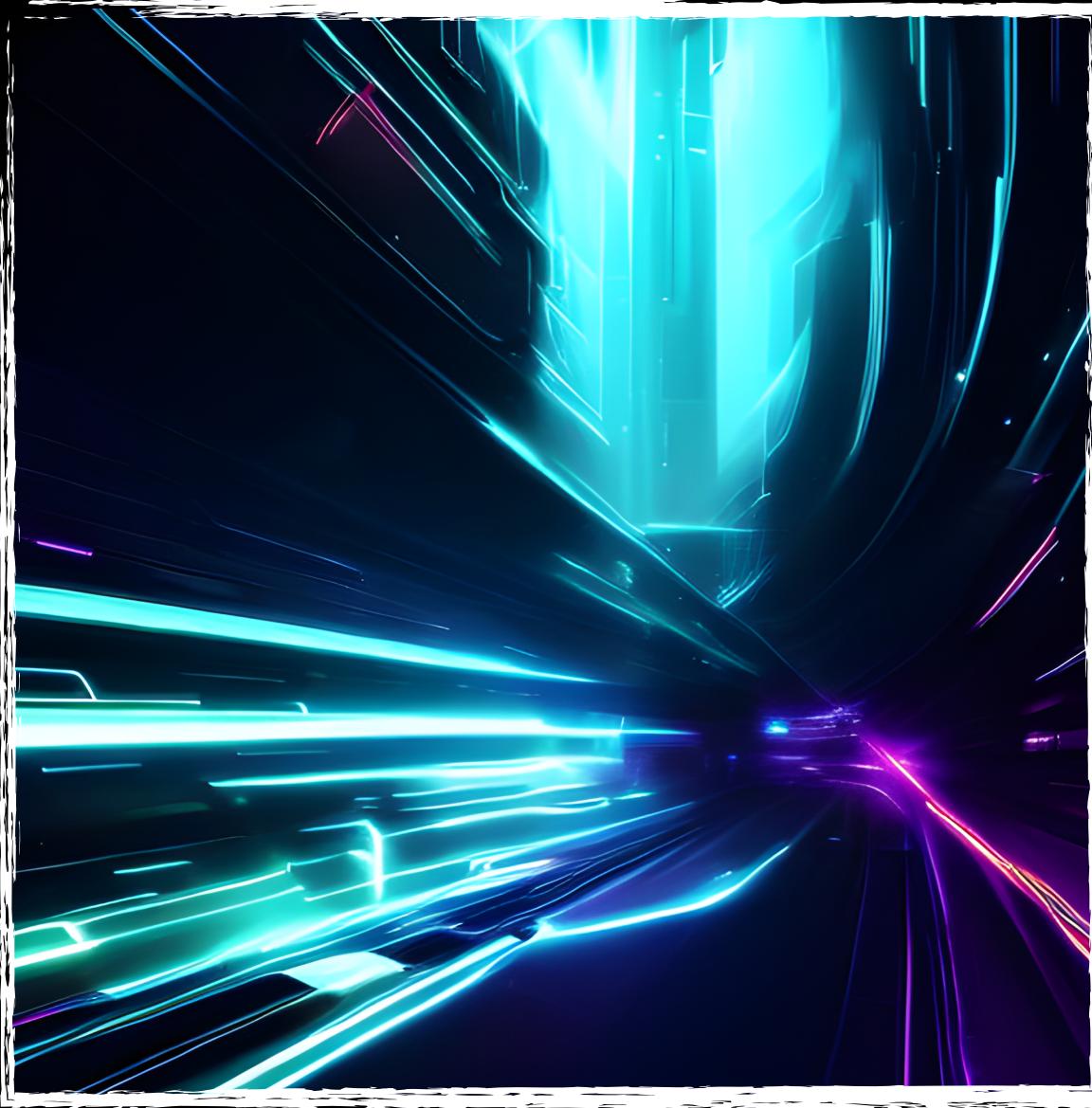
Curious @ Taiwan Holy Young

2023.07.18 2023.07.19

## >\_ 目錄

- ▶ 對稱密碼
- ▶ 偽隨機數
- ▶ 雜湊函數
- ▶ RSA

# >\_ WhoAmI



- ▶ ID : Curious
- ▶ DC : curious\_lucifer
- ▶ SCIST 3rd 總召 / LoTuX CTF 創辦人

## >\_ 資安倫理宣傳

本課程目的在提升學員對資訊安全之認識及資安實務能力，深刻體認到資安的重要性！所有課程學習內容不得從事非法攻擊或違法行為，所有非法行為將受法律規範，提醒學員不要以身試險。

>\_ Lab

Lab : <http://35.201.230.193/>

Flag Format : FLAG{.\*}

Discord : <https://discord.gg/jTV6dA4PxP>

## >\_ 先備知識

Linux : <https://tg.pe/x8PS>, <https://tg.pe/xqyp>

Python : <https://tg.pe/x5yC>, <https://tg.pe/xazN>

Pwntools : <https://tg.pe/xr1E>

## >\_ 虛擬機

x86 安裝說明 : [https://hackmd.io/@akvo-fajro/VirtualBox\\_Guide](https://hackmd.io/@akvo-fajro/VirtualBox_Guide)

ARM 安裝說明 : [https://hackmd.io/@akvo-fajro/UTM\\_Guide](https://hackmd.io/@akvo-fajro/UTM_Guide)

>\_ Pwntools

安裝說明 : [https://hackmd.io/@akvo-fajro/Pwntools\\_Installation\\_Guide](https://hackmd.io/@akvo-fajro/Pwntools_Installation_Guide)

>\_ Crypto Ubuntu

Download : <https://tg.pe/xmMP>

User : user

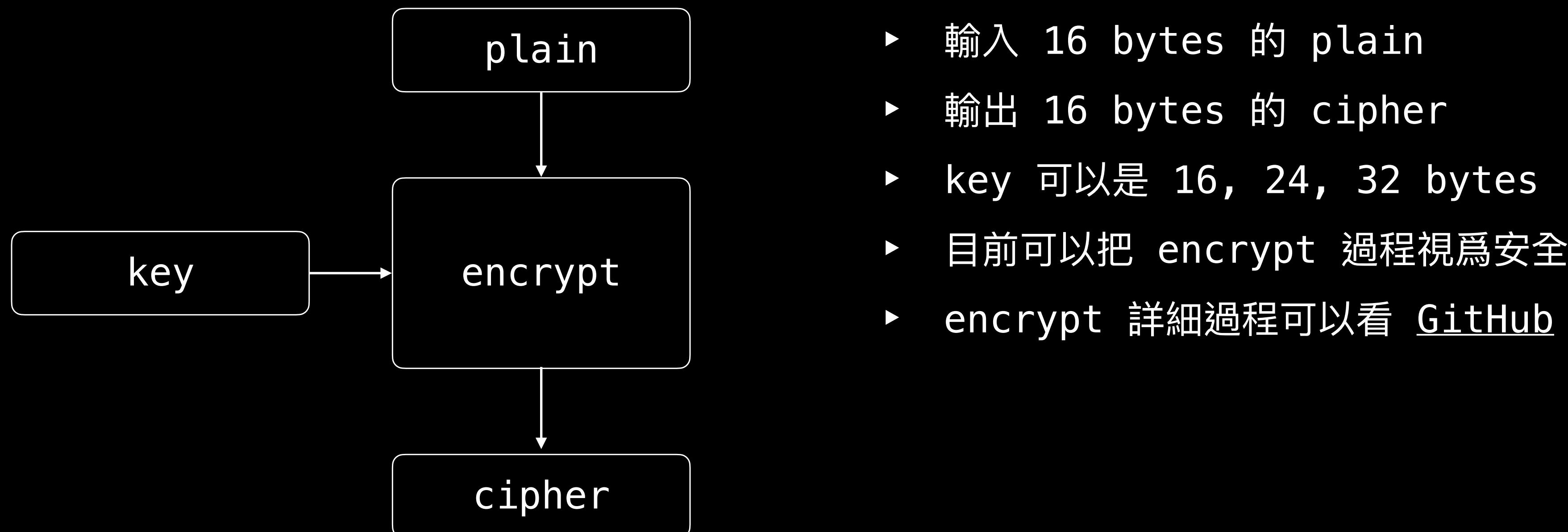
Password : user

# 對稱密碼

# >\_ AES

## Advance Encryption Standard

- ▶ 加密
- ▶ 解密



# >\_ AES

## Advance Encryption Standard

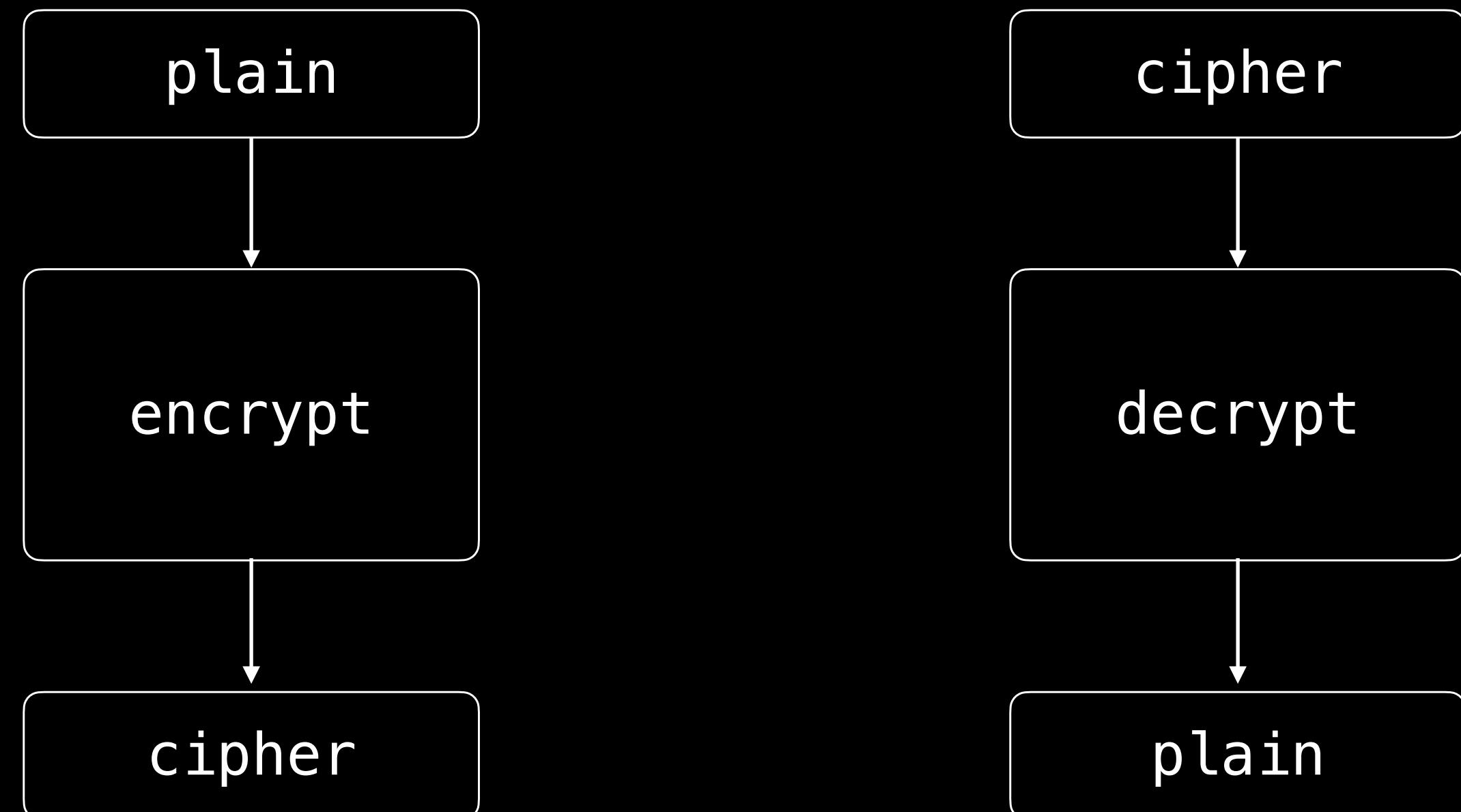
- ▶ 加密
- ▶ 解密



# >\_ AES

## Advance Encryption Standard

- ▶ 因為加密或解密的過程都可以視為安全，所以加密和解密的過程可以簡化成



plain & cipher 都是 16 bytes

## >\_ AES

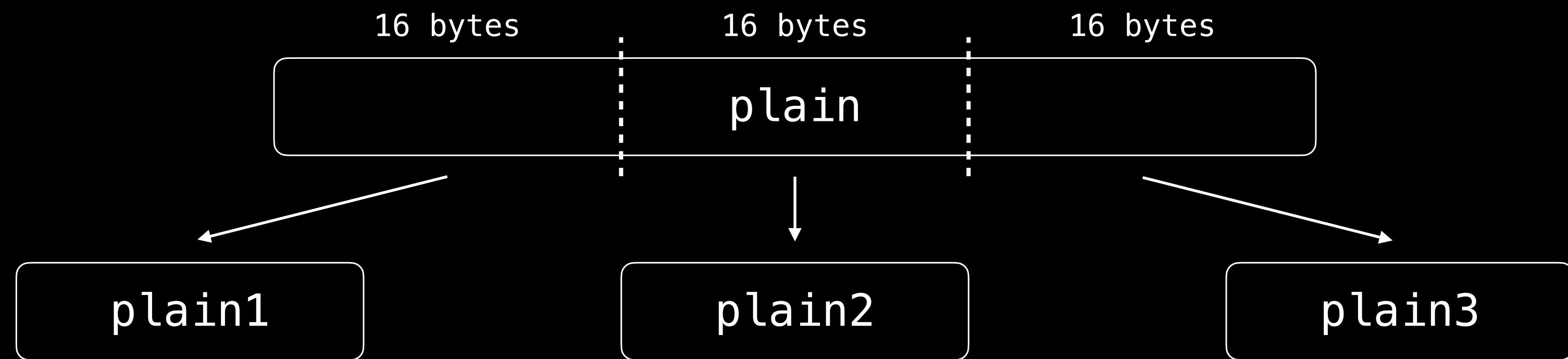
### Advance Encryption Standard

- ▶ 雖然說單純的 AES 是安全的，但是它只能加密 16 bytes，所以就有 **Padding** 和 **Block Cipher Mode** 的機制

AES ECB Mode

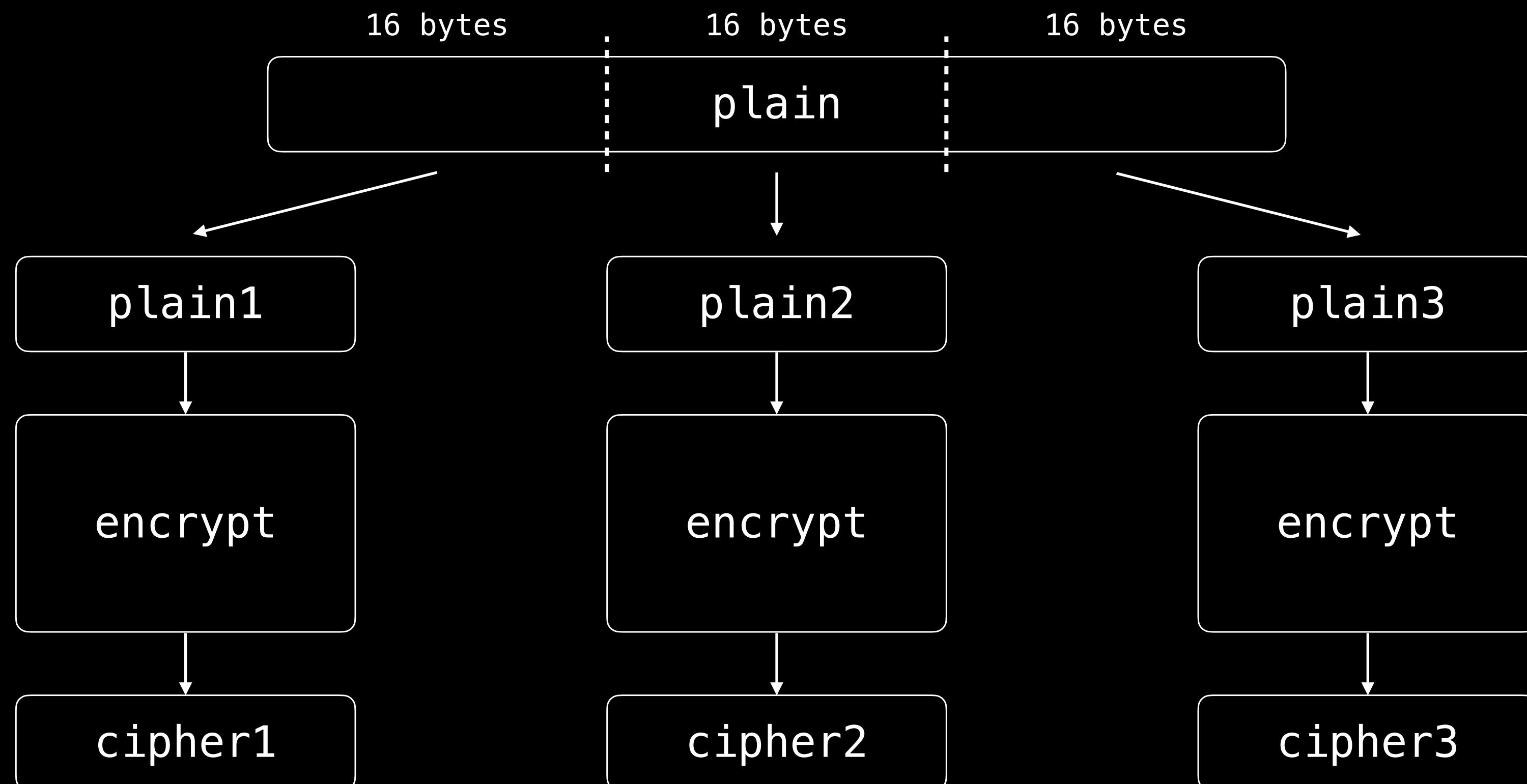
## >\_ Block Cipher Mode – ECB

- ▶ 如果我需要用 ECB Mode 加密一個 48 bytes 的明文



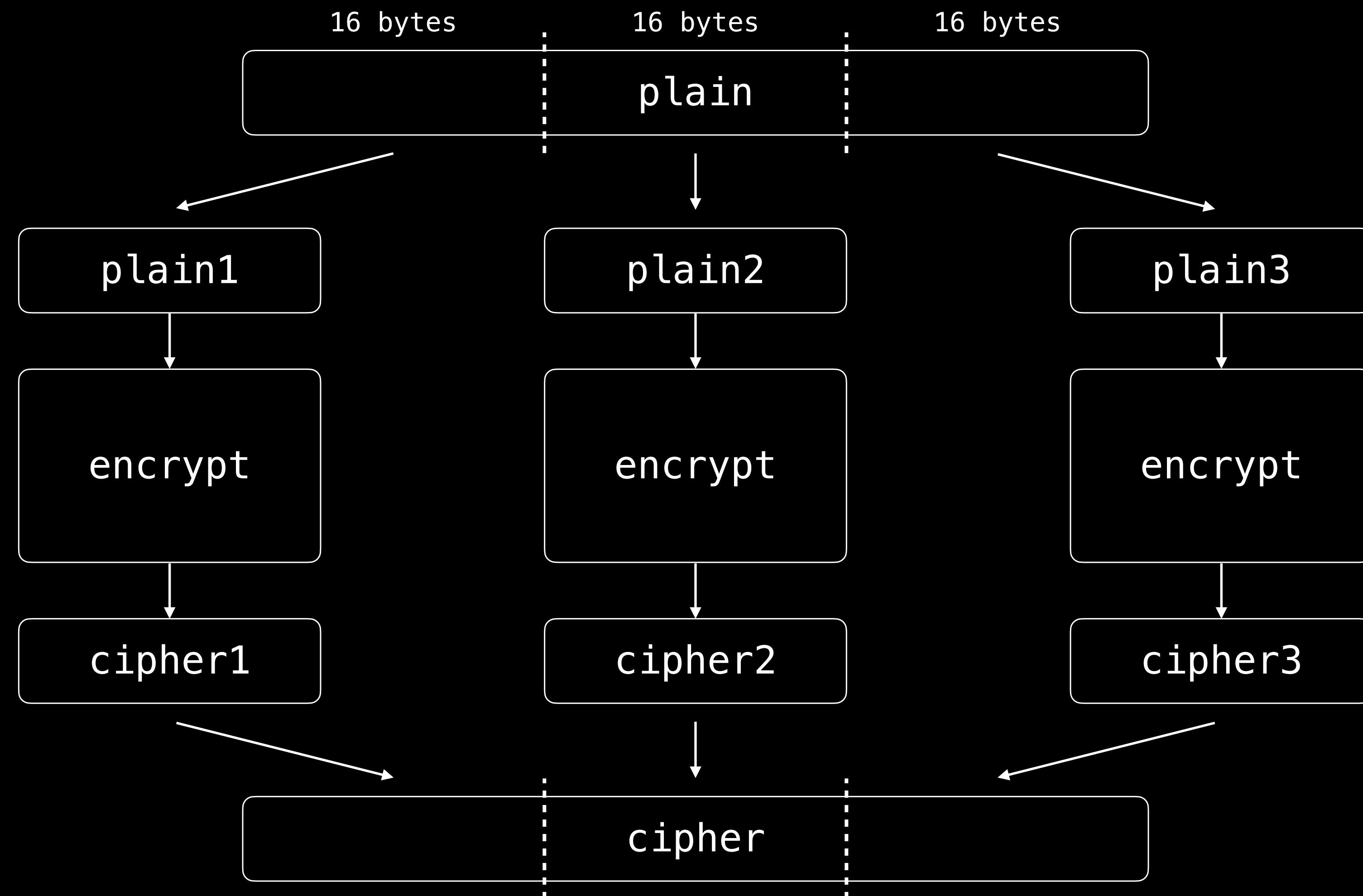
## >\_ Block Cipher Mode – ECB

- ▶ 如果我需要用 ECB Mode 加密一個 48 bytes 的明文



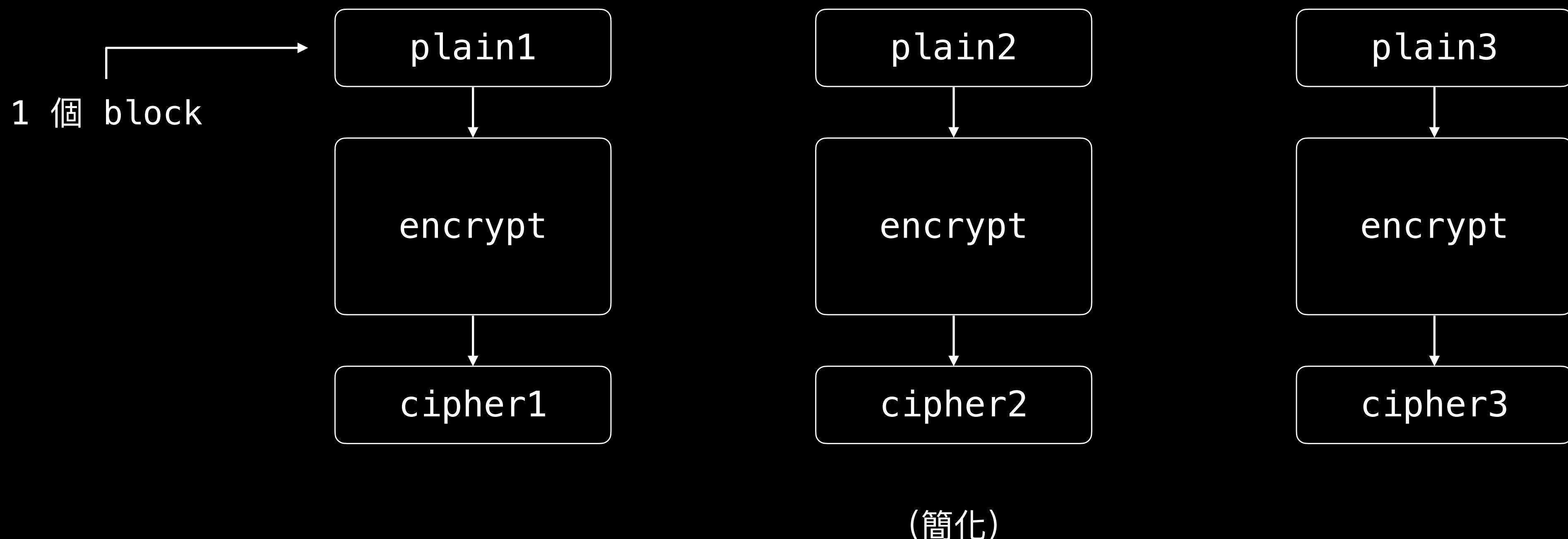
# >\_ Block Cipher Mode – ECB

- ▶ 如果我需要用 ECB Mode 加密一個 48 bytes 的明文



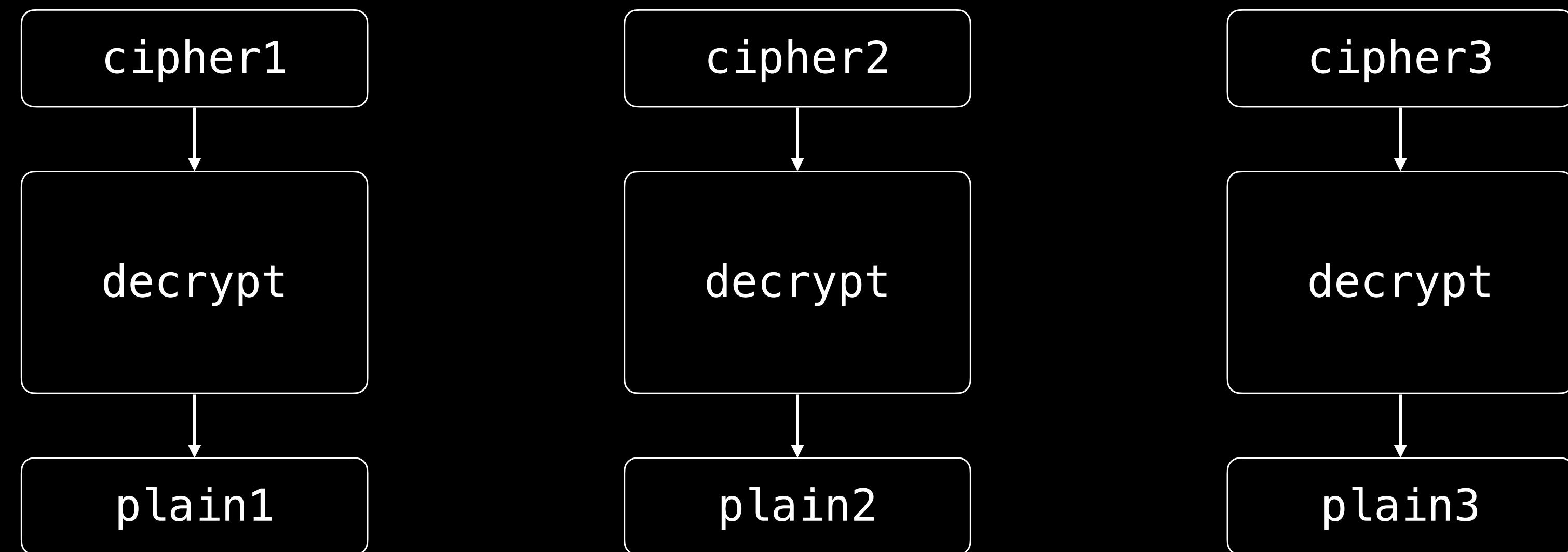
## >\_ Block Cipher Mode – ECB

- ▶ 如果我需要用 ECB Mode 加密一個 48 bytes 的明文



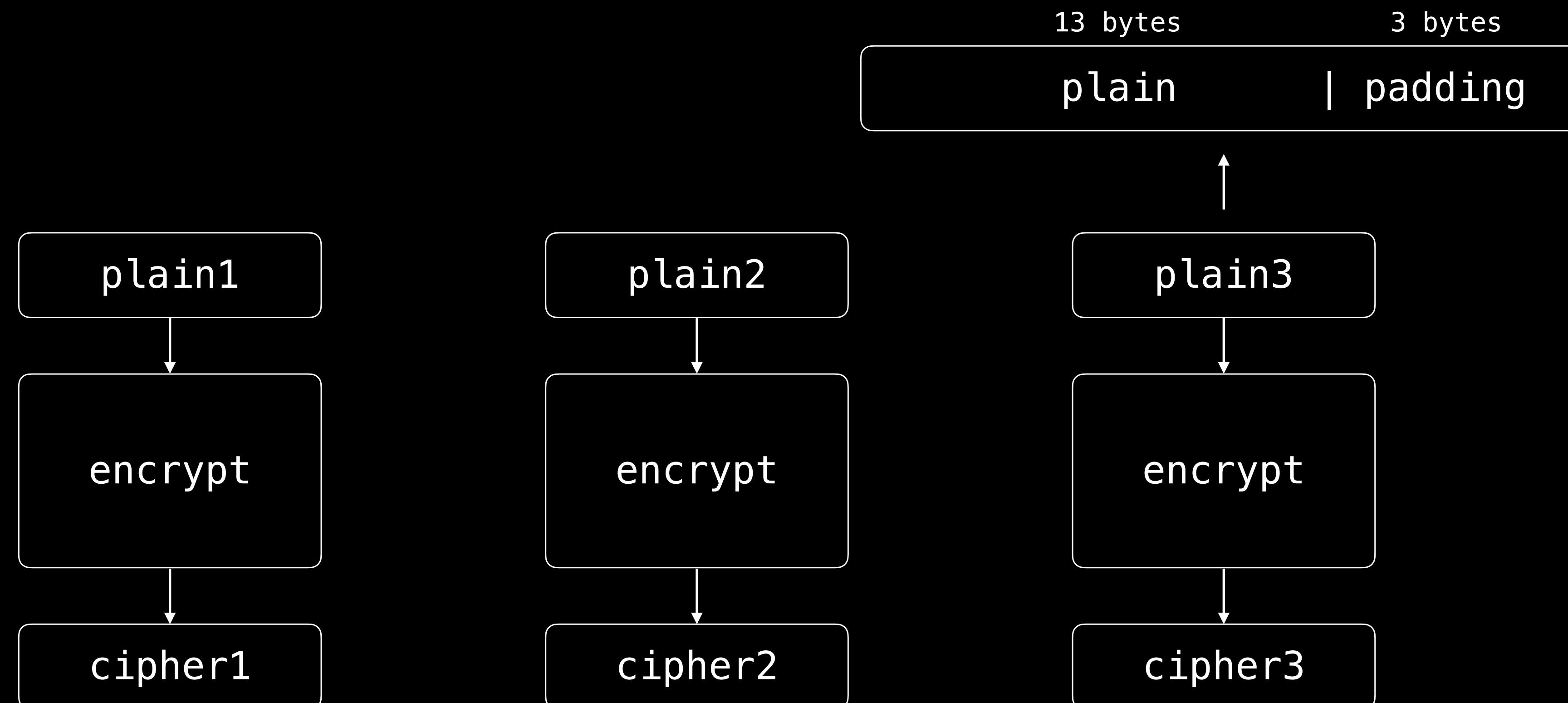
## >\_ Block Cipher Mode – ECB

- ▶ 如果我需要用 ECB Mode 解密一個 48 bytes 的明文



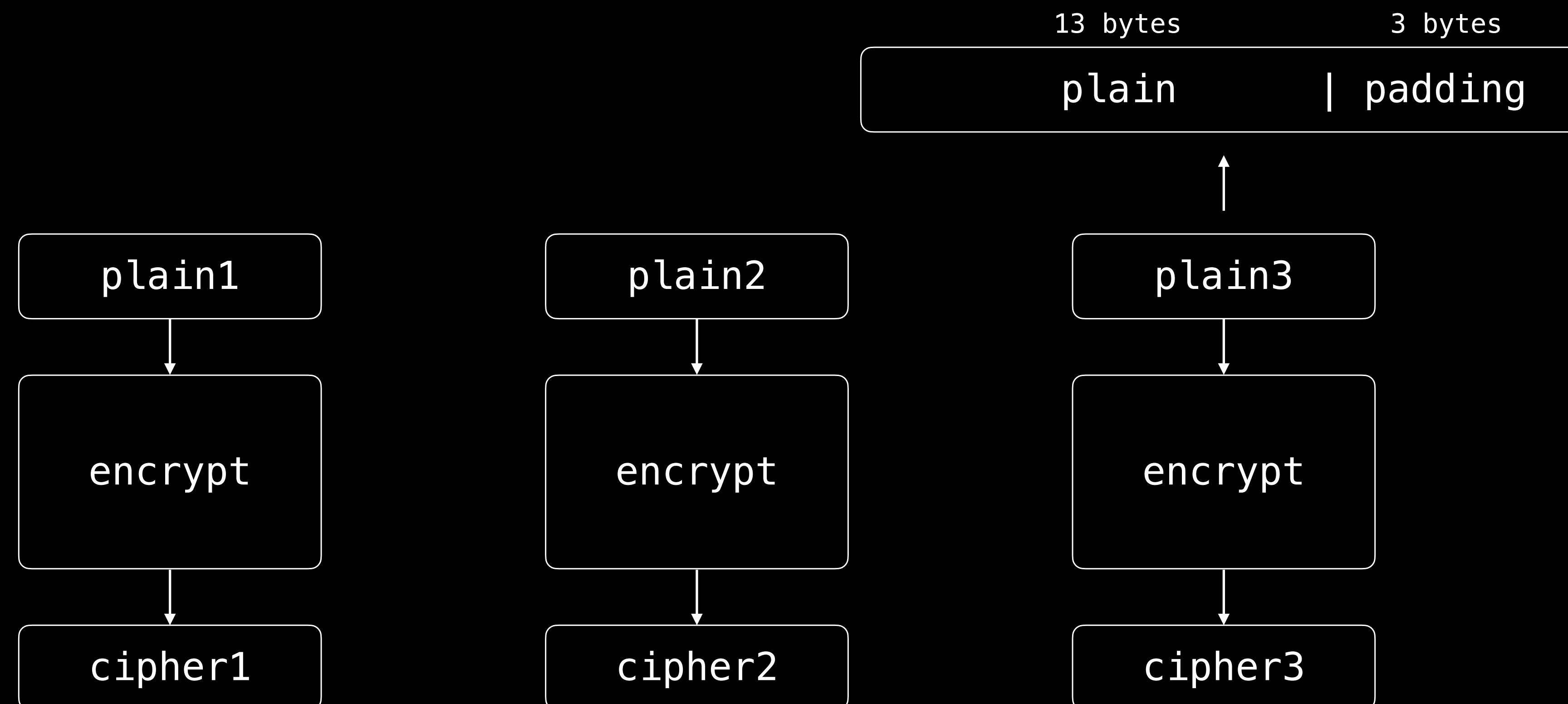
# >\_ Padding

- ▶ 如果我需要用 ECB Mode 加密一個 45 bytes 的明文



# >\_ Padding

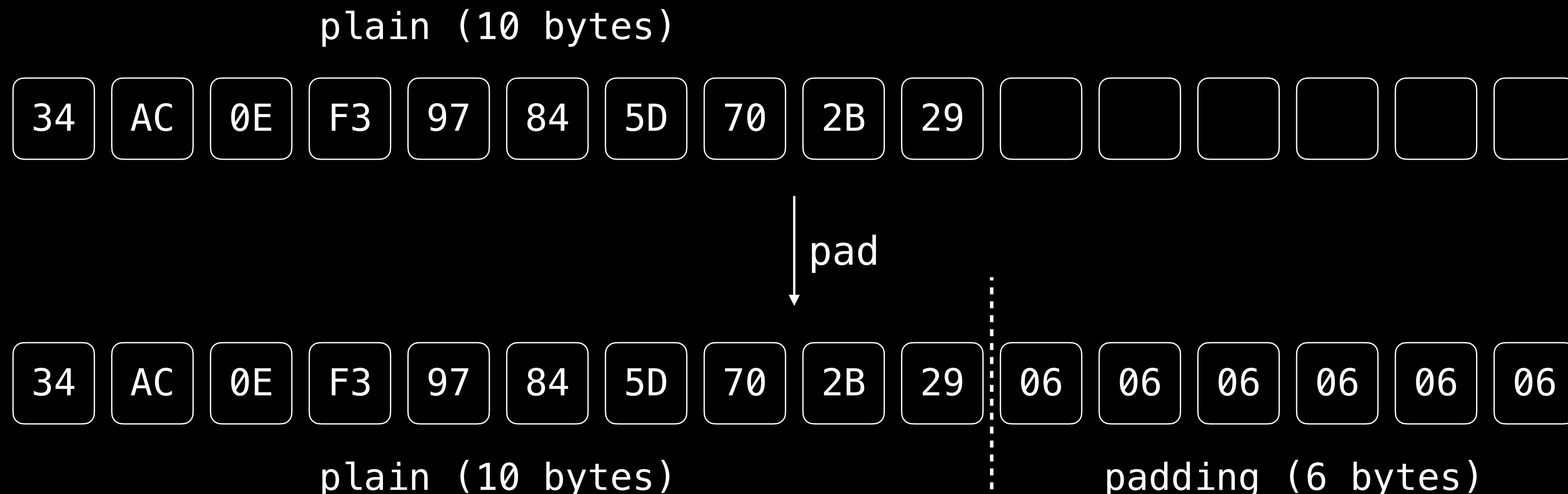
- ▶ 如果我需要用 ECB Mode 加密一個 45 bytes 的明文
- ▶ 解密時會先做正常的 ECB Mode Decryption，再把 Padding 去掉



# > Padding

# PKCS#7

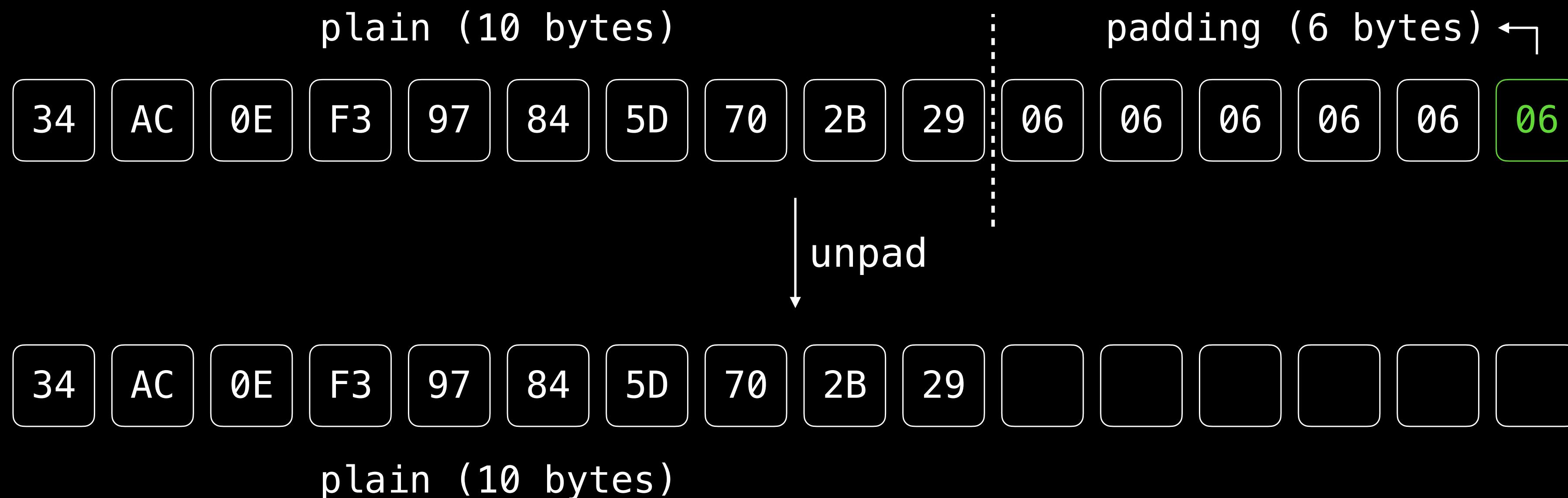
- pad : 最後一個 block 缺 n 個 bytes 就填 n 個 bytes([n])
  - unpad : 如果最後一個 bytes 是 n，則把最後 n 個 bytes 拿掉



# >\_ Padding

## PKCS#7

- ▶ pad : 最後一個 block 缺 n 個 bytes 就填 n 個 bytes([n])
- ▶ unpad : 如果最後一個 bytes 是 n，則把最後 n 個 bytes 拿掉



## >\_ Padding

### PKCS#7

- ▶ pad : 最後一個 block 缺 n 個 bytes 就填 n 個 bytes([n])
- ▶ unpad : 如果最後一個 bytes 是 n，則把最後 n 個 bytes 拿掉

注意：

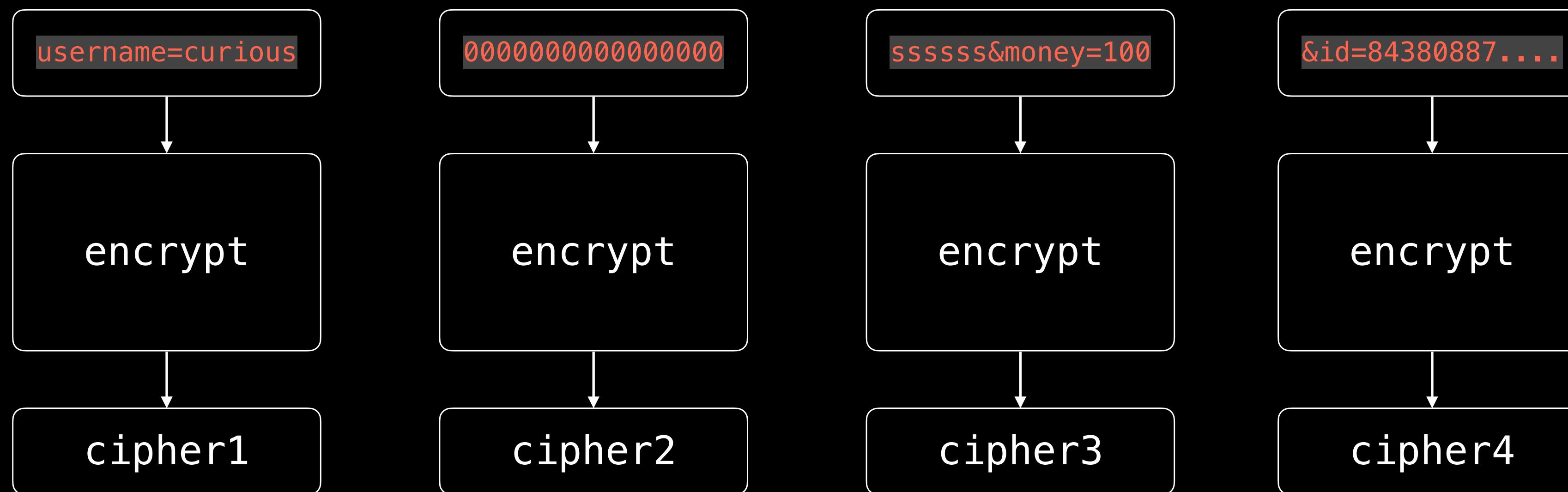
- ▶ 如果 plain 的長度剛剛好是 16 的倍數，那就最後一個 block 就是缺 16 個 bytes
- ▶ 在 unpad 時會檢查 n 是否  $0 < n \leq 16$ ，且檢查最後 n bytes 是否都是 bytes([n])

AES ECB Mode - Cut & Paste

# >\_ Cut & Paste

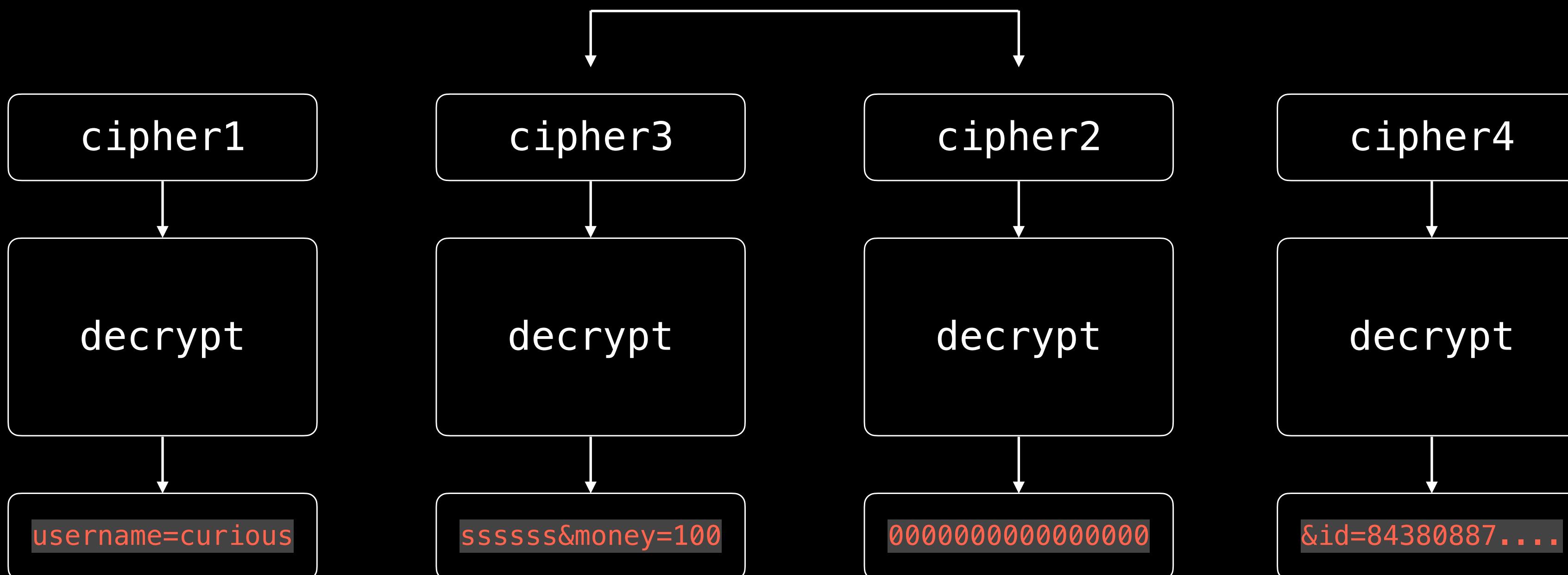
- 如果某一個系統它會把使用者相關資訊用 AES ECB 加密過後給使用者，然後使用者可以用這個 cipher 給系統驗證他的身份和資料，則

plain : **username=curious0000000000000000ssssss&money=100&id=84380887**



# > Cut & Paste

- ▶ 如果某一個系統它會把使用者相關資訊用 AES ECB 加密過後給使用者可以用這個 cipher 給系統驗證他的身份和資料，則



我好有錢

>\_ Cut & Paste

Lab : Cut & Paste Ø

AES CBC Mode

## >\_ XOR

- ▶  $19 \wedge 8 = 0b10011 \wedge 0b1000 = 0b11011 = 27$
- ▶  $'a' \oplus '?' = 97 \wedge 63 = 94 = '^'$
- ▶  $'curious' \oplus 'crypto!' = '\x00\x07\x0b\x19\x1b\x1aR'$

	1	0
1	0	1
0	1	0

# >\_ XOR

- ▶  $19 \wedge 8 = 0b10011 \wedge 0b1000 = 0b11011 = 27$
- ▶  $'a' \oplus '?' = 97 \wedge 63 = 94 = '^'$
- ▶  $'curious' \oplus 'crypto!' = '\x00\x07\x0b\x19\x1b\x1aR'$

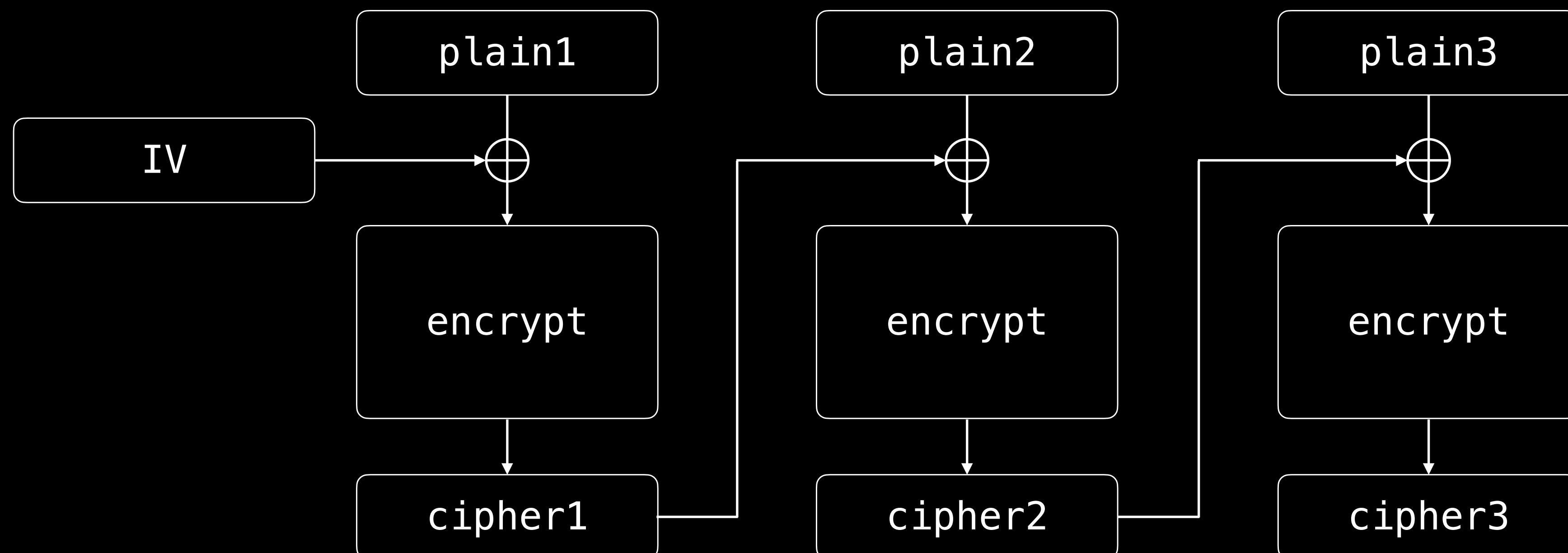
	1	0
1	0	1
0	1	0

## 特性

- ▶  $a \oplus a = 0$
- ▶  $a \oplus 0 = a$
- ▶  $a \oplus b = b \oplus a$
- ▶  $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

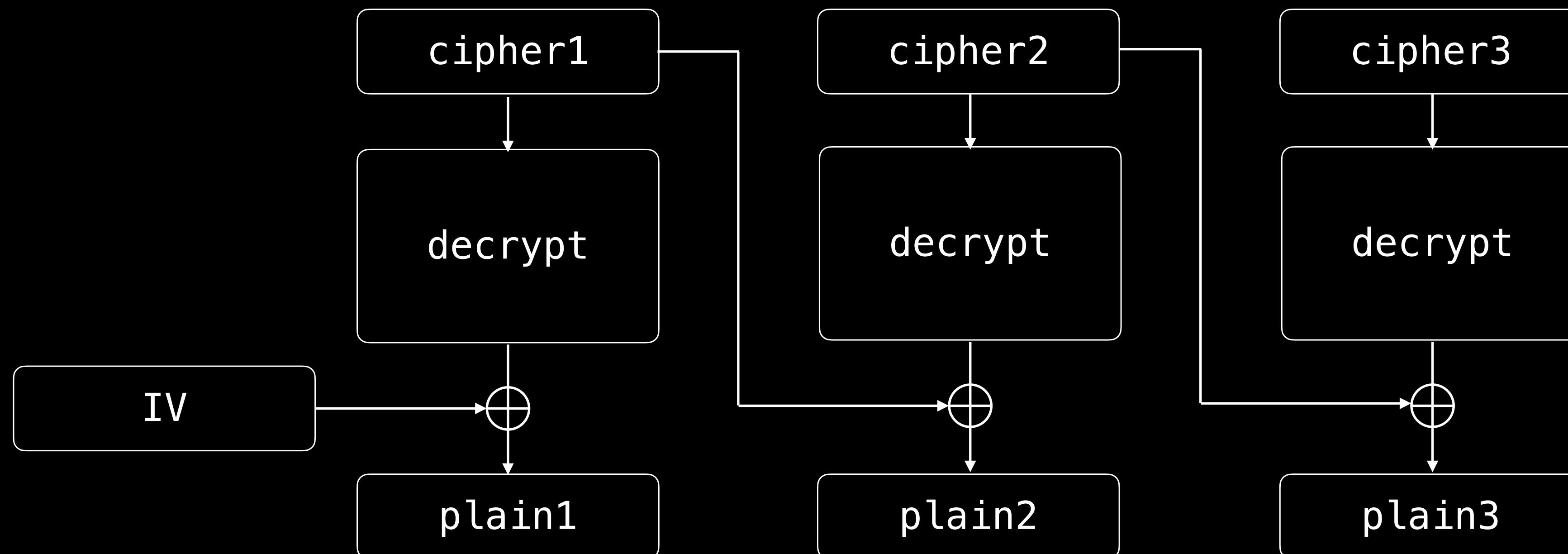
# >\_ Block Cipher Mode - CBC

- ▶ 加密
- ▶ 解密



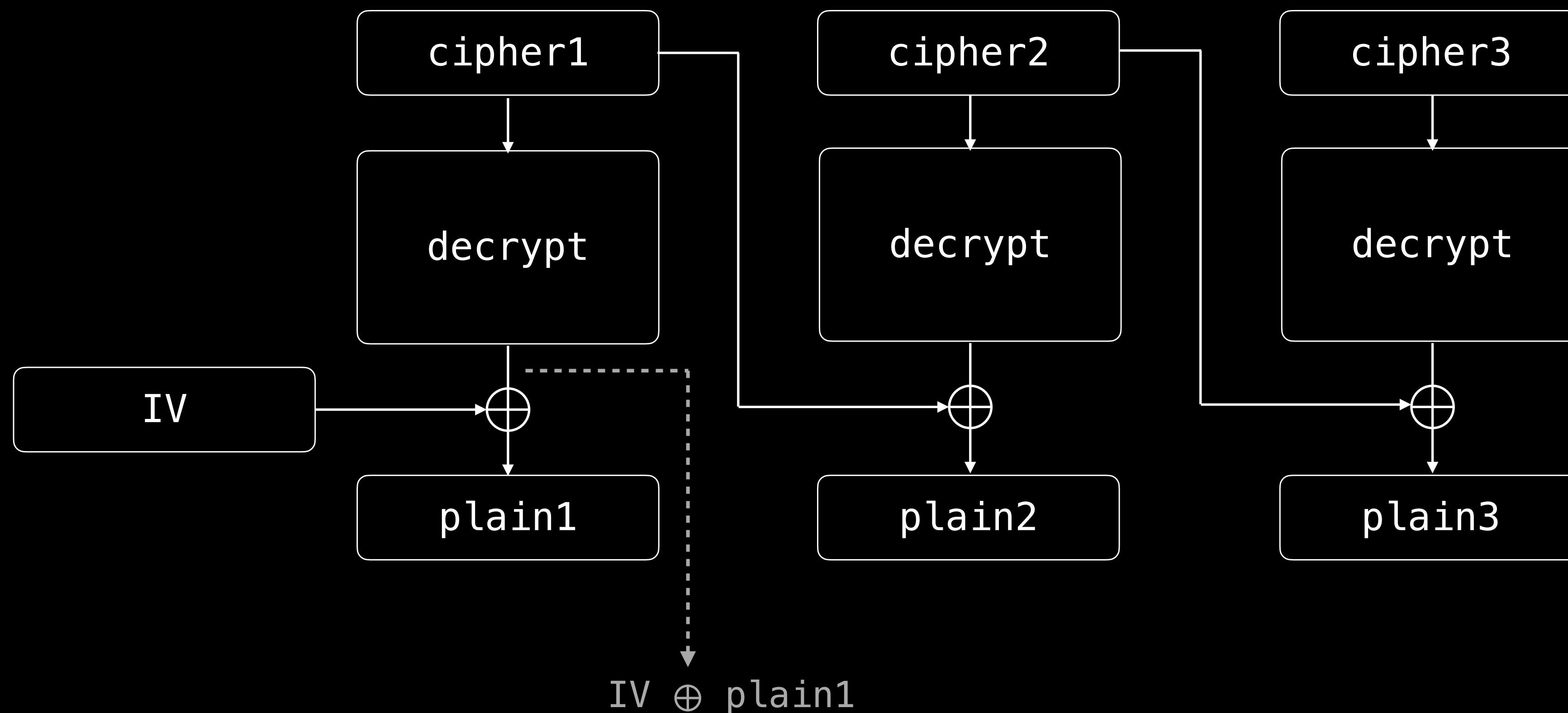
# >\_ Block Cipher Mode - CBC

- ▶ 加密
- ▶ 解密



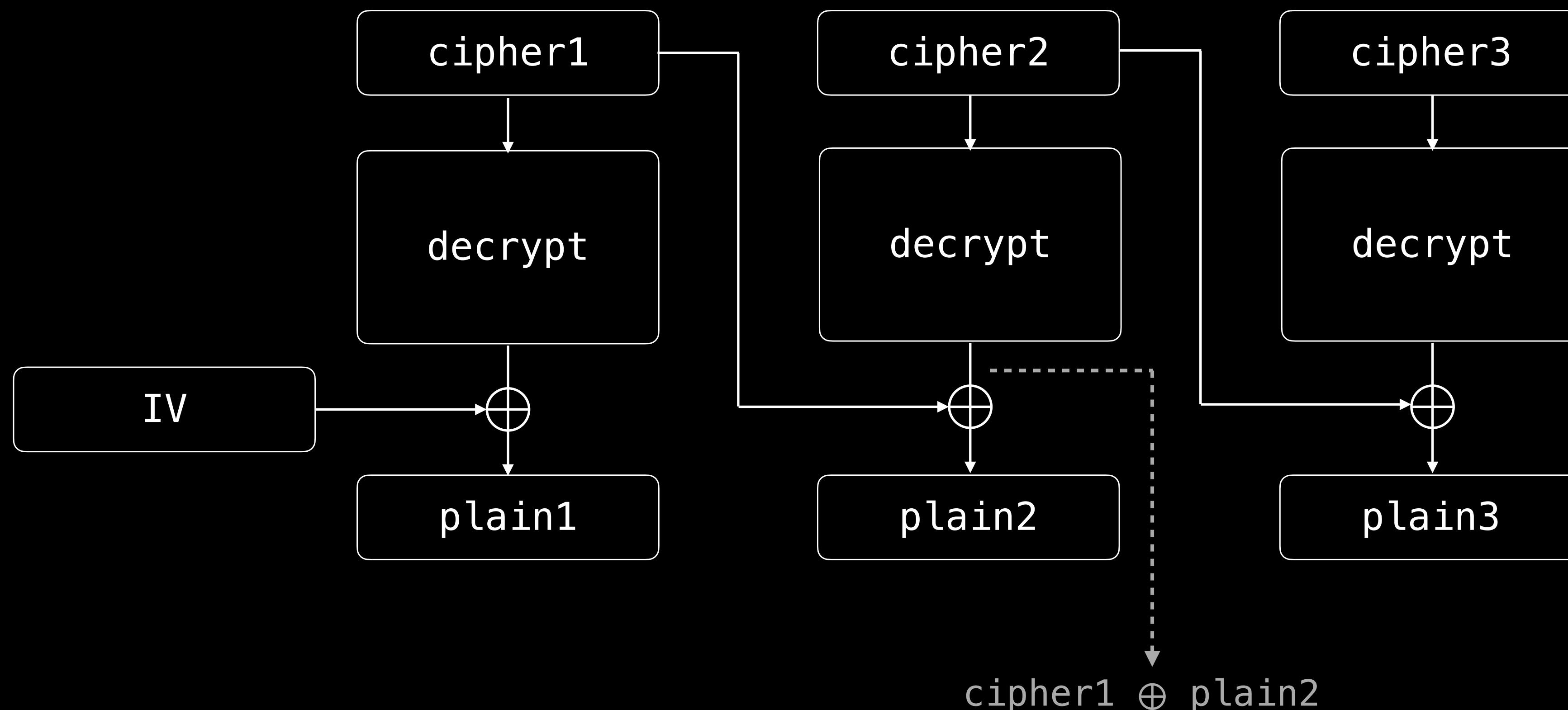
# >\_ Block Cipher Mode - CBC

- ▶ 加密
- ▶ 解密



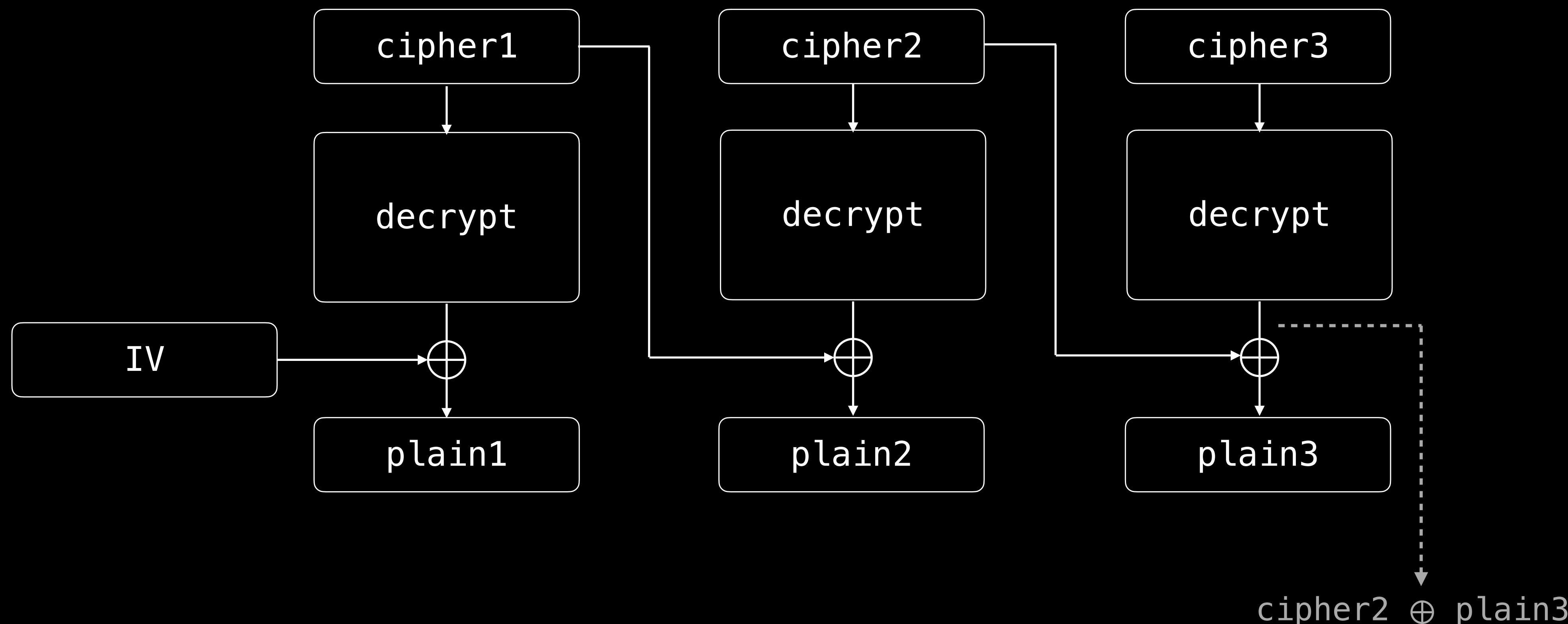
# >\_ Block Cipher Mode – CBC

- ▶ 加密
- ▶ 解密



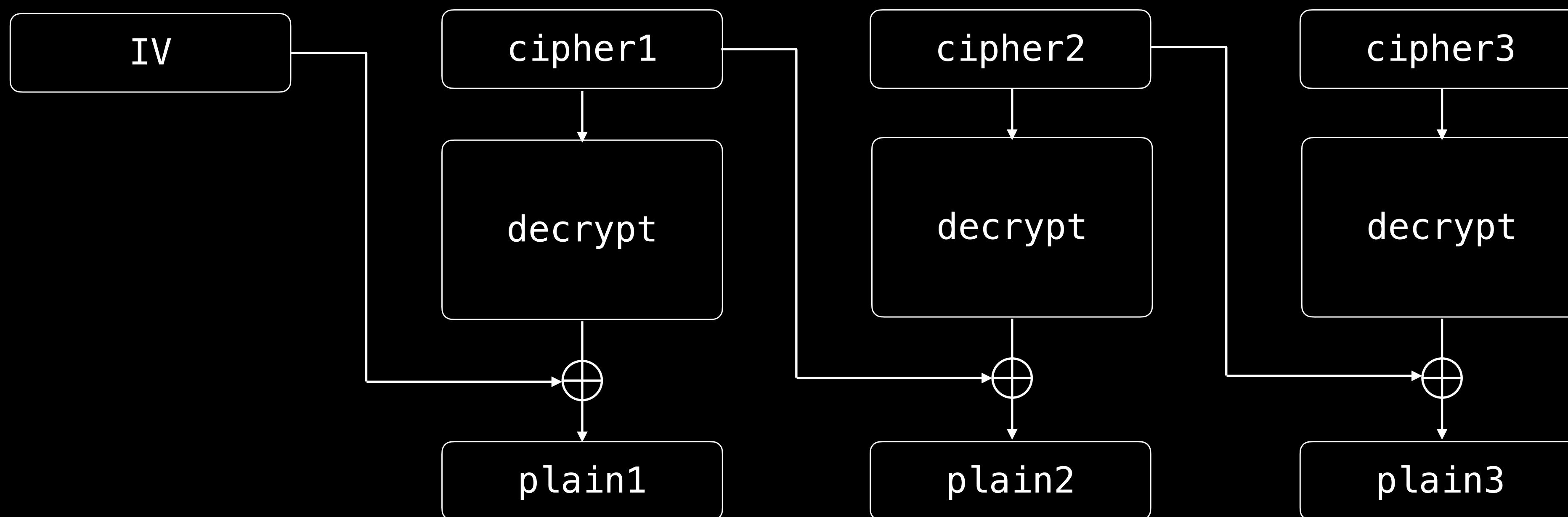
# >\_ Block Cipher Mode – CBC

- ▶ 加密
- ▶ 解密



# >\_ Block Cipher Mode - CBC

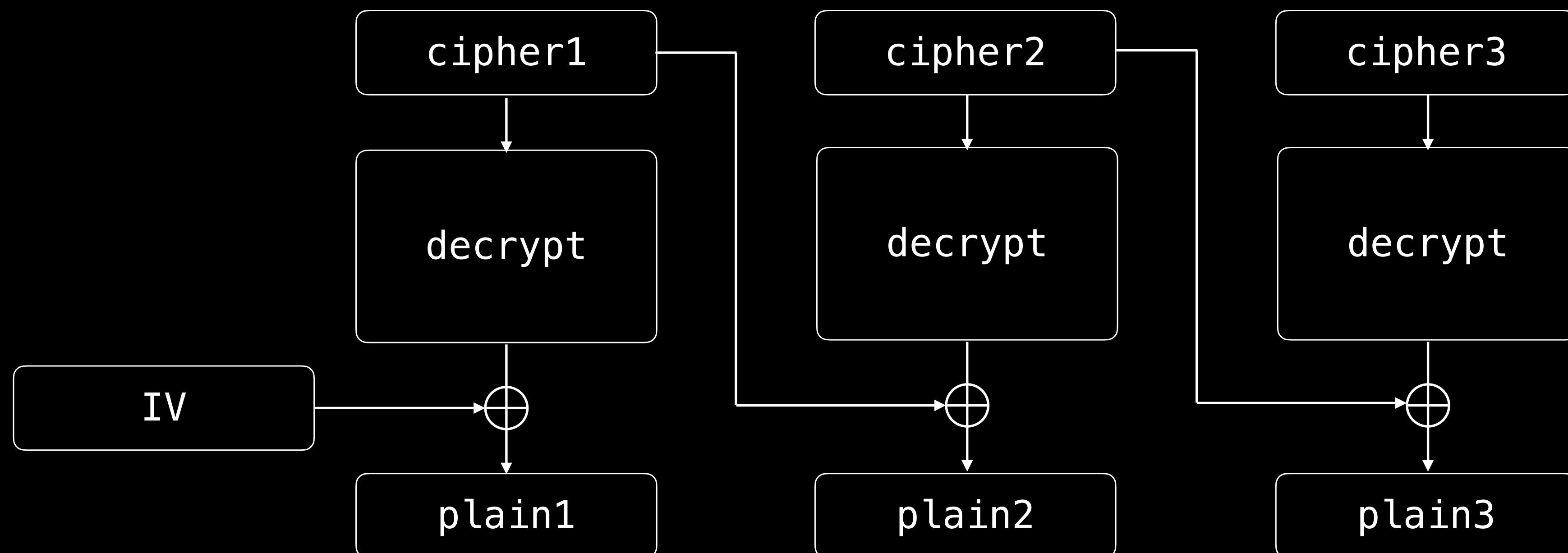
- ▶ 加密
- ▶ 解密



# AES CBC Mode – Bit Flipping

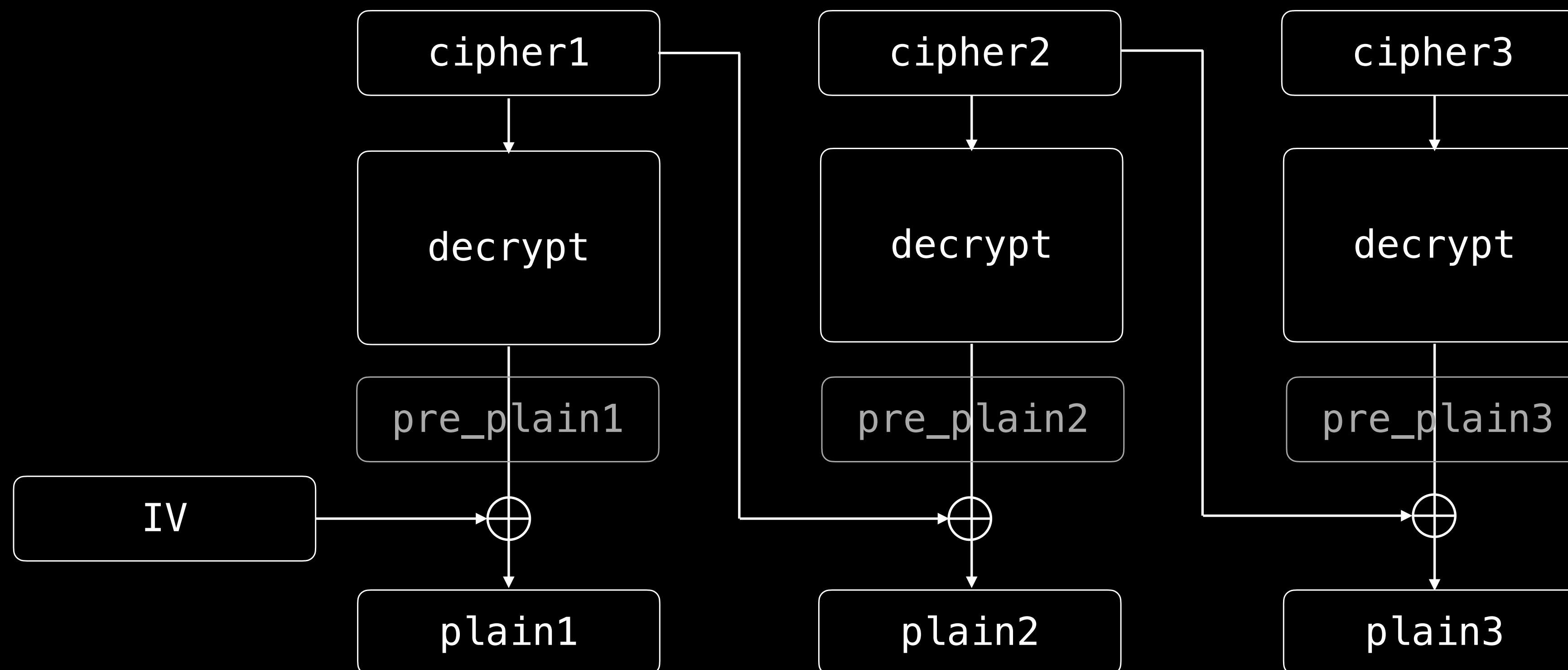
# >\_ Bit Flipping

- ▶ 回憶一下 CBC Mode 是怎麼解密的



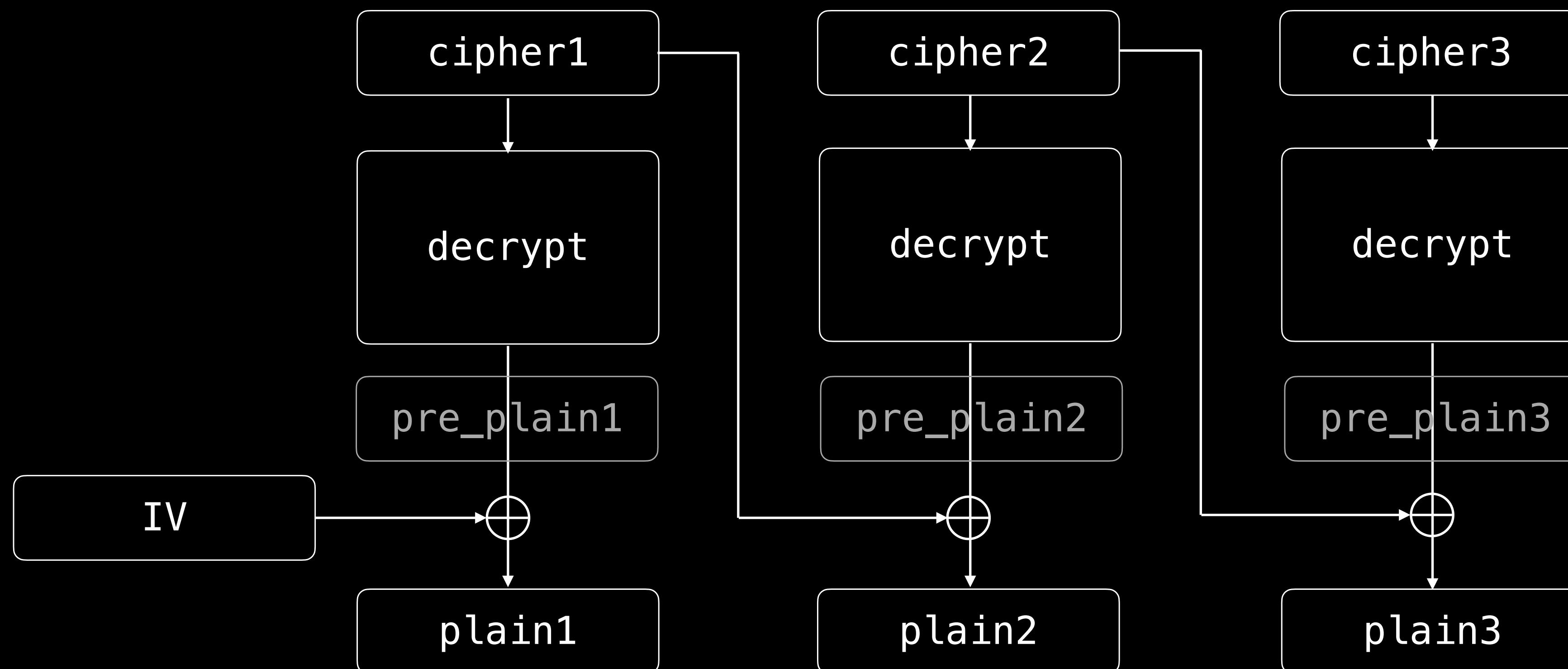
## >\_ Bit Flipping

- ▶ 我們把 cipher1 decrypt 出來，還沒跟 IV xor 的 bytes 叫做 pre\_plain1
- ▶ 同樣的 cipher2 decrypt 出來，還沒跟 cipher1 xor 的 bytes 叫做 pre\_plain2



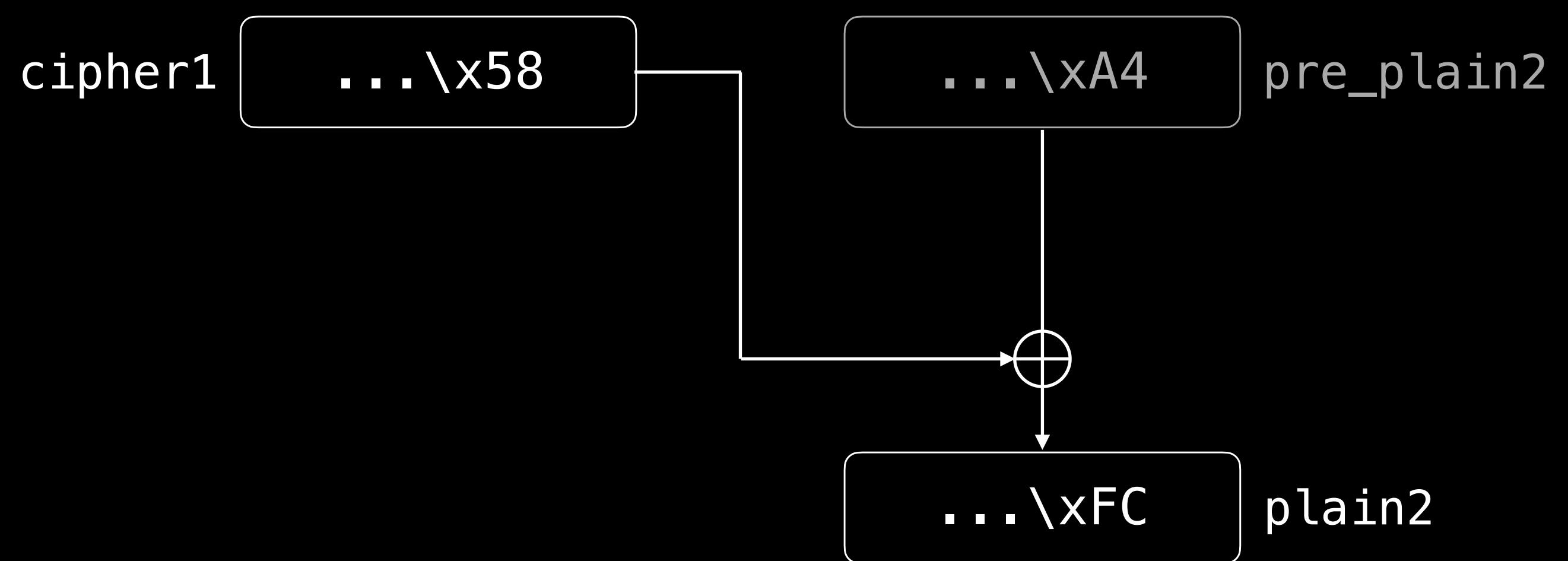
## >\_ Bit Flipping

- ▶ 如果我把 cipher1 的最後一個 bytes 改了，那 plain1 會整個亂掉，plain2 因為 pre\_plain2 並沒有改變，所以 plain2 只有最後一個 bytes 改了，plain3 沒有受到任何影響



## >\_ Bit Flipping

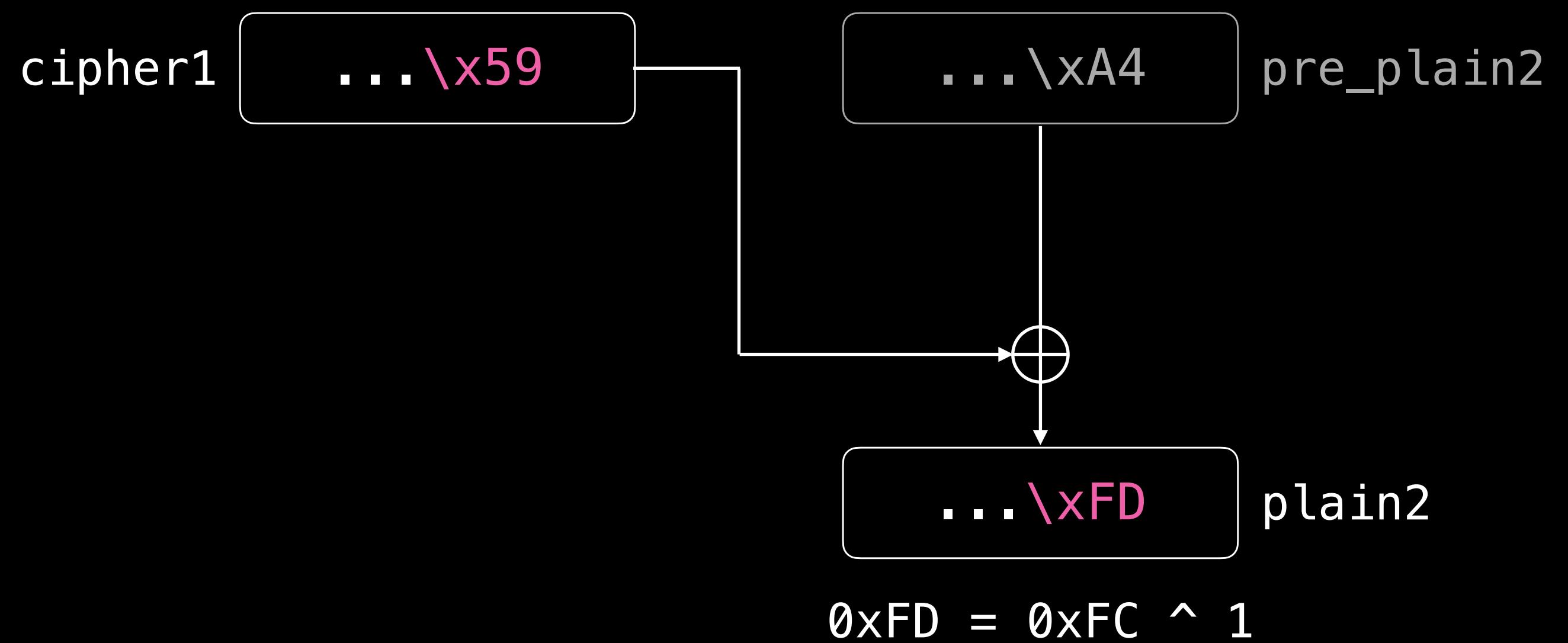
- ▶ 把不重要的東西去掉後，分析一下 plain2 的改變跟 cipher1 的改變有什麼關係



## >\_ Bit Flipping

- ▶ 把不重要的東西去掉後，分析一下 plain2 的改變跟 cipher1 的改變有什麼關係

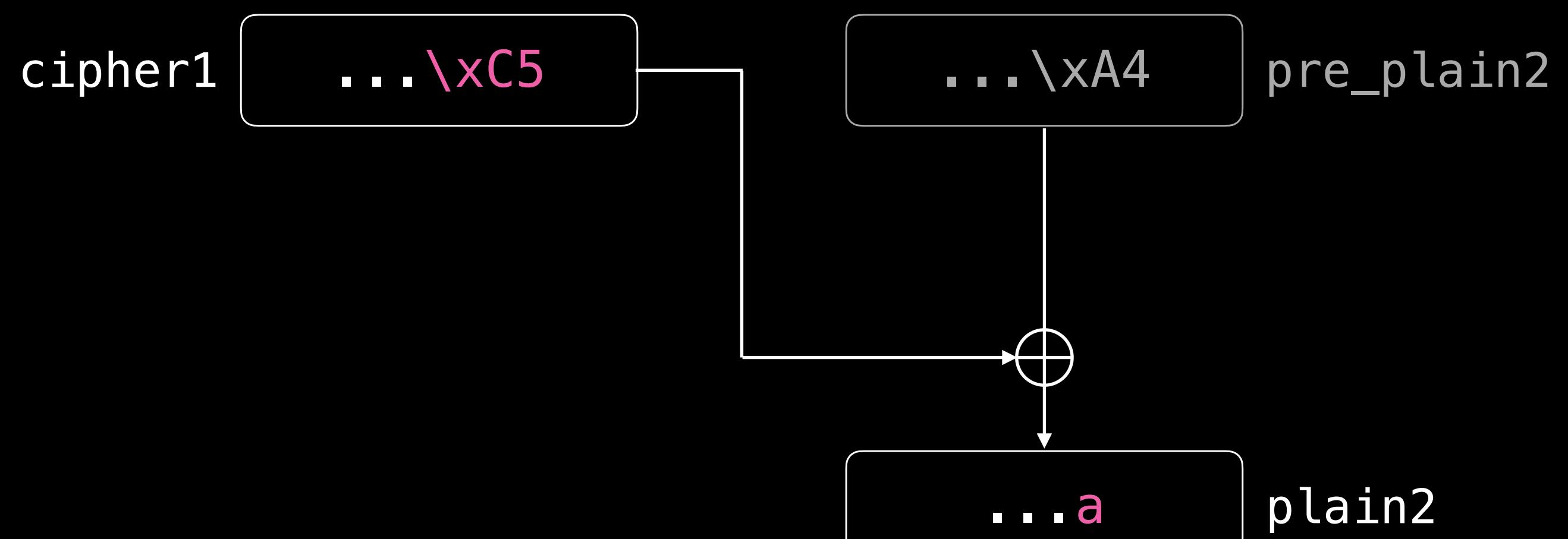
$$0x59 = 0x58 \wedge 1$$



## >\_ Bit Flipping

- ▶ 把不重要的東西去掉後，分析一下 plain2 的改變跟 cipher1 的改變有什麼關係

$0xC5 = 0x58 \wedge 0x9D$

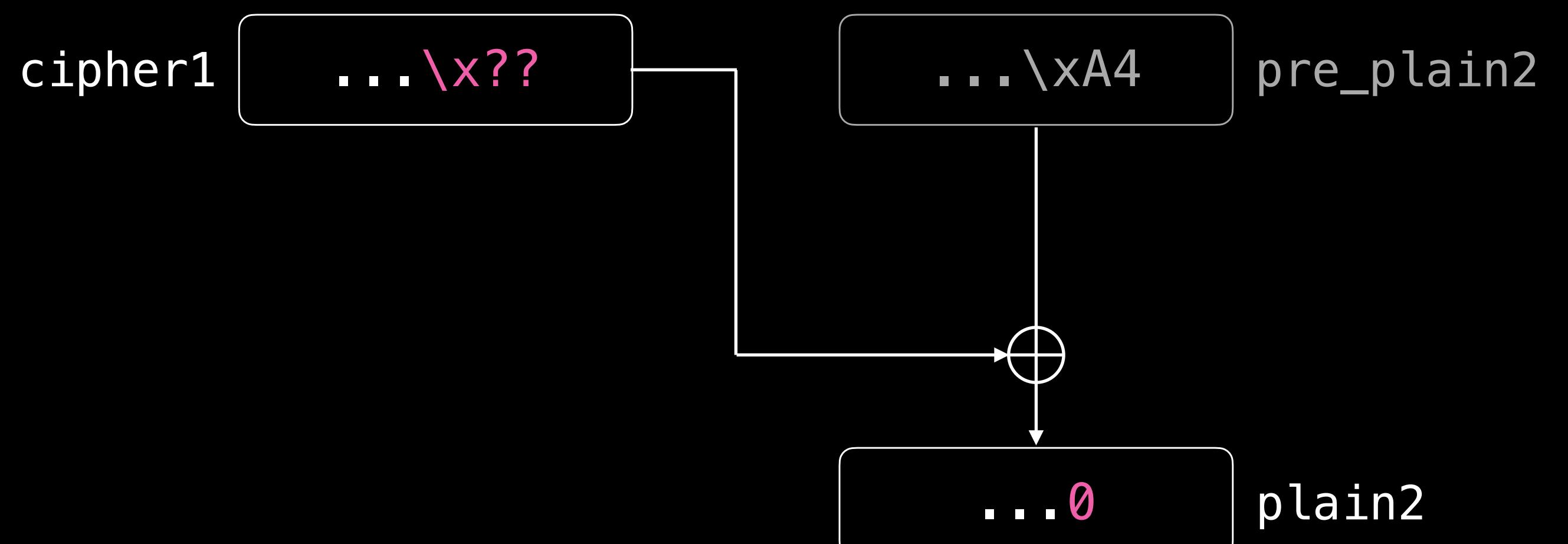


$0x61 = 0xFC \wedge 0x9D$

# >\_ Bit Flipping

- ▶ 如果我想要 plain1 的最後一 bytes 是 '0'，那

$0x?? = 0x58 \wedge 0x??$

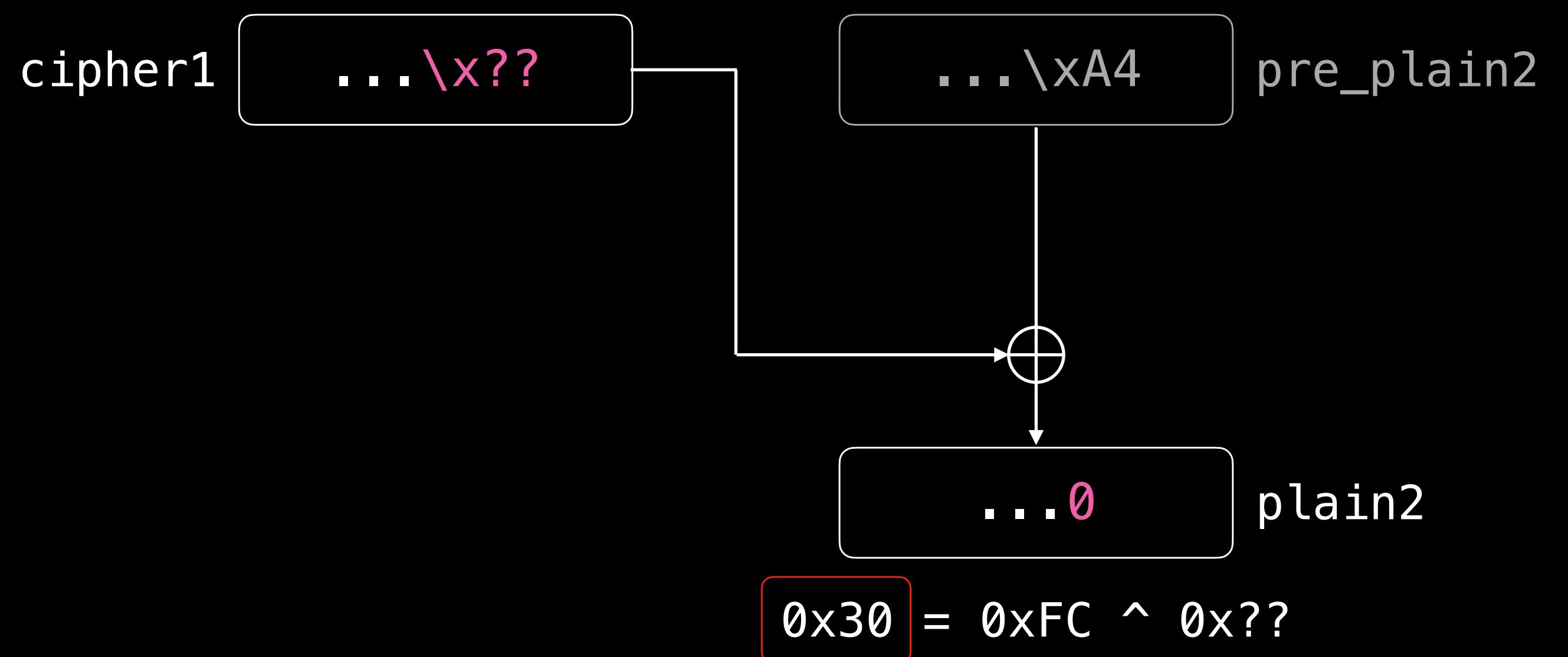


$0x?? = 0xFC \wedge 0x??$

# >\_ Bit Flipping

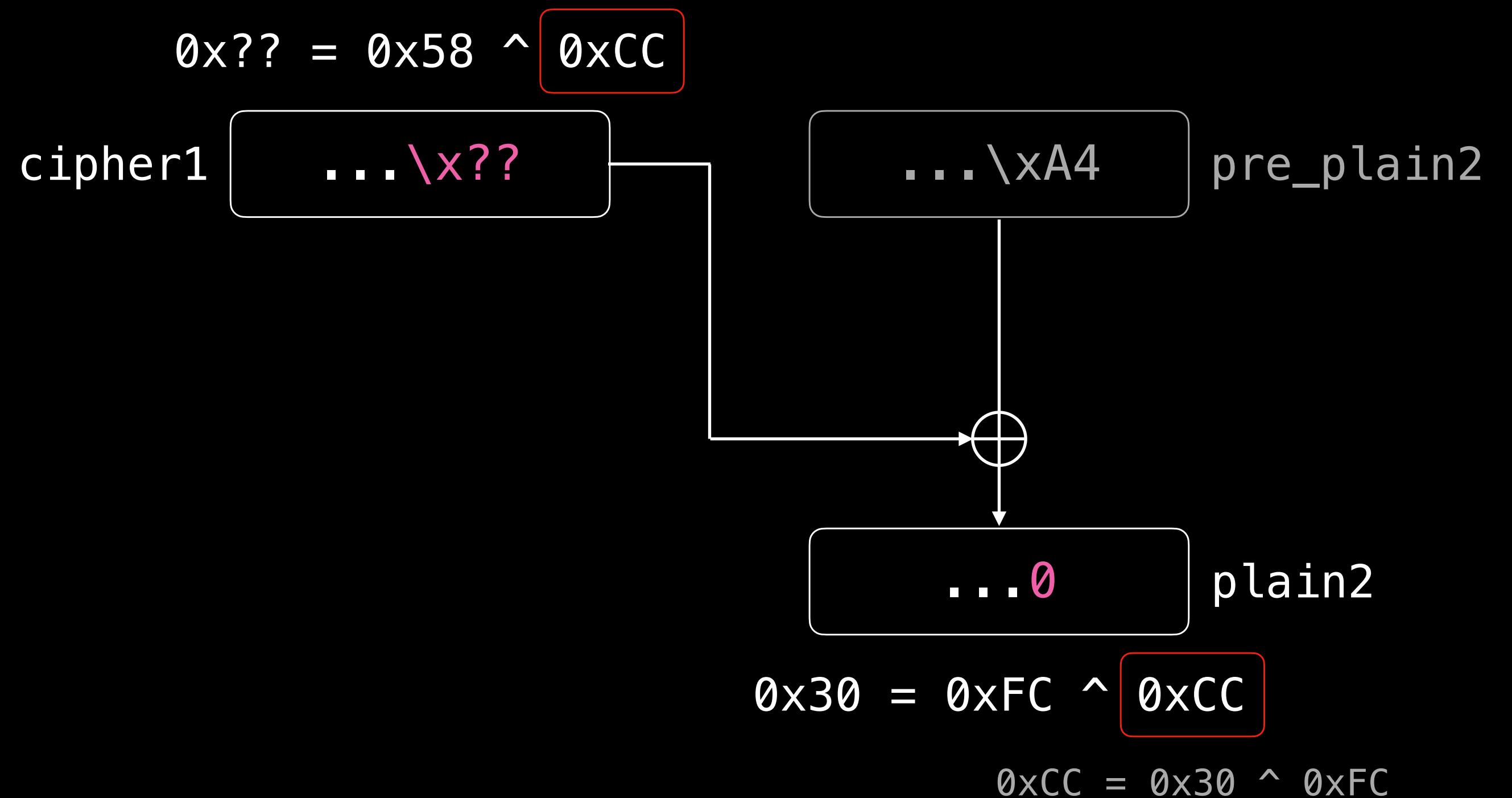
- ▶ 如果我想要 plain1 的最後一 bytes 是 '0'，那

$0x?? = 0x58 \wedge 0x??$



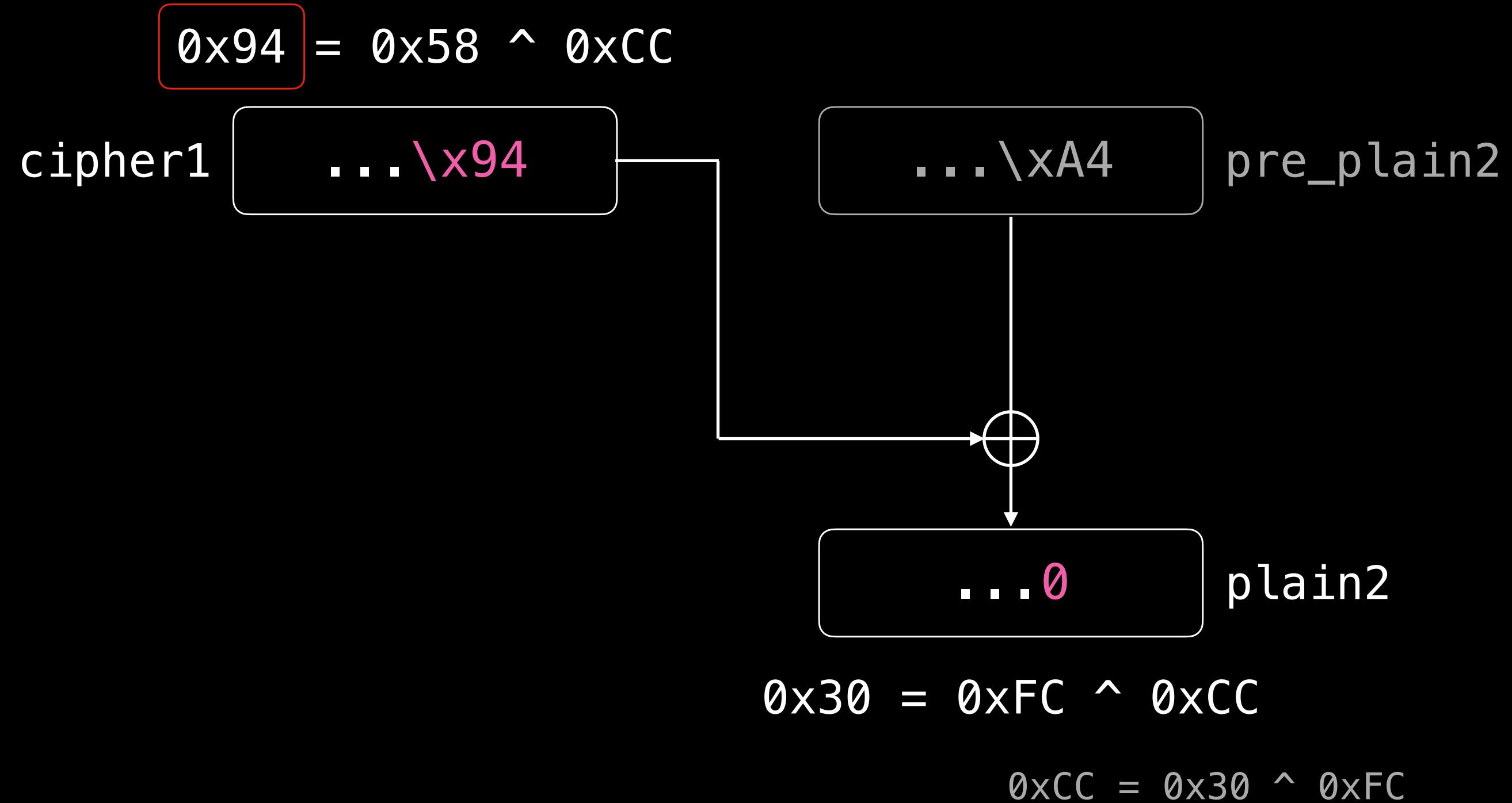
# >\_ Bit Flipping

- ▶ 如果我想要 plain1 的最後一 bytes 是 '0'，那



# >\_ Bit Flipping

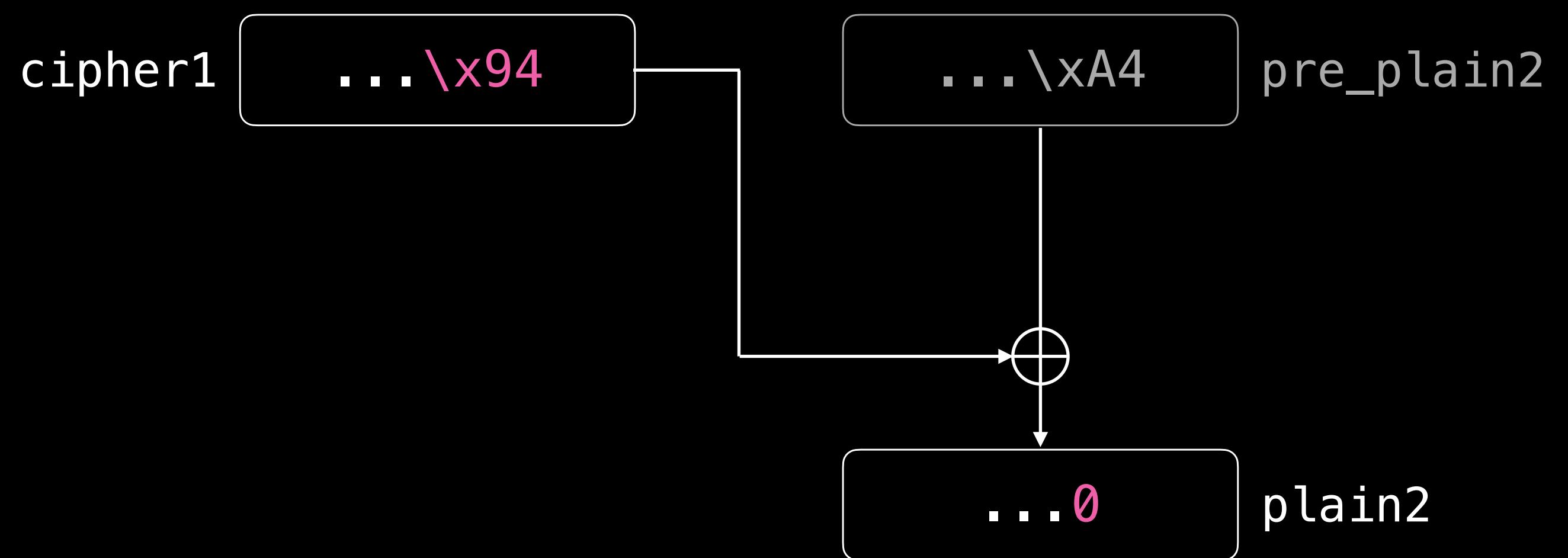
- 如果我想要 plain1 的最後一 bytes 是 '0'，推出來我要把 cipher[1] 的最後一 bytes 改成 \x94



# >\_ Bit Flipping

- 如果我想要 plain1 的最後一 bytes 是 '0'，推出來我要把 cipher[1] 的最後一 bytes 改成 \x94

$$0x94 = 0x58 \wedge 0xCC = 0x58 \wedge (0x30 \wedge 0xFC)$$



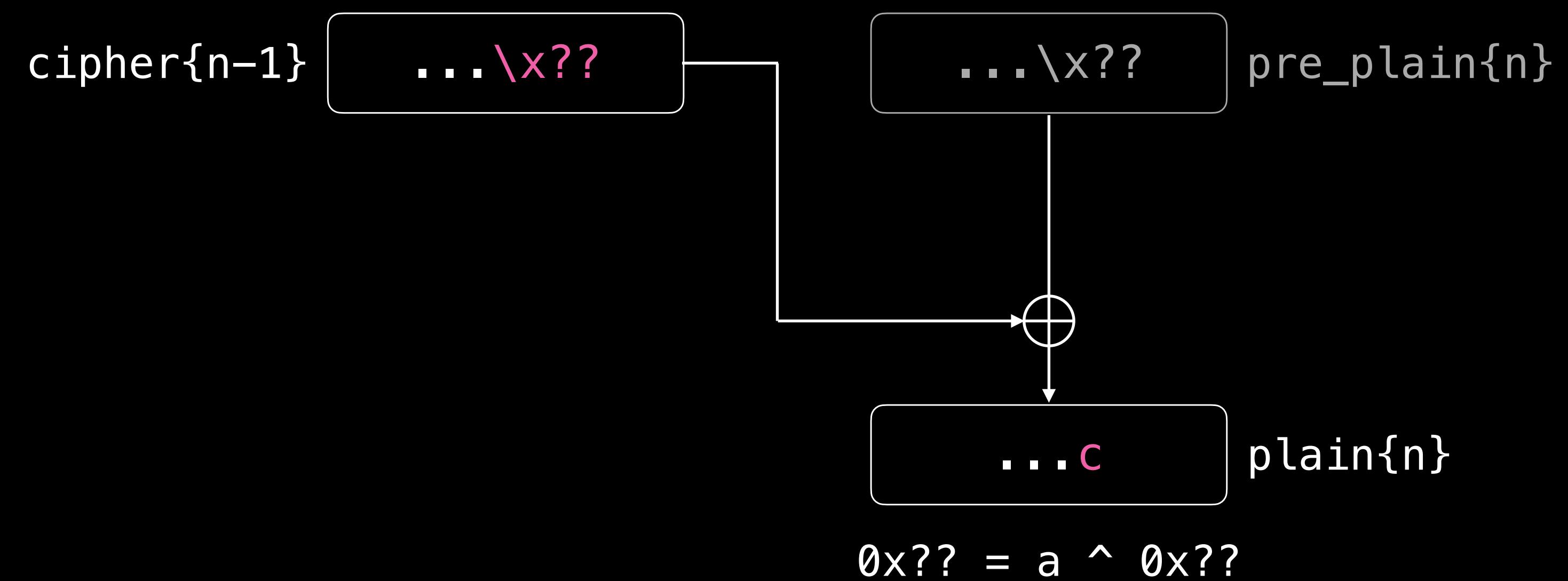
$$0x30 = 0xFC \wedge 0xCC$$

$$0xCC = 0x30 \wedge 0xFC$$

## >\_ Bit Flipping

- 所以說假設  $\text{plain}\{n\}$  的最後一 byte 原本是  $a$ ， $\text{cipher}\{n-1\}$  的最後一 byte 是  $b$ ，我們想要把  $\text{plain}\{n\}$  的最後一 byte 改成  $c$ ，那  $\text{cipher}\{n-1\}$  的最後一 byte 要改成  $b \oplus a \oplus c$

$0x?? = b \wedge 0x??$



## >\_ Bit Flipping

- ▶ 以上 Bit Flipping 的技巧不只對最後 1 bytes 有用，對於 block 中的每一 bytes 都有效

>\_ Bit Flipping

Lab : Bit Flipping Ø

# AES CBC Mode – Padding Oracle

## >\_ Padding Oracle

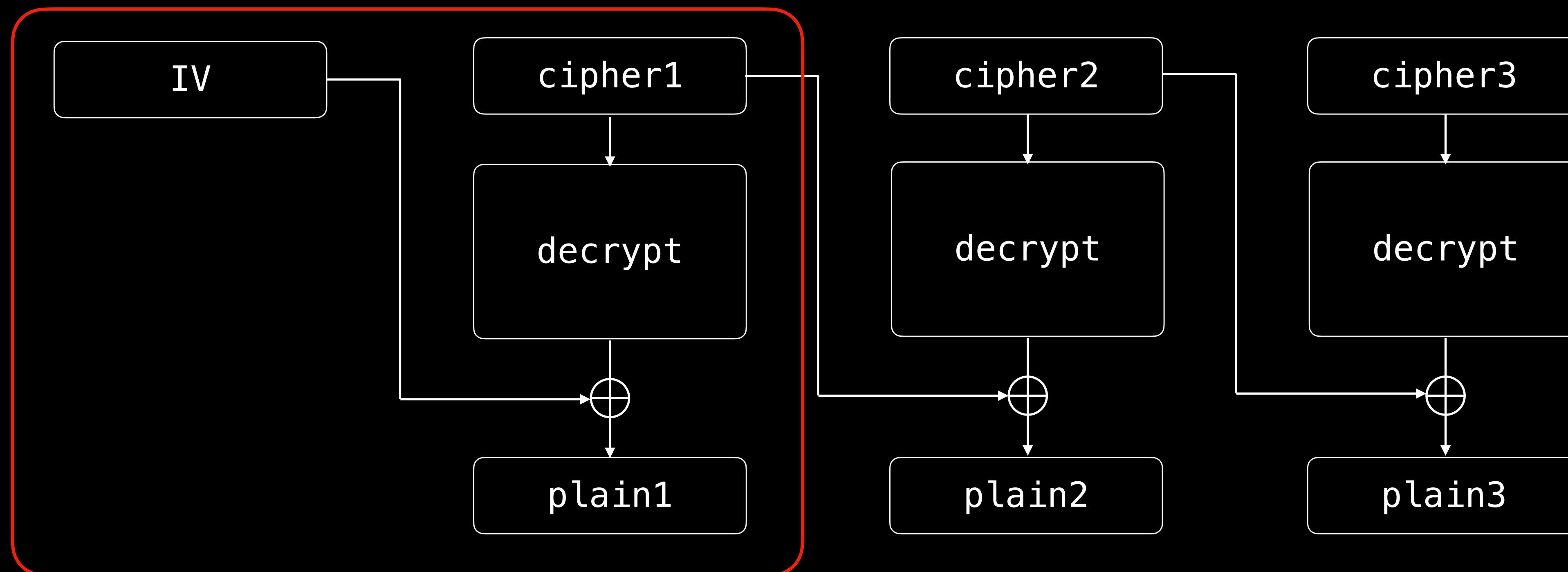
- ▶ 回想一下 PKCS#7 的 unpad，假設最後一個 bytes 是 n  
則如果 n 不滿足  $0 < n \leq 16$  或最後 n bytes 並不都是 bytes([n]) 時 unpad 就會報錯

## >\_ Padding Oracle

- ▶ 如果今天有一個 service，他會告訴你傳過來的 cipher 解密之後 padding 有沒有正確

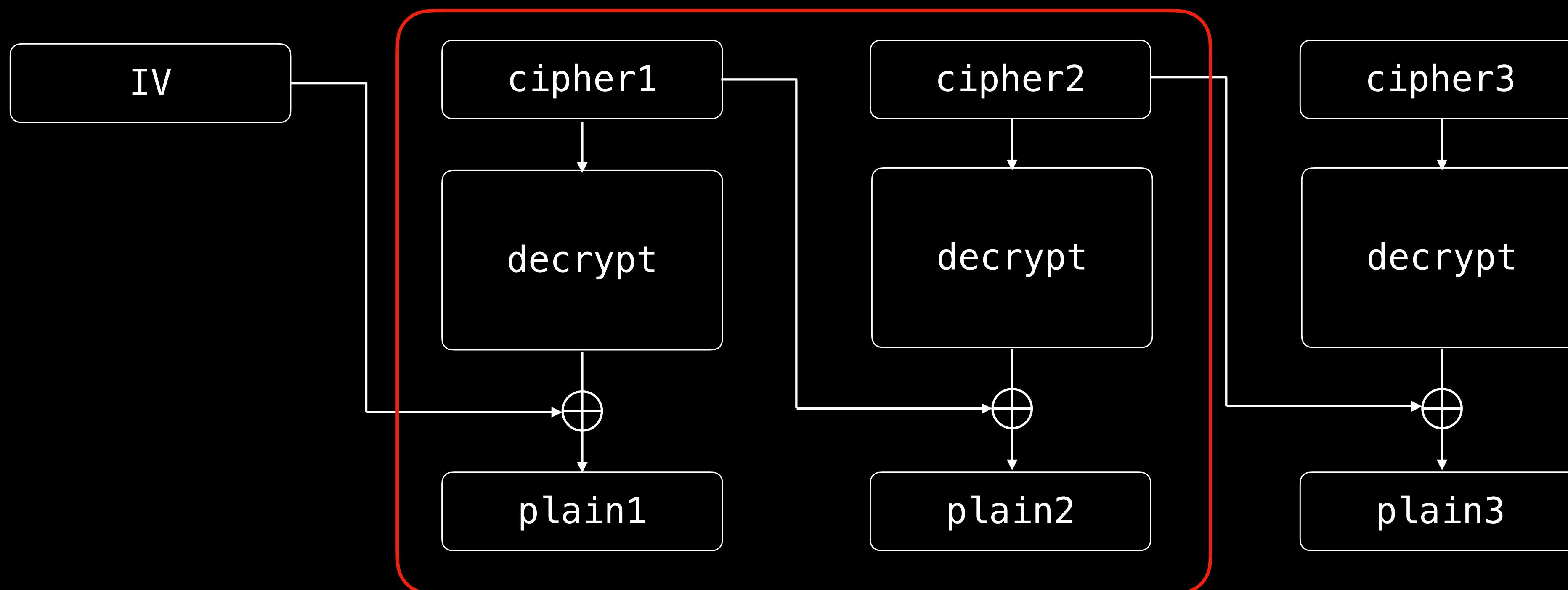
## >\_ Padding Oracle

- ▶ 如果今天有一個 service，他會告訴你傳過來的 cipher 解密之後 padding 有沒有正確
- ▶ Padding Oracle Attack : 如果 IV 可以改，那透過 IV 和 cipher1 可以推出 plain1



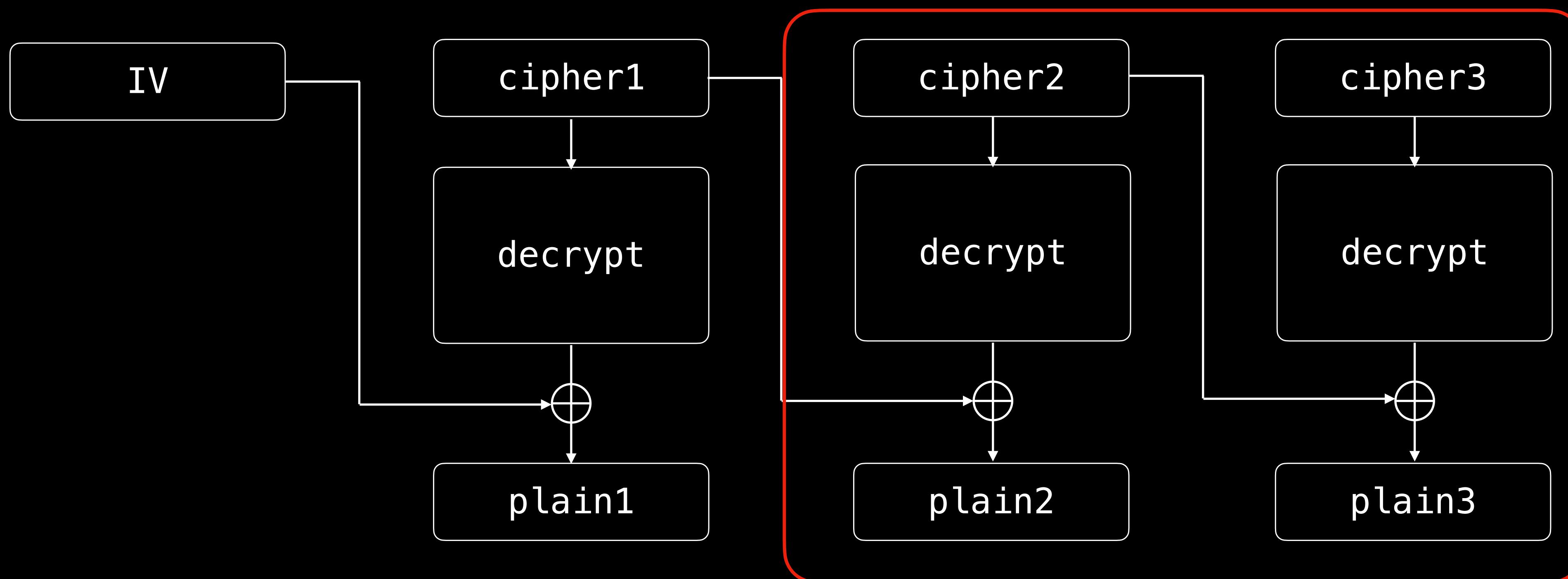
## >\_ Padding Oracle

- ▶ 如果今天有一個 service，他會告訴你傳過來的 cipher 解密之後 padding 有沒有正確
- ▶ Padding Oracle Attack : 如果 cipher1 可以改，那透過 cipher1 和 cipher2 可以推出 plain2



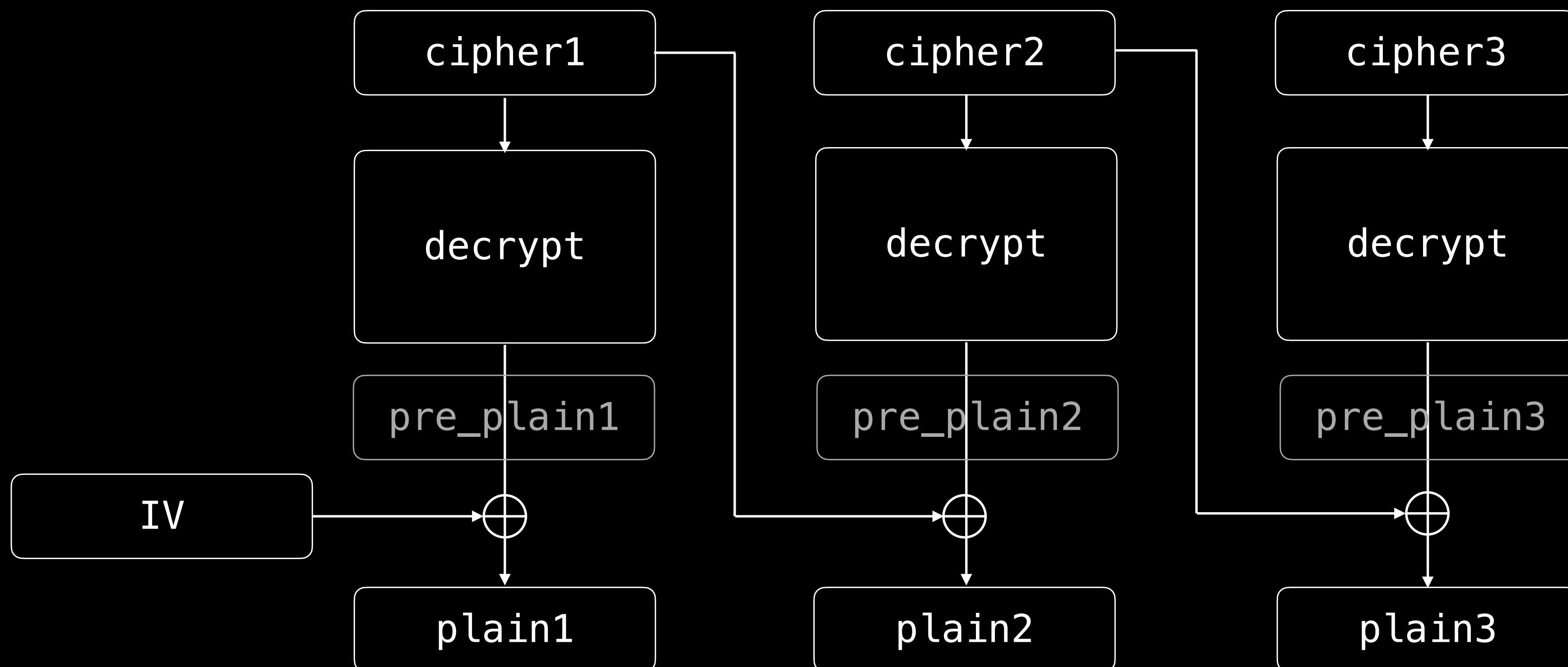
## >\_ Padding Oracle

- ▶ 如果今天有一個 service，他會告訴你傳過來的 cipher 解密之後 padding 有沒有正確
- ▶ Padding Oracle Attack : 如果 cipher2 可以改，那透過 cipher2 和 cipher3 可以推出 plain3



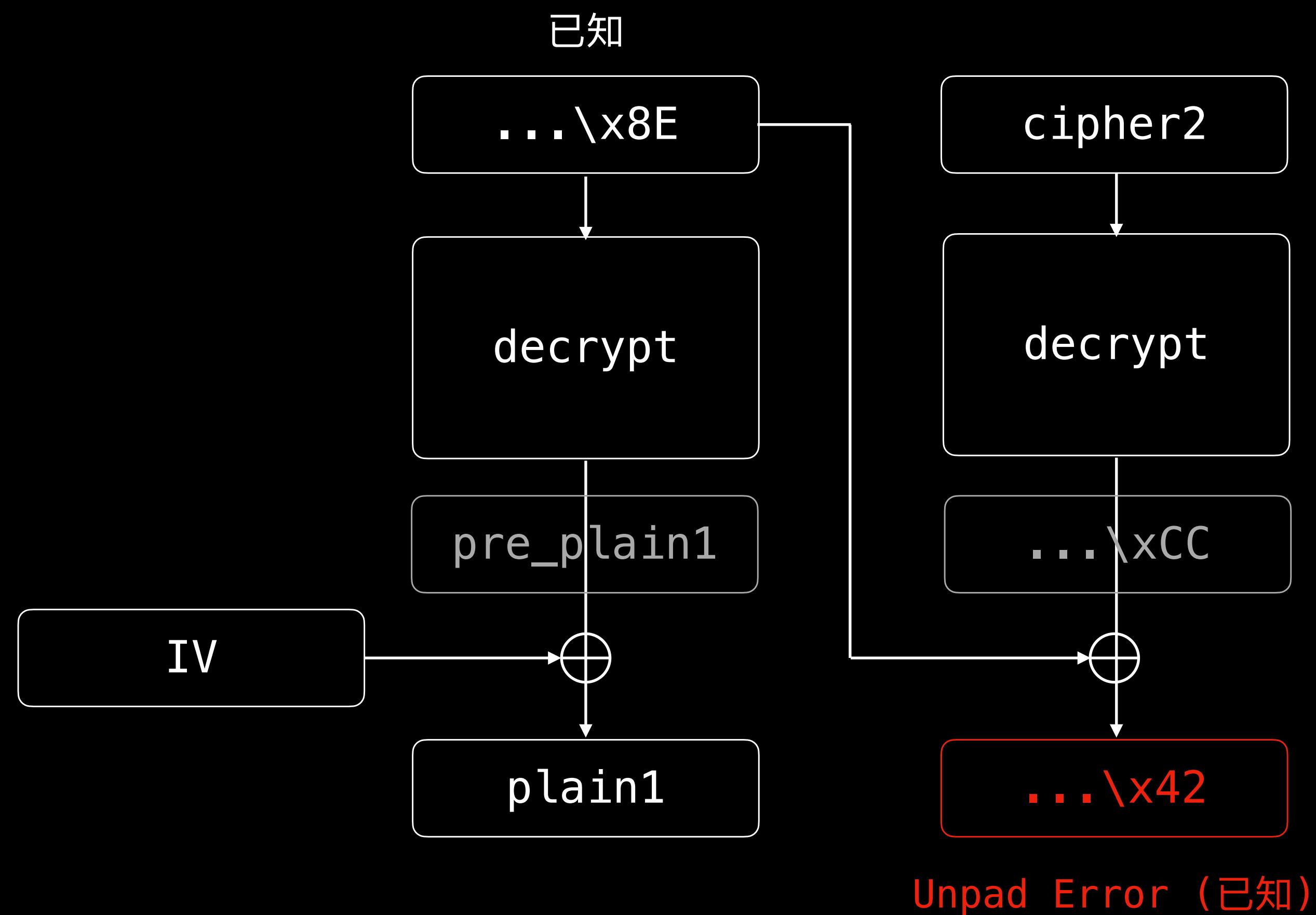
# >\_ Padding Oracle

- ▶ 假設已知 cipher，我們想要知道 plain2 的最後 1 bytes 是什麼



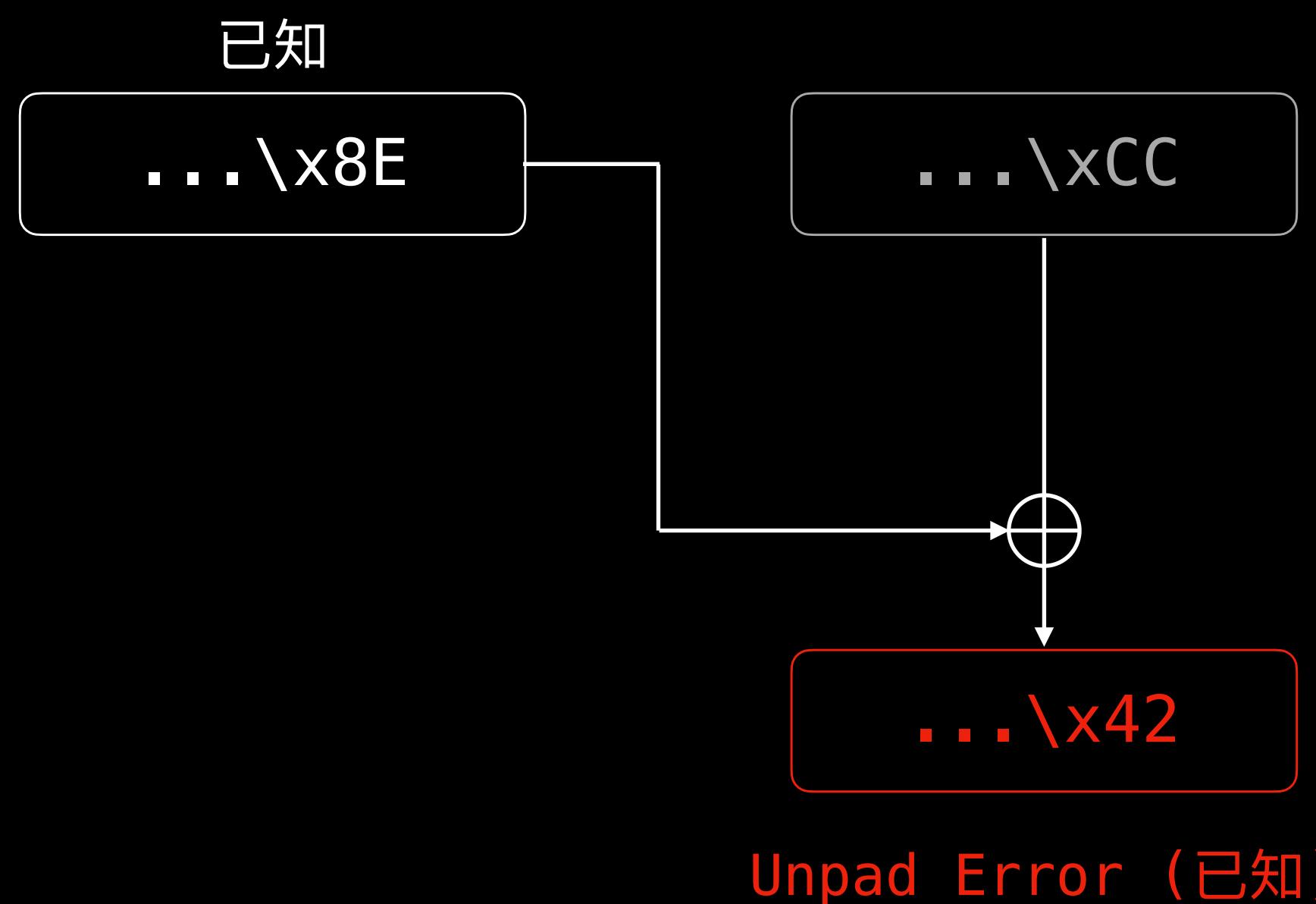
# >\_ Padding Oracle

- decrypt 只傳入 cipher1 + cipher2



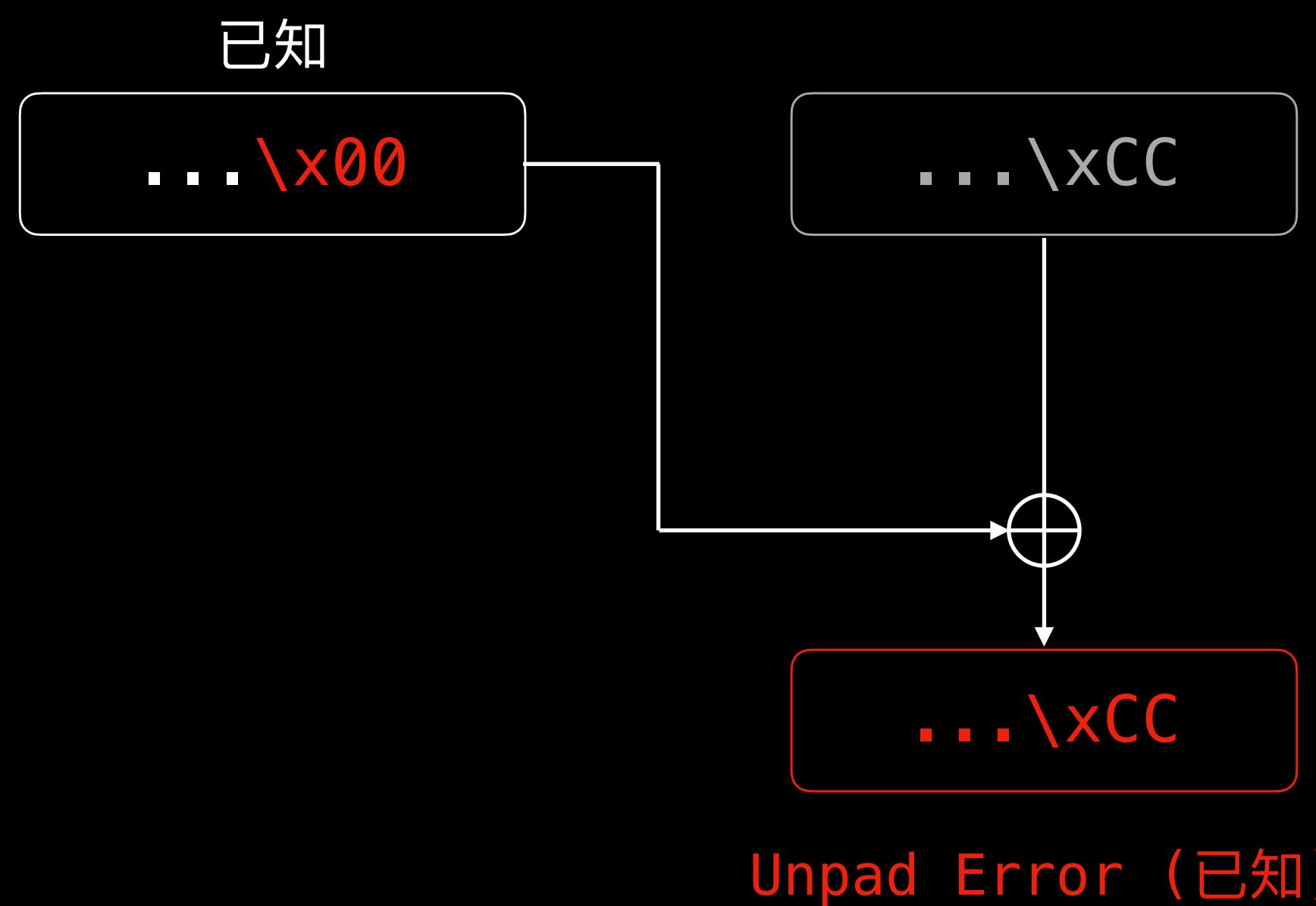
# >\_ Padding Oracle

- ▶ 找 plain2 的最後 1 bytes



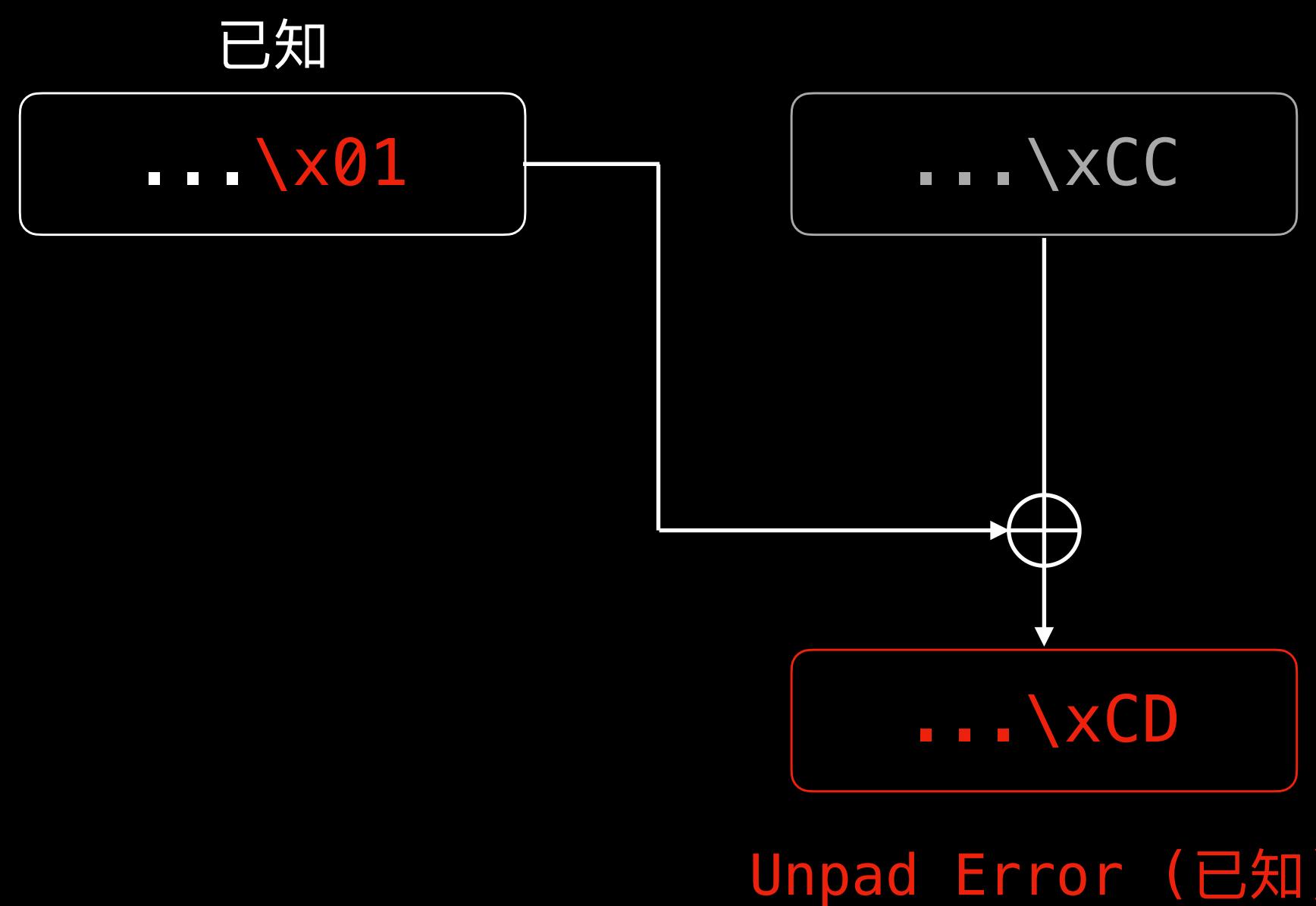
# >\_ Padding Oracle

- ▶ 找 plain2 的最後 1 bytes



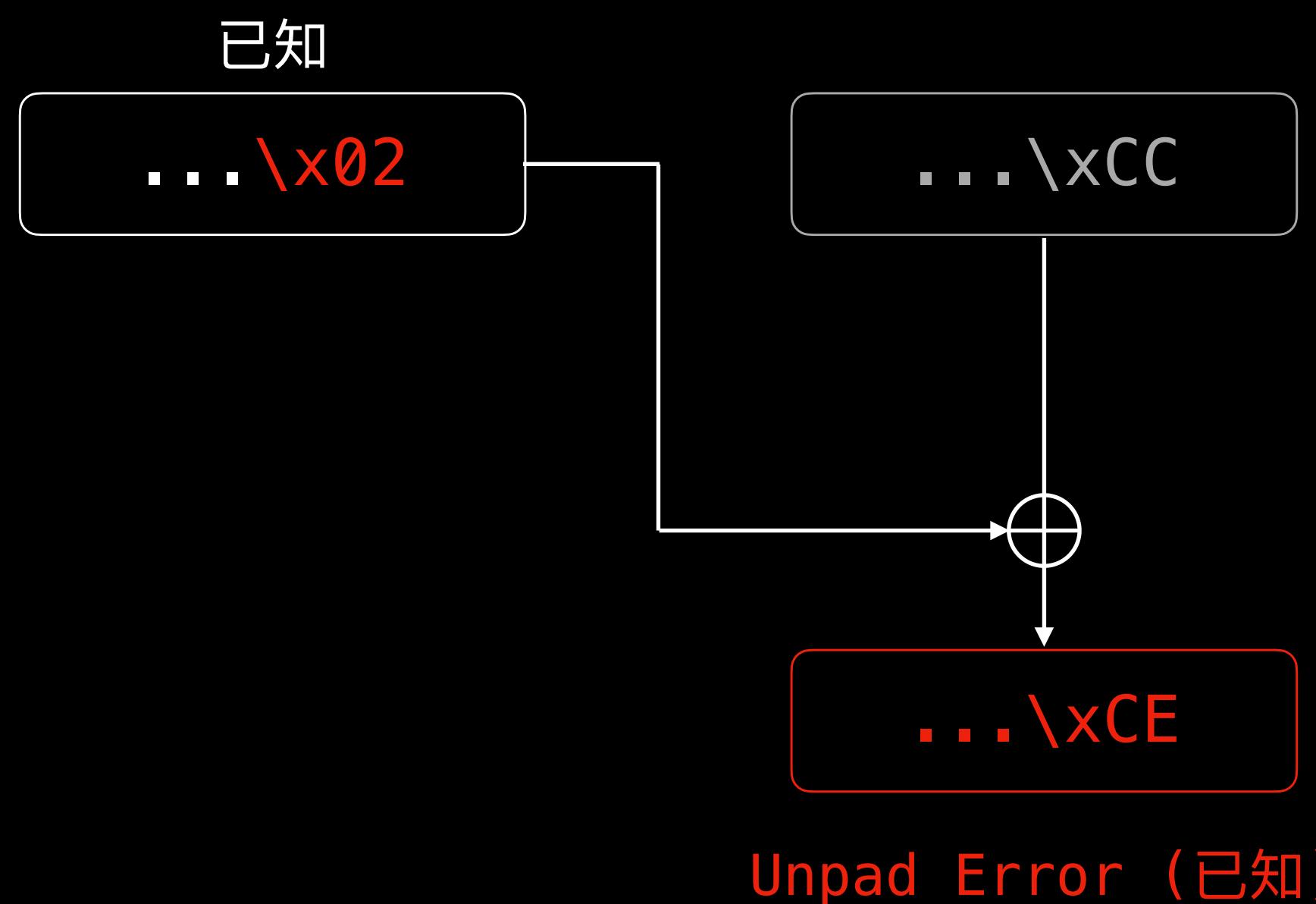
# >\_ Padding Oracle

- ▶ 找 plain2 的最後 1 bytes



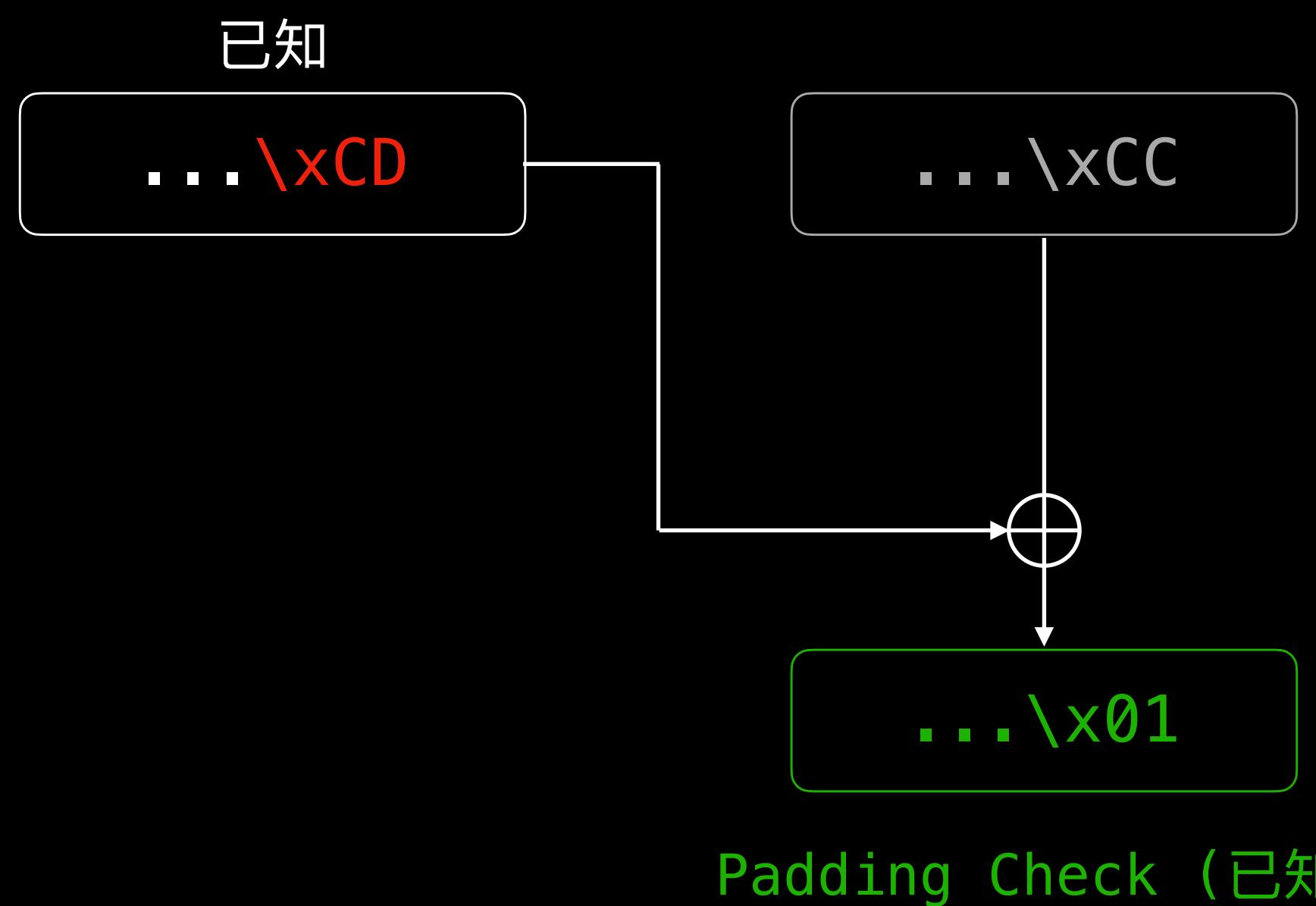
# >\_ Padding Oracle

- ▶ 找 plain2 的最後 1 bytes



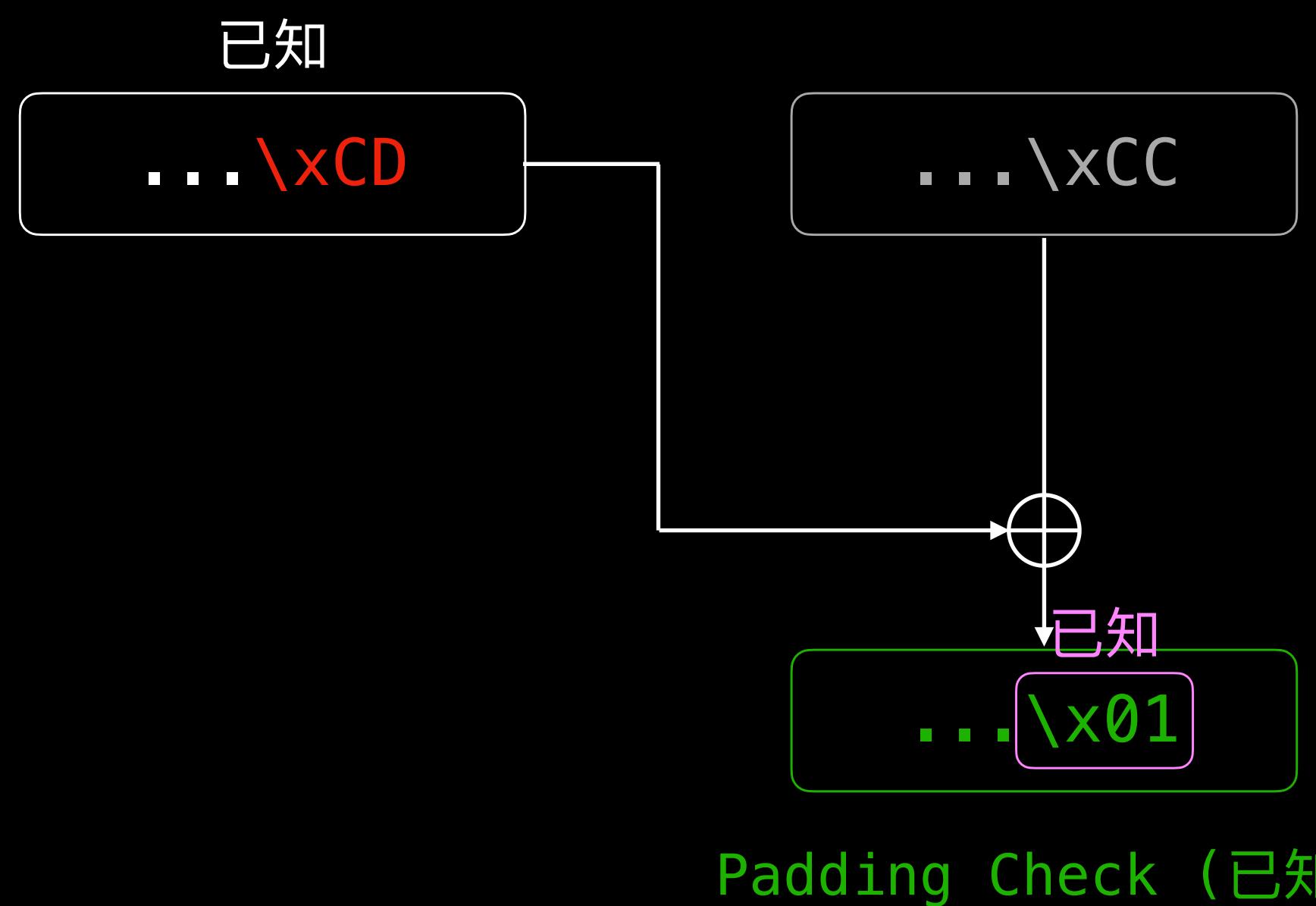
# >\_ Padding Oracle

- ▶ 找 plain2 的最後 1 bytes



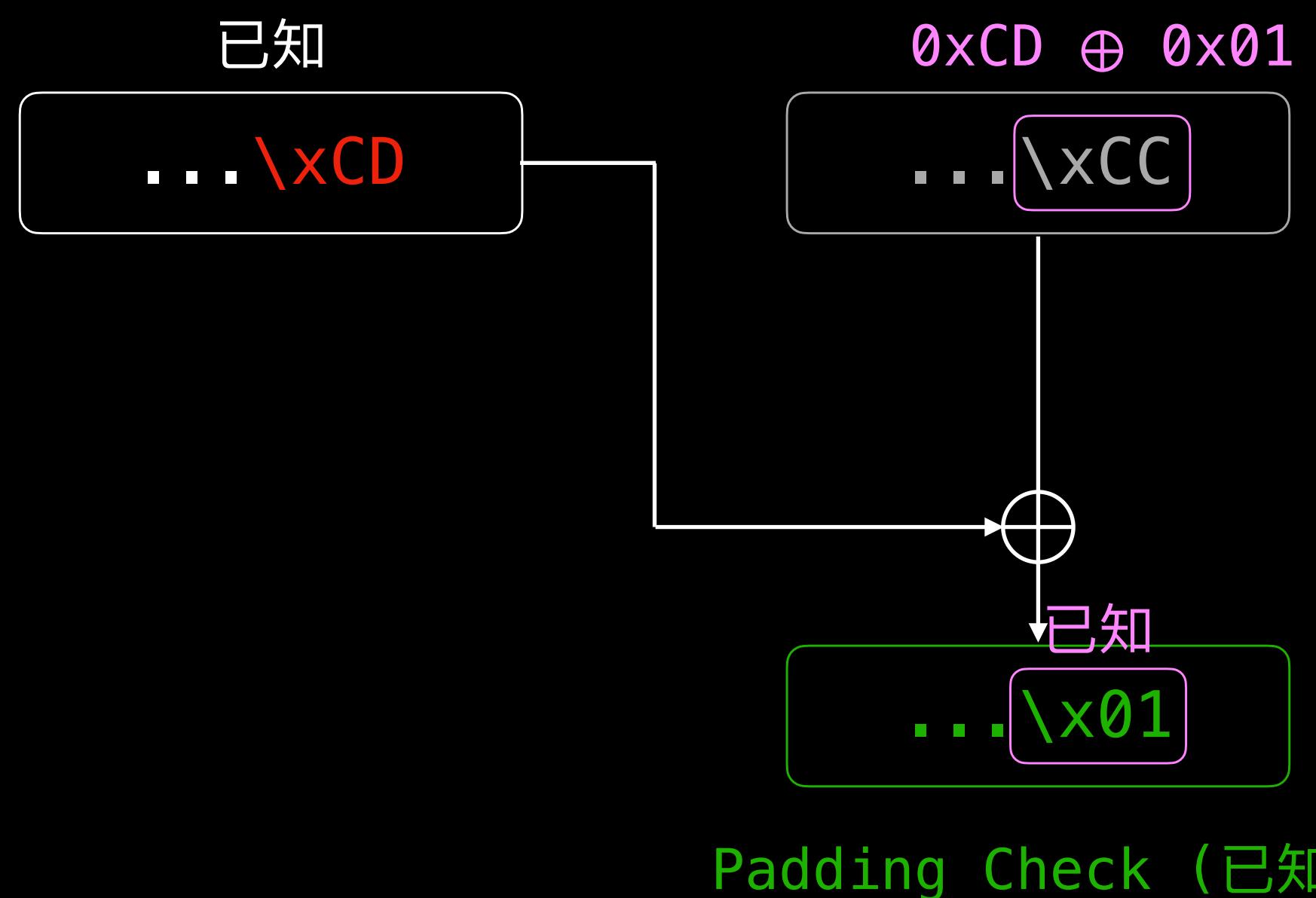
# >\_ Padding Oracle

- ▶ 找 plain2 的最後 1 bytes



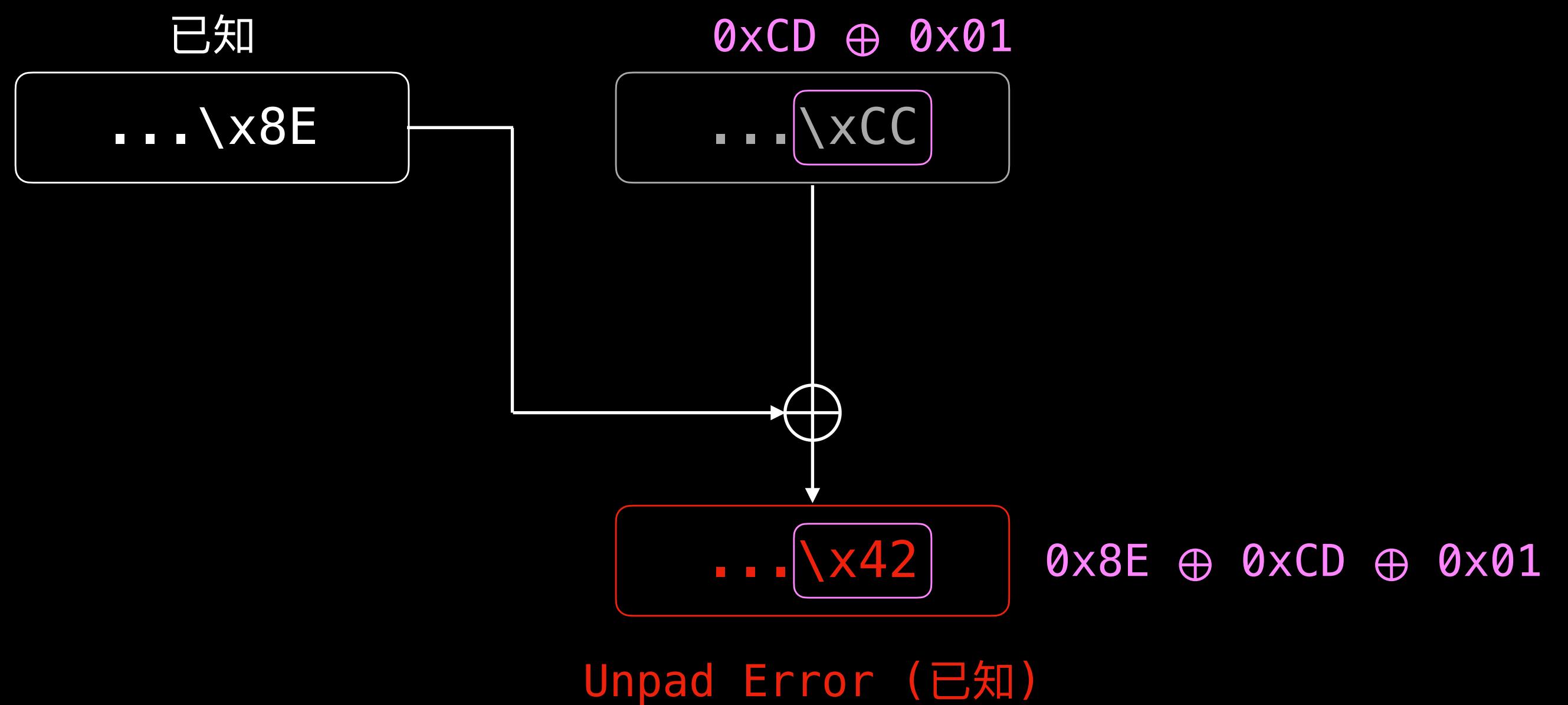
# >\_ Padding Oracle

- ▶ 找 plain2 的最後 1 bytes



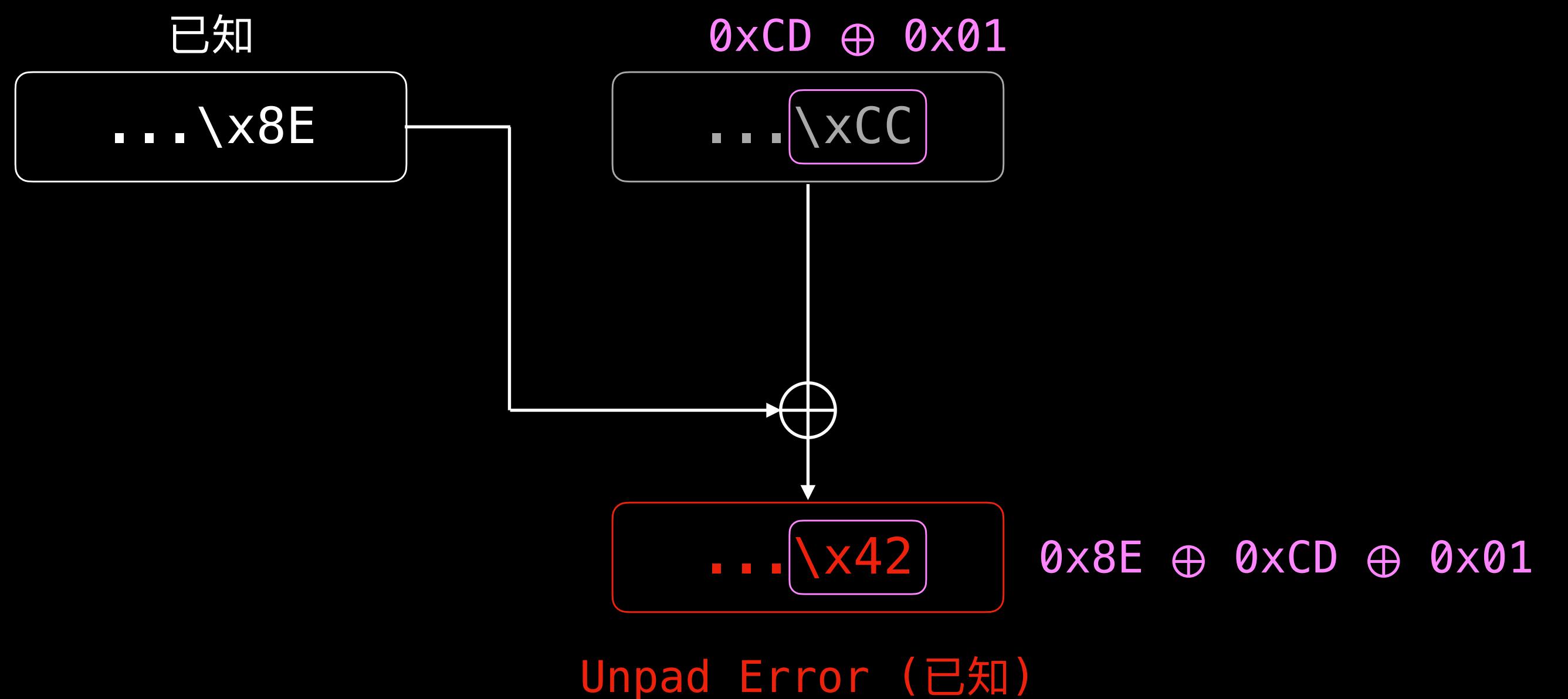
# >\_ Padding Oracle

- ▶ 找 plain2 的最後 1 bytes



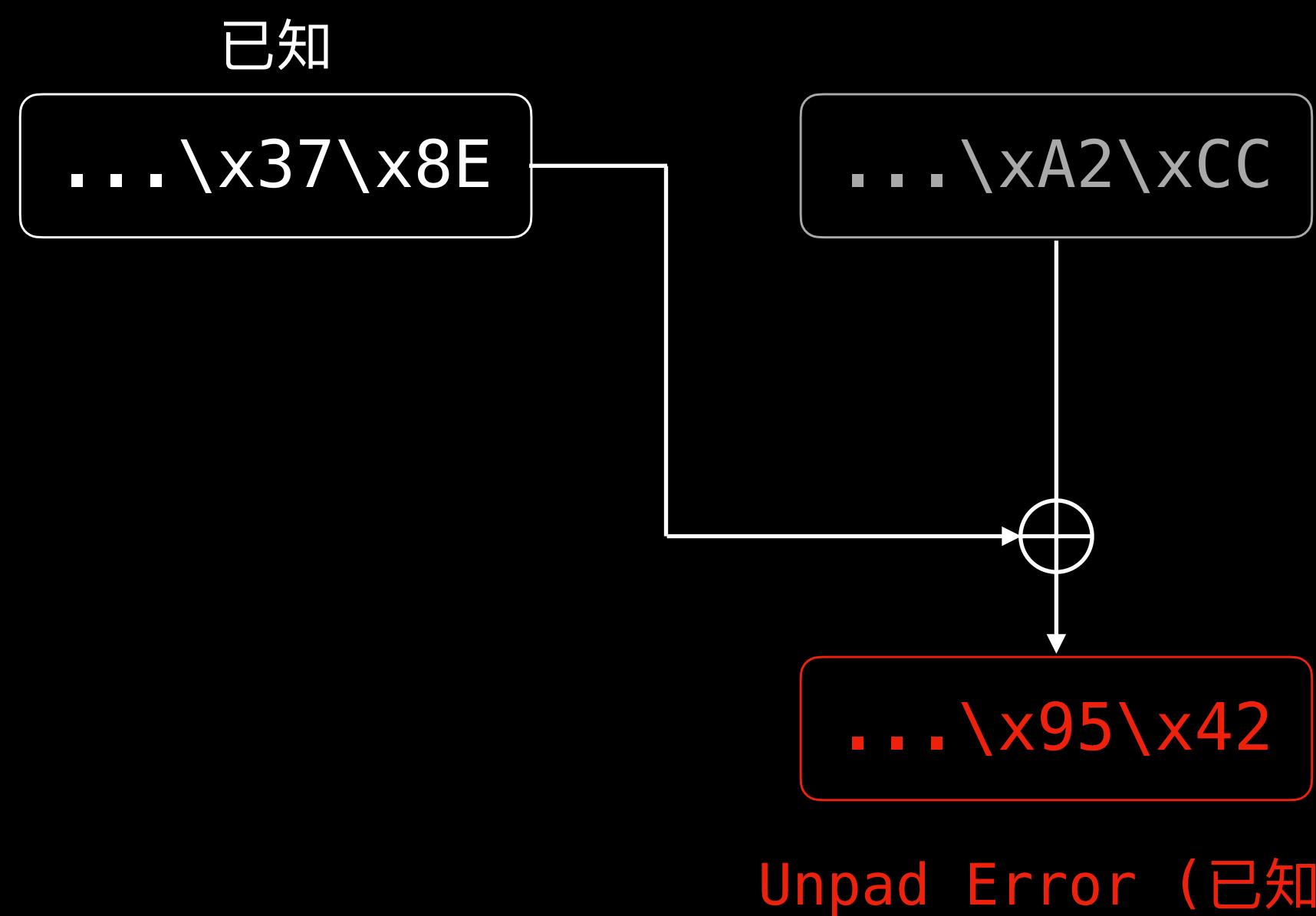
# >\_ Padding Oracle

- ▶ 這樣就找到 plain2 的最後一個 bytes 了



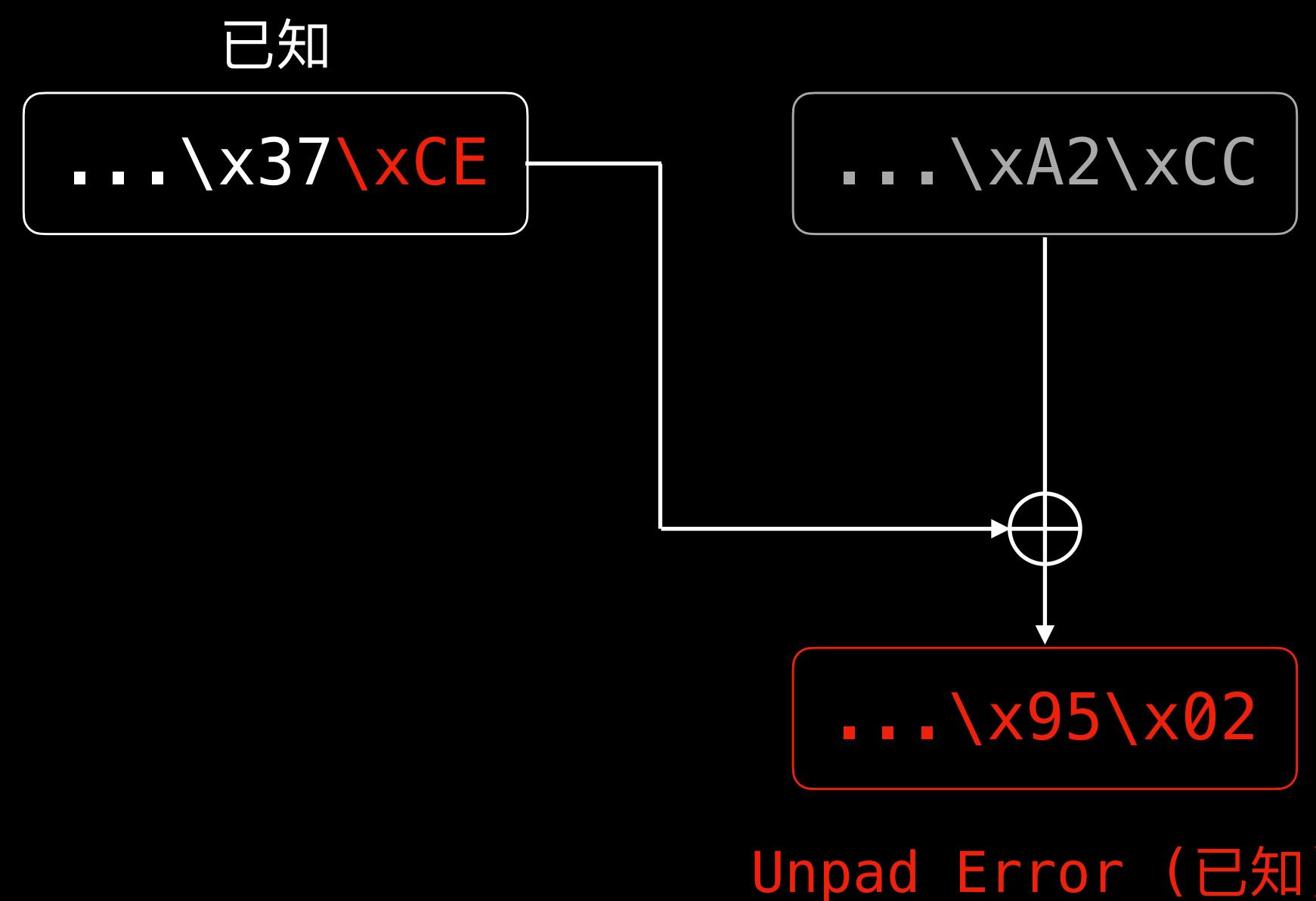
## >\_ Padding Oracle

- 知道了 plain2 的最後 1 bytes 是什麼後，我們可以用 Bit Flipping 的技巧把最後一 bytes 翻成 '\x02'，然後去找 plain2 的倒數第 2 bytes 是什麼



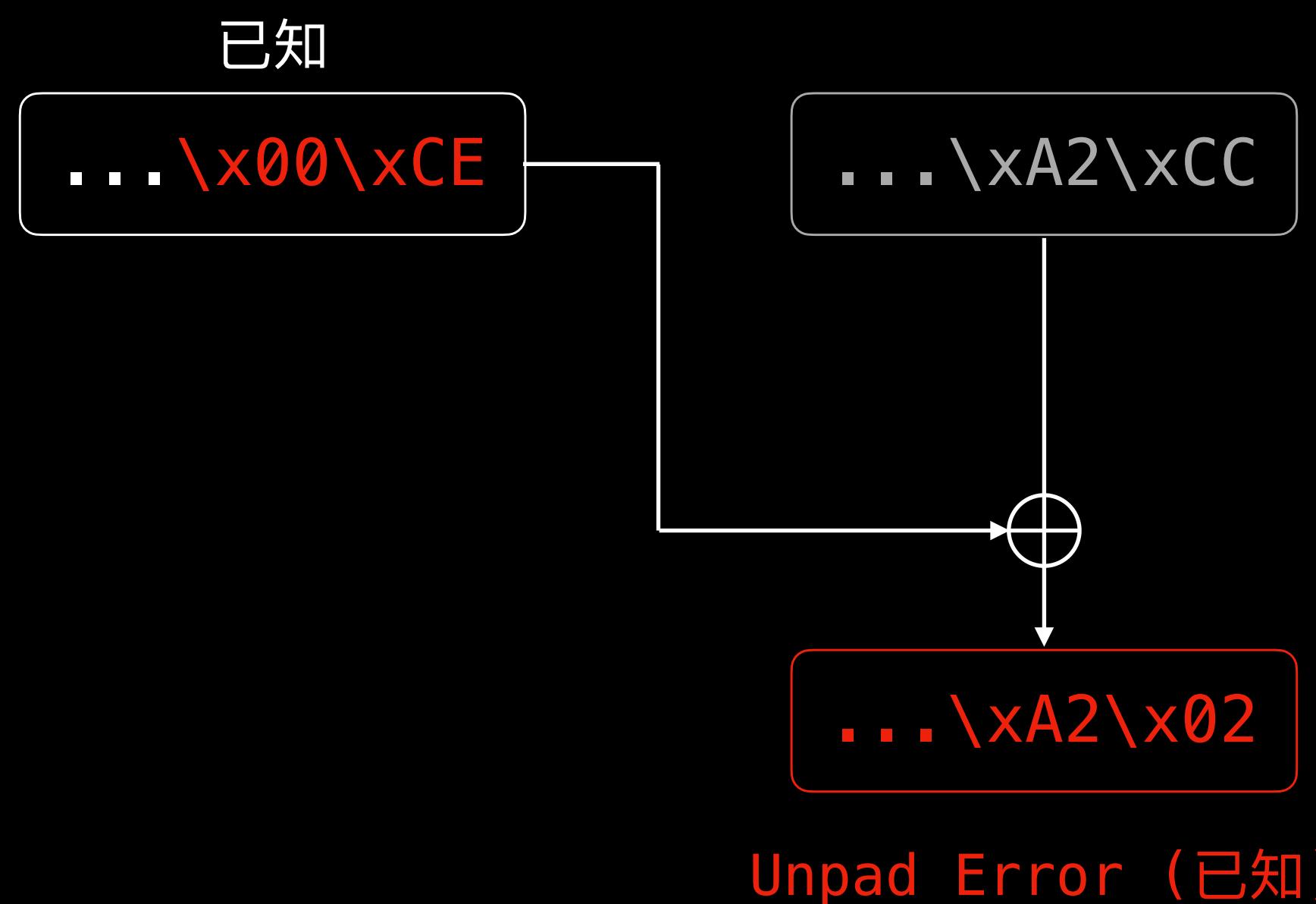
# >\_ Padding Oracle

- ▶ 找 plain2 的倒數第 2 bytes



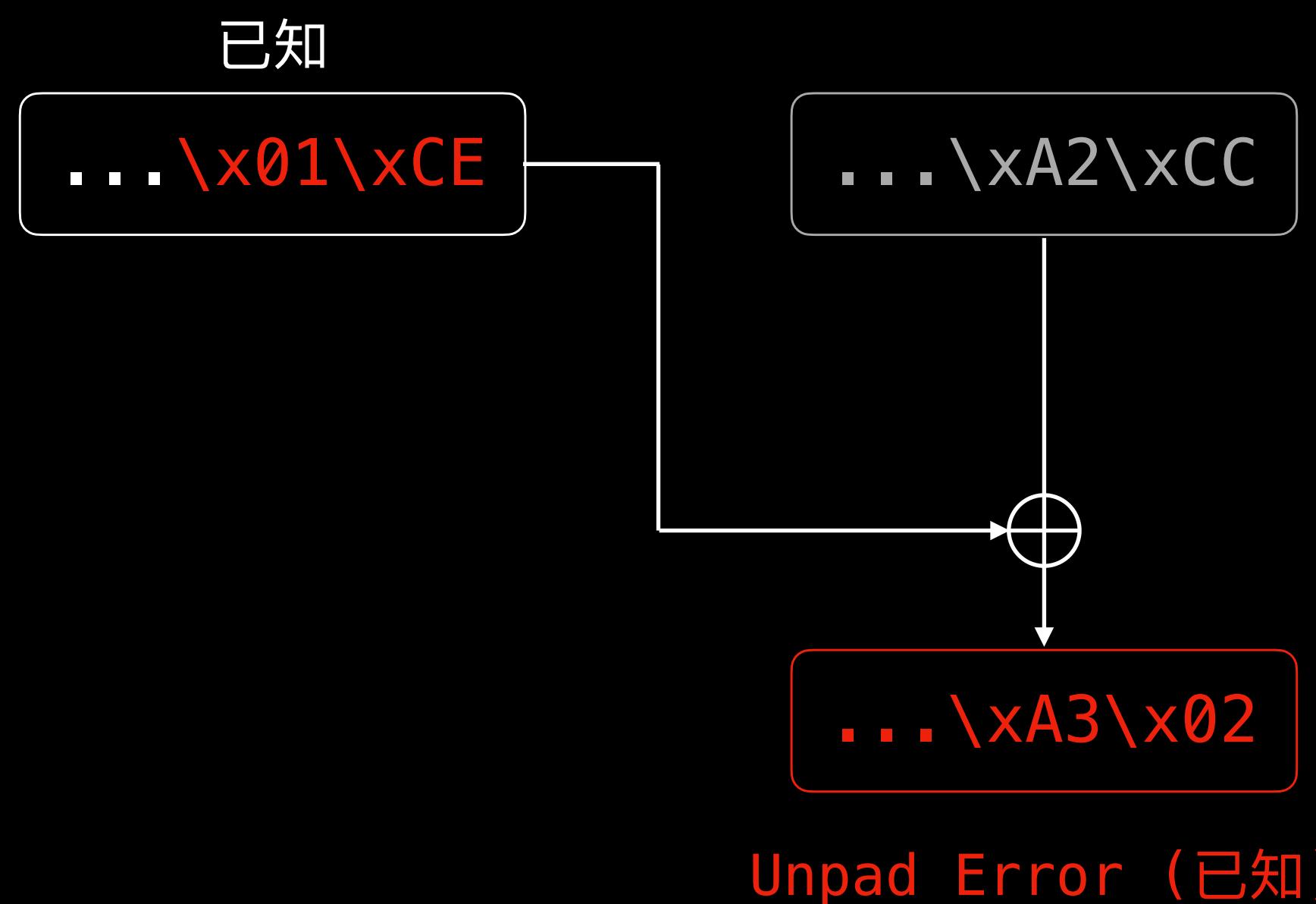
# >\_ Padding Oracle

- ▶ 找 plain2 的倒數第 2 bytes



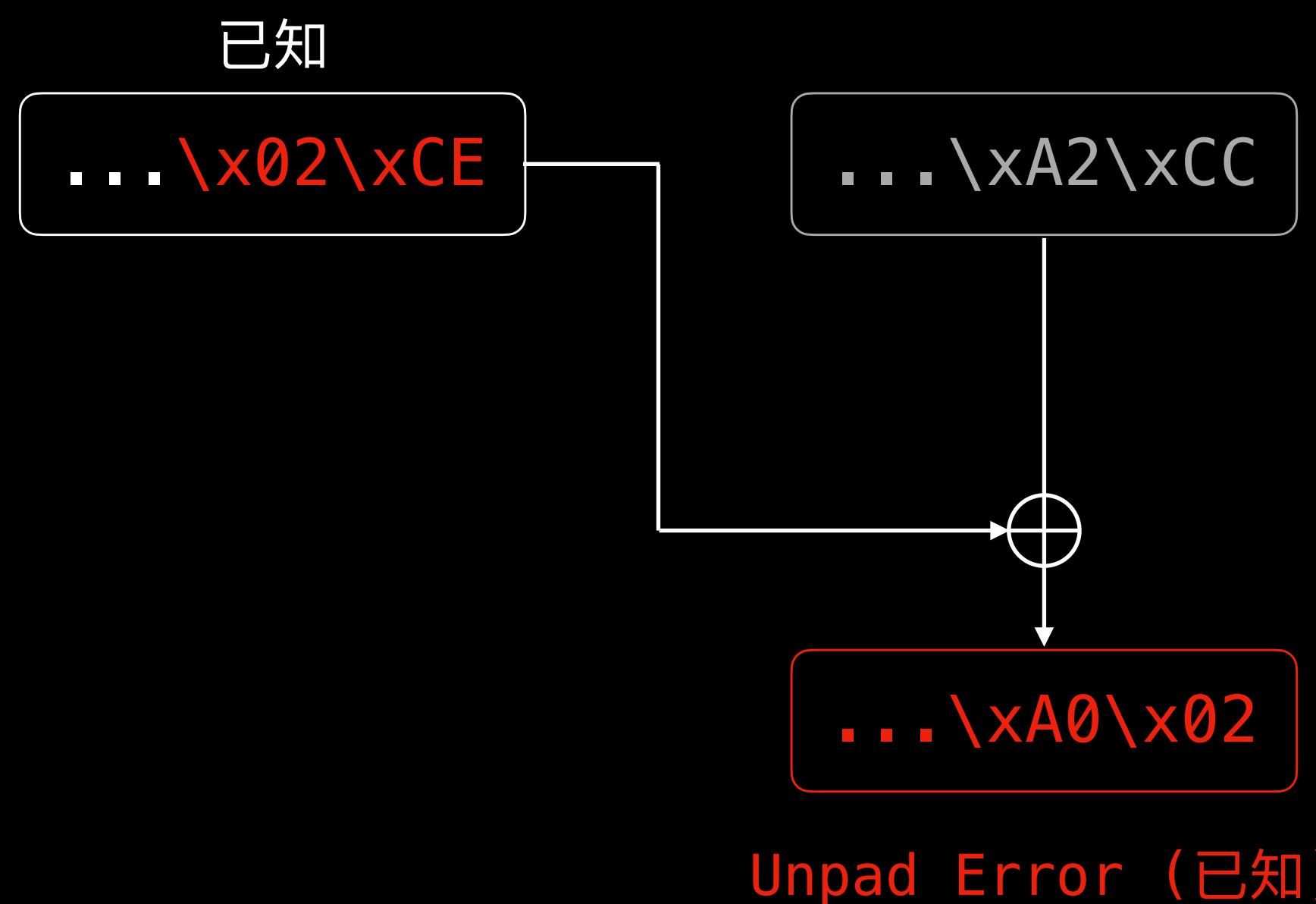
# >\_ Padding Oracle

- ▶ 找 plain2 的倒數第 2 bytes



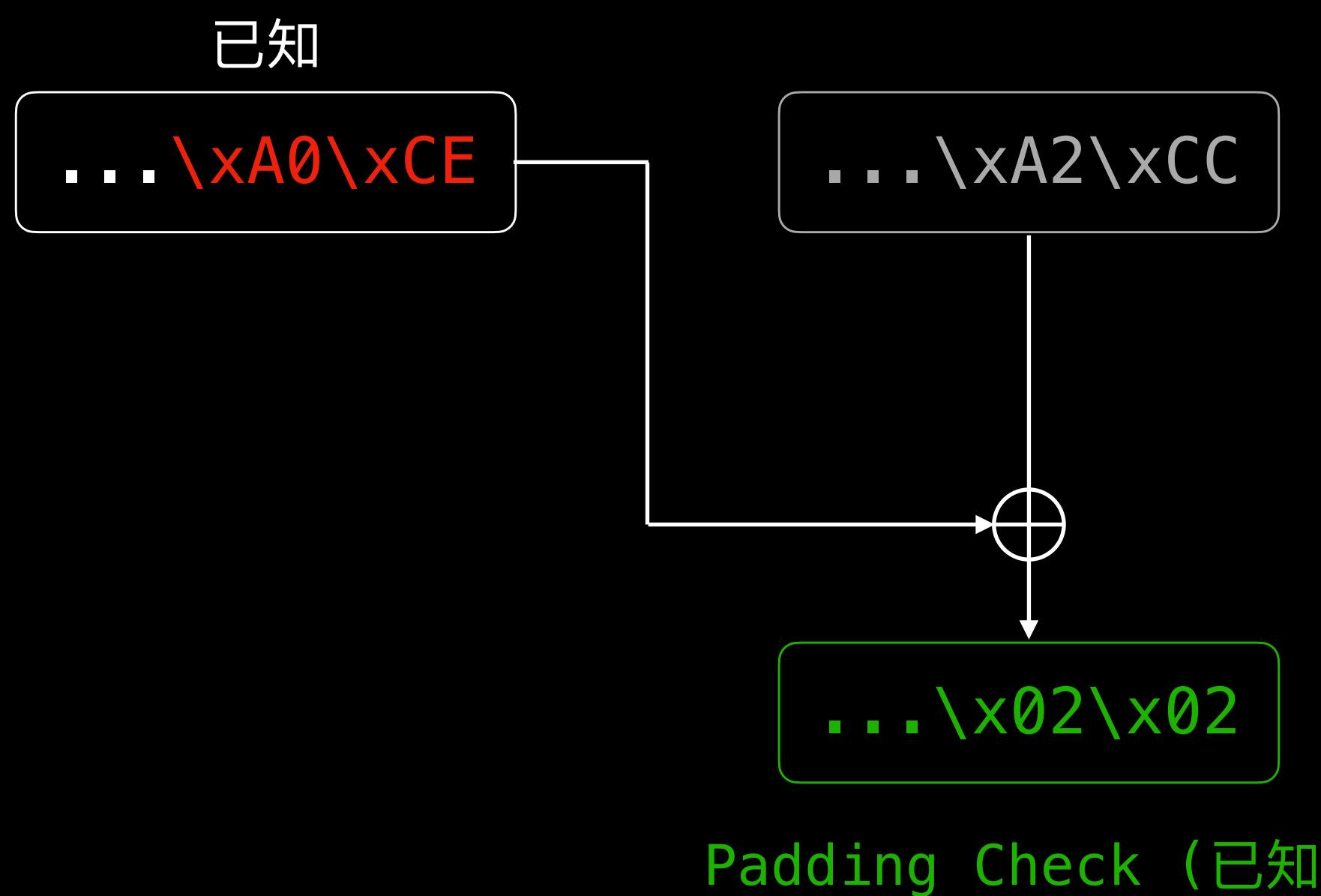
# >\_ Padding Oracle

- ▶ 找 plain2 的倒數第 2 bytes



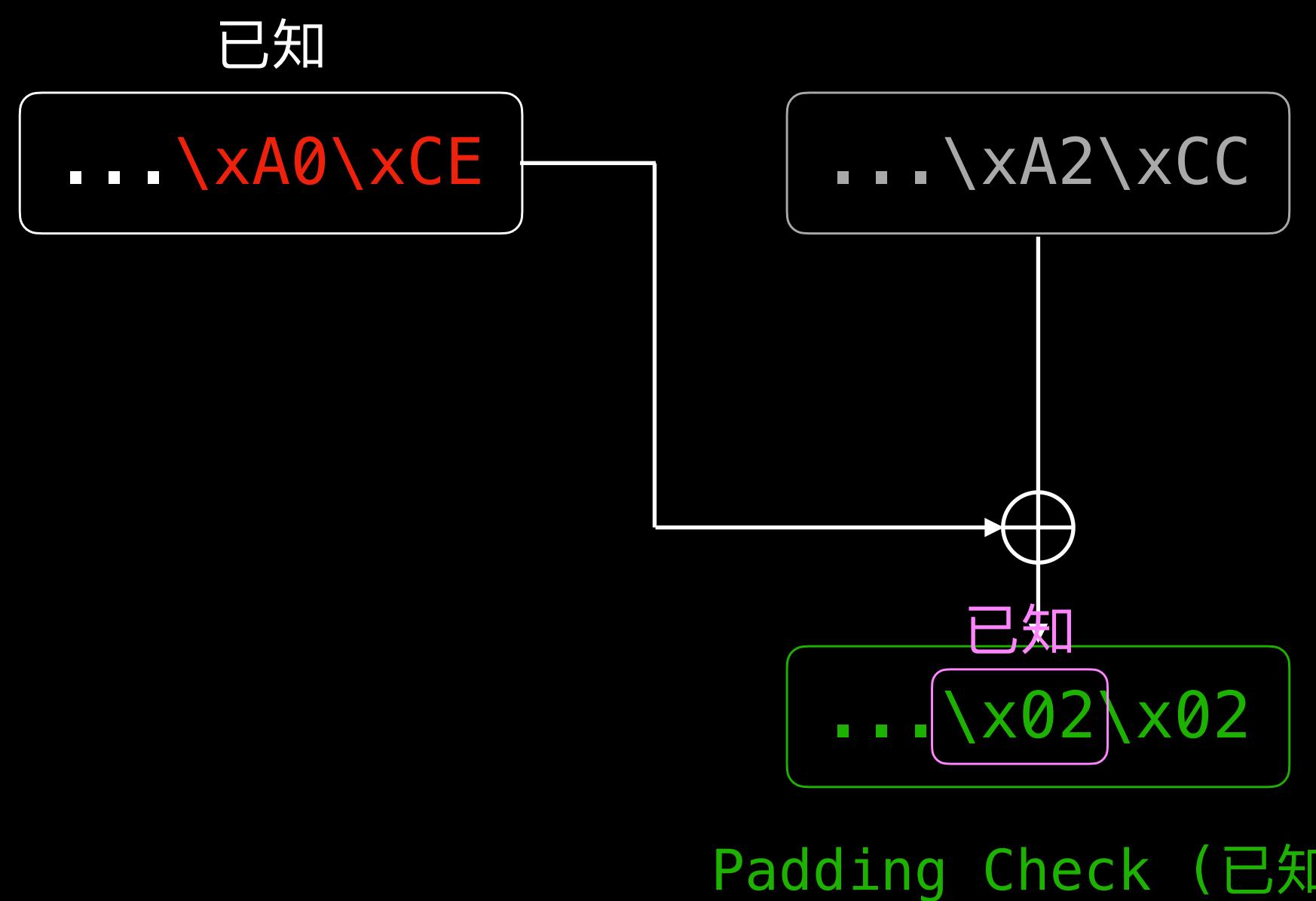
# >\_ Padding Oracle

- ▶ 找 plain2 的倒數第 2 bytes



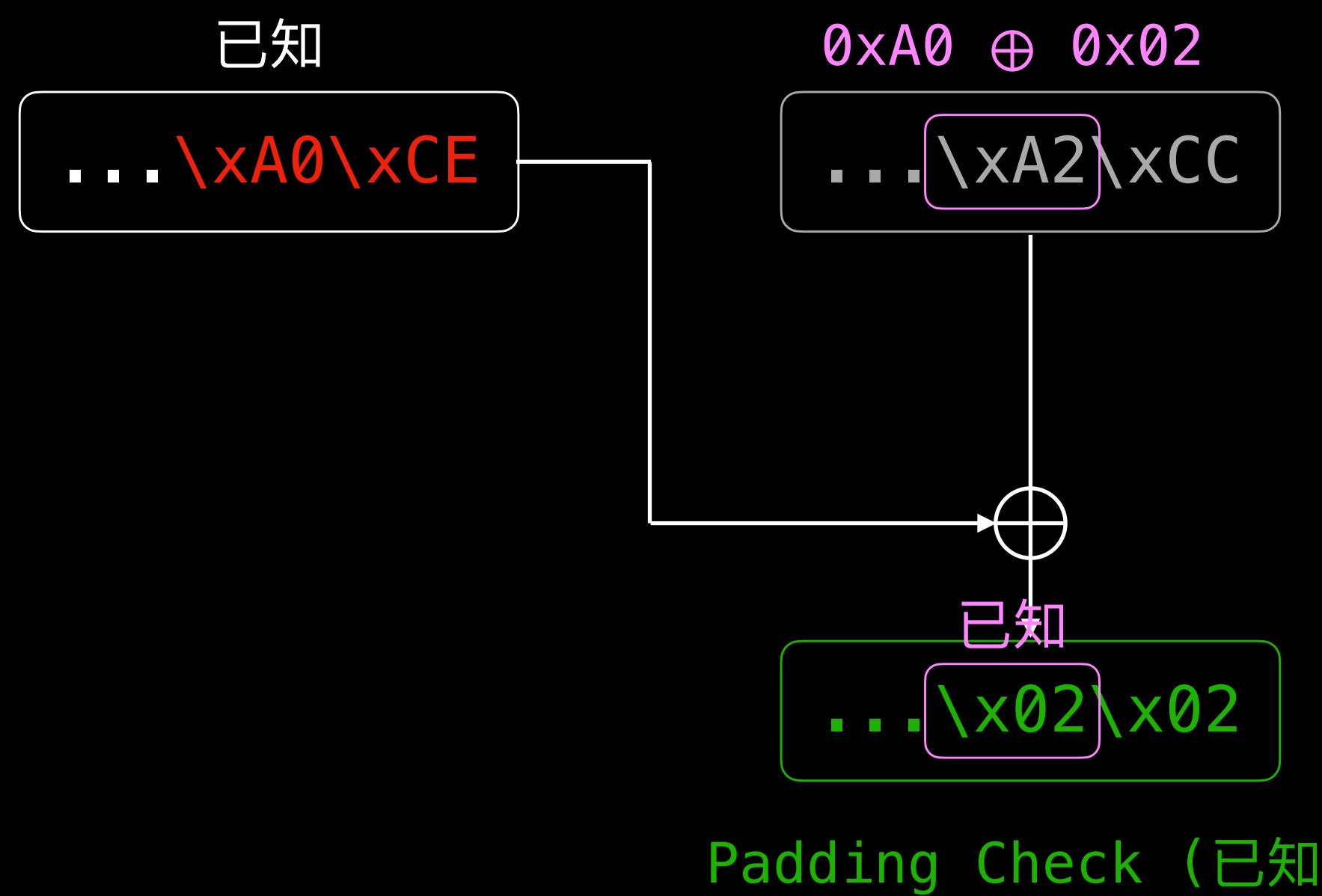
# >\_ Padding Oracle

- ▶ 找 plain2 的倒數第 2 bytes



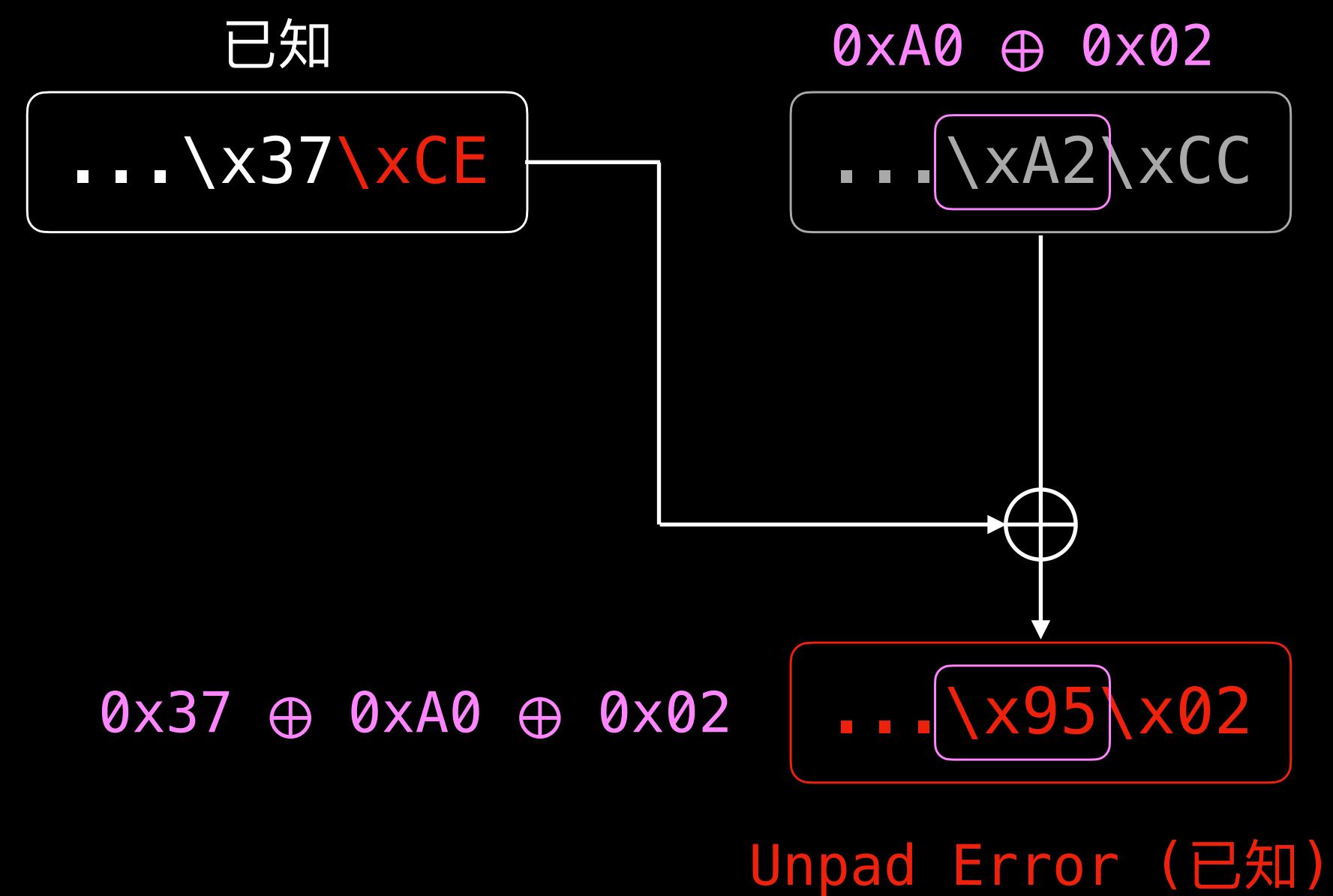
# >\_ Padding Oracle

- ▶ 找 plain2 的倒數第 2 bytes



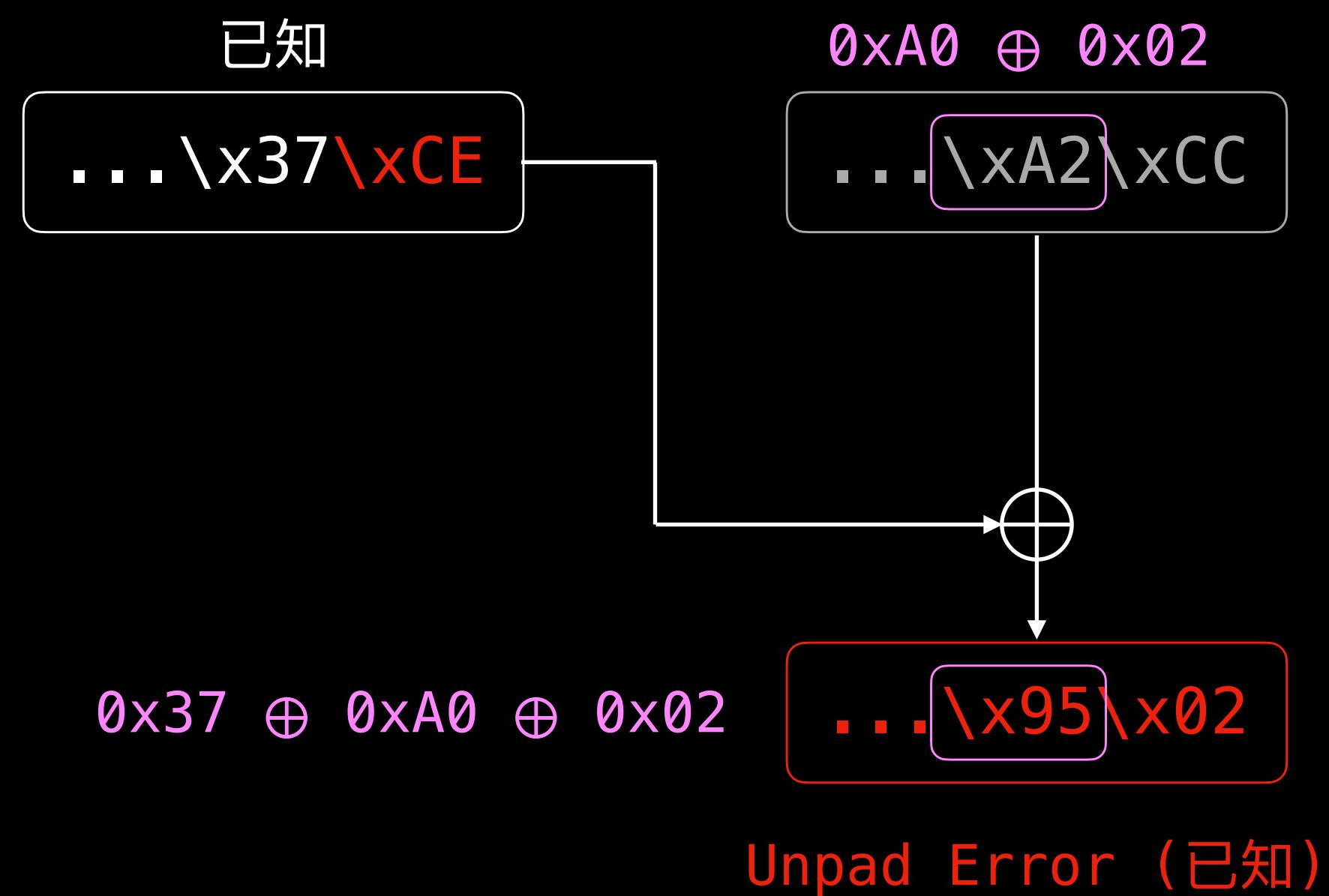
# >\_ Padding Oracle

- ▶ 找 plain2 的倒數第 2 bytes



# >\_ Padding Oracle

- ▶ 這樣就找到 plain2 的倒數第 2 bytes 了



## >\_ Padding Oracle

- ▶ 如此往復下去就可以把整個 Block 找出來了
- ▶ 找出來後可以把 cipher1 + cipher2 換成 IV + cipher1 或 cipher2 + cipher3，來找其他 Block

>\_ Padding Oracle

Lab : Padding Oracle 0

# 偽隨機數

## >\_ LCG

### 線性同餘方法

- ▶ 需要  $s_0, m, inc, N$
- ▶  $s_i \equiv m \cdot s_{i-1} + inc \pmod{N}$
- ▶  $s_i, m, inc < N$
- ▶ 如果拿到足夠多的  $s_i$  就可以把  $m, inc, N$  都推出來，進而預測偽隨機數
- ▶ LCG Attack

## >\_ LCG

- ▶ 如果我拿到一串  $s_i$ ，那我要怎麼算出  $m, inc, N$

# >\_ LCG

- ▶ 如果我拿到一串  $s_i$ ，那我要怎麼算出  $m, inc, N$

1. 求  $N$

2. 求  $m$

3. 求  $inc$

$$t_0 \equiv s_1 - s_0 \pmod{N}$$

$$t_1 \equiv s_2 - s_1 \equiv m \cdot (s_1 - s_0) \equiv m \cdot t_0 \pmod{N}$$

$$t_2 \equiv s_3 - s_2 \equiv m \cdot (s_2 - s_1) \equiv m \cdot t_1 \pmod{N}$$

$$\Rightarrow N_0 = t_0 \cdot t_2 - t_1^2 \equiv t_0 \cdot (m^2 \cdot t_0) - (m \cdot t_0)^2 \equiv 0 \pmod{N}$$

把  $N_0, N_1, N_2, \dots$  取他們的最大公因數，足夠多的話最大公因數就會是  $N$

# >\_ LCG

- ▶ 如果我拿到一串  $s_i$ ，那我要怎麼算出  $m, inc, N$

1. 求  $N$
2. 求  $m$
3. 求  $inc$

已知  $s_2 - s_1 \equiv m \cdot (s_1 - s_0) \pmod{N}$

$$\Rightarrow m \equiv (s_2 - s_1) \cdot (s_1 - s_0)^{-1} \pmod{N}$$

# >\_ LCG

- ▶ 如果我拿到一串  $s_i$ ，那我要怎麼算出  $m, inc, N$

1. 求  $N$
2. 求  $m$
3. 求  $inc$

已知  $s_1 \equiv m \cdot s_0 + inc \pmod{N}$

$$\Rightarrow inc \equiv s_1 - m \cdot s_0 \pmod{N}$$

>\_ LCG

Lab : LCG

## >\_ MT19937

### 梅森旋轉算法

- ▶ 由 seed 擴展到 624 個 state，每個 state 都 32 bits
- ▶ 每個 state 經過 state\_to\_rand 轉換後輸出（一樣 32 bits）
- ▶ 624 個 state 用完之後會用舊的 state generate 一個新的 state
- ▶ 拿到 624 個 32 bits 的 rand，反推它們的 state，然後就可以依照 MT19937 的算法預測偽隨機數
- ▶ Python 的 random module 就是用 MT19937 當作偽隨機數的產生算法
- ▶ MT19937 Attack

>\_ MT19937

Lab : MT19937

# 雜湊函數

# >\_ Hash

## 預期特性

- ▶ 對於任何長度的輸入，輸出的長度固定
- ▶ 對於兩種不同的輸入，輸出要不同(或是輸出相同的機率要小於一個值)
- ▶ 對於兩個很相近的輸入，輸出的內容不會有關連

>\_ Hash - Length Extension Attack

Lab : Length Extension Attack

RSA

# >\_ Math

## 歐拉定理

如果  $a > 1, n > 1, a, n \in \mathbb{N}$  且  $\gcd(a, n) = 1$ ，則  $a^{\varphi(n)} \equiv 1 \pmod{n}$ ，其中  $\varphi(n)$  是有多少小於  $n$  且和  $n$  互質的數

### 證明

設  $\{\alpha_1, \alpha_2, \dots, \alpha_{\varphi(n)}\}$  是  $\varphi(n)$  個不同的和  $n$  互質的數字，已知  $\gcd(a, n) = 1$ ，對  $i, j \in \{1, 2, \dots, \varphi(n)\}$  且  $i \neq j$

， $\gcd(a\alpha_i, n) = 1$  且  $a\alpha_i \not\equiv a\alpha_j$ ，所以說  $\{\alpha_1, \alpha_2, \dots, \alpha_{\varphi(n)}\}$  和  $\{a\alpha_1, a\alpha_2, \dots, a\alpha_{\varphi(n)}\}$  是同一個集合

$$\Rightarrow \alpha_1 \cdot \alpha_2 \cdot \dots \cdot \alpha_{\varphi(n)} \equiv a\alpha_1 \cdot a\alpha_2 \cdot \dots \cdot a\alpha_{\varphi(n)} \pmod{n}$$

把兩邊約一約就可以看到  $a^{\varphi(n)} \equiv 1 \pmod{n}$

# >\_ Math

## 中國剩餘定理

$$\text{if } \begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ x \equiv a_3 \pmod{m_3} \\ \dots \\ x \equiv a_n \pmod{m_n} \end{cases} \quad \text{let } \begin{cases} M = m_1 \cdot m_2 \cdot \dots \cdot m_n \\ M_i = \frac{M}{m_i} \\ t_i \equiv M_i^{-1} \pmod{m_i} \end{cases}$$

$$\Rightarrow x \equiv a_1 t_1 M_1 + a_2 t_2 M_2 + \dots + a_n t_n M_n \pmod{M}$$

# >\_ RSA

## 金鑰生成

產生兩個大質數  $p, q$  和一個正整數  $e$ ，令  $n = p \cdot q$  且  $e$  要滿足  $\gcd(e, \varphi(n)) = 1$ ， $d \equiv e^{-1} \pmod{\varphi(n)}$

- ▶ 公鑰： $(e, n)$
- ▶ 私鑰： $d$

## 加解密

- ▶ 加密： $c \equiv m^e \pmod{n}$
- ▶ 解密： $m \equiv c^d \pmod{n}$

# >\_ RSA

## 證明

因  $d \equiv e^{-1} \pmod{\varphi(n)}$ ，所以  $ed = 1 + k\varphi(n)$

如果  $\gcd(m, p) = 1$ ：

$$m^{ed} \equiv m^{1+k(p-1)(q-1)} \equiv m \cdot (m^{p-1})^{k(q-1)} \equiv m \pmod{p}$$

如果  $\gcd(m, p) = p$ ：

$$m^{ed} \equiv 0 \equiv m \pmod{p}$$

對於  $q$  也是一樣的，所以

$$\begin{cases} m^{ed} \equiv m \pmod{p} \\ m^{ed} \equiv m \pmod{q} \end{cases}$$

由貝祖定理可以知道存在  $(x, y)$  滿足  $xp + yq = 1$ ，且  $x \equiv p^{-1} \pmod{q}$  和  $y \equiv q^{-1} \pmod{p}$  時

用 CRT：

$$\begin{aligned} m^{ed} &\equiv m \cdot q \cdot (q^{-1} \pmod{p}) + m \cdot p \cdot (p^{-1} \pmod{q}) \pmod{n} \\ &\equiv m \cdot (q \cdot (y + kp) + p \cdot (x + lq)) \pmod{n} \\ &\equiv m \pmod{n} \end{aligned}$$

>\_ RSA

Lab : Calc RSA

## >\_ RSA

- ▶ Break RSA 最直觀的方法就是把  $n$  分解成  $p, q$ ，這樣就可以照著金鑰生成的方式用  $e$  生成  $d$  來解密
- ▶ FactorDB

>\_ RSA

Lab : RSA Baby 0

## >\_ Fermat Factorization

假設  $p > q$  且  $p - q$  相較於  $p$  或  $q$  很小，則假設  $p - q = 2b$

$$\begin{cases} p = a + b \\ q = a - b \end{cases} \Rightarrow n = pq = a^2 - b^2 \Rightarrow n + b^2 = a^2$$

因為  $p - q$  相較於  $p$  或  $q$  很小，所以  $b^2$  相較於  $n$  或  $a^2$  也很小，嘗試不同的  $a$  使得  $\sqrt{a^2 - n} \in \mathbb{N}$

>\_ Fermat Factorization

Lab : Fermat Factorization

## >\_ Pollard's p-1 Algorithm

如果  $p - 1$  的最大質因數  $B$  很小的話， $(p - 1) | 1 \times 2 \times \dots \times B$ ，所以：

$$2^{1 \times 2 \times \dots \times B} = 2^{k(p-1)} \equiv 1 \pmod{p} \Rightarrow \gcd(2^{1 \times 2 \times 3 \times \dots \times B} - 1, n) > 1$$

只要暴力嘗試  $2^1, 2^{1 \times 2}, 2^{1 \times 2 \times 3}, \dots$  和  $n$  最大公因數有沒有大於 1，有的話就可以分解  $n$  了

>\_ Pollard's p-1 Algorithm

Lab : Pollard's p-1 Algorithm

## >\_ Broadcast Attack

如果我們拿到  $e$  個不同的  $n$  加密相同  $m$  的結果，可以用 CRT 直接解  $m$

$$\begin{cases} m^3 \equiv c_1 \pmod{n_1} \\ m^3 \equiv c_2 \pmod{n_2} \\ m^3 \equiv c_3 \pmod{n_3} \end{cases}$$

用 CRT 可以解出來  $m^3 \equiv c \pmod{n_1 \cdot n_2 \cdot n_3}$

因為  $m < n_1, m < n_2, m < n_3$ ，所以說  $m^3 < n_1 \cdot n_2 \cdot n_3$ ，直接對 CRT 解出來的  $c$  開三次方就可以得到  $m$

>\_ Broadcast Attack

Lab : Broadcast Attack

## >\_ Common Modulus Attack

如果用兩把公鑰  $(n, e_1), (n, e_2)$  加密  $m$ ，且  $\gcd(e_1, e_2) = 1$ ，我們可以利用擴展歐幾里德算法直接解  $m$ 。  
令  $c_1 \equiv m^{e_1} \pmod{n}$ ,  $c_2 \equiv m^{e_2} \pmod{n}$ ，因為  $\gcd(e_1, e_2) = 1$ ，所以可以用擴展歐幾里德算法求出  $(s_1, s_2)$  滿足  
 $s_1 \cdot e_1 + s_2 \cdot e_2 = 1$ ，故  $c_1^{s_1} \cdot c_2^{s_2} \equiv m^{e_1 \cdot s_1} \cdot m^{e_2 \cdot s_2} = m$

>\_ Common Modulus Attack

Lab : Common Modulus Attack

## >\_ Wiener Attack

如果私鑰  $d$  很小的話，可以用 Wiener Attack 去分解  $n$

- ▶ Wiener Attack

>\_ Wiener Attack

Lab : Wiener Attack

## >\_ 關於 RSA Attack

- ▶ 我的筆記
- ▶ CTFLib

>\_ 回饋表單

南區回饋表單連結

北區回饋表單連結