

# Cryptography 2

Ice1187 2024/03/15

# Speaker

黃俊嘉 (Ice1187)

- ▶ TNFSH 百八級
- ▶ Master's student at NTU CSIE NSLab
- ▶ Member of Balsn CTF Team
- ▶ Member of UNDEFINED
- ▶ Intern Researcher at CyCraft
- ▶ Speaker of CyberSec, SECCON



✉ [hcc001202@gmail.com](mailto:hcc001202@gmail.com)

GitHub <https://github.com/Ice1187>

# 聲明

本課程目的在提升學員對資安產業之認識及資安實務能力。本課程所授與的知識和技巧僅做為資安實務教育訓練目的。

所有課程學習內容應於雙方知情、同意且合法的情況下進行實作和練習，並且不得從事非法攻擊或違法行為，以免受到法律制裁。請勿利用所習得之技術從事非法或惡意的攻擊及入侵行為！

# 大綱

- Python 環境安裝與語法複習
- 現代密碼學：區塊加密
- AES 加解密演算法
- 區塊加密工作模式
  - 運作原理
  - 弱點與攻擊



# Python 環境安裝

- `python3 -m pip --version`
- `python3 -m pip install pycryptodomex pwntools`

# Python 複習

- str
  - 對字串的抽象表達，底層為 UTF-8
  - 不能直接對 str 的元素做運算
- bytes
  - 儲存 raw data 的 bytes
  - 操作概念與 C 中的 char 相同
  - 可以直接對 bytes 的元素做運算
- bytearray
  - mutable：可以對內容做修改的 bytes

變數型別

```
a = 1                      # int
b = [1, 2, 3]                # list
c = 'hello'                  # str
d = b'hello'                 # bytes
e = bytes([0x65])            # bytes
f = bytearray()               # bytes (mutable)
```

成員運算子

```
0      in [1, 2, 3]    # False
'a'    in [1, 'a']     # True
'he'   in 'hello'      # True
'heo'  in 'hello'      # False
```

# Python 複習

- str.index(c)
  - 回傳 c 在 str 中的第幾個位置
- list.append(e)
  - 將 e 添加為 list 的最後一個元素
- int.to\_bytes(n, endian)
  - 將 int 以 endian 轉成長度為 n 的 bytes
- int.from\_bytes(bytes, endian)
  - 將 bytes 以 endian 轉成 int

## 常用的型態內建 Methods

```
str.index(c)
list.append(e)
int.to_bytes(n, endian)
int.from_bytes(bytes, endian)
```

## 範例

```
'abcde'.index('d')           # 3
[1, 2].append(3)              # [1, 2, 3]
(16).to_bytes(1, 'big')       # 'b\x10'
int.from_bytes(b'\x10', 'big') # 16
```

# Python 複習

- `chr(int)`
  - 將 int 轉成 ASCII 對應的 str 字元
- `ord(str)`
  - 將 str 字元轉成 ASCII 對應的 int
- `len(str), len(list)`
  - 取得 str、list 的長度
- `print()`
  - 將參數印出到螢幕

常用內建函數

```
chr(65)          # 'A'  
ord('A')         # 65  
len('abc')       # 3  
len([1, 2, 3])   # 3  
print('hello')    # hello
```

# Python 複習

- string

- ascii\_lowercase : 小寫字母
- ascii\_letters : 小、大寫字母

```
from string import ascii_lowercase, ascii_letters, digits
ascii_lowercase # 'abc...z'
ascii_letters   # 'abc...zABC...Z'
digits         # '0123456789'
```

- digits : 數字

- random

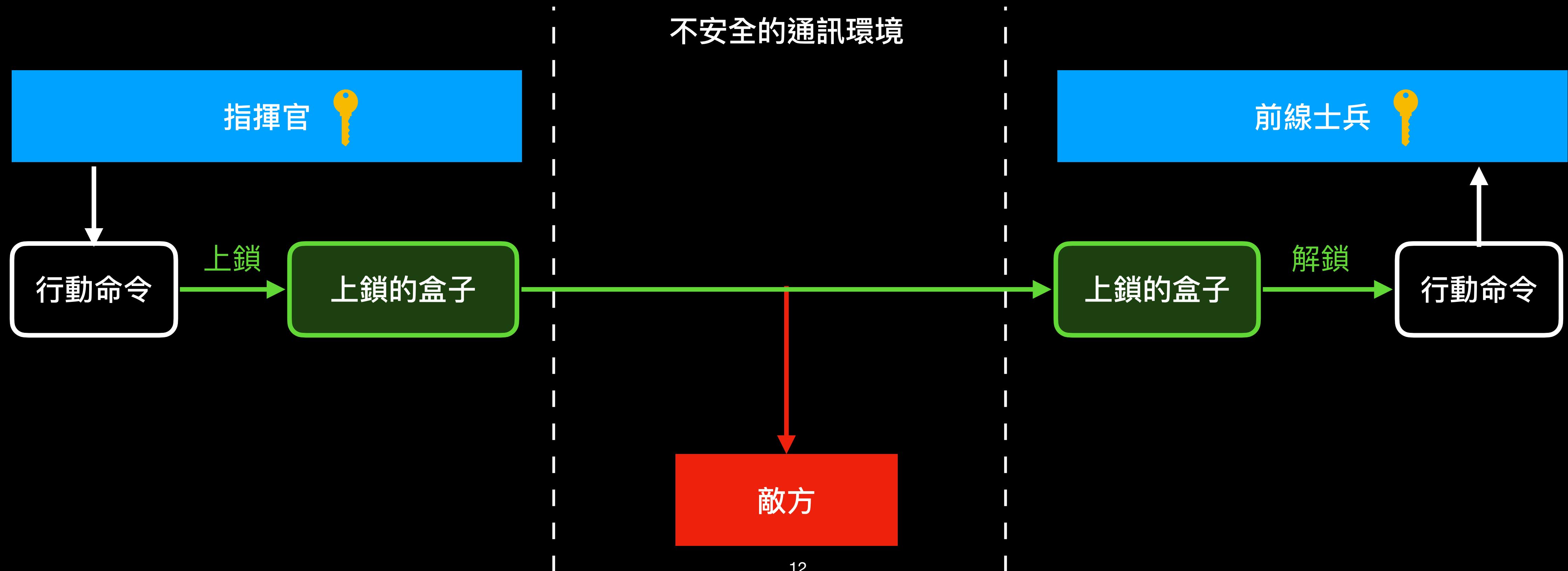
- choices(list, k=1) : 從 list 中選 k 個
- randint(a, b) : 回傳 n ,  $a \leq n \leq b$

```
import random
random.choices(list, k=1)
random.randint(a, b)
```

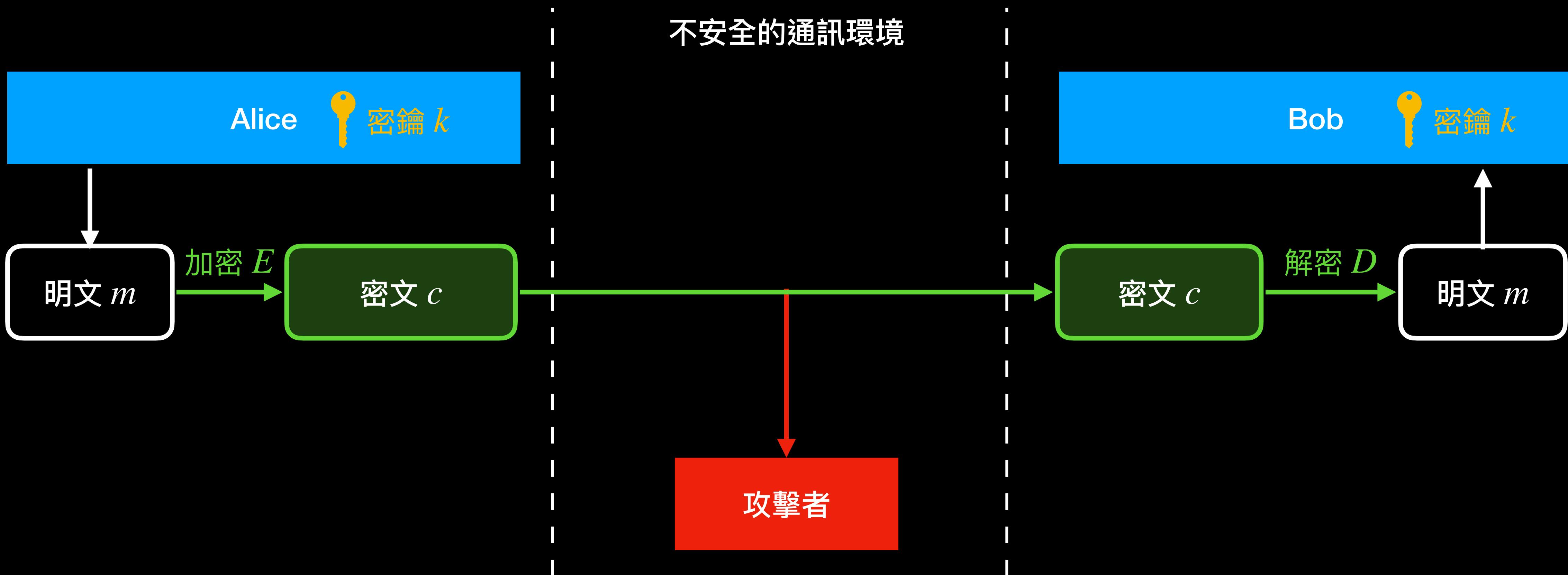
# 現代密碼學

# 密碼學簡介

- 在不安全的環境中建立安全通訊管道的技術，如：戰爭時的軍事通訊



# 名詞與符號



# 密碼學名詞與符號

## Terminology

- 明文 *plaintext / m*：要傳遞的訊息
- 密文 *ciphertext / c*：經過加密後的明文
- 密鑰 *key / k*：加解密時使用的「鑰匙」，必須避免洩漏以確保加密安全性
- 加密 *encryption / E*：將明文轉換成密文的程序， $E(m, k) = c$
- 解密 *decryption / D*：將密文轉換成明文的程序， $D(c, k) = m$

# 現代密碼學

- 以數學為基礎設計，經過多年公開的檢視與研究，被公認為安全的加密演算法
  - 安全性
  - 計算成本
  - 計算時的其他特點（軟體優化、泛用性、複雜程度）
- 安全性並非經過數學證明，最新研究或算力提升可能使其變得不安全
  - DES、RC4

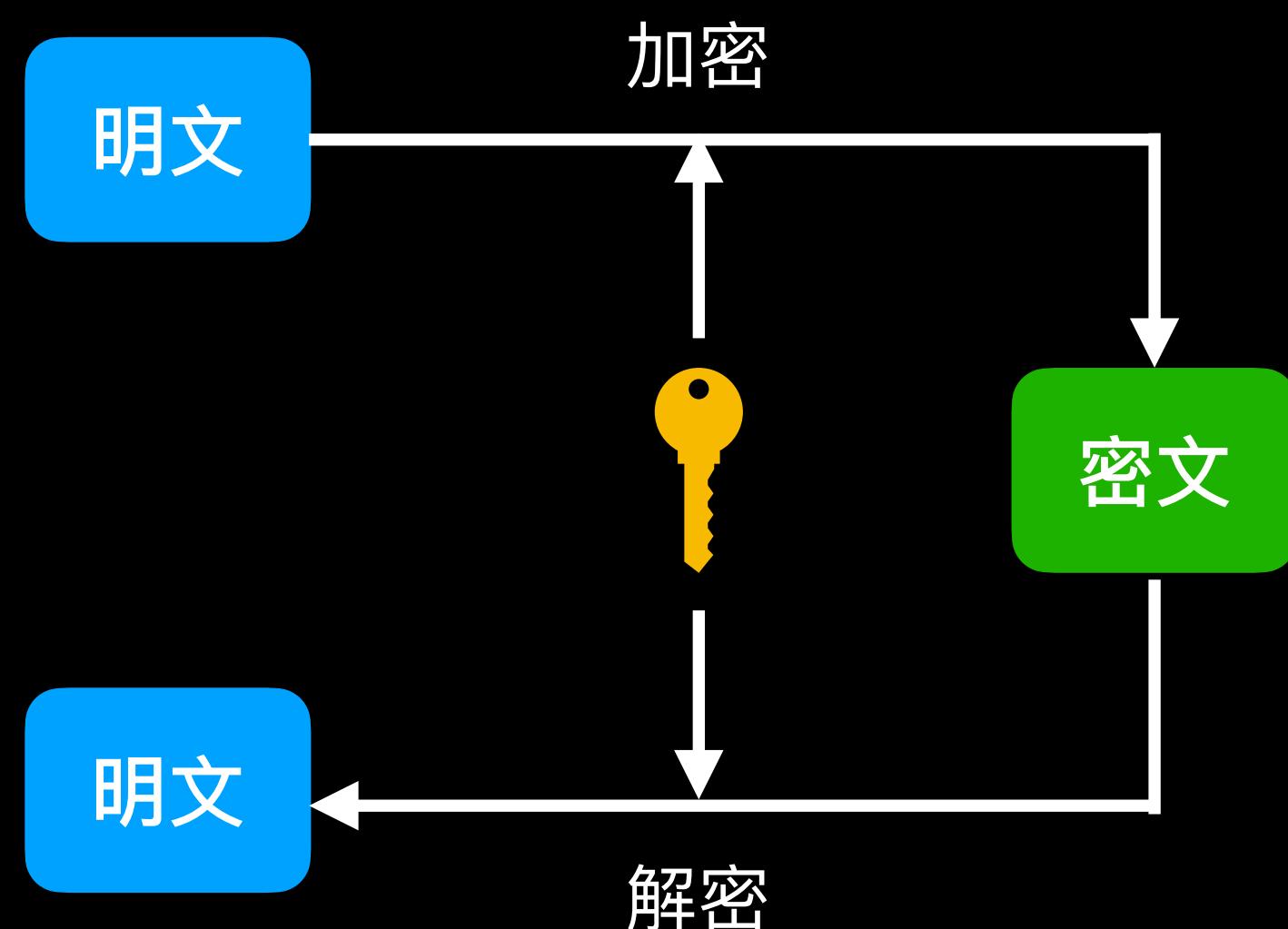
# 現代密碼學的分類

- 對稱式加密
  - 串流加密
  - 區塊加密
- 非對稱式加密

# 對稱式加密 v.s. 非對稱式加密

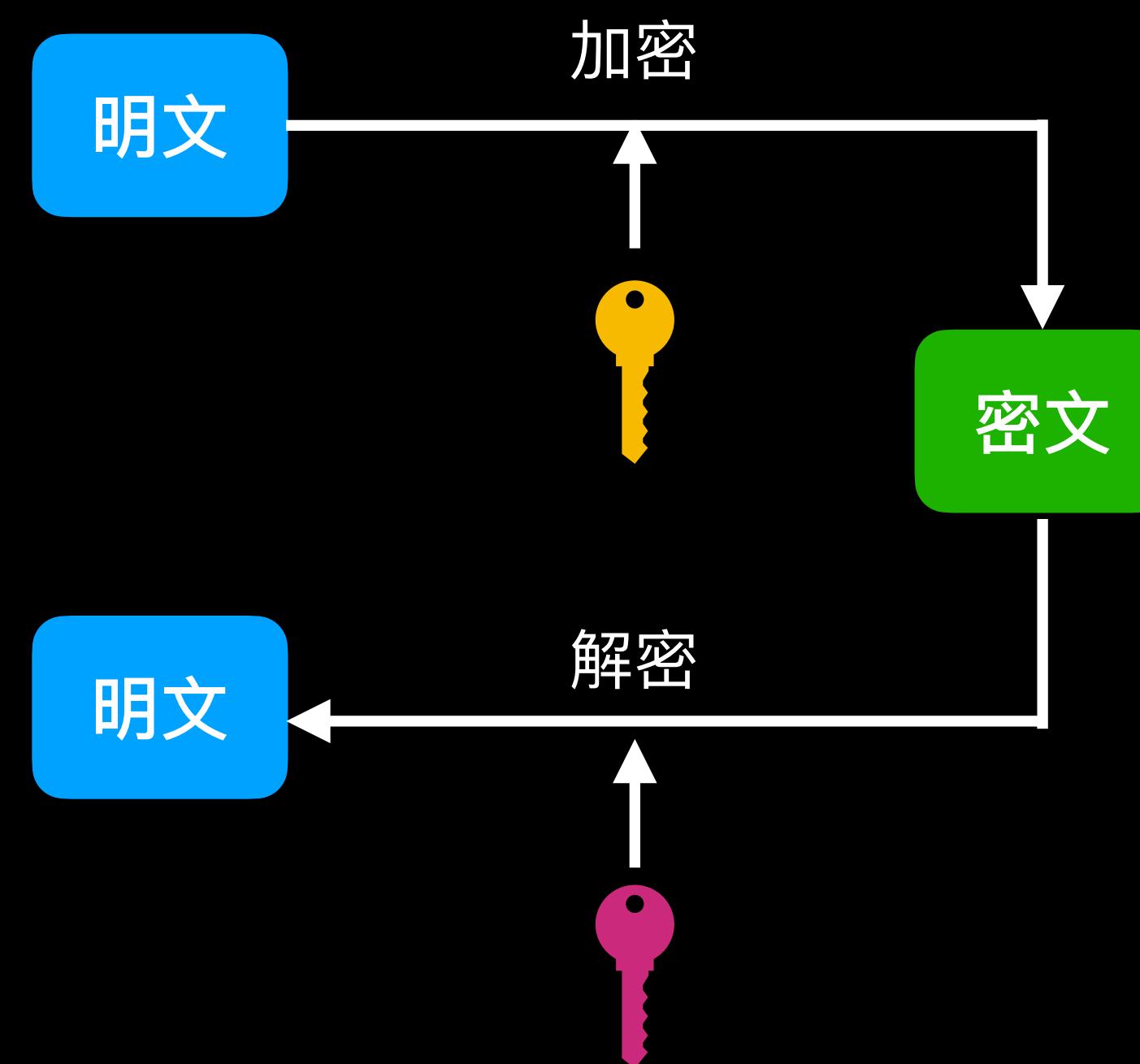
## Symmetric v.s. Asymmetric Encryption

對稱式加密



加、解密使用**相同**金鑰

非對稱式加密



加、解密使用**不同**金鑰

# 串流加密 v.s. 區塊加密

## Stream Cipher v.s. Block Cipher

- 串流加密

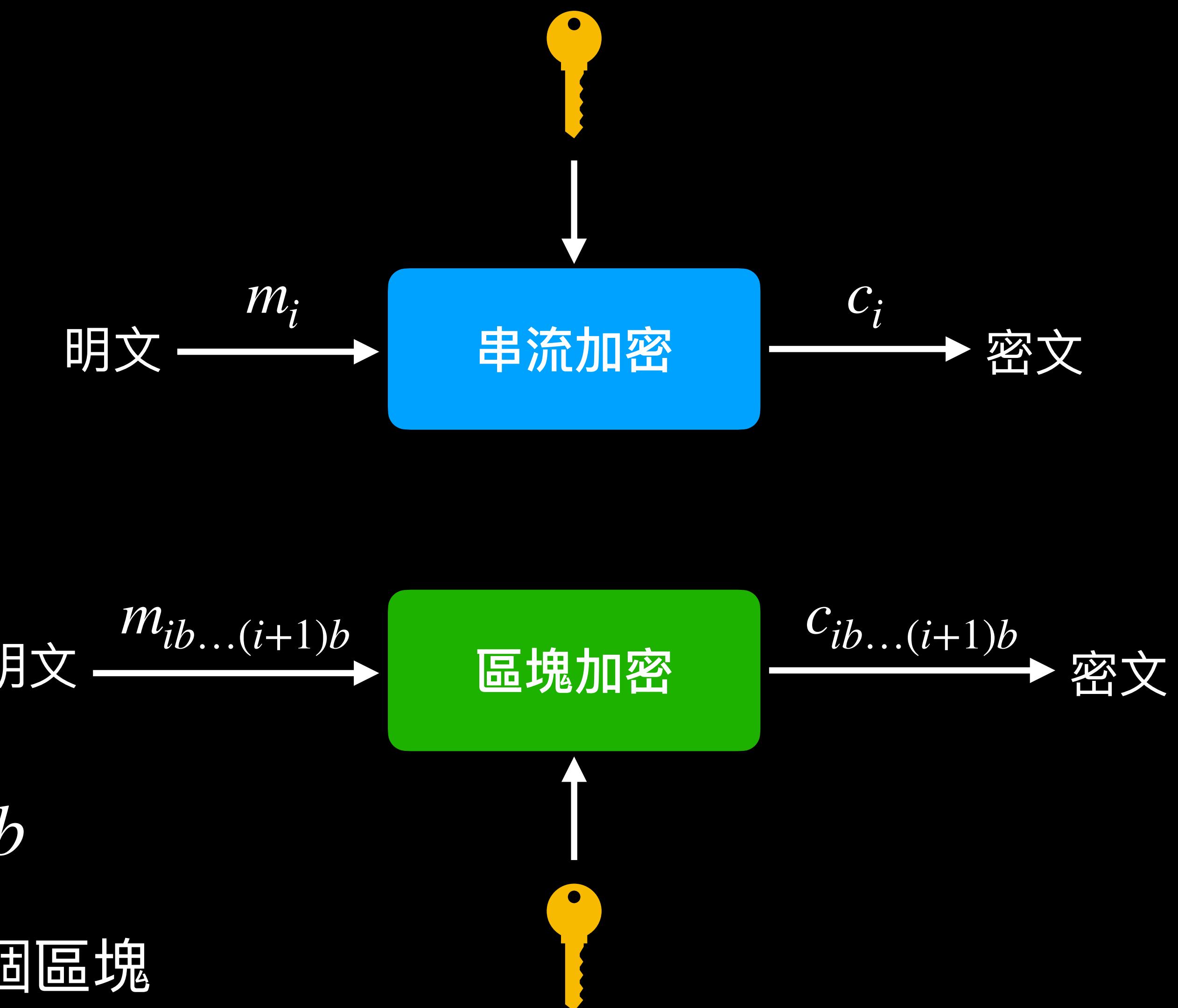
- 一次加密一個 bit

- 區塊加密

- 一次加密一個區塊

- 區塊為加密演算法預先定義的大小  $b$

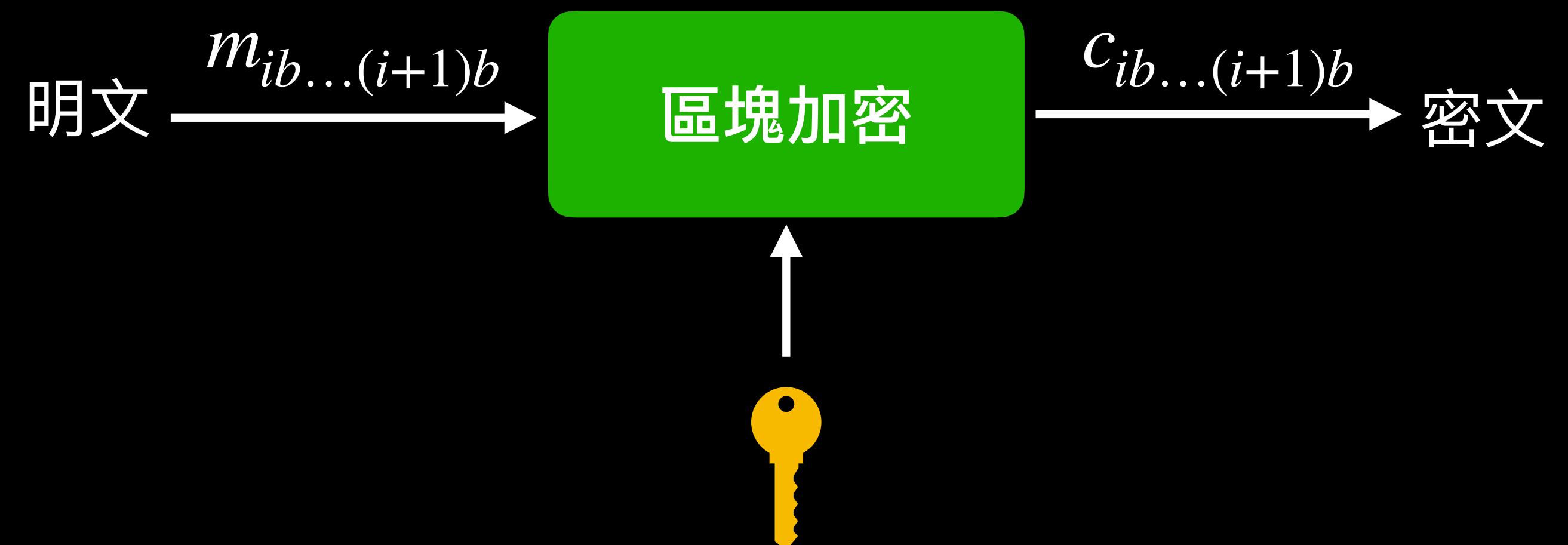
- 剩餘資料不足時需做填充，補足一個區塊



# 區塊加密 Block Cipher

# 區塊加密

- 基於乘積加密 (Product Cipher) 概念設計的加密方式
- 加密： $E(m_{ib\dots(i+1)b}, k_i) = c_{ib\dots(i+1)b}$
- 解密： $D(c_{ib\dots(i+1)b}, k_i) = m_{ib\dots(i+1)b}$



# 密碼分析 Cryptanalysis

## Product Cipher

- 在不知道密鑰的情況下，破解密文的技術
  - 數學分析
  - 統計
    - 頻率分析
  - 旁通道攻擊

# 乘積加密

## Product Cipher

- Shannon 提出的一套建立能抵抗密碼分析的加密系統的方式
  - 混淆 Confusion：模糊密文與密鑰間的統計特徵
  - 擴散 Diffusion：將明文的統計特徵均勻分散於密文中

# 混淆 Confusion

- 目的：模糊密文與密鑰之間的關聯性
  - 即使攻擊者能得知明文的對應密文，也無法推論出密鑰
- 理想：使輸入與輸出之間的非線性關係和差分均勻性最大化
- 作法：代換盒 Substitution-Box

S <sub>5</sub>		Middle 4 bits of input															
		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Outer bits	00	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

# 擴散 Diffusion

- 目的：將明文的統計特徵均勻分散於密文中
- 理想：改變明文的 1 bit，會導致平均約一半的密文 bit 被改變
- 作法：置換盒 Permutation-Box

01001011  $\Rightarrow$  10110111

01000011  $\Rightarrow$  00111001

# 乘積加密

## Product Cipher

- Shannon 提出的一套建立能抵抗密碼分析的加密系統的方式
  - 混淆 Confusion：模糊密文與密鑰間的統計特徵  $\rightarrow$  S-box
  - 擴散 Diffusion：將明文的統計特徵均勻分散於密文中  $\rightarrow$  P-box
- S-box 與 P-box 單獨使用都容易被密碼分析破解
- 乘積加密：組合使用 S-box 與 P-box 來建構加密系統
- 現今主流的區塊加密都是基於乘積加密的概念進行設計，如：DES、AES

# AES 簡介

## Advanced Encryption Standard

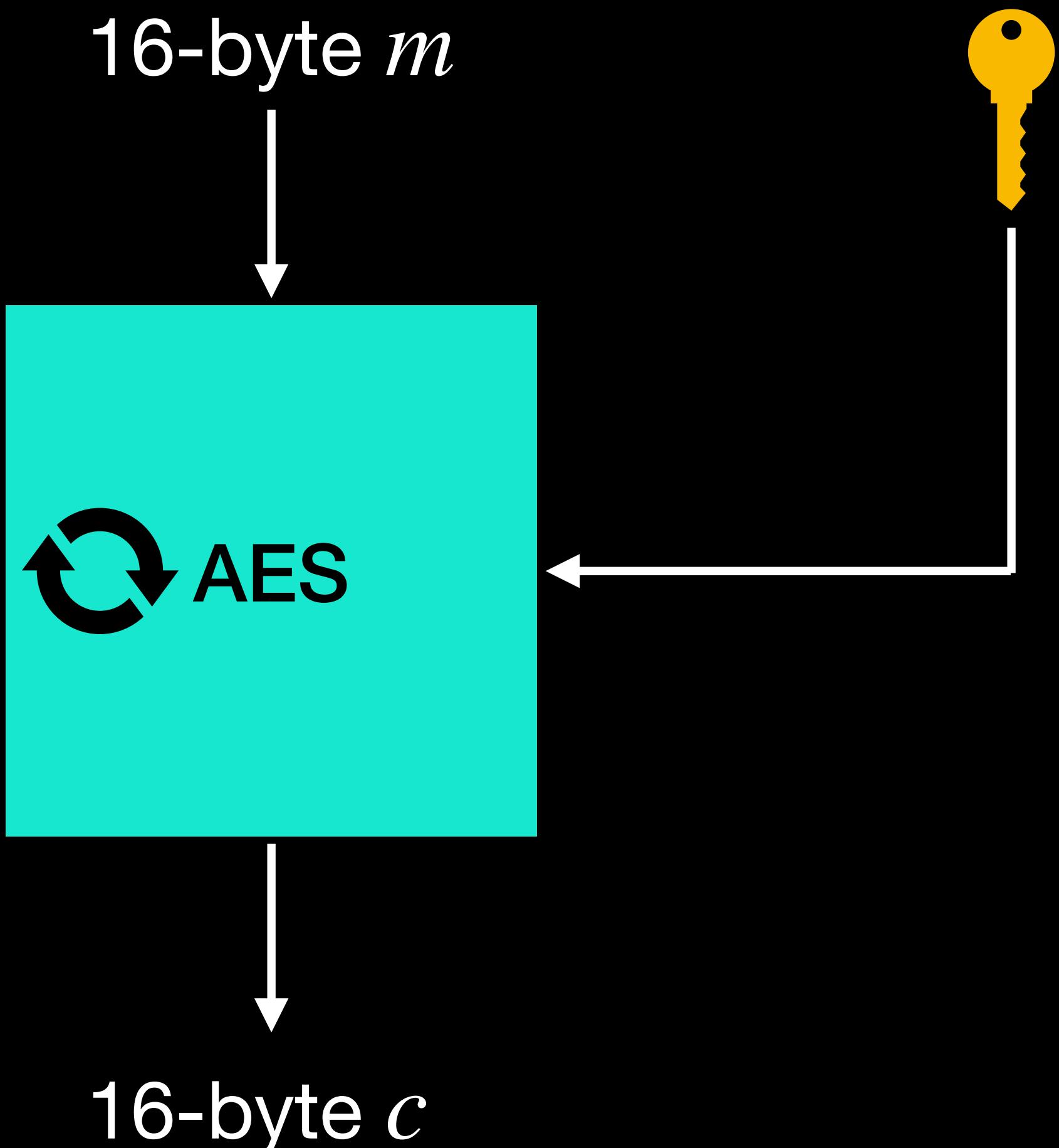
- 世界上最廣泛使用的對稱式區塊加密演算法之一
- 1997 年由美國國家標準暨技術研究院 NIST 公開徵選
  - block size: 128 bit
  - key length: 128 / 192 / 256 bit
  - 適合於軟體和硬體進行運算
- 經 NIST 與學術社群篩選，由 *Rijndael* 勝出，成為 AES 演算法

# AES 應用

- 加密通訊系統
  - Wi-Fi
  - SSH
  - Skype
  - IPSec VPN
  - TLS
  - ...
- 密碼學元件
  - 串流加密 Stream Cipher
  - 偽隨機數生成器 PRNG
  - 雜湊函數 Hash Function

# AES 計算過程

- Input  $x$  / Block size: 16 bytes
- Key  $k$  length: 16 / 24 / 32 bytes
- 重複計算 10 / 12 / 14 回合



# AES 計算過程

每回合由下列元件組成：

- 生成該回合使用的 round key
  - Key Transform
- 加密明文
  - Byte Substitution
  - ShiftRows
  - MixColumns
  - Key Addition

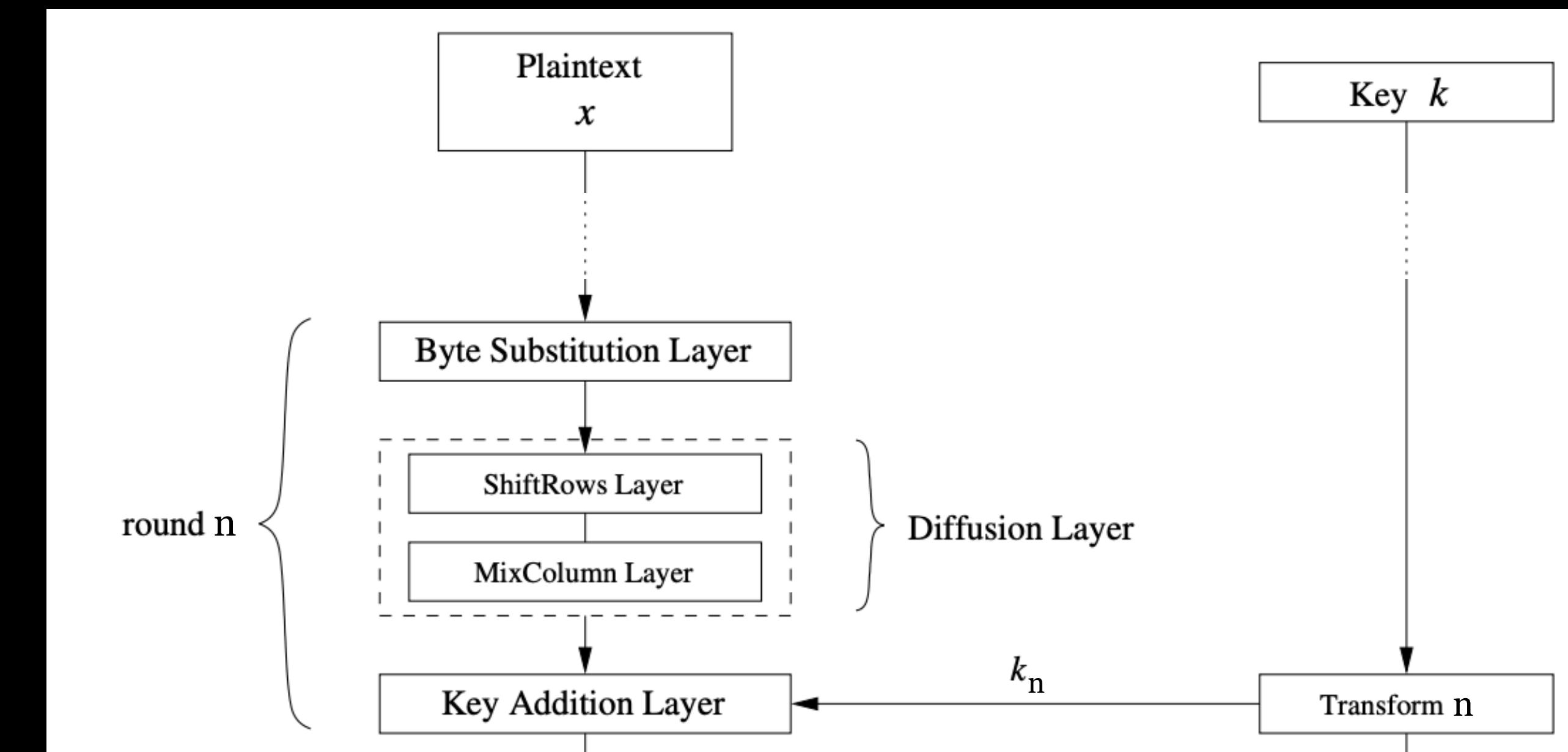
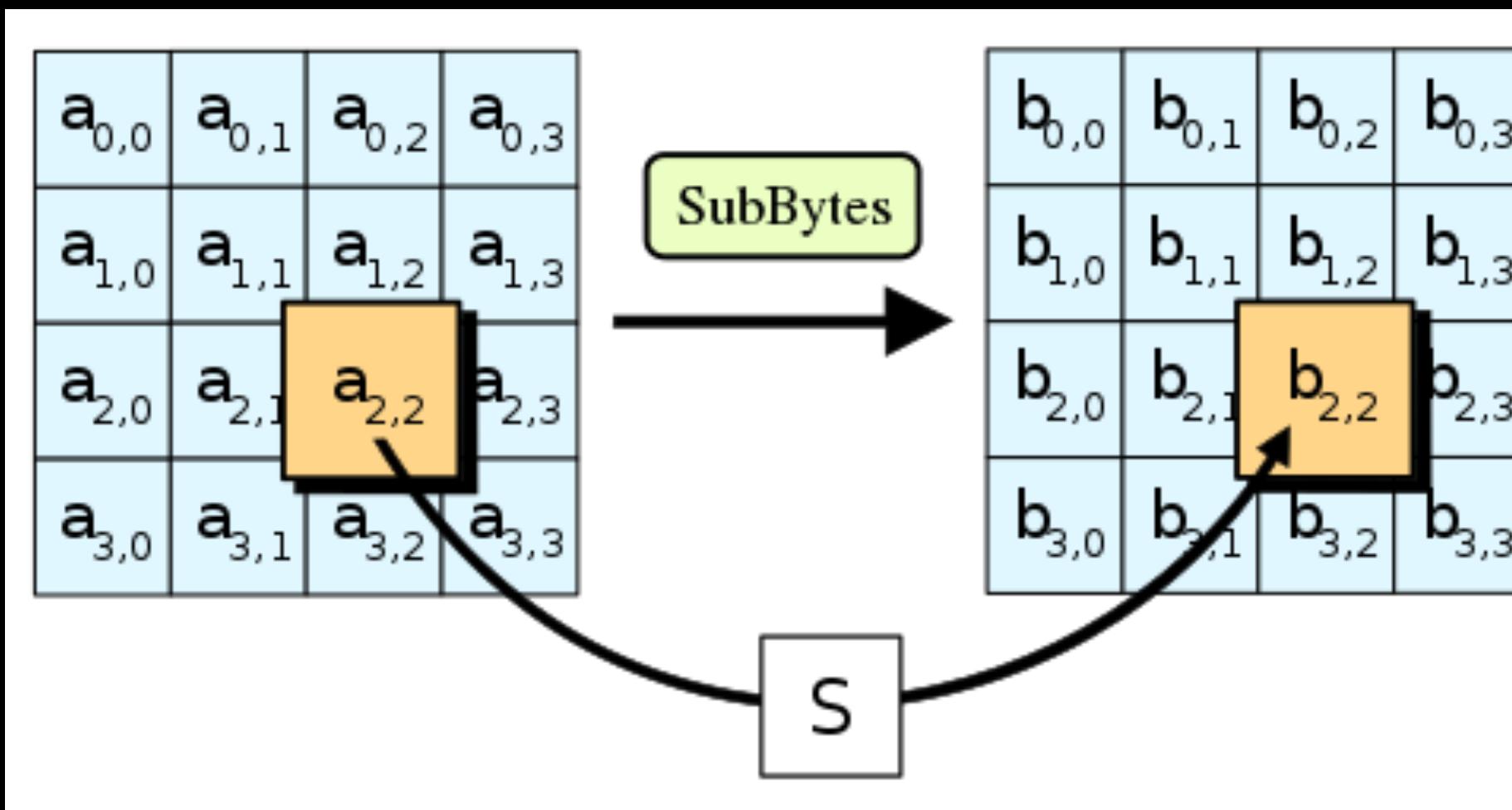


Image modified from:  
Understanding Cryptography A Textbook for Students and Practitioners

# AES 計算過程 - Byte Substitution

- 將輸入 bytes 依 S-box 進行替換
- 混淆：模糊密文與密鑰間的統計特徵
- 範例： $a_7 \rightarrow 5c$

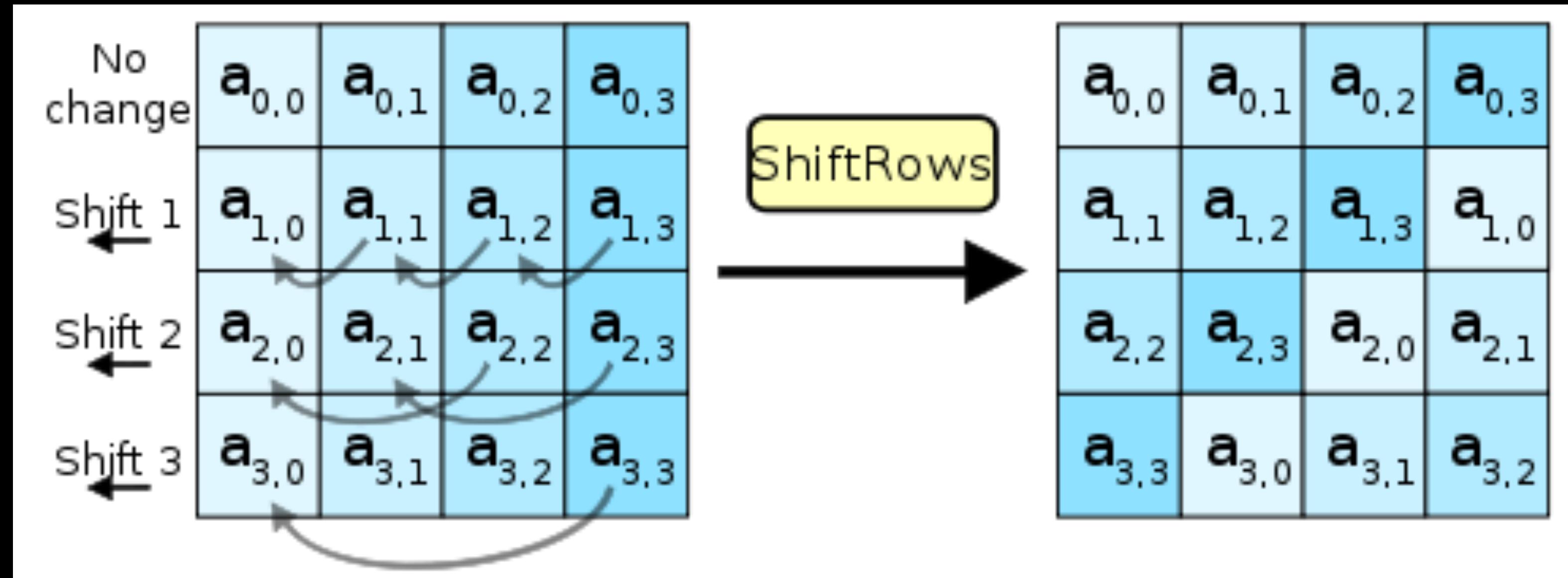


	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

AES S-box, src: [https://en.wikipedia.org/wiki/Rijndael\\_S-box](https://en.wikipedia.org/wiki/Rijndael_S-box)

# AES 計算過程 - ShiftRows

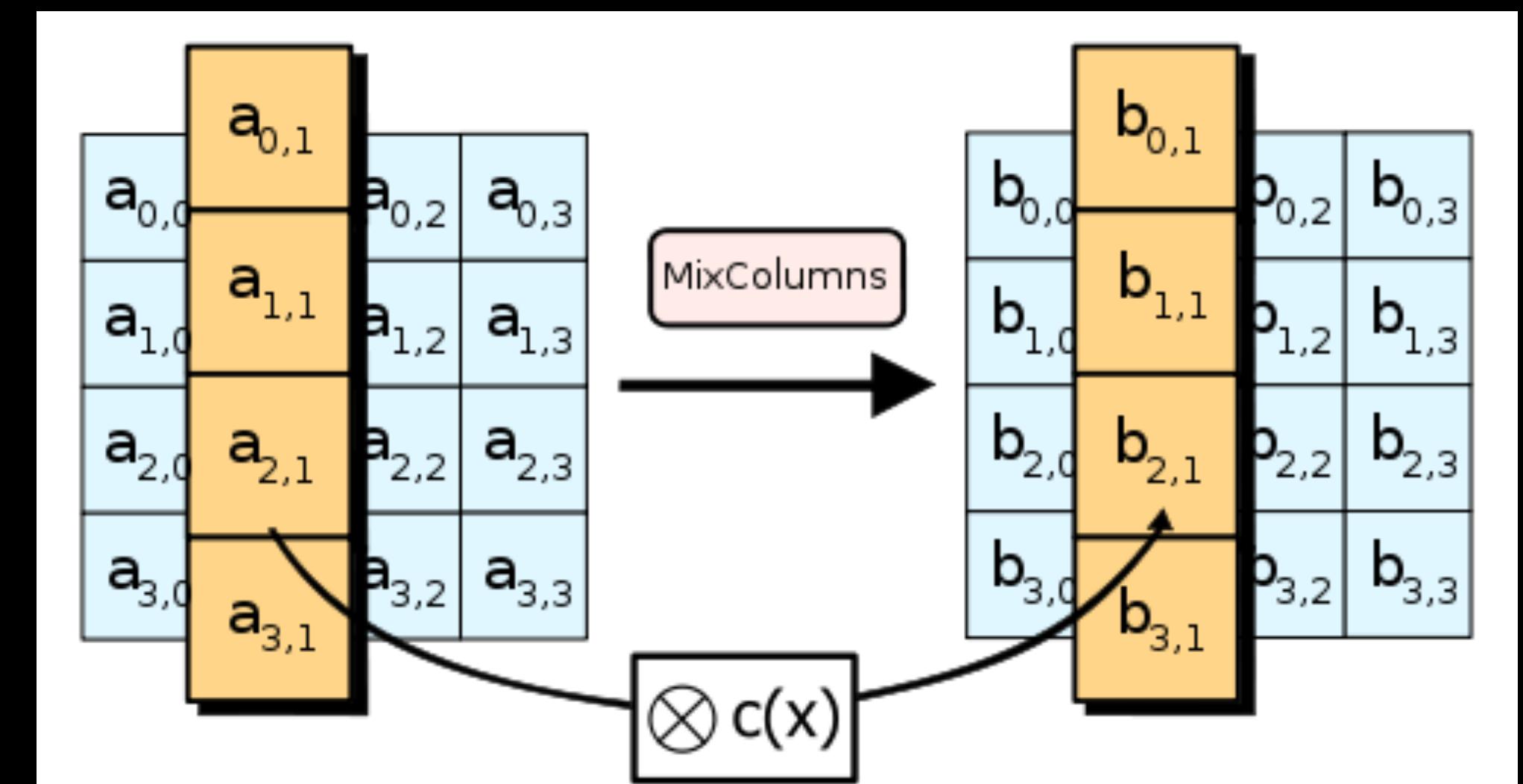
- 每個 row 向左平移
- 擴散：將明文的統計特徵均勻分散於密文中



# AES 計算過程 - MixColumns

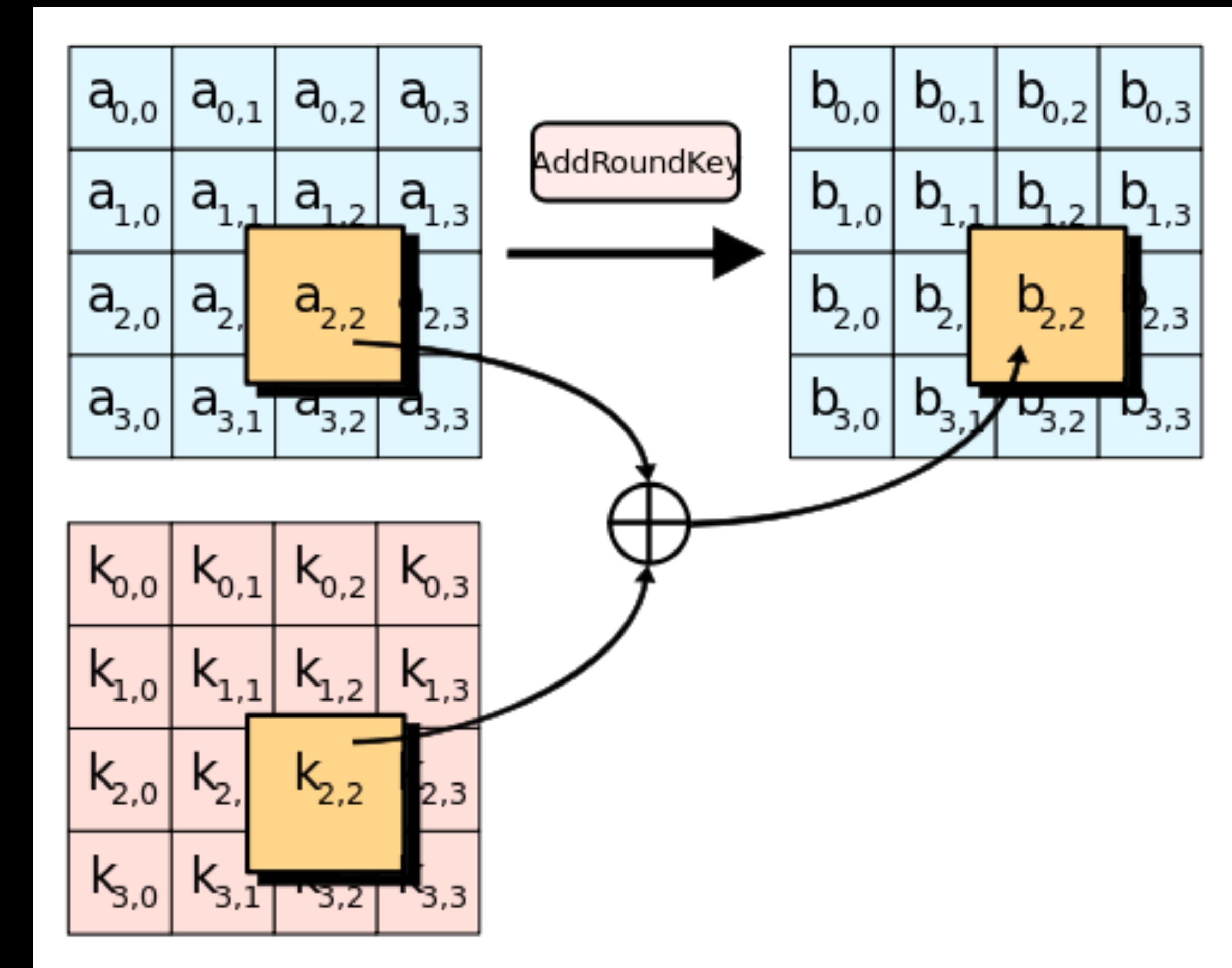
- 將同 column 數值進行混合的多項式運算
- 擴散：將明文的統計特徵均勻分散於密文中
- 參見 Rijndael MixColumns

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \quad 0 \leq j \leq 3$$



# AES 計算過程 - Key Addition

- 將輸入與該回合的 round key 做 XOR

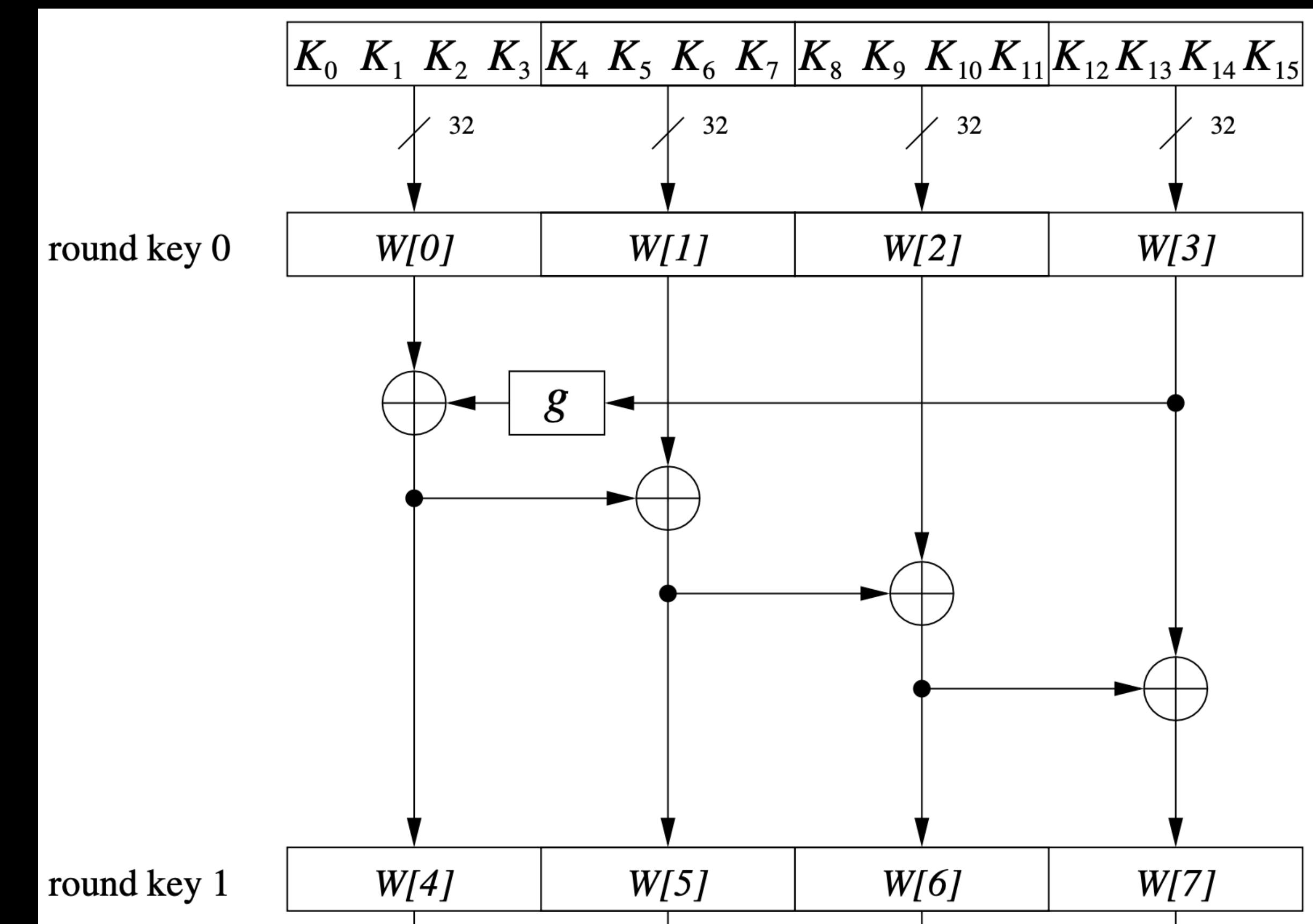


# AES 計算過程 - Key Transform

- 每一回合使用不同的 key 進行加密
- 傳入的密鑰 : key  $k$
- 該回合使用的 key : round key  $k_i$

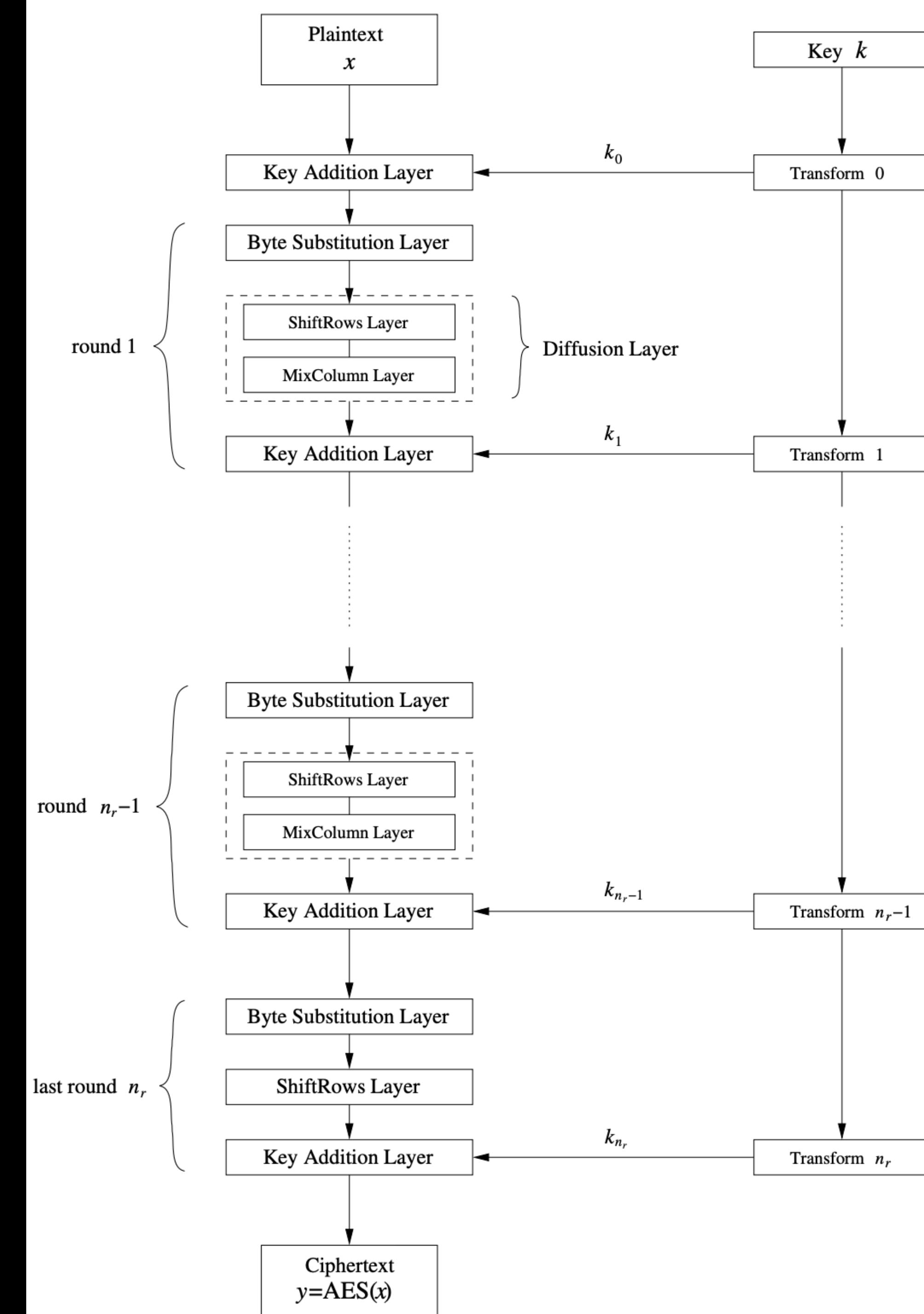
$$k_0 = \text{Transform}(k)$$

$$k_{i+1} = \text{Transform}(k_i)$$



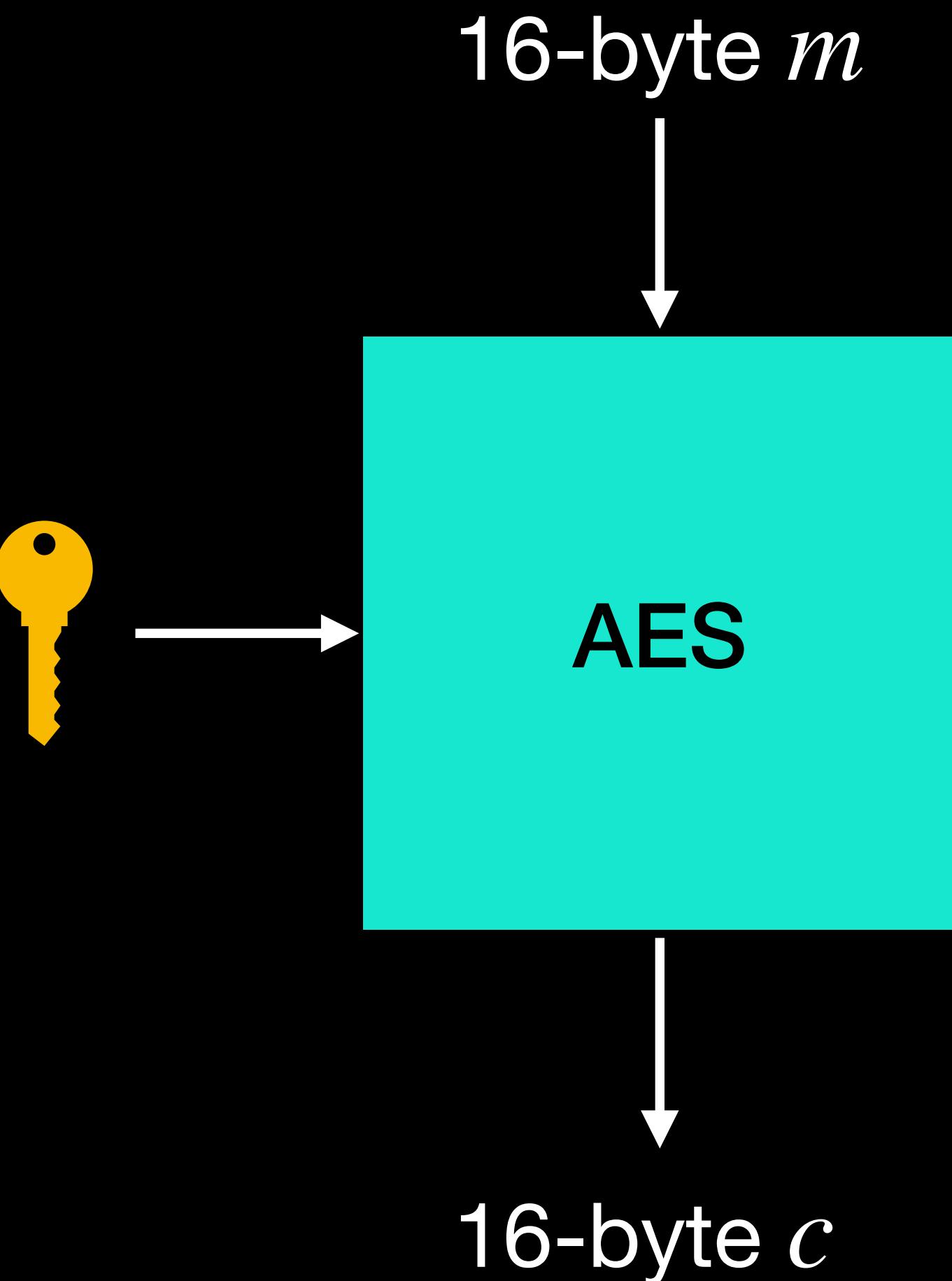
# AES 計算過程

- Input  $x$  / Block size: 16 bytes
- Key  $k$  length: 16 / 24 / 32 bytes
- 重複計算 10 / 12 / 14 回合
- 注意
  - 第一回合前進行 Key Addition
  - 最後回合沒有 MixColumns
- 解密：每個過程皆存在逆運算



# 小結 : AES

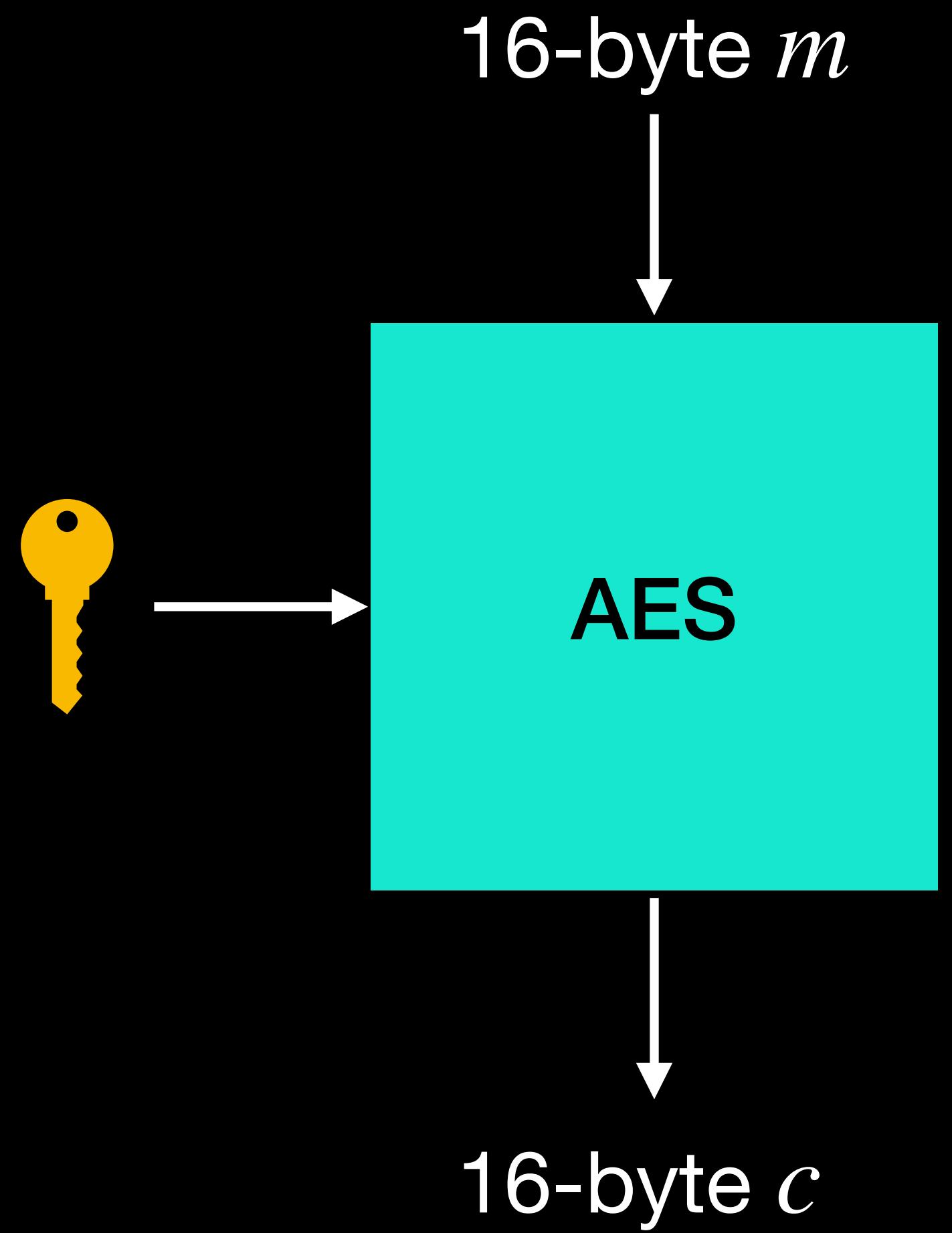
- 輸入 : 16-byte 明文  $m$
- 輸出 : 16-byte 密文  $c$
- 密鑰長度 : 16 / 24 / 32 bytes
- 可將 AES 視作一個黑箱函數
  - 加密 :  $AES(m_{16 \text{ bytes}}, k) = c_{16 \text{ bytes}}$
  - 解密 :  $AES^{-1}(c_{16 \text{ bytes}}, k) = m_{16 \text{ bytes}}$



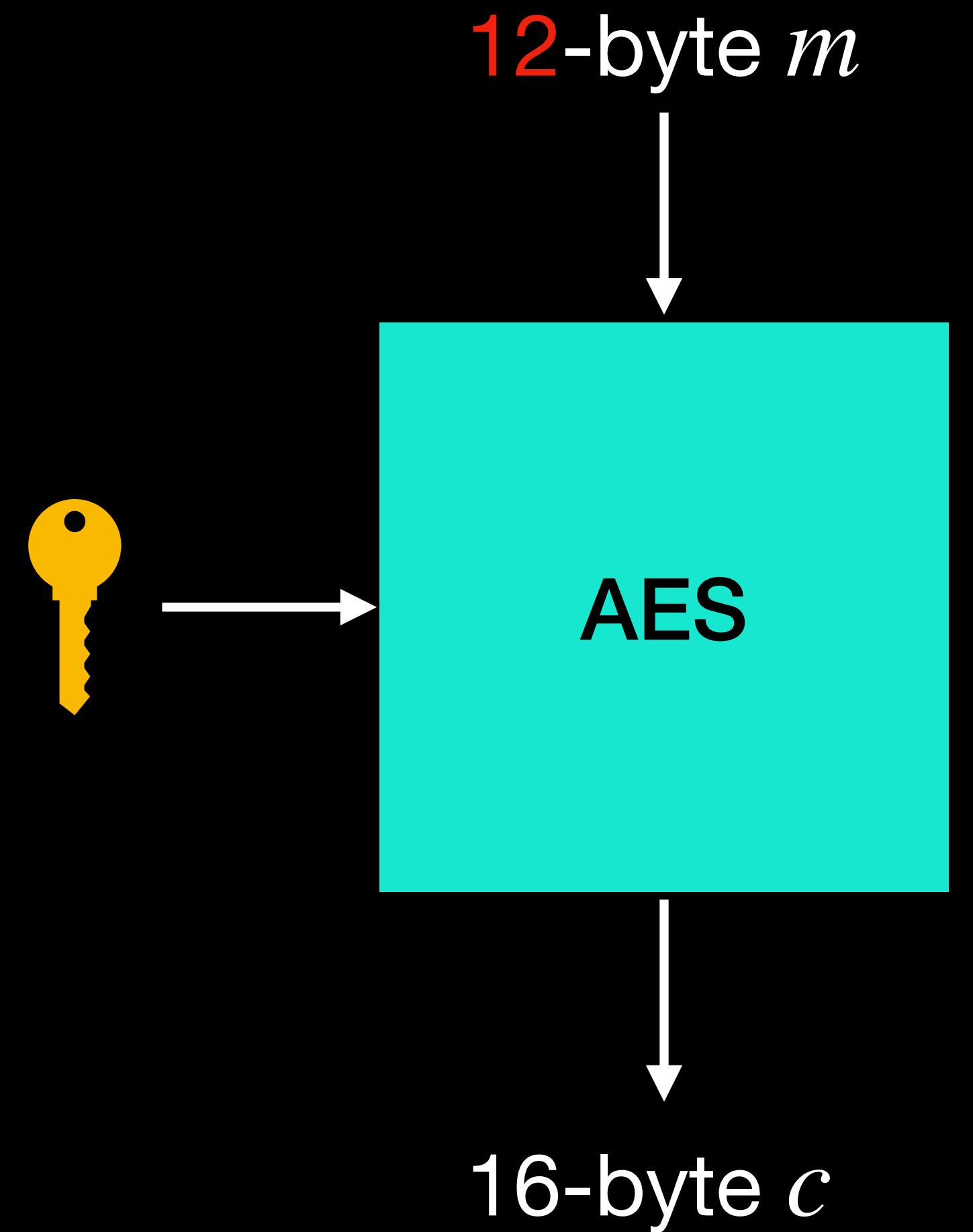
休息一下~

# 填充與工作模式

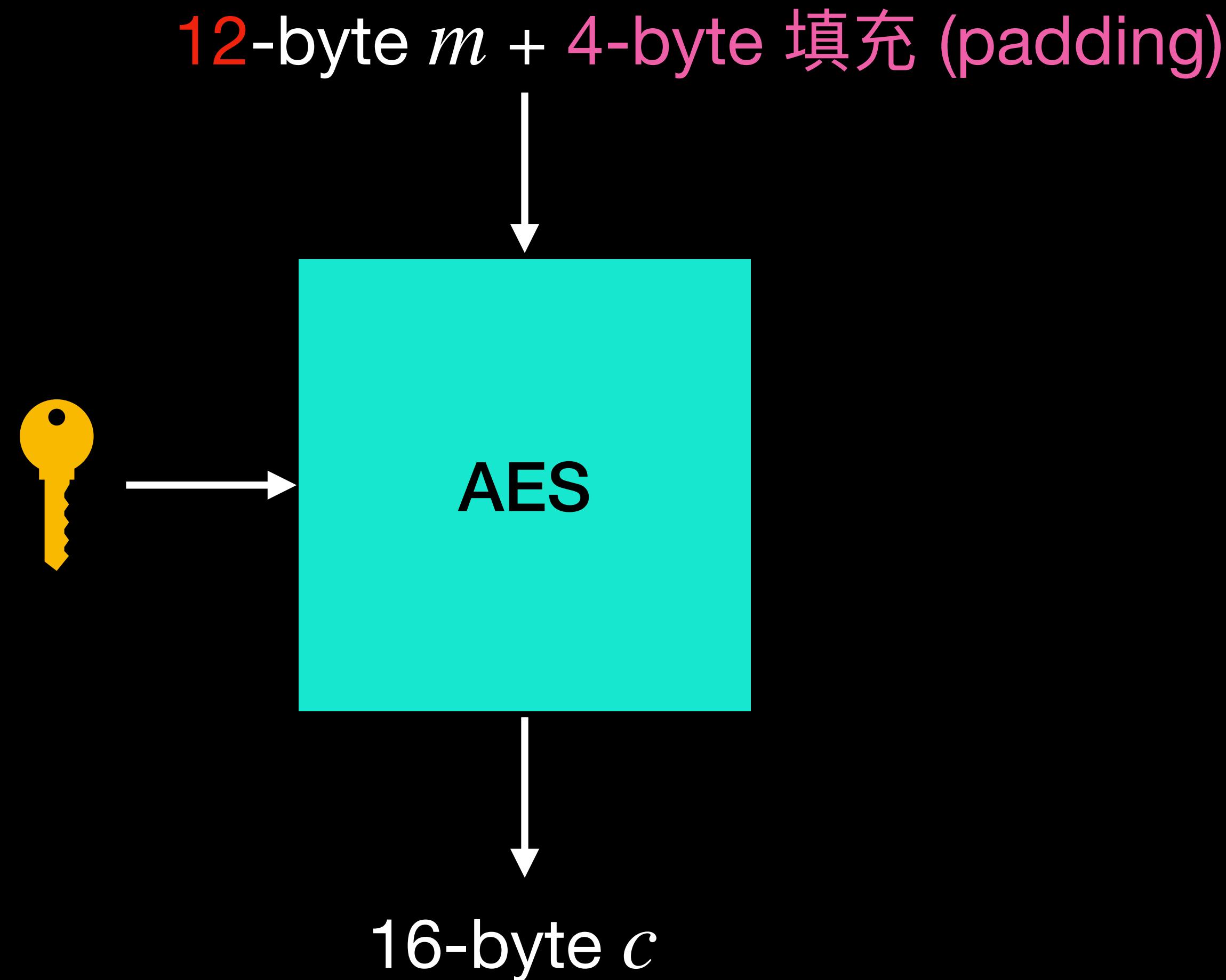
# AES



# 資料長度不足 16 bytes 怎麼辦？



# 資料長度不足 16 bytes 怎麼辦？→ 填充 Padding



# 填充 Padding

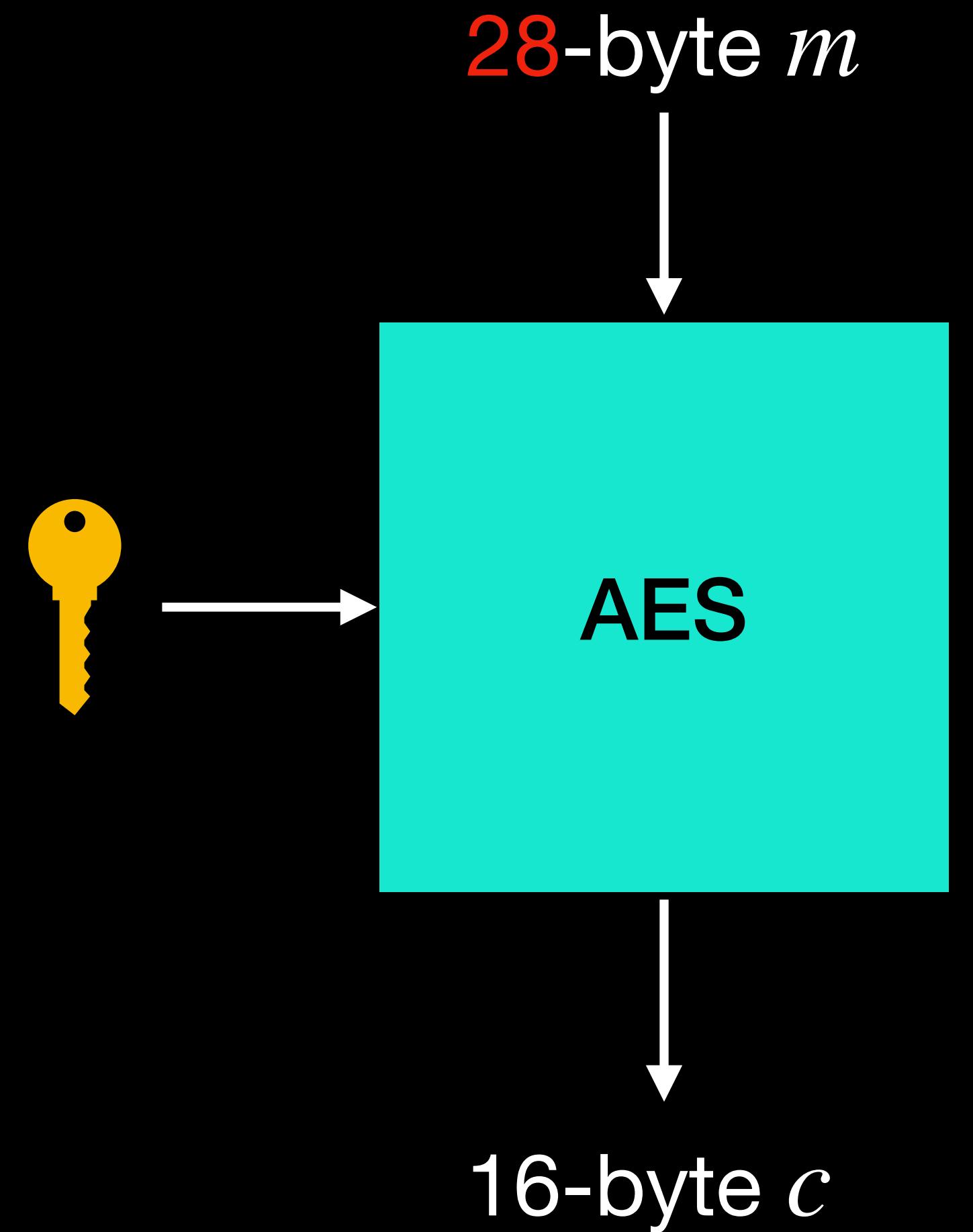
- AES 輸入區塊長度為 16 bytes，不足時需做填充
- 填充方法
  - PKCS#7
  - ANSI X9.23
  - ISO 10126
  - Zero Padding
  - ...

# PKCS#7 Padding for AES

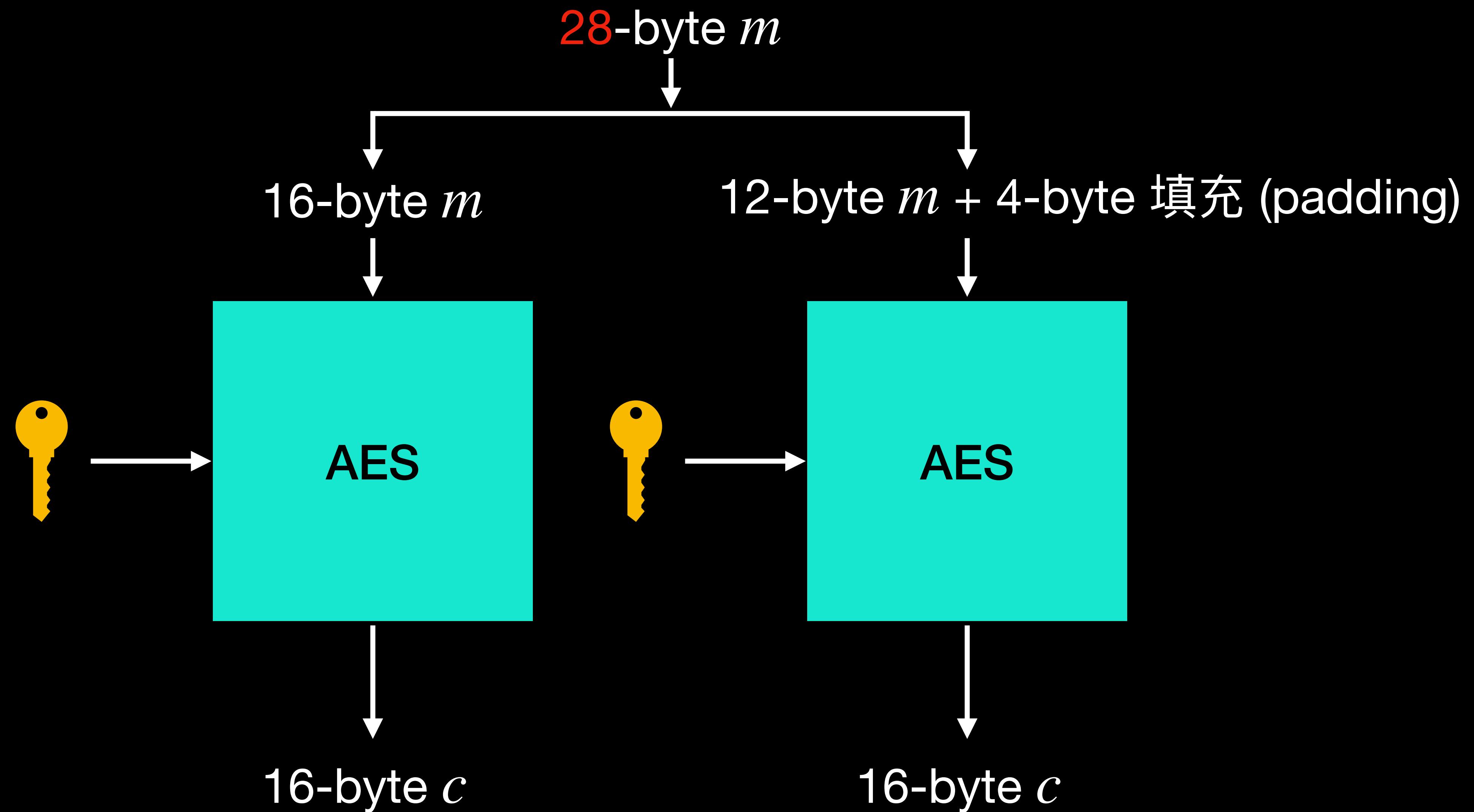
- 填充  $k$  個數值為  $k$  的 bytes
- 不缺 bytes，填充 16 個 0x10
  - 避免歧異，如：明文結尾剛好是 0x0202
- 根據最後一個 byte 即可知道填充長度



# 資料長度超過 16 bytes 怎麼辦？



# 資料長度超過 16 bytes 怎麼辦？→ 工作模式 Mode



# 區塊密碼工作模式

## Block Cipher Mode

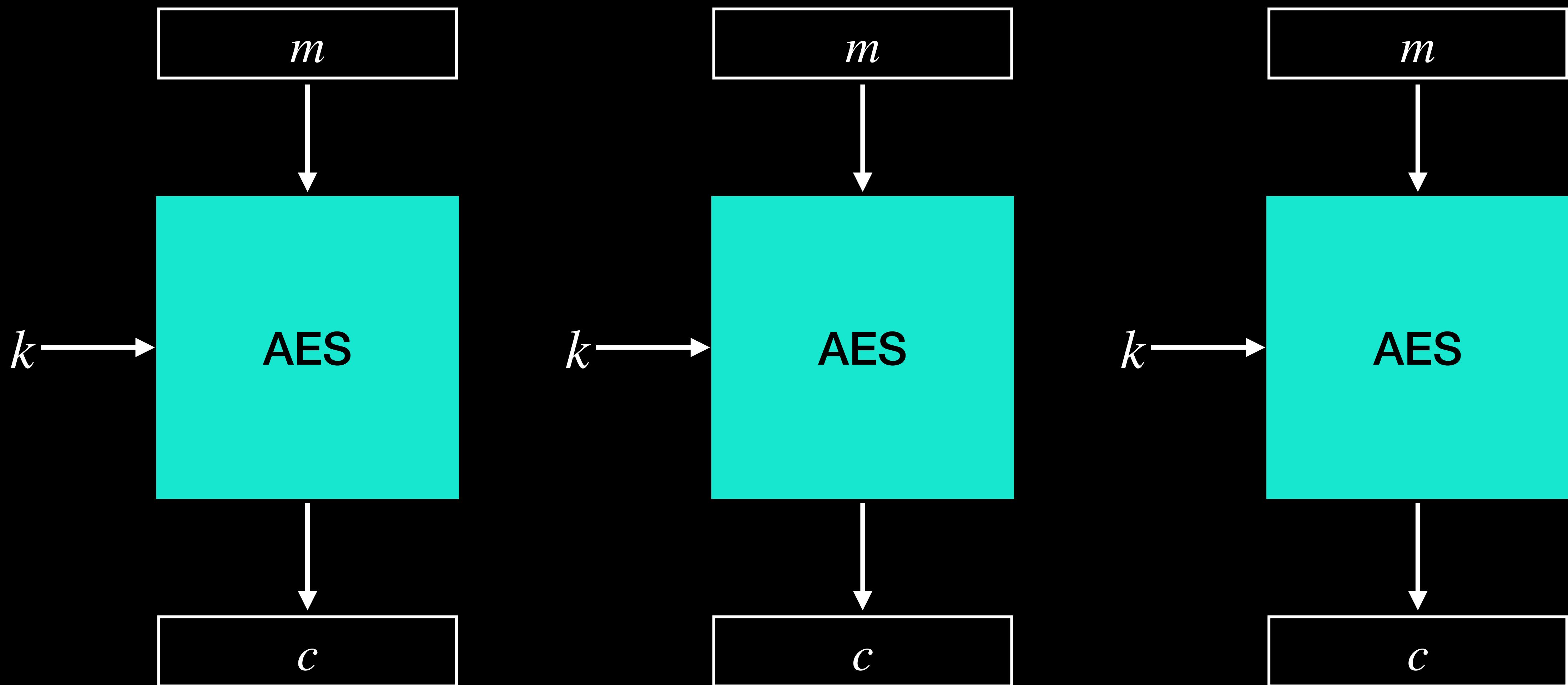
- 區塊密碼只適用於單一密鑰對單一區塊進行加密  $AES(m_{16\ bytes}, k) = c_{16\ bytes}$
- 工作模式：如何使單一密鑰能對多個區塊的資料進行加密
  - 設計考量：安全性、平行化、誤差傳播、特殊目的
  - 工作模式
    - ECB Mode
    - CBC Mode
    - CTR Mode

# 電子密碼本工作模式 ECB Mode

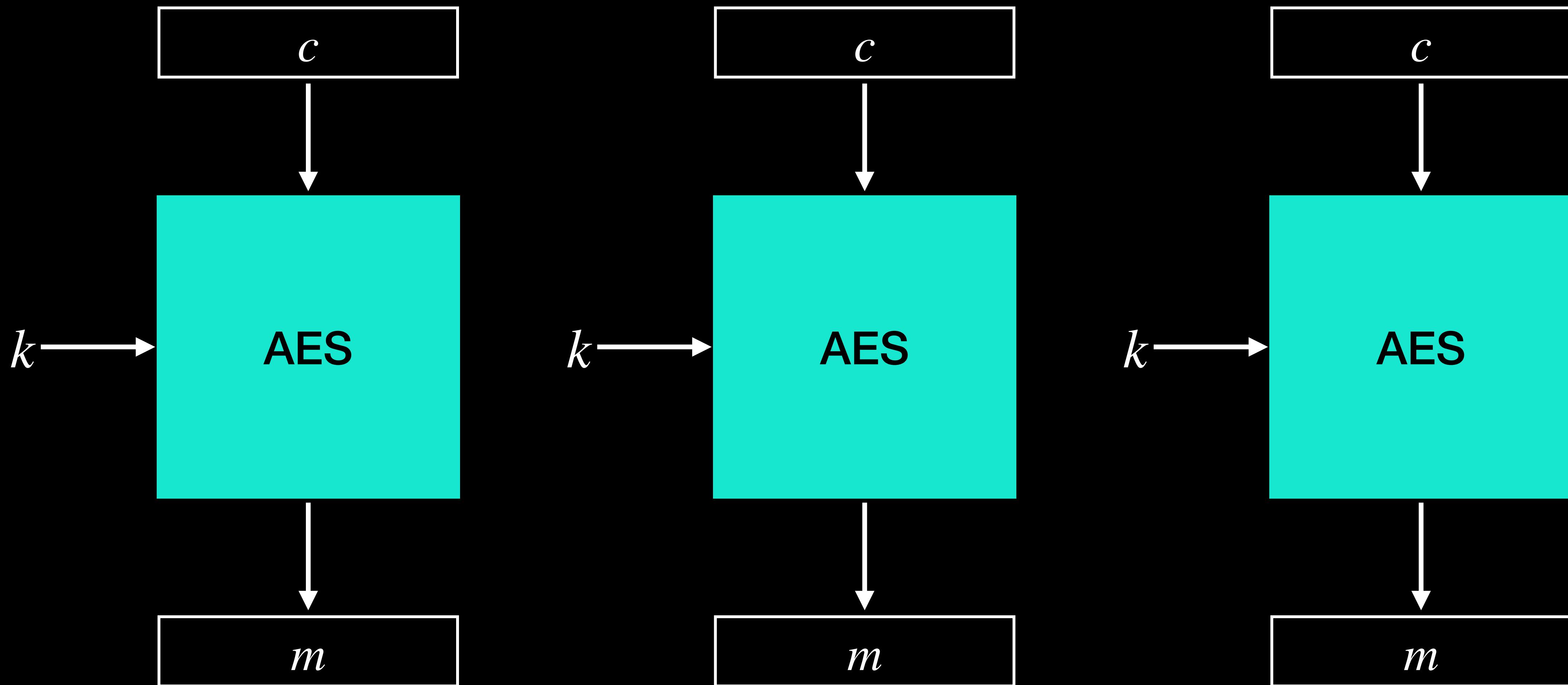
## Electronic CodeBook Mode

- 使用單一密鑰對每個區塊分別進行加解密
- 優點
  - 直覺
  - 加密平行化
  - 解密平行化
- 缺點
  - 區塊之間缺乏擴散，導致安全性不足

# ECB Mode 加密



# ECB Mode 解密



# ECB Mode 缺點

- 使用相同的密鑰與演算法，因此相同的明文會加密出相同的密文

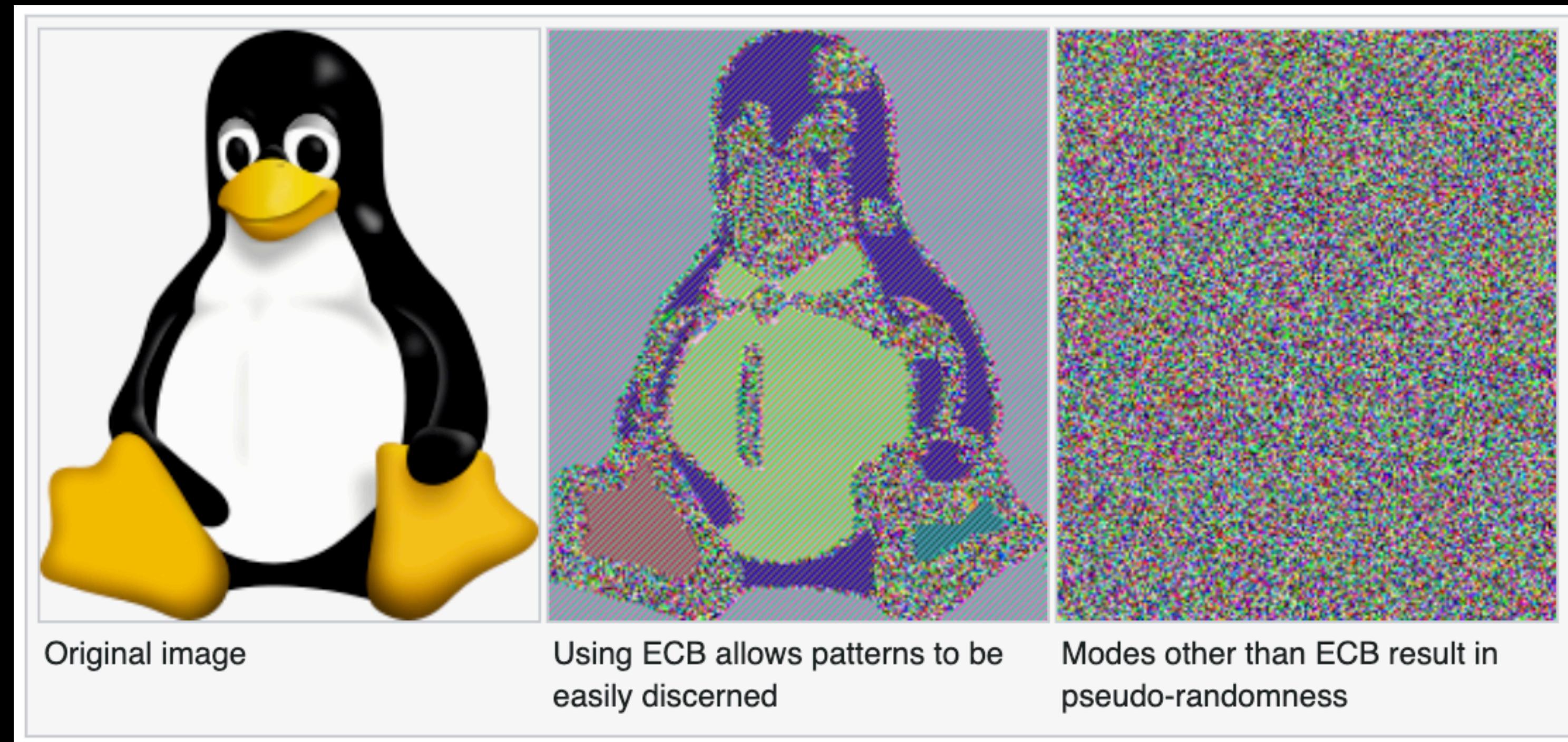
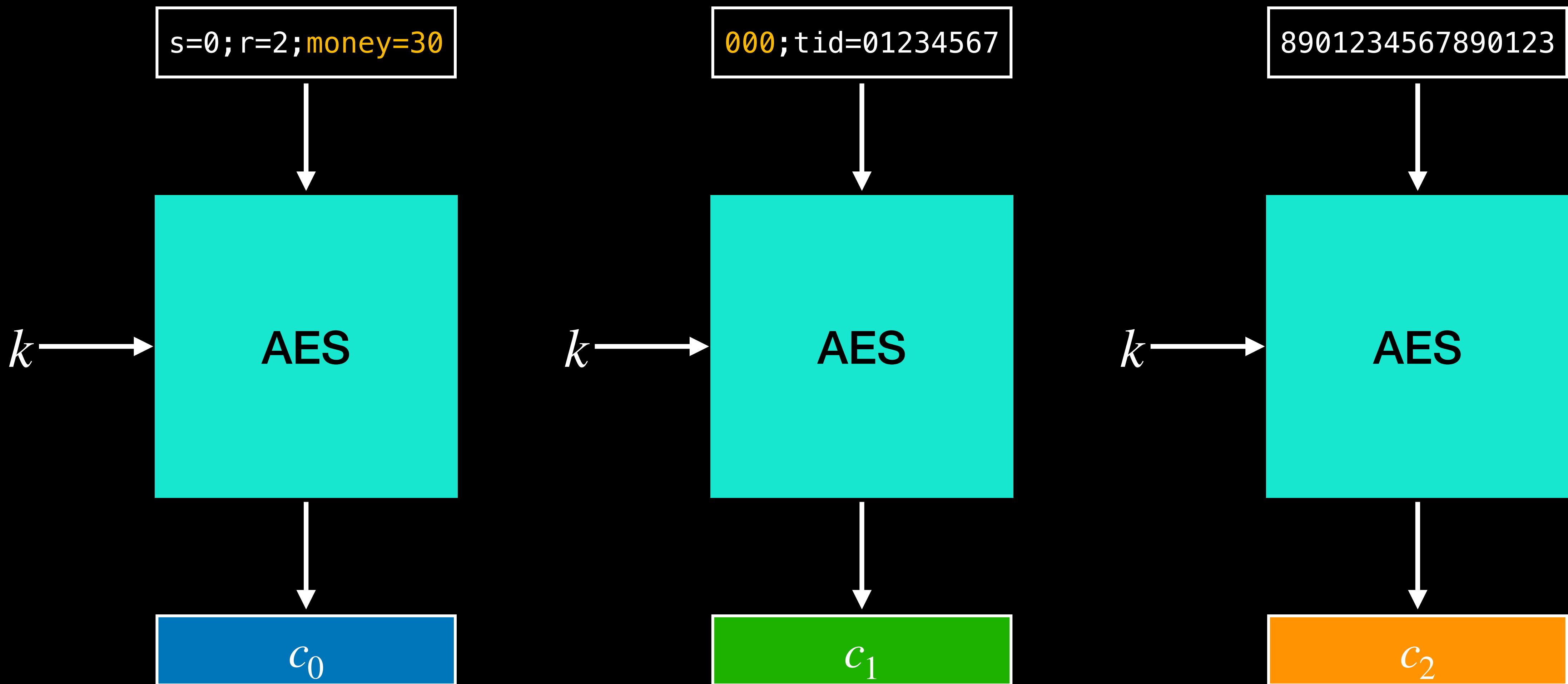
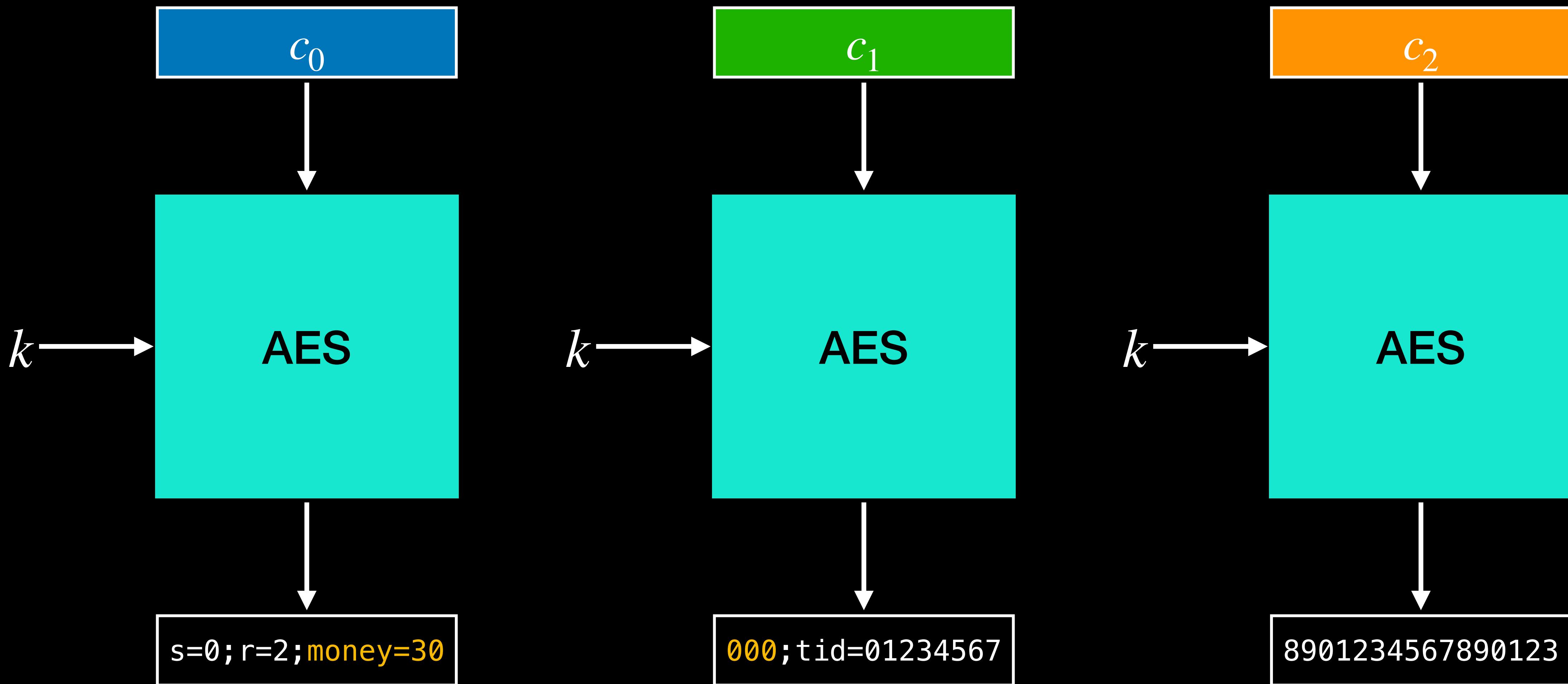


Image from: [https://en.wikipedia.org/wiki/Block\\_cipher\\_mode\\_of\\_operation](https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation)

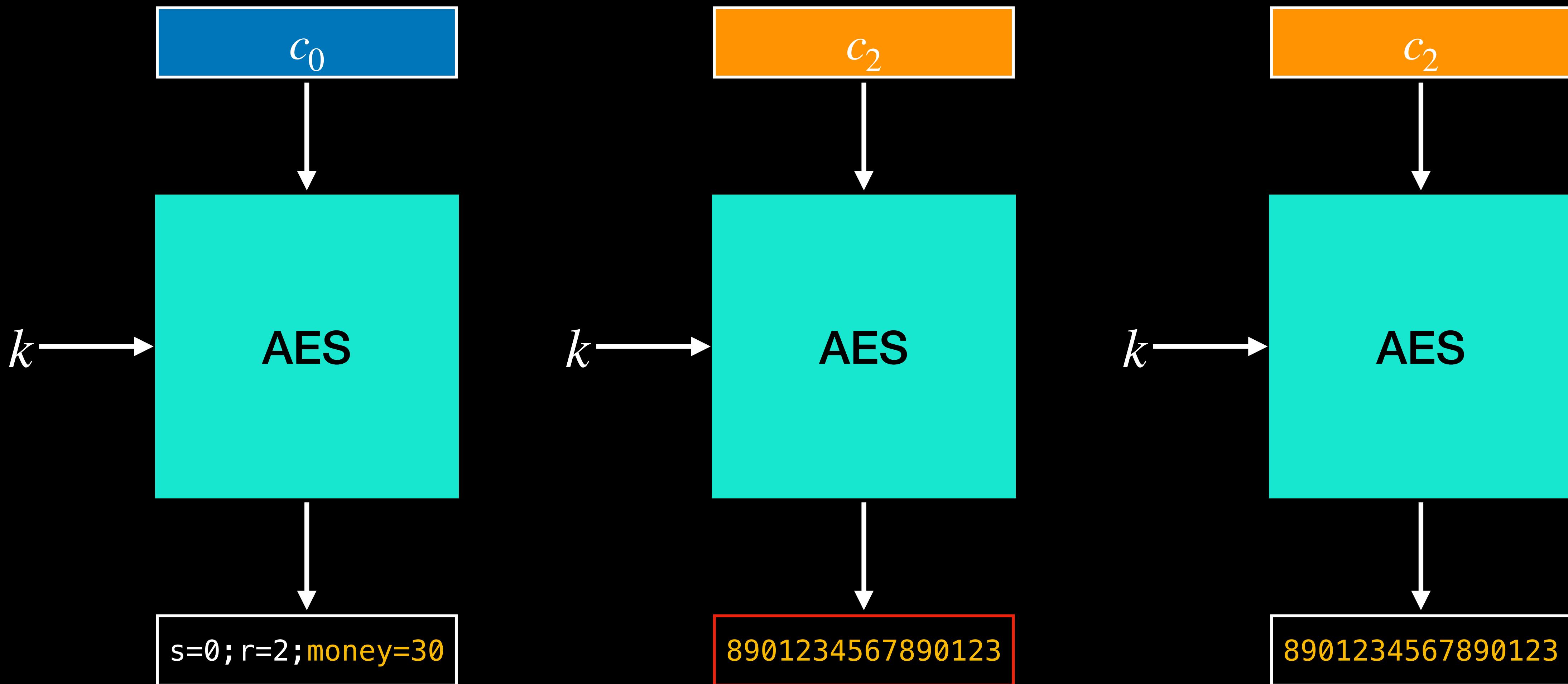
# 針對 ECB Mode 的 Cut & Paste 攻擊



# 針對 ECB Mode 的 Cut & Paste 攻擊



# 針對 ECB Mode 的 Cut & Paste 攻擊



# Lab : ECB / Cut & Paste

## ECB / Cut & Paste

100

閱讀題目程式碼 `chal.py`，然後嘗試修改 `solve_template.py` 中標示 `TODO` 的部分，對 ECB 工作模式發出 Cut & Paste 攻擊！

連線資訊：`nc 210.70.138.222 11005`

Flag 格式：`FLAG{...}`

Author: Ice1187

`chal.py`

`solve_template...`

# Lab : ECB / Cut & Paste

```
6  with open('flag', 'r') as f:
7      flag = f.read()
8
9  key = os.urandom(16)
10 aes = AES.new(key, AES.MODE_ECB)
11
12 m = {
13     'sid': 0,
14     'rid': 2,
15     'money': 100000,
16     'tid': 0xfedcba987654321cafe,
17 }
18 json_m = json.dumps(m).encode('ascii')
19 pad_m = pad(json_m, block_size=16)
20
21 print('Alice is transferring $30000 to Bob.')
22 print('m:', pad_m)
23 print('')
24
25 c = aes.encrypt(pad_m)
26 print('You have hijacked the encrypted message of the money transfer.')
27 print('c:', c)
28 print('')
```

# Lab : ECB / Cut & Paste

```
|> nc 210.70.138.222 11005
```

Alice is transferring \$30000 to Bob.

```
m: b'{"sid": 0, "rid": 2, "money": 10000, "tid": 75222050998243593145086}
\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b\x0b'
```

You have hijacked the encrypted message of the money transfer.

```
c: b'\x85\x83\x0e\x2\xfa\x02T\xd9\xb8\xbd\x0bU\x95\x87\xda\x7-:2mPJ\x18\
\x94\x17\x81v@\x15\x825\x1a\x99\xe6n\xeaI\xc1\xf4\xbb=\x16\x99d\x0\xef$\x9
\x9\x8e\xf6\xc5\xca\xfd\x99\x08\x00\'W\xf6"D\x81\xb2\x85\xef\xac0S\x18\x
\x14RC\xc2\xceP\xbe\x8ecv'
```

# Lab : ECB / Cut & Paste

```
print('Try to do the cut&paste attack to transfer more money. Paste your attack_c (in decimal)')  
attack_c = int(input('attack_c: ')).to_bytes(512, 'big').strip(b'\x00')  
  
attack_m = aes.decrypt(attack_c)  
attack_m = unpad(attack_m, block_size=16)  
#print(attack_m)  
attack_m = json.loads(attack_m)  
print(attack_m)  
  
if attack_m['sid'] == m['sid'] and \  
    attack_m['rid'] == m['rid'] and \  
    attack_m['tid'] == m['tid'] and \  
    attack_m['money'] > m['money']: \  
    print('flag:', flag)
```

# Lab : ECB / Cut & Paste

You have hijacked the encrypted message of the money transfer.

```
c: b'\x82\x9f\xbe\xcb\x158(6\x0e\x8b\xeb\xb70)\x87e\\x9a\x9e}\x97\x8b\x8ba\x  
a9\x94s\xab\n\x80\xc7\x96\xdfW\xb9\x84[M\xf7W\x  
b3\xba`9\xc3\xaa BHV4W\xd8\xe00[\xd5\xf1\xe9`>$  
\x0c^\\x05%\xc9\x9b\xda+\xb1\xde\x7c/\xab\x02\x  
\xd8\xe00[\xd5\xf1\xe9`>$\x0c^\\x05%\xc9\x9b\xda+\x  
xb1\xde\x7c/\xab\x02\x8a\xe5'
```

Try to do the cut&paste attack to transfer more money. Paste your attack\_c (in decimal)

```
attack_c: 232798541505500065905648951182786469259  
3571630998381962258163119461471803959485567648308  
2904658795604825108217522574546012248424704239699  
6962230360722260475688534101037421143541764337672  
0382693  
{'sid': 0, 'rid': 2, 'money': 100000, 'tid': 7522  
2050998243593145086}
```

```
>>> int.from_bytes(b'\x82\x9f\xbe\xcb\x158(6\x0e\x8b\xeb\xb70)\x87e\\x9a\x9e}\x97\x8b\x8ba\x  
a9\x94s\xab\n\x80\xc7\x96\xdfW\xb9\x84[M\xf7W\x  
b3\xba`9\xc3\xaa BHV4W\xd8\xe00[\xd5\xf1\xe9`>$  
\x0c^\\x05%\xc9\x9b\xda+\xb1\xde\x7c/\xab\x02\x  
8a\xe5', 'big')  
23279854150550006590564895118278646925935716309  
98381962258163119461471803959485567648308290465  
879560482510821752257454601224842470423969962  
23036072226047568853410103742114354176433767203  
82693  
>>>  
>>>  
>>>  
>>>  
>>>  
>>>
```

# Lab : ECB / Cut & Paste - solve\_template.py

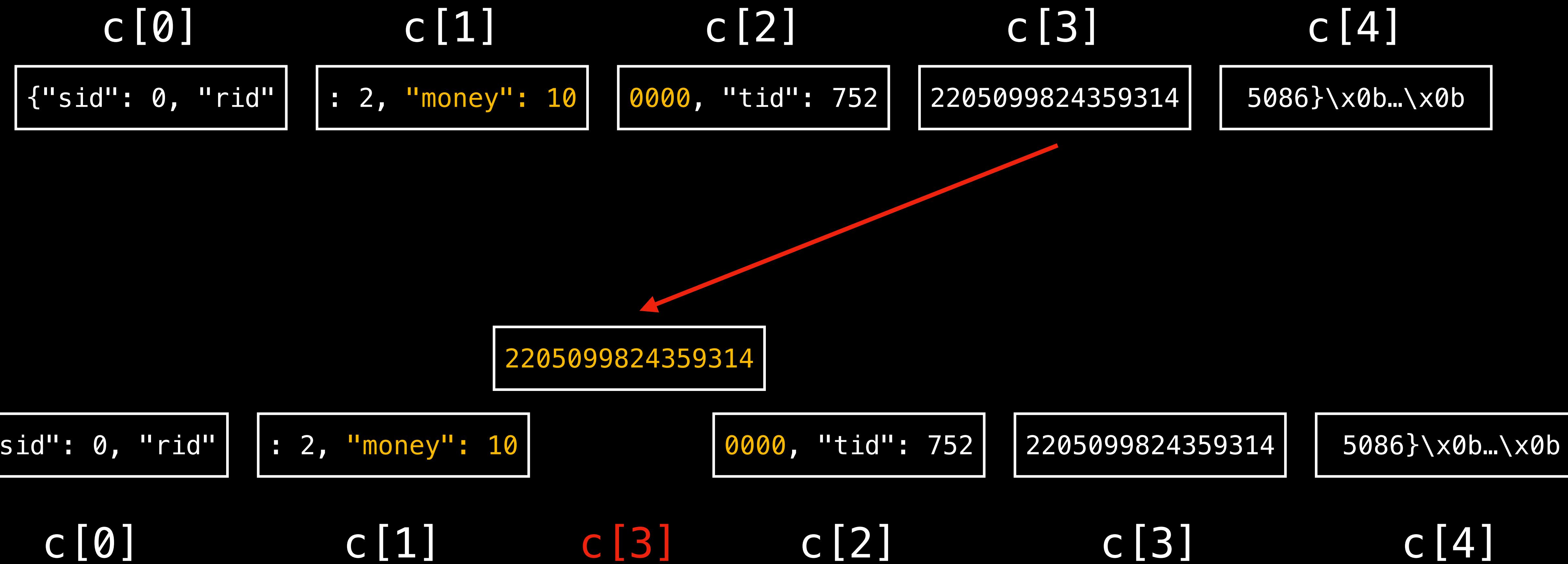
```
# TODO: paste the c you received 1. 複製貼上題目給出的 c (每次重新連線後會不同)
c_in = b"\xe4\xb7\xe6..." # change me

c = []
for i in range(0, len(c_in), 16):
    c.append(c_in[i:i+16])

# TODO: combining c[0], c[1], ..., c[4] to do the cut&paste attack
attack_c = c[0] + c[1] + c[2] + c[3] + c[4] # change me
attack_c = int.from_bytes(attack_c, 'big')
print(attack_c)
```

2. 組合  $c[0], c[1], \dots, c[4]$  來進行 cut & paste 攻擊

# 解法：ECB / Cut & Paste



# 解法：ECB / Cut & Paste

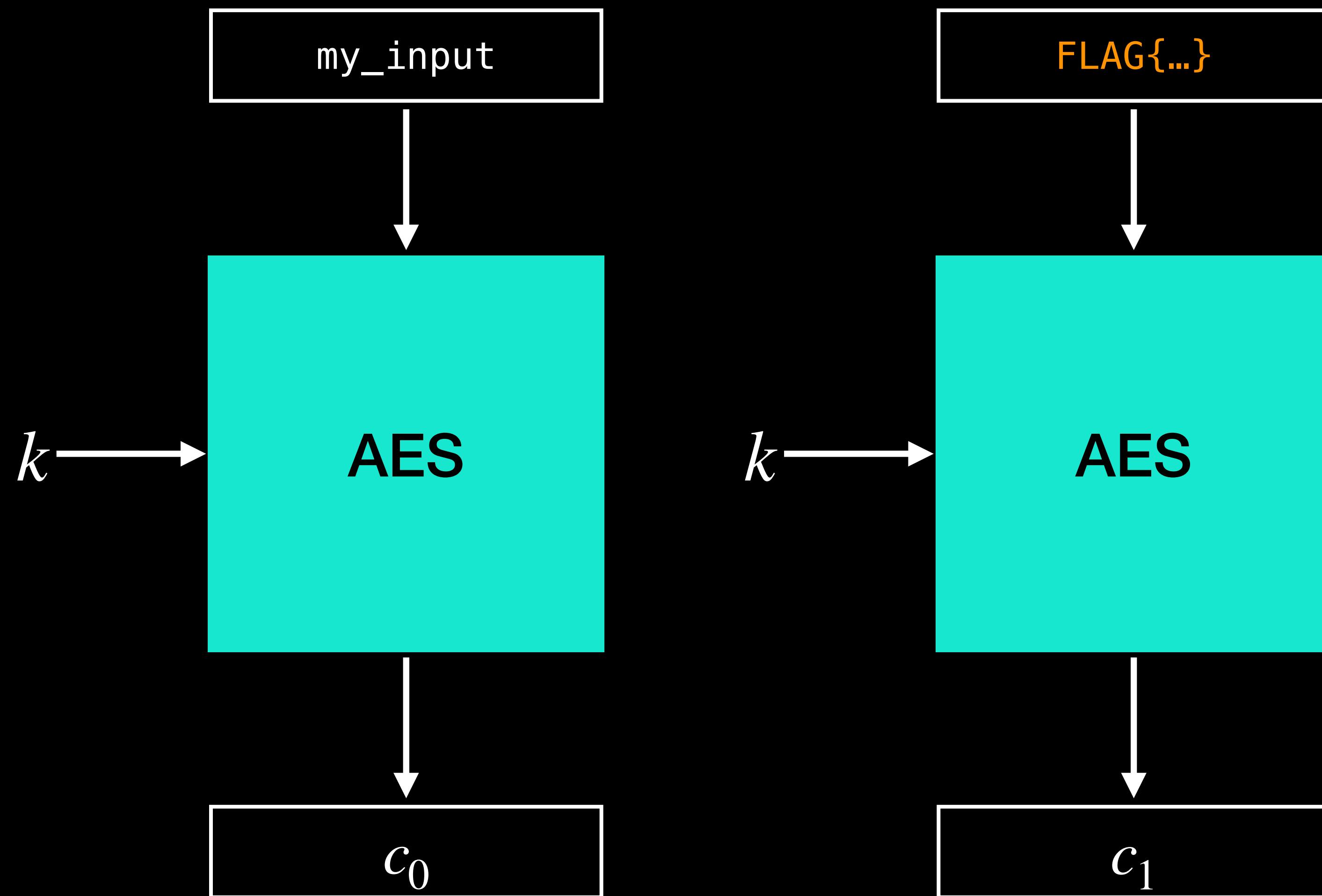
```
c_in = b"\xcb\x8a\x1a\xcd\x89FY\x1f)<gv\xd5\xba\xcb\x89`  
  
c = []  
for i in range(0, len(c_in), 16):  
    c.append(c_in[i:i+16])  
  
attack_c = c[0] + c[1]+ c[3] + c[2] + c[3] + c[4]  
attack_c = int.from_bytes(attack_c, 'big')  
print(attack_c)
```

# 針對 ECB Mode 的 Prepend Oracle Attack

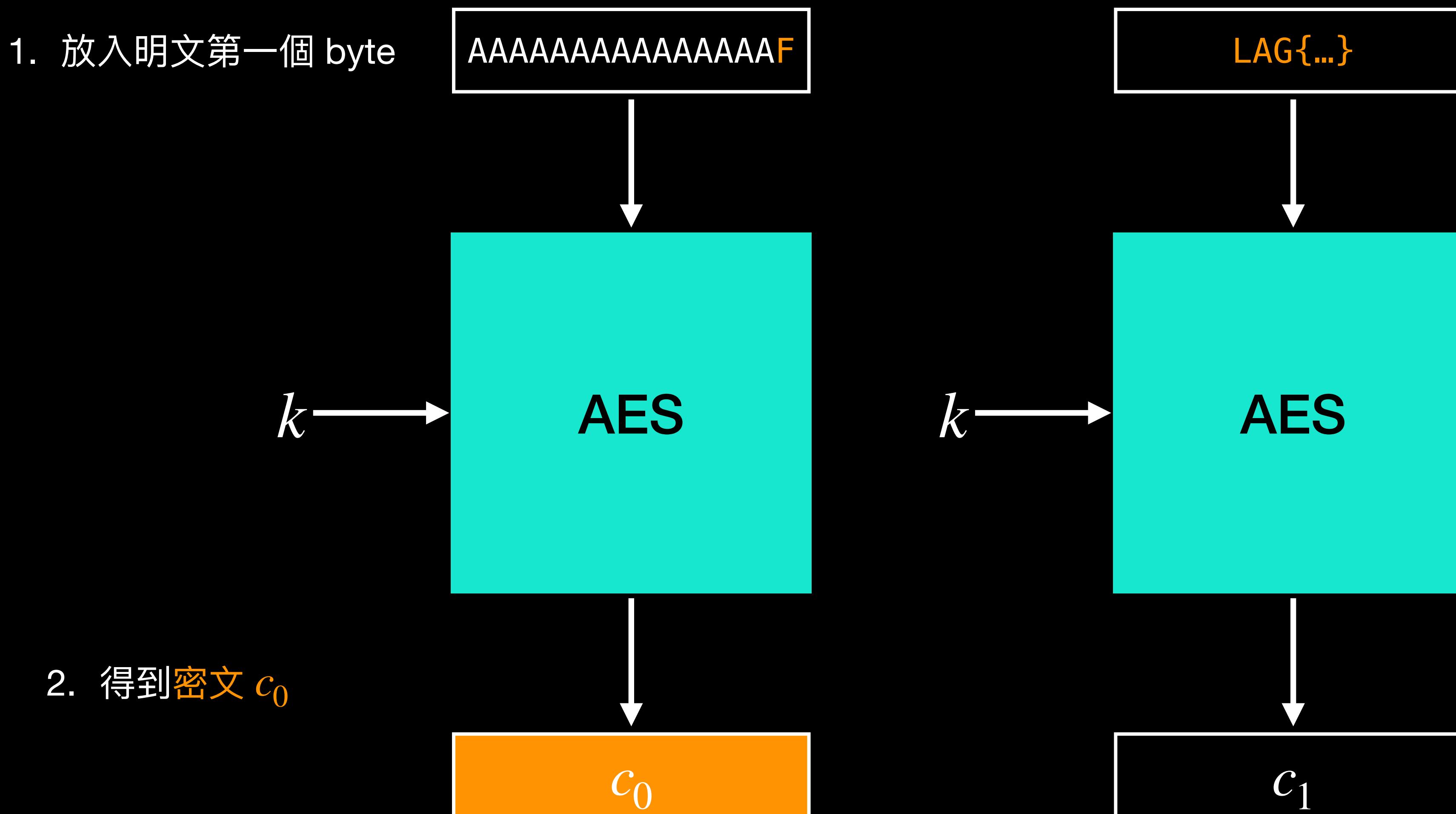
- 將可控制的輸入串接在明文前 (prepend)，再進行加密

```
m = my_input + flag  
pad_m = pad(m, block_size=16)  
c = aes.encrypt(pad_m)
```

# 針對 ECB Mode 的 Prepend Oracle Attack

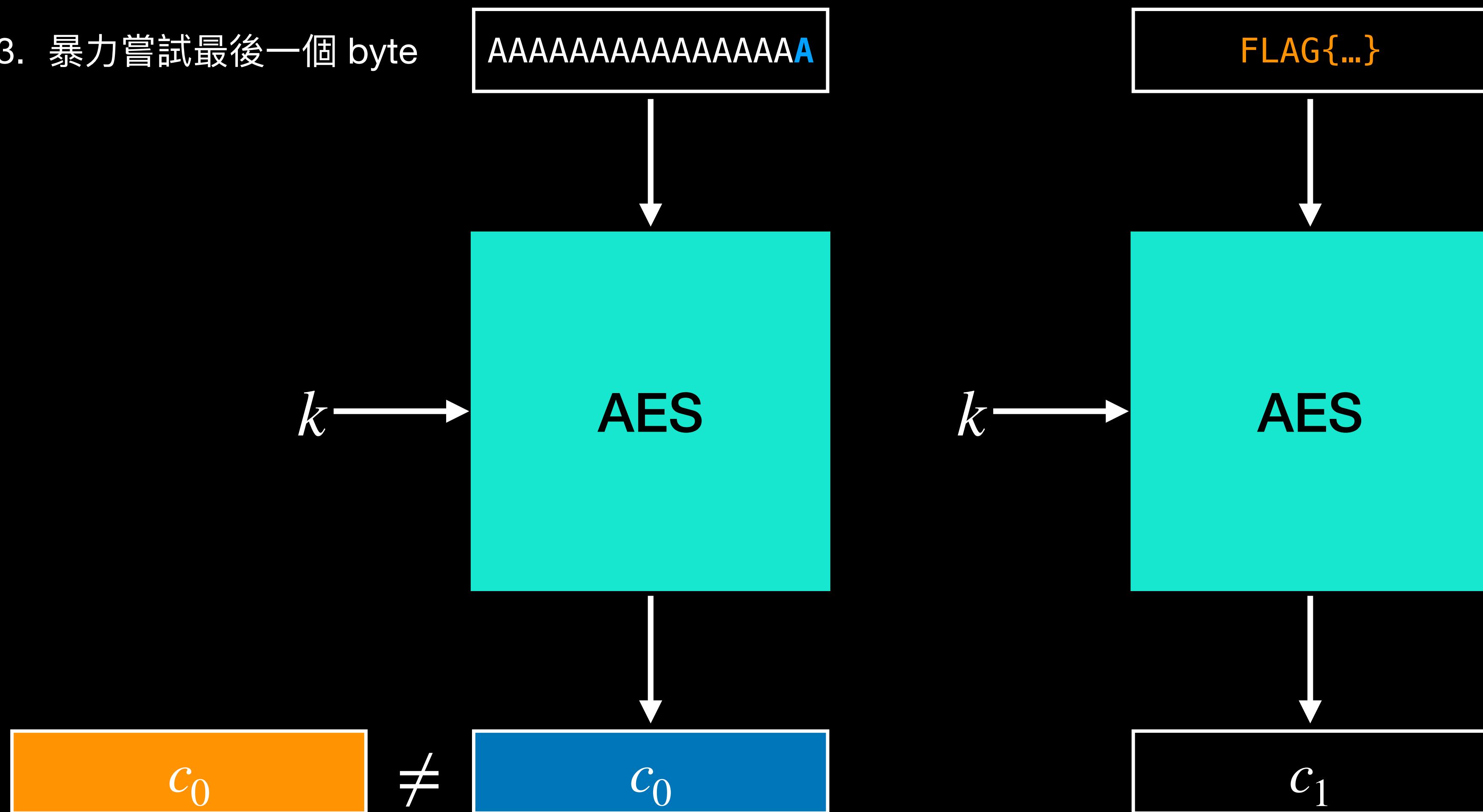


# 針對 ECB Mode 的 Prepend Oracle Attack



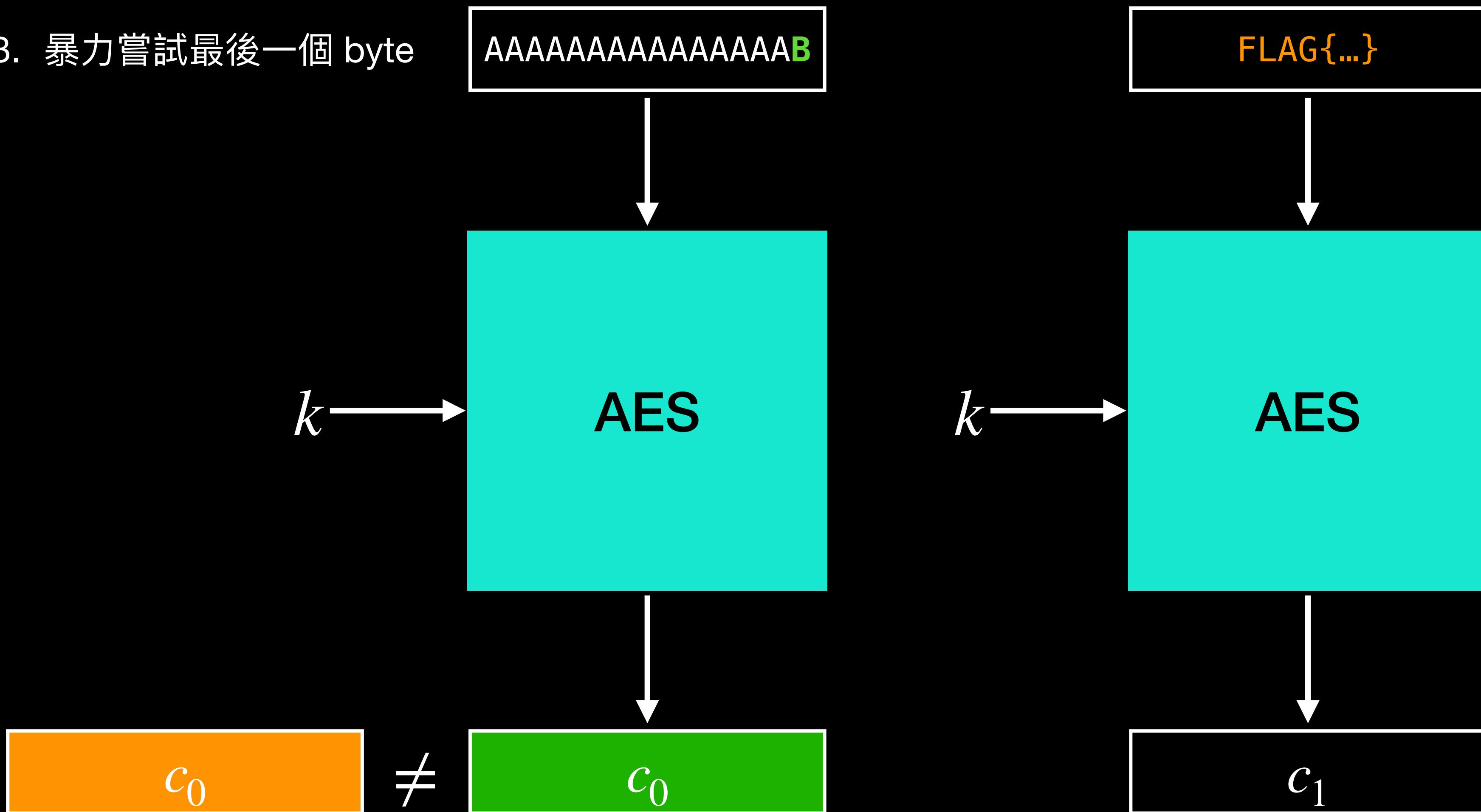
# 針對 ECB Mode 的 Prepend Oracle Attack

3. 暴力嘗試最後一個 byte



# 針對 ECB Mode 的 Prepend Oracle Attack

3. 暴力嘗試最後一個 byte



# 針對 ECB Mode 的 Prepend Oracle Attack

3. 暴力嘗試最後一個 byte

AAAAAAA  
AAAAAA  
F

$k \rightarrow$

AES

FLAG{...}

$k \rightarrow$

AES

4. 密文相同，表示為明文第一個 byte

$c_0$

=

$c_0$

$c_1$

# 針對 ECB Mode 的 Prepend Oracle Attack

5. 放入明文第二個 byte

AAAAAAAAAAAAAAFL

$k \rightarrow$

AES

AG{...}

$k \rightarrow$

AES

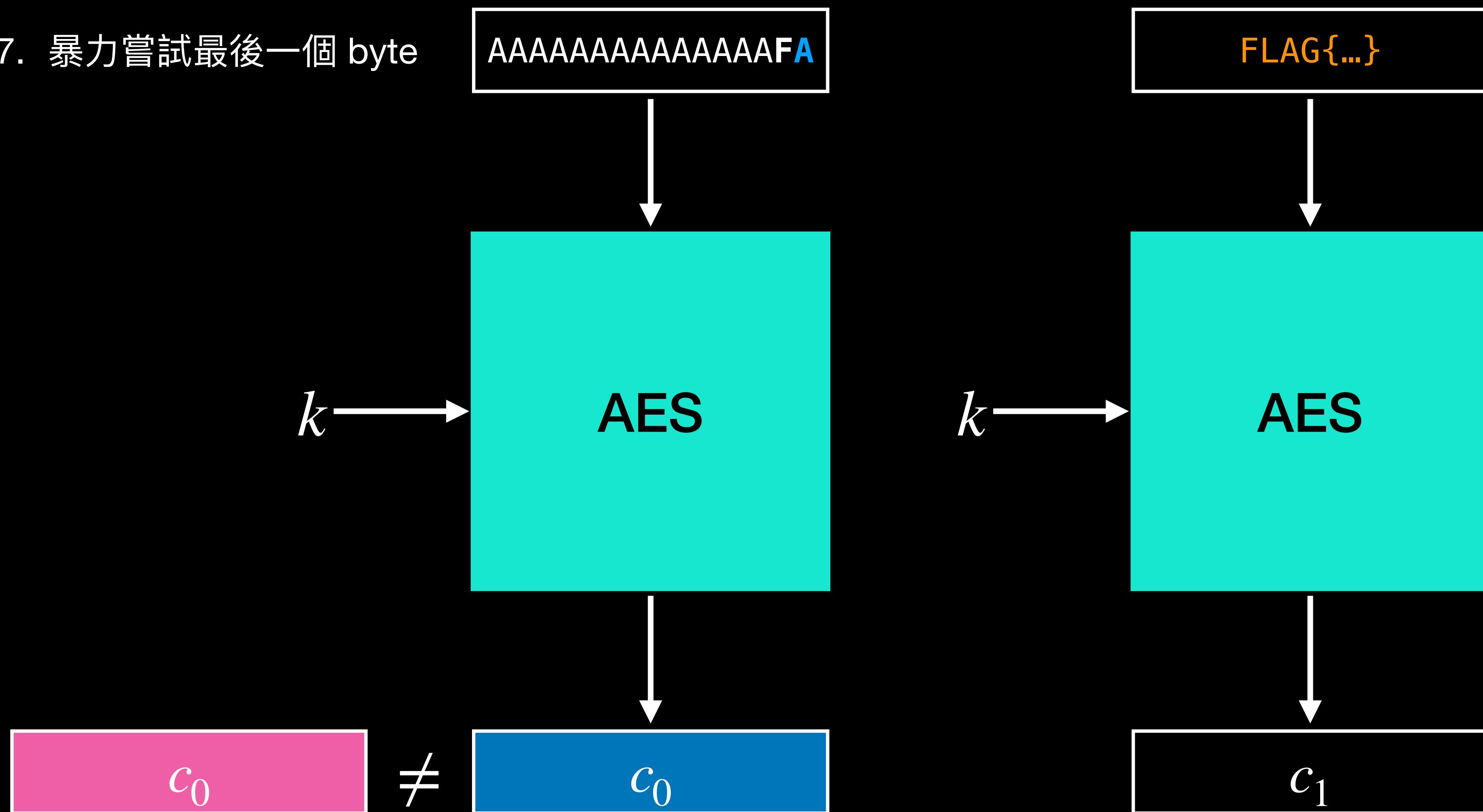
6. 得到密文  $c_0$

$c_0$

$c_1$

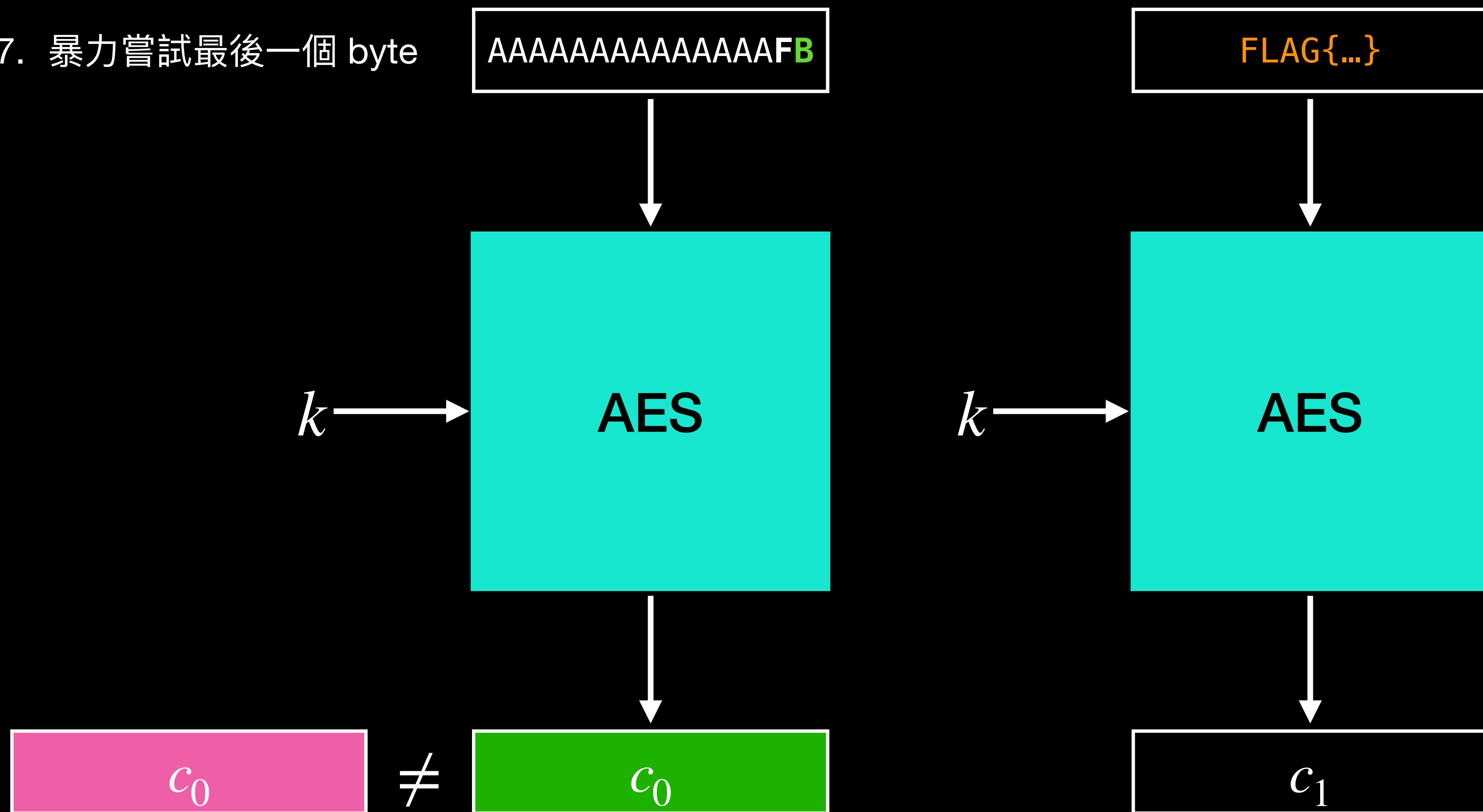
# 針對 ECB Mode 的 Prepend Oracle Attack

7. 暴力嘗試最後一個 byte



# 針對 ECB Mode 的 Prepend Oracle Attack

7. 暴力嘗試最後一個 byte



# 針對 ECB Mode 的 Prepend Oracle Attack

7. 暴力嘗試最後一個 byte

AAAAAAA  
AAAAAFL

$k \rightarrow$

AES

FLAG{...}

$k \rightarrow$

AES

8. 密文相同，表示為明文第二個 byte

$c_0$

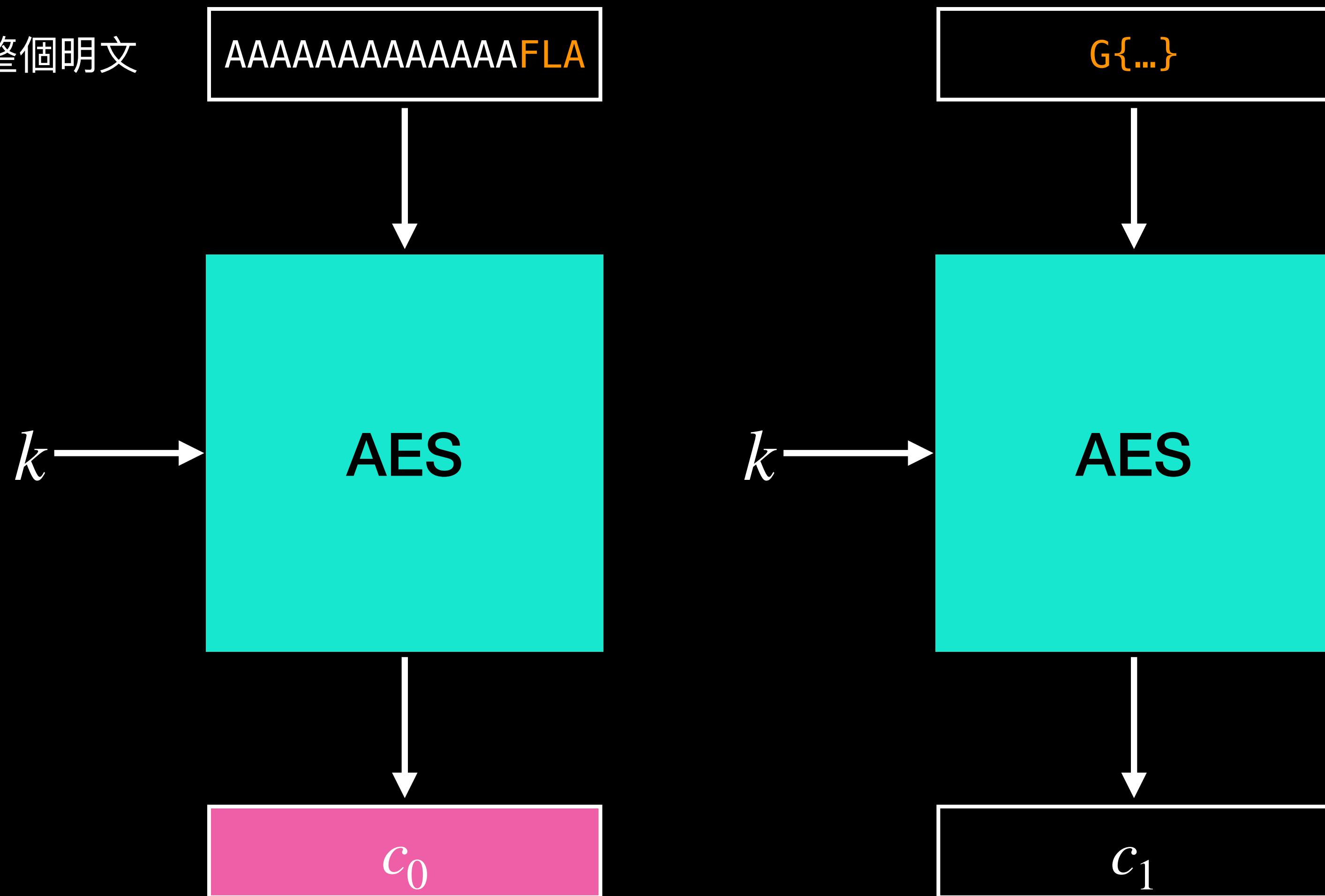
=

$c_0$

$c_1$

# 針對 ECB Mode 的 Prepend Oracle Attack

9. 重複直到找出整個明文



# 針對 ECB Mode 的 Prepend Oracle Attack

- 最多需要計算  $(1 + 256) \times \text{len}(m)$  次
  - 放入明文第 # 個 byte，計算一次
  - 每次暴力嘗試一個 byte，最多  $2^8 = 256$  種組合
  - 明文  $m$  長度為  $\text{len}(m)$

# Lab : ECB / Prepend Oracle Attack

## ECB / Prepend Oracle Attack

100

閱讀題目程式碼 `chal.py`，然後嘗試修改 `solve_template.py` 中標示 TODO 的部分，對 ECB 工作模式發出 Prepend Oracle 攻擊！

連線資訊：`nc 210.70.138.222 11006`

Flag 格式：`FLAG{...}`

Author: Ice1187

`chal.py`

`solve_template...`

# Lab : ECB / Prepend Oracle Attack

```
while True:
    len_my_input = int(input('len of my_input (in bytes): '))
    if len_my_input > 0:
        my_input = input('my_input (in hex): ')
        my_input = int(my_input, 16).to_bytes(len_my_input, 'big')
    else:
        my_input = b''
    m = my_input + flag
    pad_m = pad(m, block_size=16)
    c = aes.encrypt(pad_m)
    print('ciphertext:', c)
```

# Lab : ECB / Prepend Oracle Attack

```
> nc 210.70.138.222 11006
len of my_input (in bytes): 0
ciphertext: b'0\xe9(\x97\xda\x18jV\xf7\xee\xedH\xb
\x1a8\x8f'
len of my_input (in bytes): 15
my_input (in hex): 41414141414141414141414141
ciphertext: b'\xC\xb6`\xe7\x8f%\xbeB\xcb\xe90<\xd4
\xdd\x06F'
```

# Lab : ECB / Prepend Oracle Attack - solve\_template.py

```
# setup connection
host = '127.0.0.1' • 將連線資訊修改成對的 IP 與 port
port = 9000
chal = remote(host, port)

def send(data):
    # send input          • send 負責處理收發資訊（可以忽略不用看）
    #print(len(data), data)
    chal.recvuntil(b'len of my_input (in bytes): ')
    chal.sendline(str(len(data)))
    if len(data) > 0:
        chal.recvuntil(b'my_input (in hex): ')
        chal.sendline(data.hex())

# receive ciphertext
chal.recvuntil(b'ciphertext: ')
ret = chal.recvuntil(b'\n').strip().decode()[2:-1]
#print(ret)
return ret
```

# Lab : ECB / Prepend Oracle Attack - solve\_template.py

```
flag = b''

# TODO: modify the following code to attack a whole block
my_input = b'A'*15
oracle = send(my_input)
for i in range(0, 256):
    print('Trying: ', bytes([i]))
    my_input = b'A'*15 + flag + bytes([i])
    c = send(my_input)

    if c[:16] == oracle[:16]:
        flag += bytes([i])
        print('Found:', flag.decode())
        break
    if i == 255:
        print('Not found!')
        exit(0)
```

# Lab : ECB / Prepend Oracle Attack - solve\_template.py

```
flag = b''

# TODO: modify the following code to attack a whole block
my_input = b'A'*15 1. 放入明文第一個 byte
oracle = send(my_input) 2. 得到密文  $c_0$  (oracle)
for i in range(0, 256): 3. 暴力嘗試最後一個 byte
    print('Trying: ', bytes([i]))
    my_input = b'A'*15 + flag + bytes([i])
    c = send(my_input)

    if c[:16] == oracle[:16]: 4. 密文相同，表示為明文第一個 byte
        flag += bytes([i])
        print('Found:', flag.decode())
        break
    if i == 255:
        print('Not found!')
        exit(0)
```

# Lab : ECB / Prepend Oracle Attack - solve\_template.py

```
> python3 solve_template.py
[+] Opening connection to 210.70.138.222 on port 11006: Done
Trying: b'\x00'
Trying: b'\x01'
Trying: b'\x02'
Trying: b'\x03'
Trying: b'\x04'
Trying: b'\x05'
.
.
.
Trying: b'D'
Trying: b'E'
Trying: b'F'
Found: F
[*] Closed connection to 210.70.138.222 port 11006
```

# Lab : ECB / Prepend Oracle Attack - solve\_template.py

```
flag = b''

# TODO: modify the following code to attack a whole block
my_input = b'A'*15
oracle = send(my_input)
for i in range(0, 256):
    print('Trying: ', bytes([i]))
    my_input = b'A'*15 + flag + bytes([i])
    c = send(my_input)

    if c[:16] == oracle[:16]:
        flag += bytes([i])
        print('Found:', flag.decode())
        break
    if i == 255:
        print('Not found!')
        exit(0)
```

TODO：修改腳本，使其能找出整個 block 的 16 個 bytes

# 解法 : ECB / Prepend Oracle Attack

```
flag = b''
for r in range(16):
    my_input = b'A'*(15 - r)
    orcale = send(my_input)
    for i in range(0, 256):
        my_input = b'A'*(15 - r) + flag + bytes([i])
        c = send(my_input)
        if c[:16] == orcale[:16]:
            flag += bytes([i])
            print('Found:', flag.decode())
            break
        if i == 255:
            print('Not fonud!')
            exit(0)
```

# 密碼塊連結工作模式 CBC Mode

## Cipher Block Chaining Mode

- 將明文與前一區塊的密文做 XOR，再進行加密

- 第一個區塊與初始化向量  $iv$  做 XOR

- $iv$  不屬於密鑰

- 優點

- 避免 ECB 相同密文的弱點

- 解密平行化

- 缺點

- 加密無法平行化

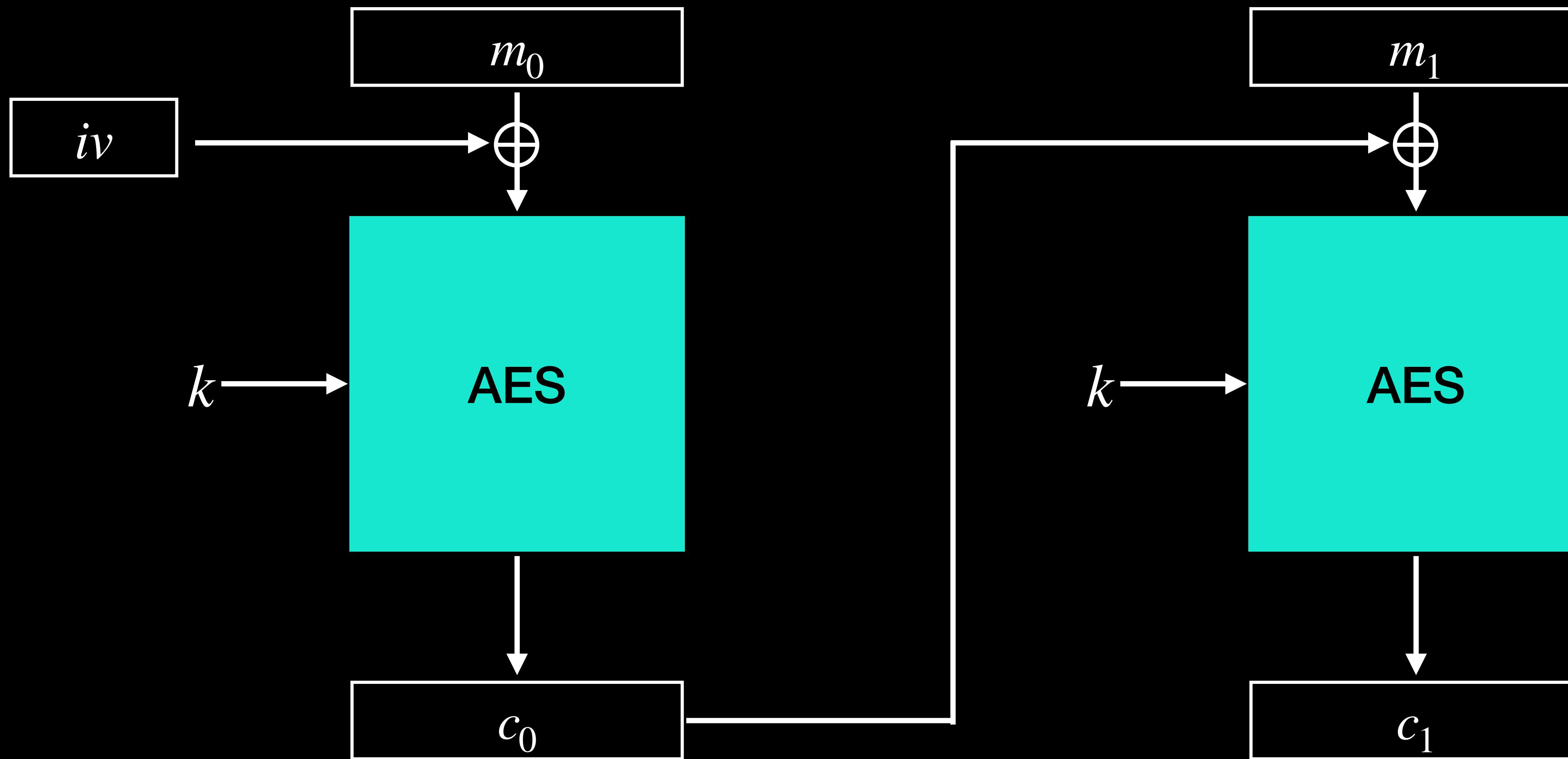
$$c_0 = E(m_0 \oplus iv)$$

$$c_i = E(m_i \oplus c_{i-1})$$

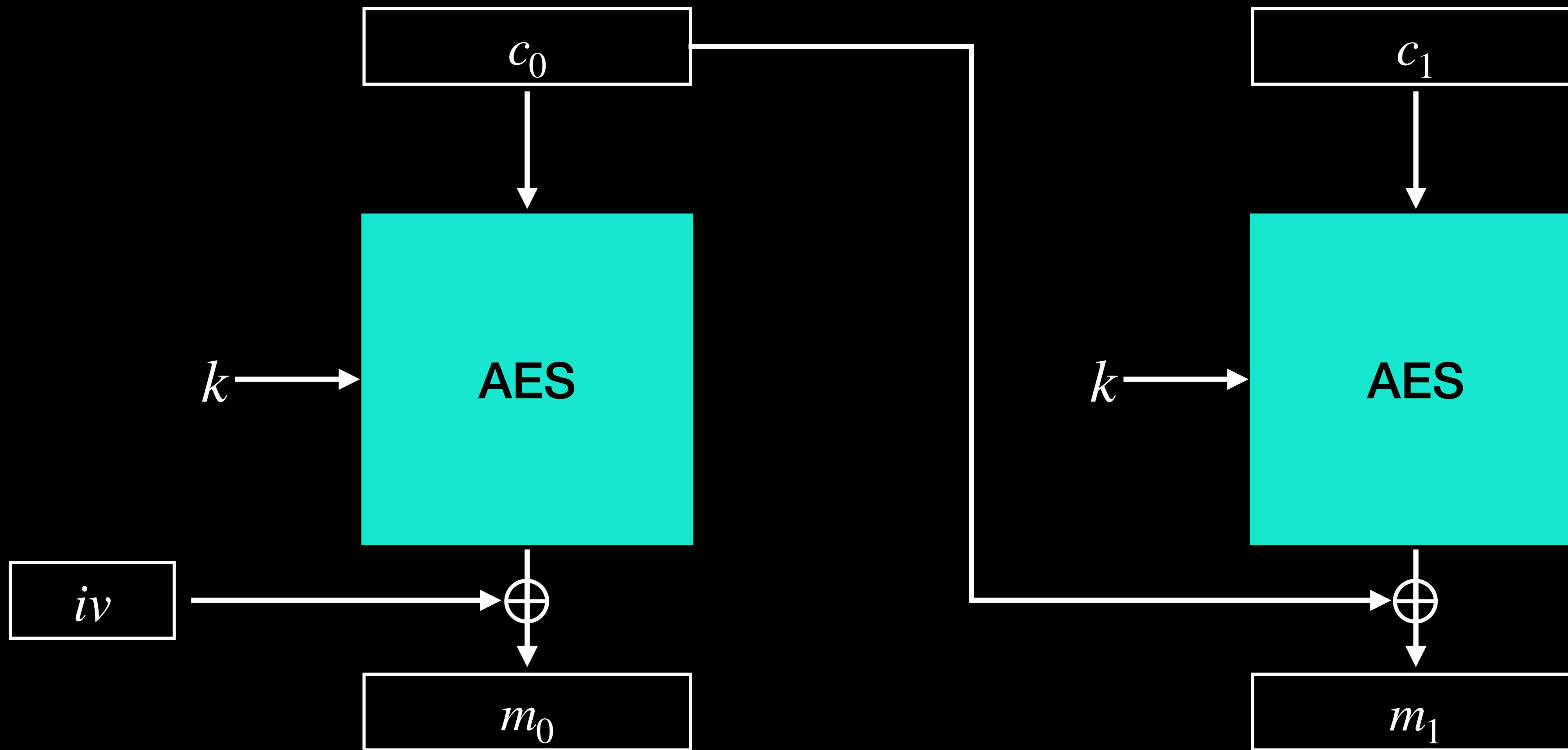
$$m_0 = D(c_0) \oplus iv$$

$$m_i = D(c_i) \oplus c_{i-1}$$

# CBC Mode 加密

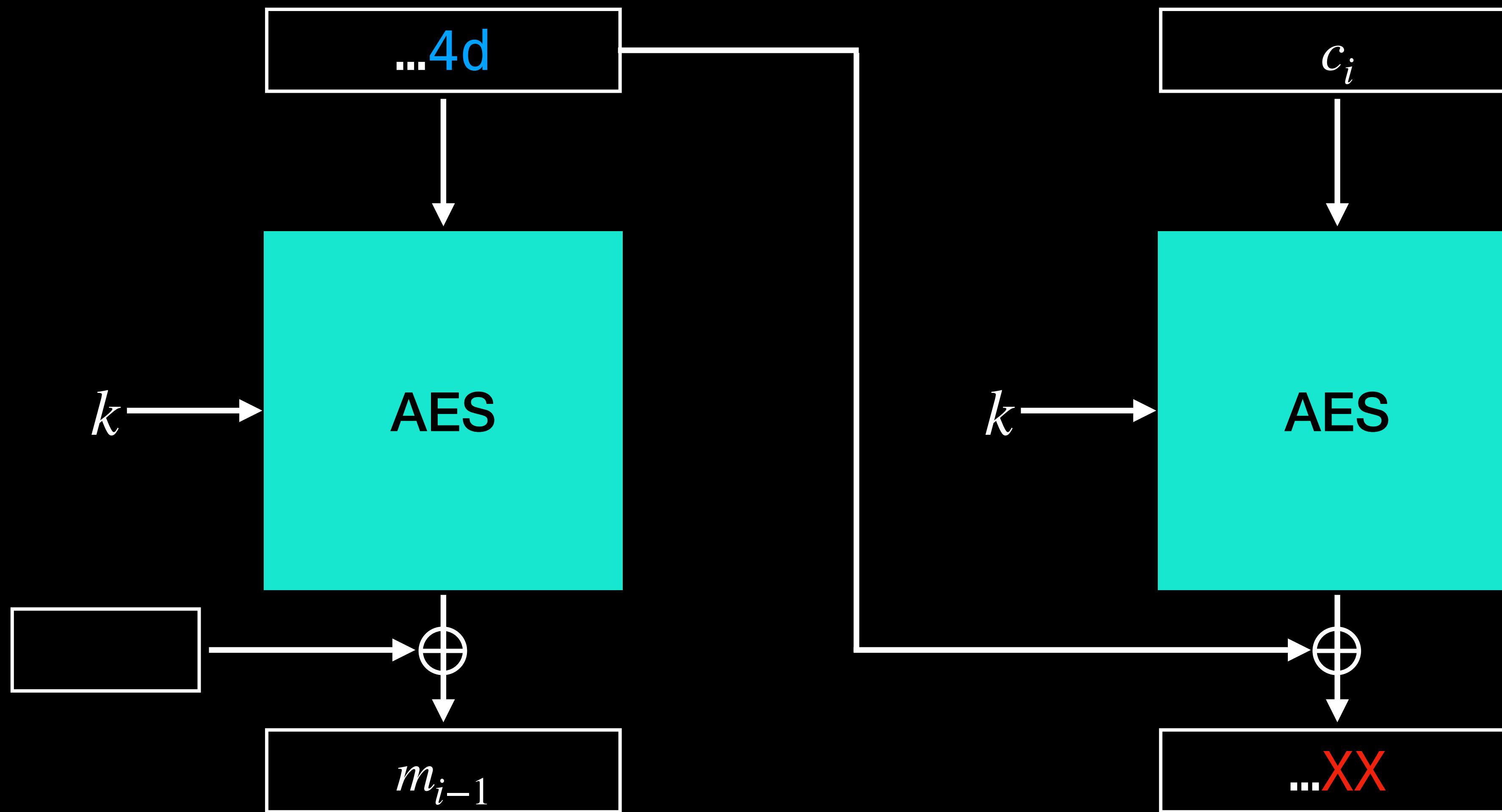


# CBC Mode 解密



# 針對 CBC Mode 的 Padding Oracle Attack

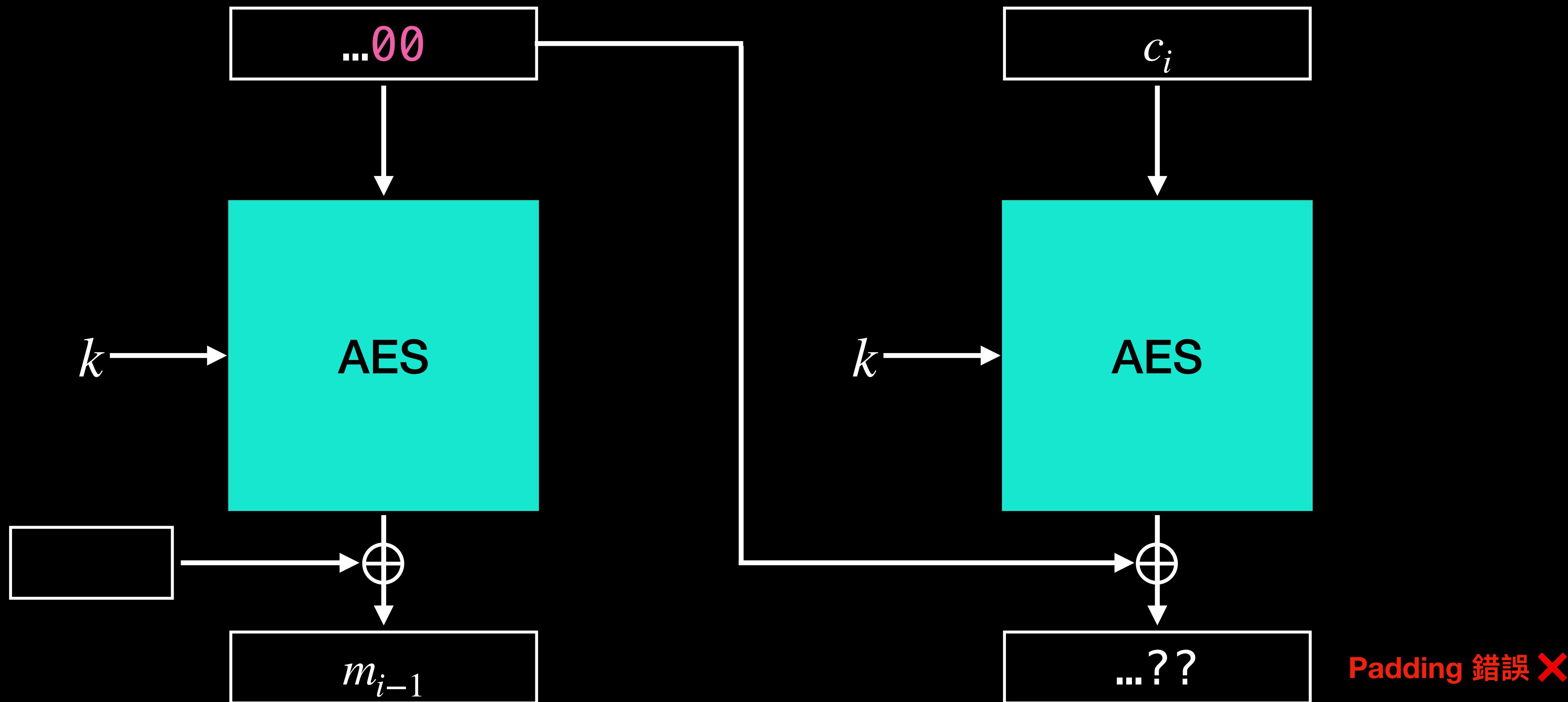
1. 已知前一區塊的密文最後一 byte



欲知此區塊的明文最後一 byte

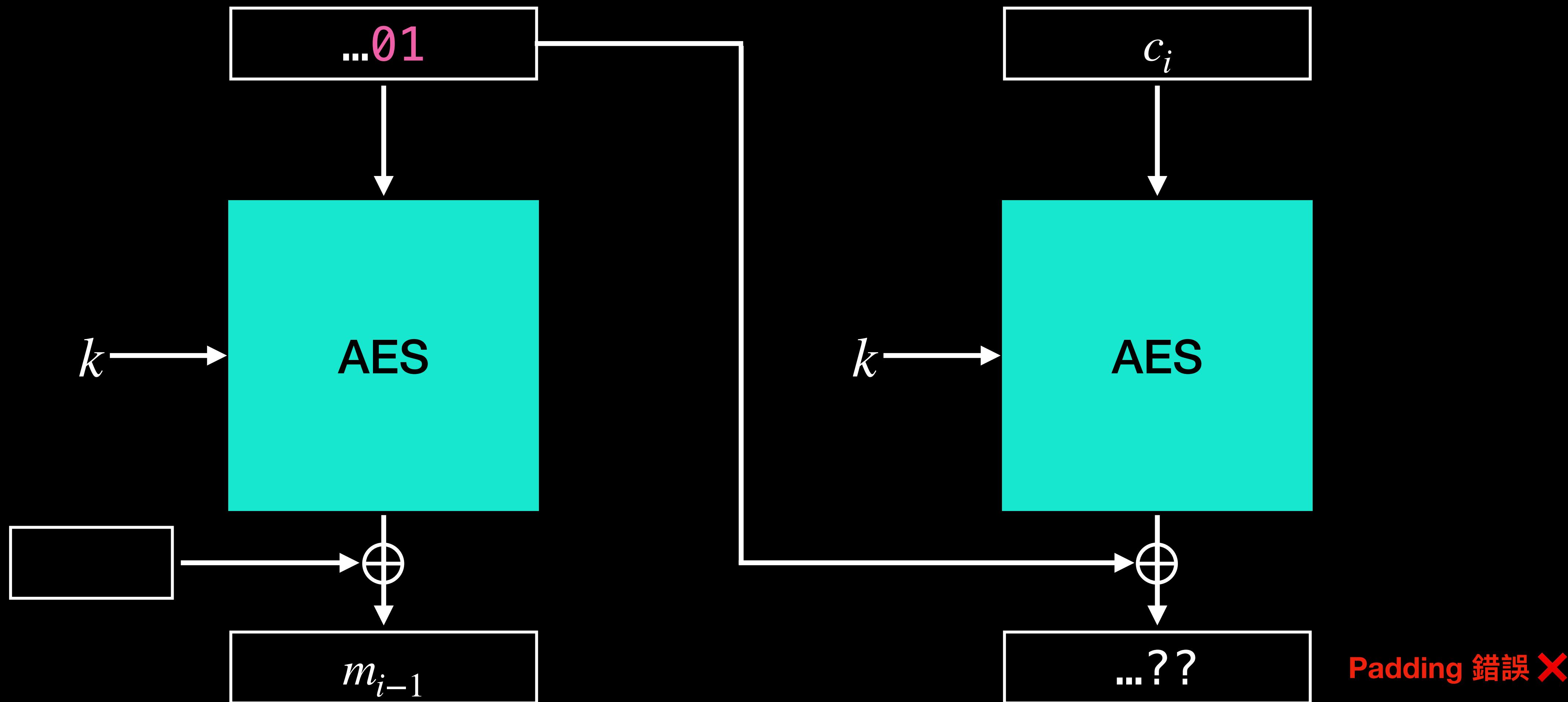
# 針對 CBC Mode 的 Padding Oracle Attack

- 暴力嘗試前一區塊的密文最後一 byte ,  
直到 padding 正確



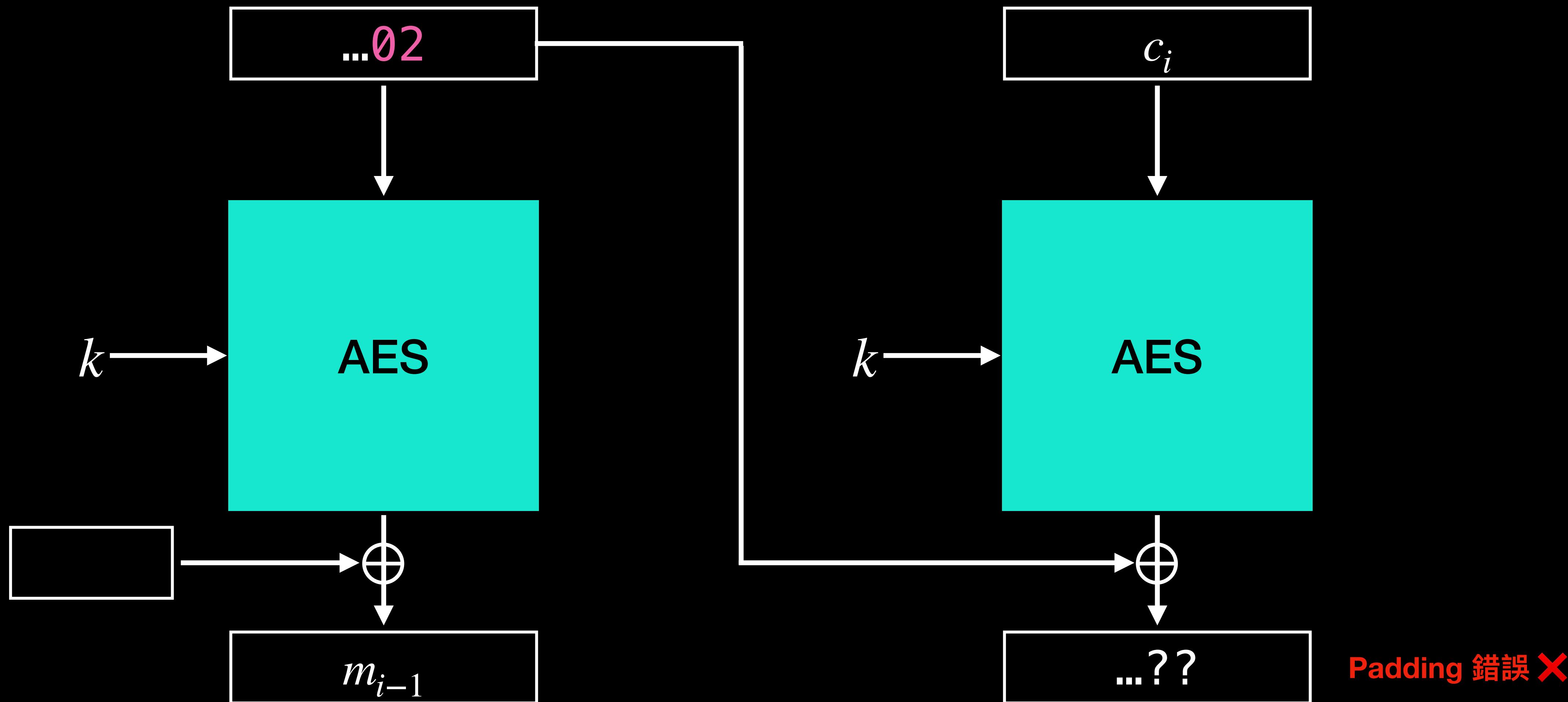
# 針對 CBC Mode 的 Padding Oracle Attack

- 暴力嘗試前一區塊的密文最後一 byte ,  
直到 padding 正確



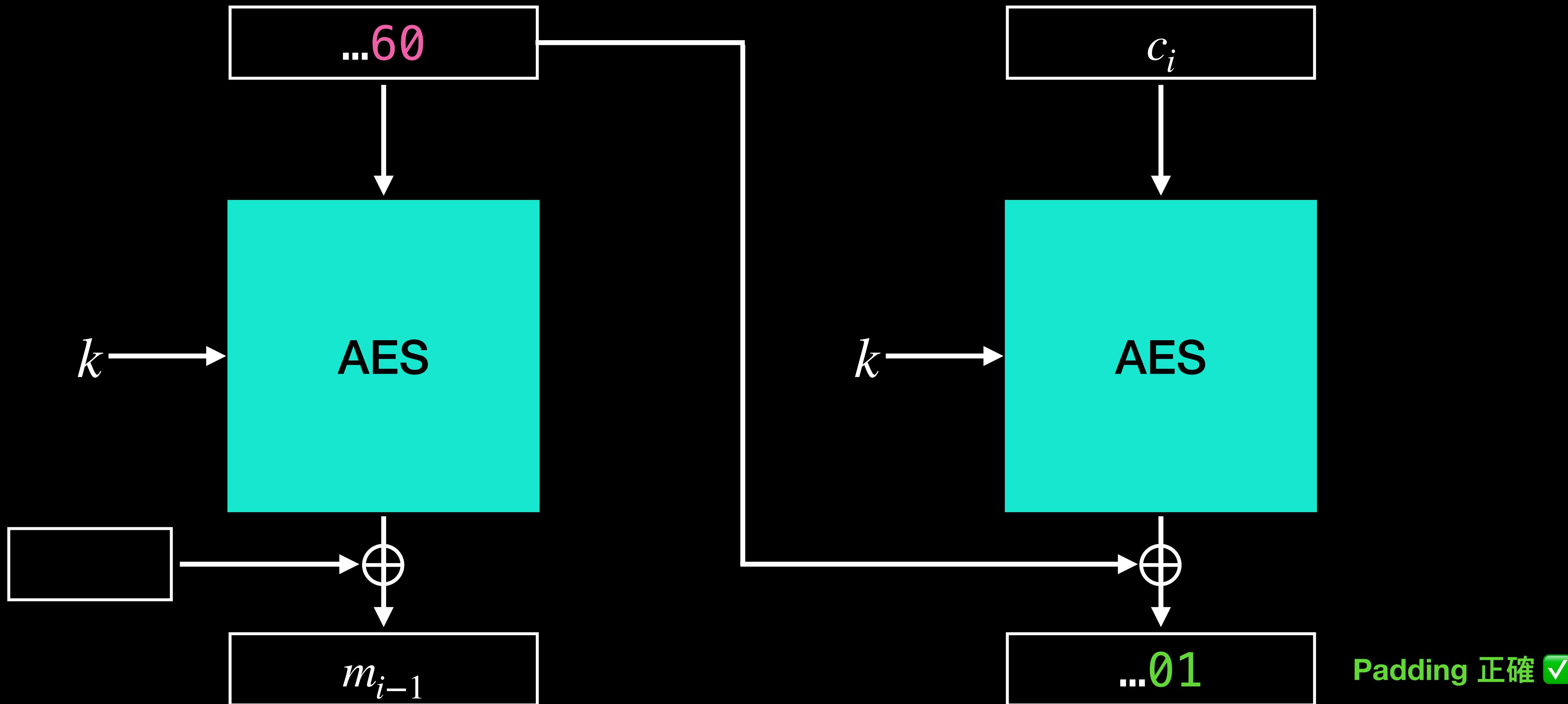
# 針對 CBC Mode 的 Padding Oracle Attack

- 暴力嘗試前一區塊的密文最後一 byte ,  
直到 padding 正確

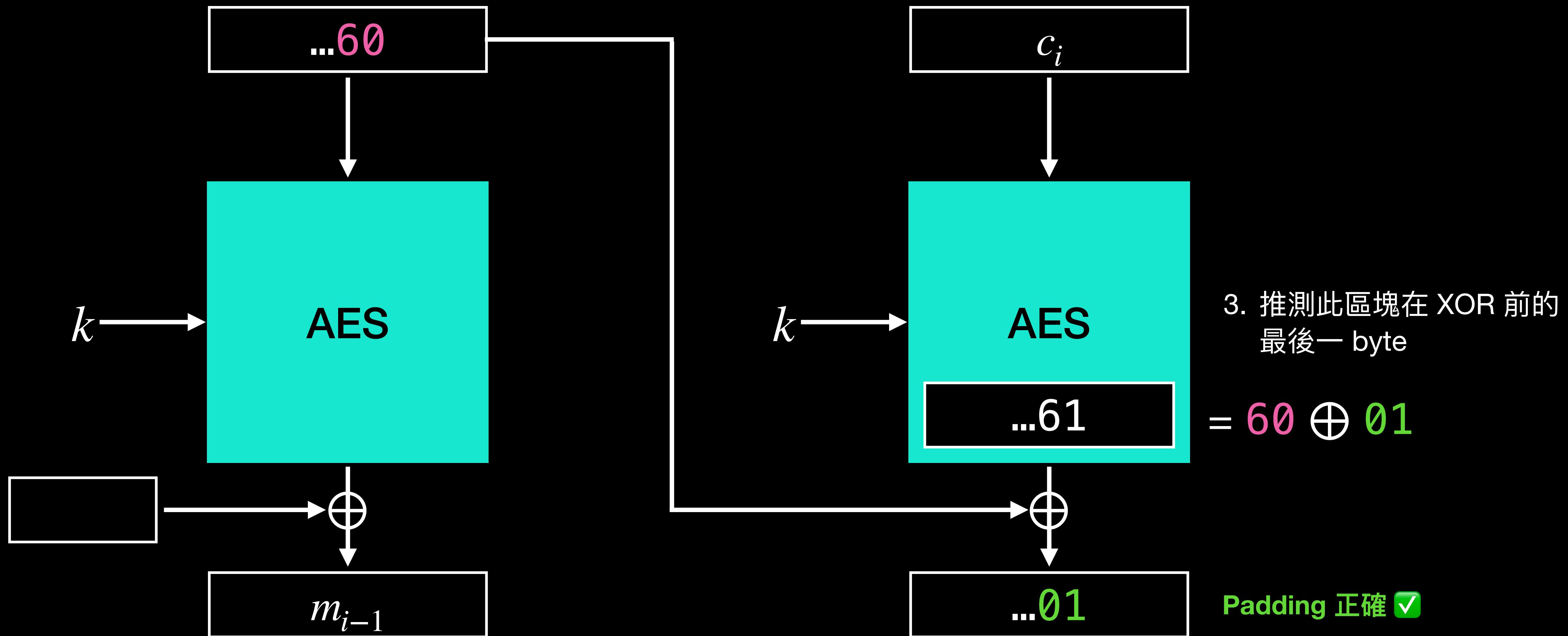


# 針對 CBC Mode 的 Padding Oracle Attack

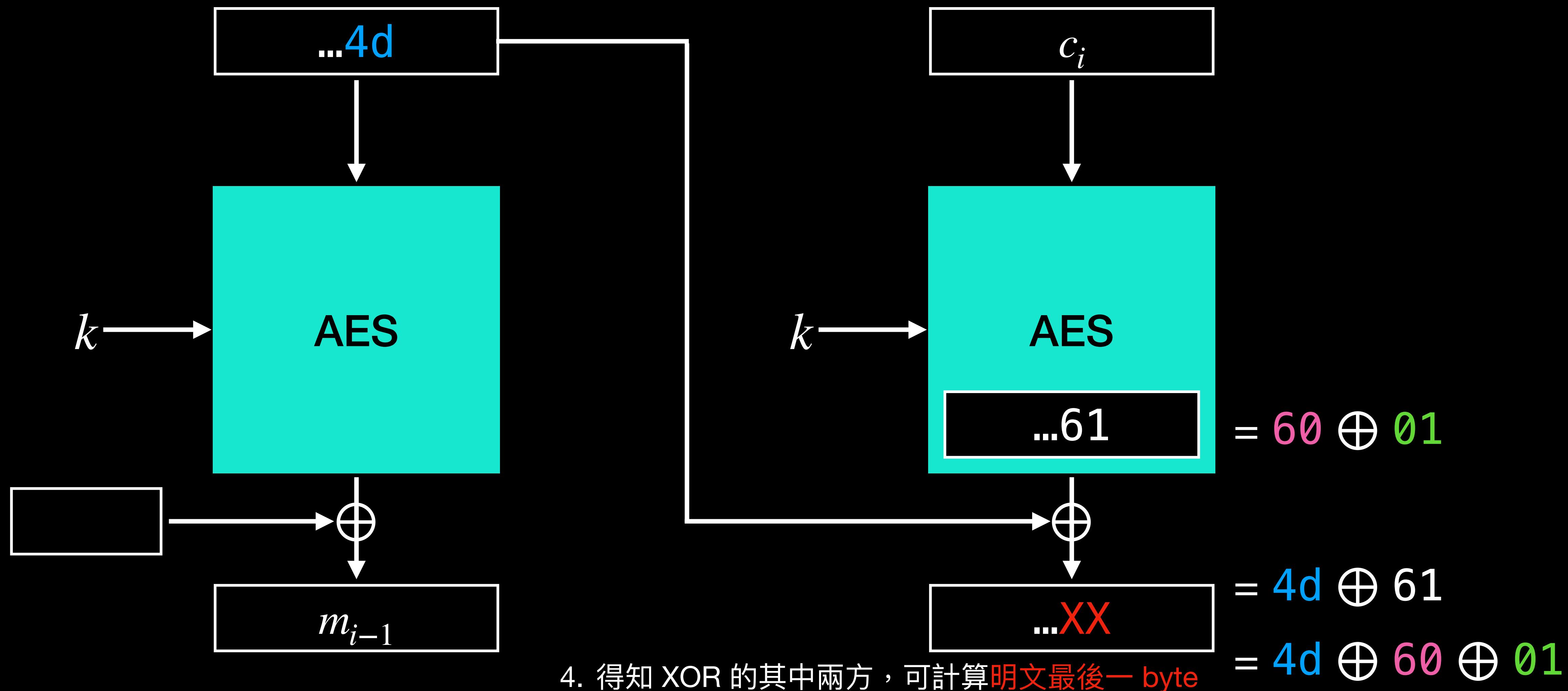
- 暴力嘗試前一區塊的密文最後一 byte ,  
直到 padding 正確



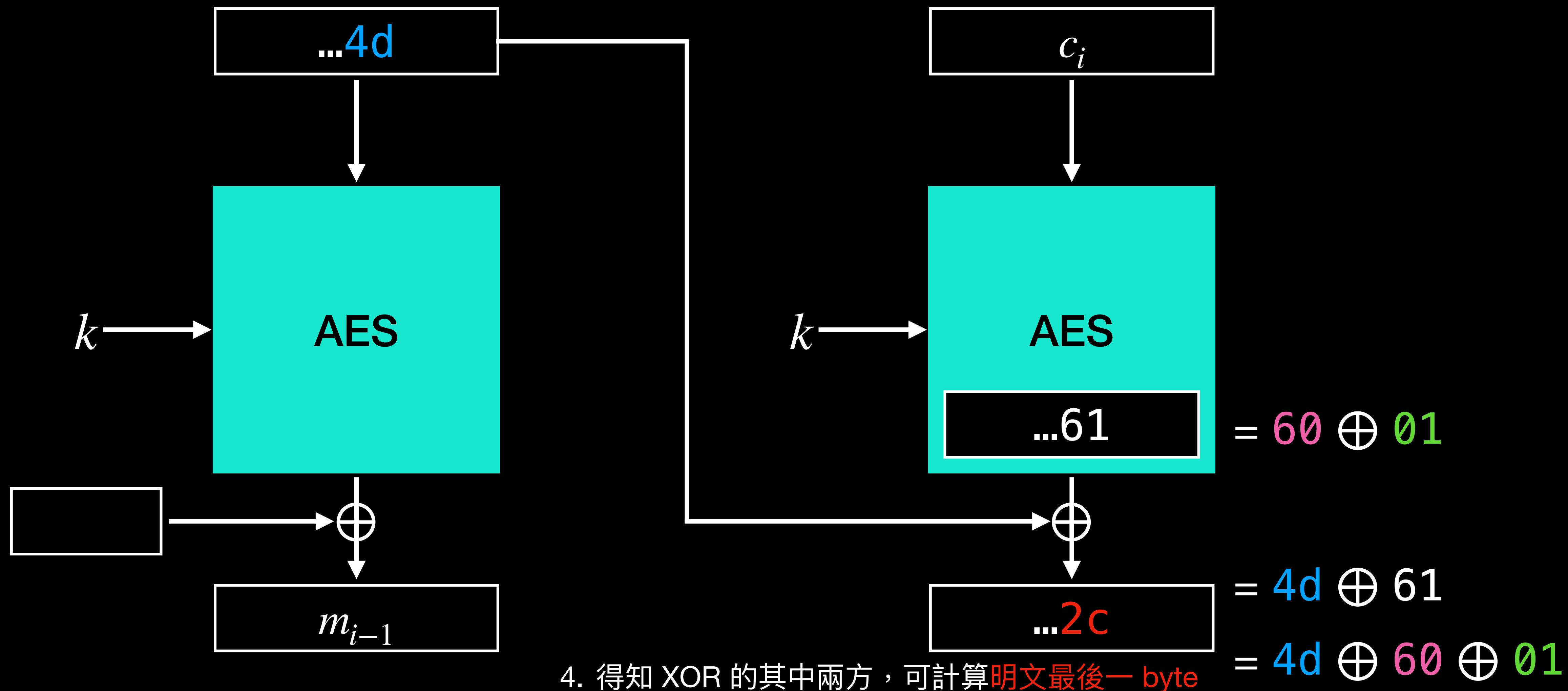
# 針對 CBC Mode 的 Padding Oracle Attack



# 針對 CBC Mode 的 Padding Oracle Attack

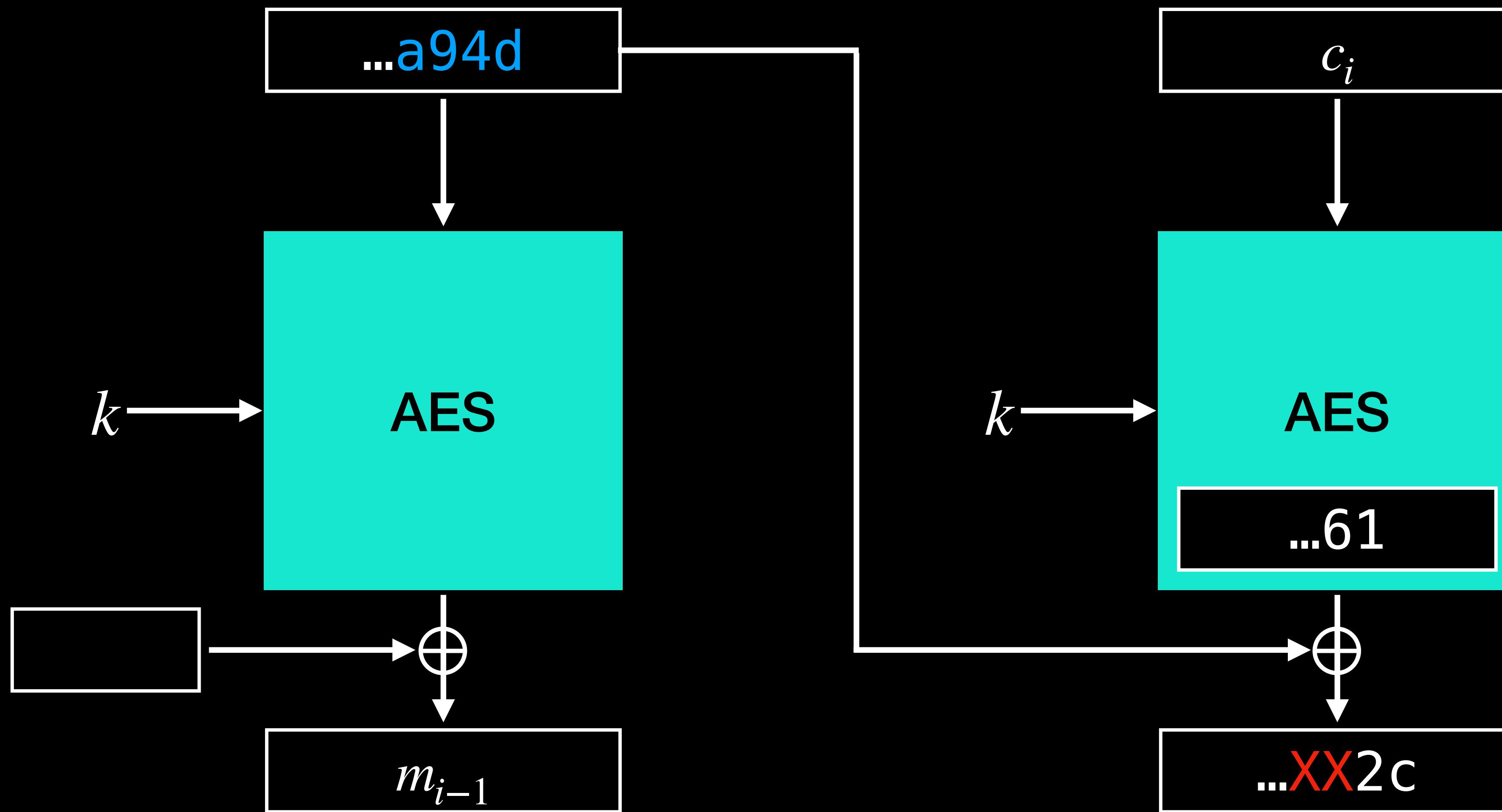


# 針對 CBC Mode 的 Padding Oracle Attack



# 針對 CBC Mode 的 Padding Oracle Attack

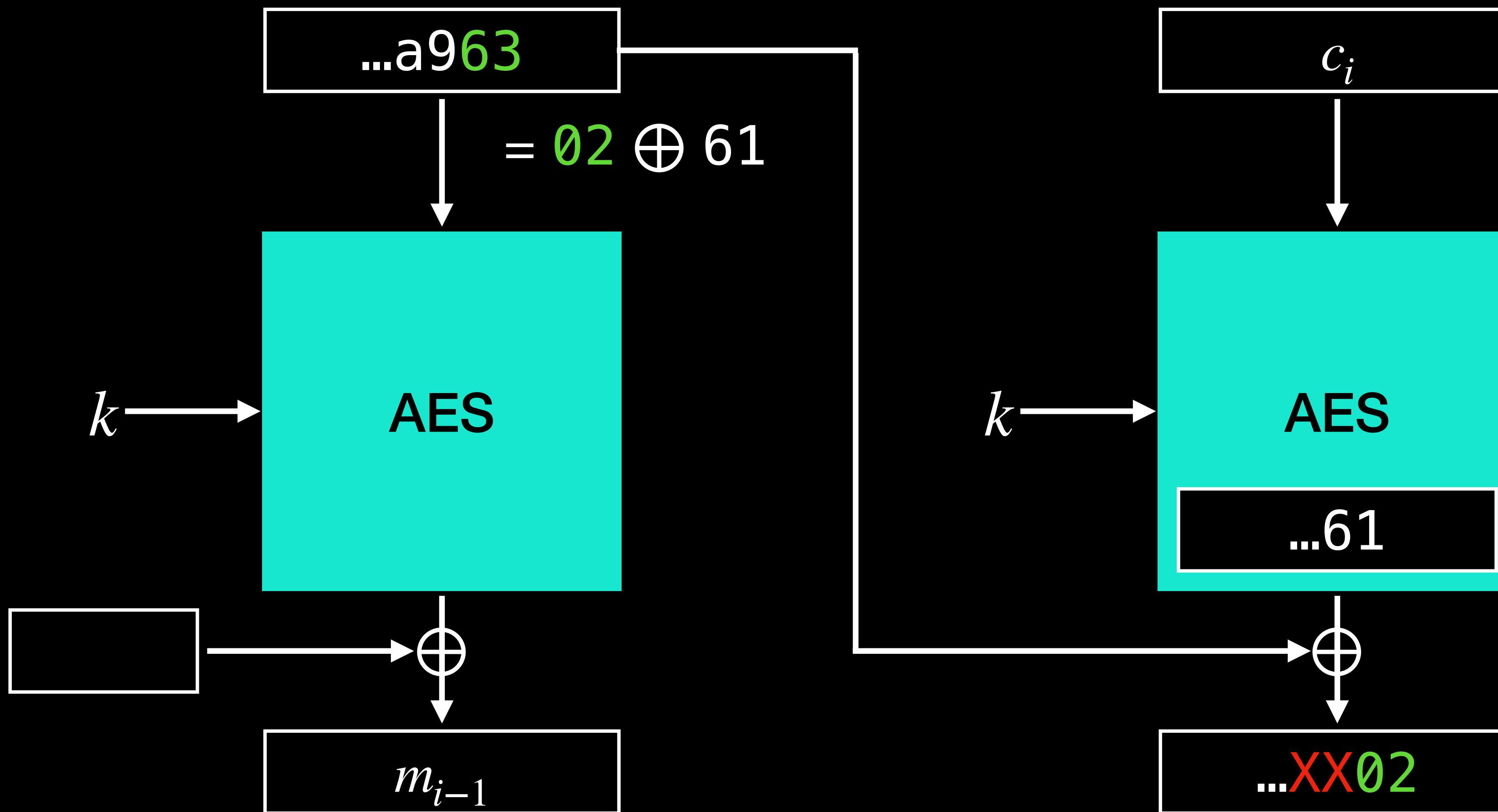
5. 已知前一區塊的密文最後二 byte



欲知此區塊的明文倒數第二 byte

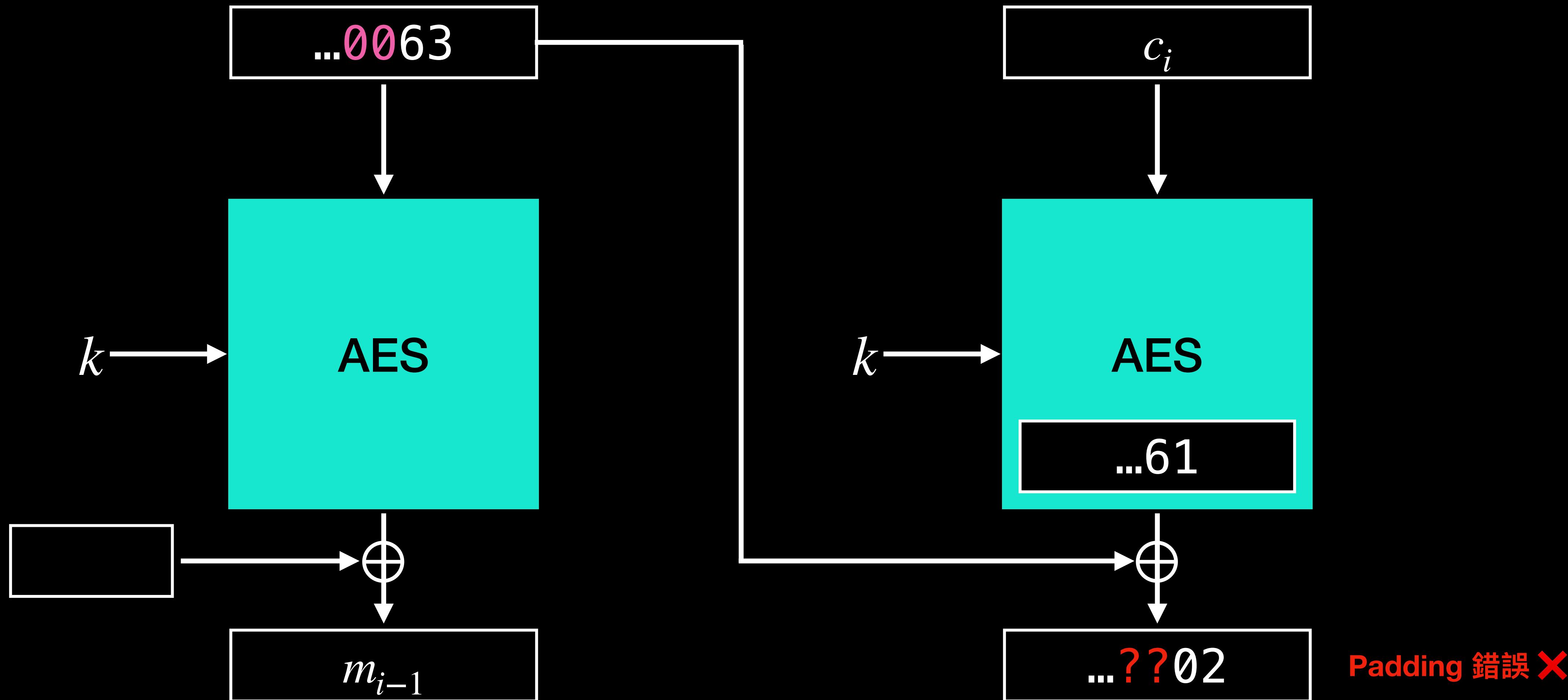
# 針對 CBC Mode 的 Padding Oracle Attack

6. 修改前一區塊的密文最後一 byte，使當前區塊明文最後一 byte 符合 padding



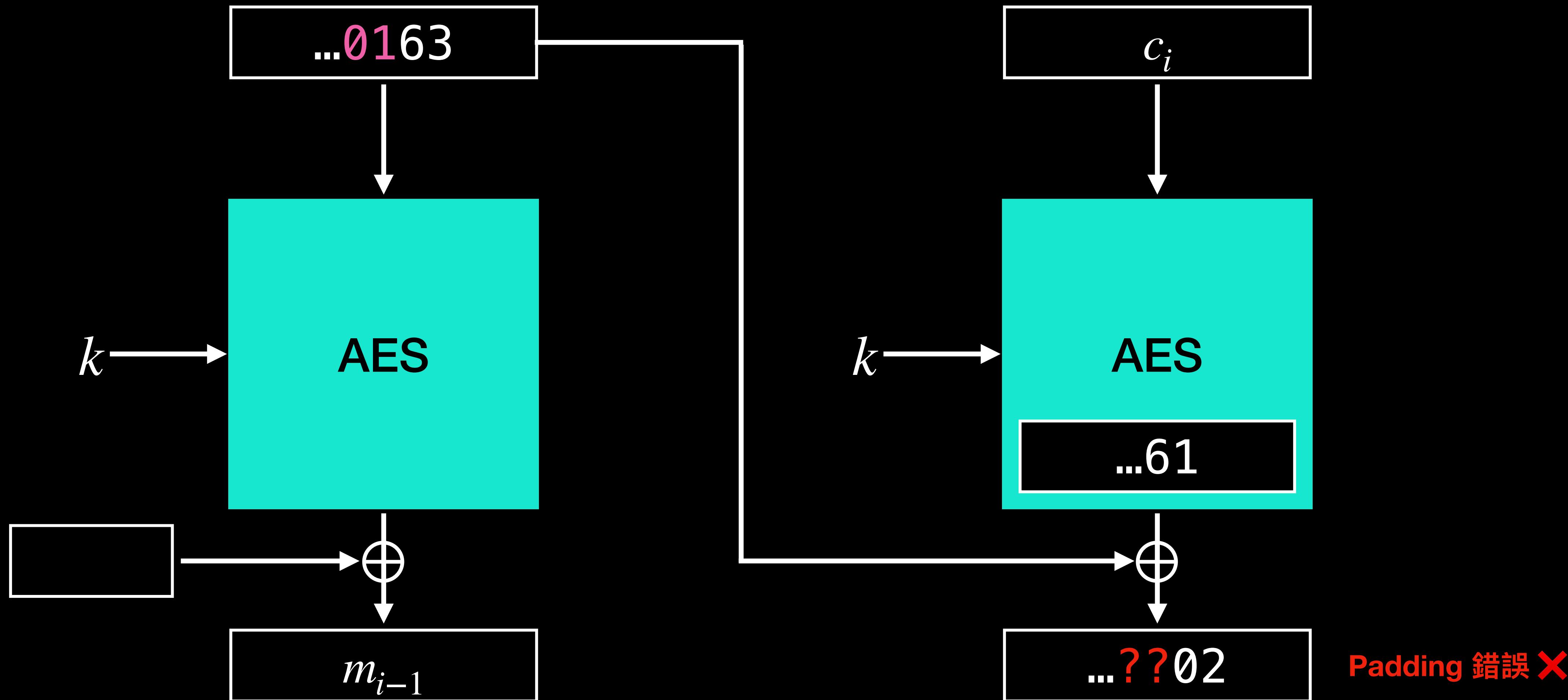
# 針對 CBC Mode 的 Padding Oracle Attack

- 暴力嘗試前一區塊的密文倒數第二 byte ,  
直到 padding 正確



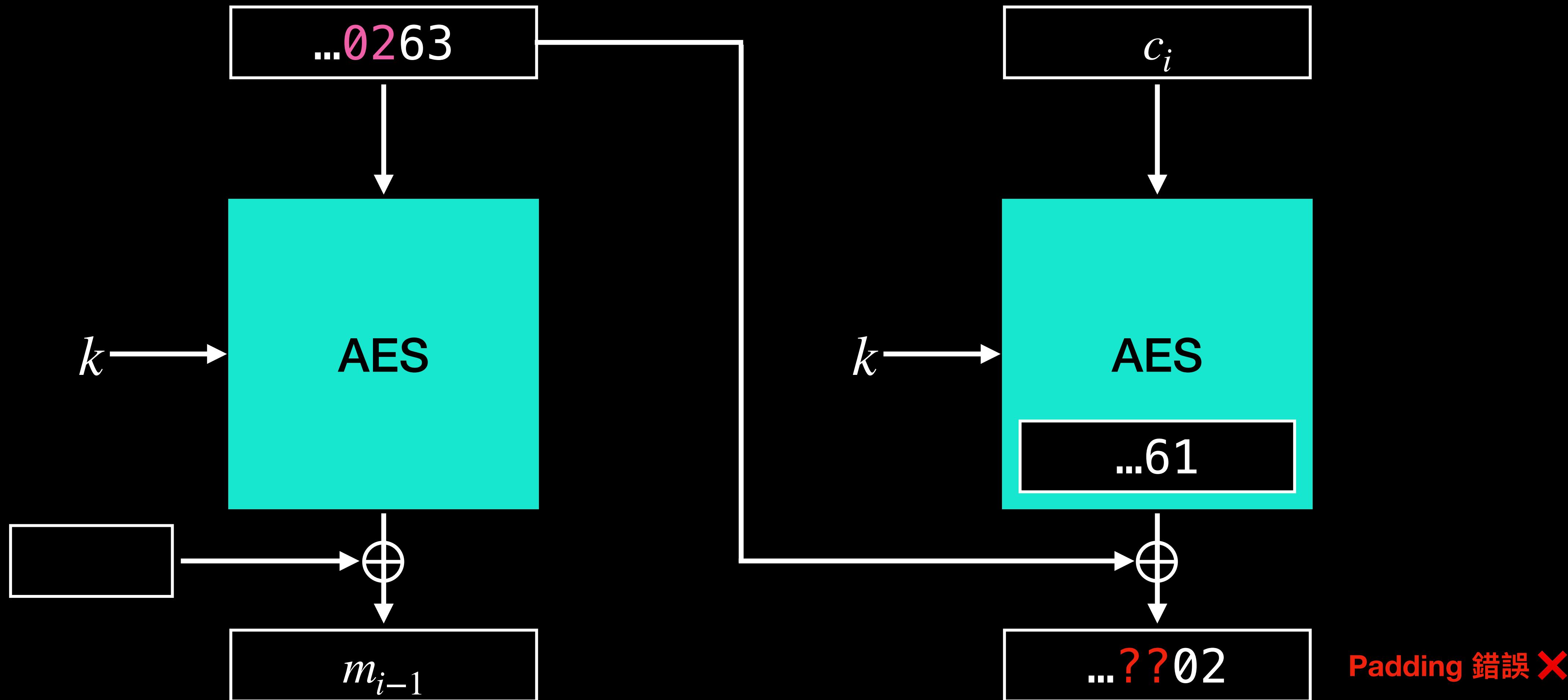
# 針對 CBC Mode 的 Padding Oracle Attack

- 暴力嘗試前一區塊的密文倒數第二 byte ,  
直到 padding 正確



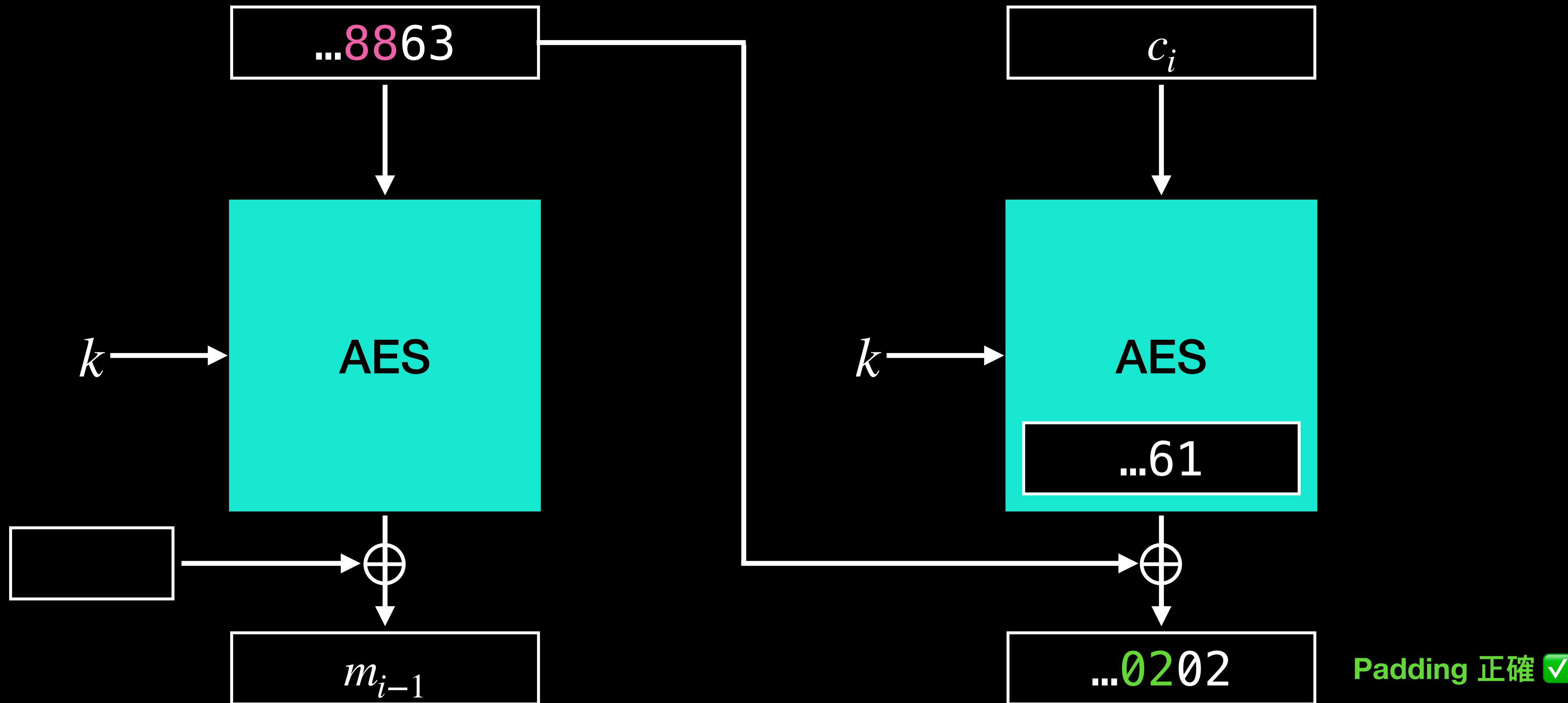
# 針對 CBC Mode 的 Padding Oracle Attack

6. 暴力嘗試前一區塊的密文倒數第二 byte ,  
直到 padding 正確

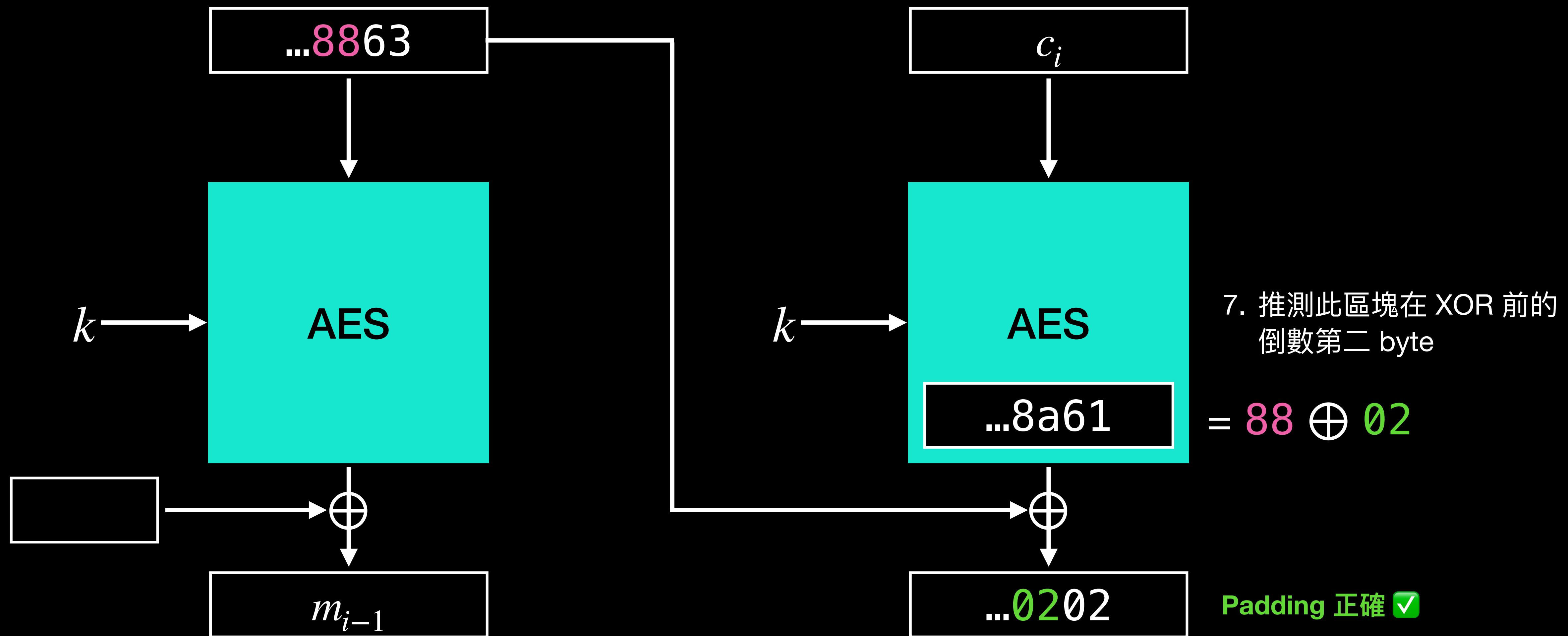


# 針對 CBC Mode 的 Padding Oracle Attack

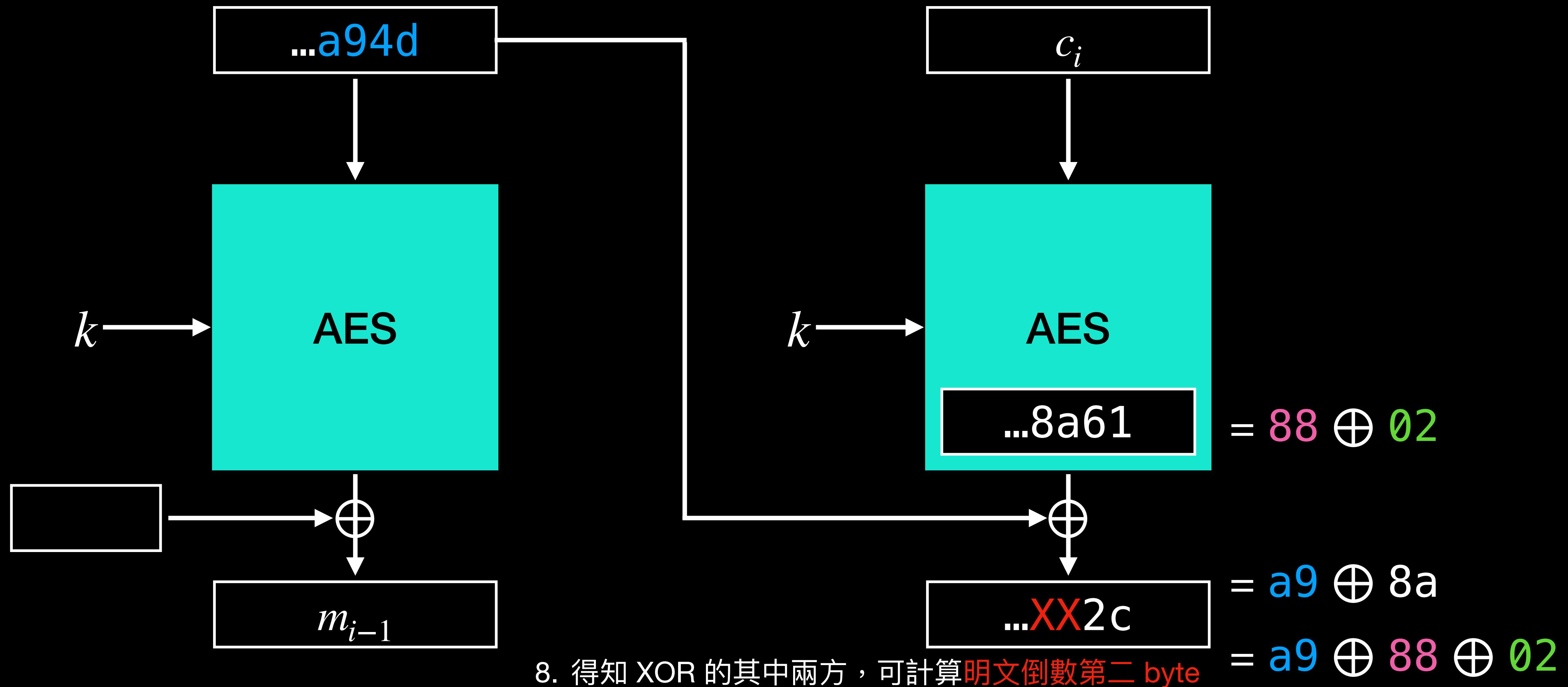
6. 暴力嘗試前一區塊的密文倒數第二 byte ,  
直到 padding 正確



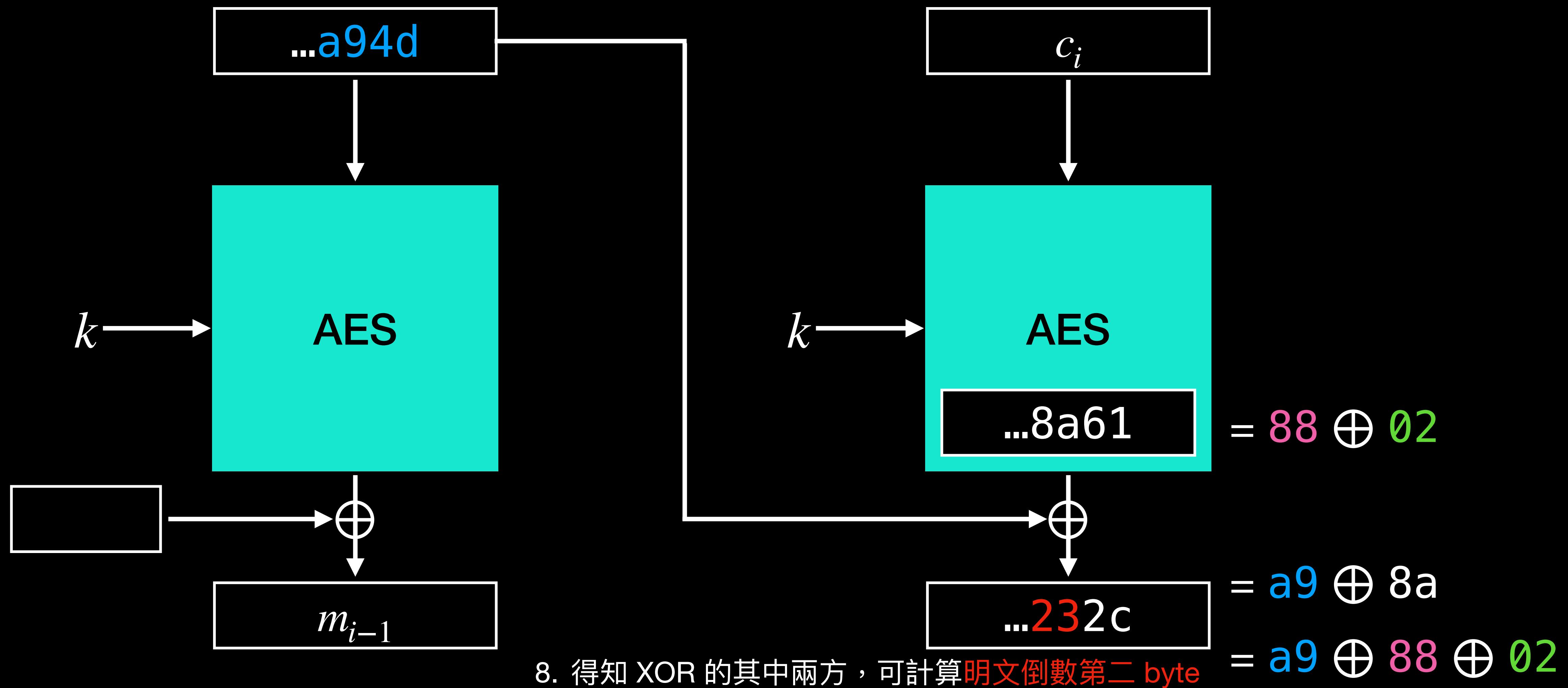
# 針對 CBC Mode 的 Padding Oracle Attack



# 針對 CBC Mode 的 Padding Oracle Attack



# 針對 CBC Mode 的 Padding Oracle Attack



# Lab : CBC / Padding Oracle Attack

## CBC / Padding Oracle Attack

100

閱讀題目程式碼 `chal.py`，然後嘗試修改 `solve_template.py` 中標示 `TODO` 的部分，對 CBC 工作模式發出 Padding Oracle 攻擊！

連線資訊：`nc 210.70.138.222 11007`

Flag 格式：`FLAG{...}`

Author: Ice1187

[chal.py](#)

[solve\\_template...](#)

# Lab : CBC / Padding Oracle Attack

```
iv  = os.urandom(16)
key = os.urandom(16)
aes = AES.new(key, AES.MODE_CBC, iv)
pad_flag = pad(flag, 16)
enc_flag = aes.encrypt(pad_flag)
print('iv:', iv.hex())
print('c:', enc_flag.hex())
```

```
while True:
    c = input('Give me c to decrypt (in hex): ')
    if c[:2] == '0x':
        c = c[2:]
    c = int(c, 16).to_bytes(len(c) // 2, 'big')
    iv,c  = c[:16], c[16:]
    aes = AES.new(key, AES.MODE_CBC, iv)

    if len(c) % 16:
        print('input size is not align to 16')
        break

    m = aes.decrypt(c)
    #print(m)
    try:
        m = unpad(m, 16)
        print('padding: correct')
    except ValueError:
        print('padding: incorrect')
```

# Lab : CBC / Padding Oracle Attack

```
> nc 210.70.138.222 11007
iv: 05570f2426aae06642b76f427436d4f0
c: 0e7e90a66bc8fccfe497f89819b5141e9642940b3169c92572aa7
7ba439da50c
Give me c to decrypt (in hex): 0e7e90a66bc8fccfe497f8981
9b5141e9642940b3169c92572aa77ba439da50c
padding: correct
Give me c to decrypt (in hex): 05570f2426aae06642b76f427
436d4f005570f2426aae06642b76f427436d4f0
padding: incorrect
```

# Lab : CBC / Padding Oracle Attack - solve\_template.py

```
# setup connection
host = '127.0.0.1'          • 將連線資訊修改成對的 IP 與 port
port = 9000
chal = remote(host, port)

def get_iv_c():
    chal.recvuntil(b'iv: ')  • get_iv_c 向題目取得 iv 和 c (可以忽略不用看)
    iv = chal.recvuntil(b'\n').strip()
    iv = int(iv, 16).to_bytes(len(iv) // 2, 'big')
    chal.recvuntil(b'c: ')
    c = chal.recvuntil(b'\n').strip()
    c = int(c, 16).to_bytes(len(c) // 2, 'big')
    return bytearray(iv), bytearray(c)

def send(data):
    # send input  • send 負責處理收發資訊 (可以忽略不用看)
    #print(data)
    chal.recvuntil(b'Give me c to decrypt (in hex): ')
    chal.sendline(data.hex().encode('ascii'))
```

# Lab : CBC / Padding Oracle Attack - solve\_template.py

```
flag = b''  
iv, c = get_iv_c()  
c = [c[i:i+16] for i in range(0, len(c), 16)]  
c = [iv] + c • 將 iv 視為第 -1 塊密文，方便操作  
num_blocks = len(c) - 1  
print('number of blocks:', num_blocks)
```

# Lab : CBC / Padding Oracle Attack - solve\_template.py

```
# TODO: support decrypt multiple block
before_xor = []
partial_flag = bytearray()
for pad in range(1, 16+1):
    # step 1, 5 1. 已知前一區塊的密文最後 / 倒數第二 byte
    prev_blk_byte = c[1][-pad]
    temp_c = deepcopy(c)

    # step 6: modify the previous block to make the suffix padding correct
    for j in range(1, len(partial_flag)+1):
        temp_c[1][-j] = before_xor[-j] ^ pad
```

# Lab : CBC / Padding Oracle Attack - solve\_template.py

```
# step 2: brute force the byte of previous block to find the byte before xor
for j in range(256): 2. 暴力嘗試前一區塊的密文最後一 byte ,
    # handle edge case      直到 padding 正確
    if partial_flag and pad == partial_flag[-1]:
        _before_xor = prev_blk_byte ^ pad
        break
    # handle edge case
    if j == prev_blk_byte:
        continue

temp_c[1][-pad] = j
pad_correct = send(b''.join(temp_c[:2+1]))
# step 3: compute the byte before xor
if pad_correct: 3. 推測此區塊在 XOR 前的最後一 byte
    _before_xor = j ^ pad
    break
```

# Lab : CBC / Padding Oracle Attack - solve\_template.py

```
# step 4: compute the byte of m
m_byte = prev_blk_byte ^ _before_xor
partial_flag.insert(0, m_byte)
before_xor.insert(0, _before_xor)

print('flag (block 2):', partial_flag.decode())
```

4. 得知 XOR 的其中兩方，  
可計算明文最後一 byte

# Lab : CBC / Padding Oracle Attack - solve\_template.py

```
# TODO: support decrypt multiple block
before_xor = []
partial_flag = bytearray()
for pad in range(1, 16+1):
    # step 1, 5
    prev_blk_byte = c[1][-pad]
    temp_c = deepcopy(c)

# step 6: modify the previous block to make the suffix padding correct
for j in range(1, len(partial_flag)+1): 6.修改前一區塊的密文最後一 byte，使當前
| temp_c[1][-j] = before_xor[-j] ^ pad      區塊明文最後一 byte 符合 padding
```

# Lab : CBC / Padding Oracle Attack - solve\_template.py

```
> python3 solve_template.py
[+] Opening connection to 210.70.138.222 on port 11007: Done
number of blocks: 2
flag (block 2): \x03
flag (block 2): \x03\x03
flag (block 2): \x03\x03\x03
flag (block 2): }\x03\x03\x03
flag (block 2): !}\x03\x03\x03
flag (block 2): 1!}\x03\x03\x03
flag (block 2): 01!}\x03\x03\x03
flag (block 2): 001!}\x03\x03\x03
flag (block 2): c001!}\x03\x03\x03
flag (block 2): _c001!}\x03\x03\x03
```

# 解法 : CBC / Padding Oracle Attack - solve\_template.py

```
flag = b''
for b in range(num_blocks, 0, -1):
    before_xor = []
    partial_flag = bytearray()
    for pad in range(1, 16+1):
        # step 1, 5
        prev_blk_byte = c[b-1][-pad]
        temp_c = deepcopy(c)

        # step 6: modify the previous block to make the suffix padding correct
        for j in range(1, len(partial_flag)+1):
            temp_c[b-1][-j] = before_xor[-j] ^ pad
```

# 解法 : CBC / Padding Oracle Attack - solve\_template.py

```
# step 2: brute force the byte of previous block to find the byte before xor
for j in range(256):
    if partial_flag and pad == partial_flag[-1]:
        _before_xor = prev_blk_byte ^ pad
        break
    if j == prev_blk_byte:
        continue

temp_c[b-1][-pad] = j
pad_correct = send(b''.join(temp_c[:b+1]))

# step 3: compute the byte before xor
if pad_correct:
    _before_xor = j ^ pad
    break
```

# 解法 : CBC / Padding Oracle Attack - solve\_template.py

```
# step 4: compute the byte of m
m_byte = prev_blk_byte ^ _before_xor
partial_flag.insert(0, m_byte)
before_xor.insert(0, _before_xor)

print(f'flag (block {b}):', partial_flag.decode())
```

# 攻擊對象：工作模式

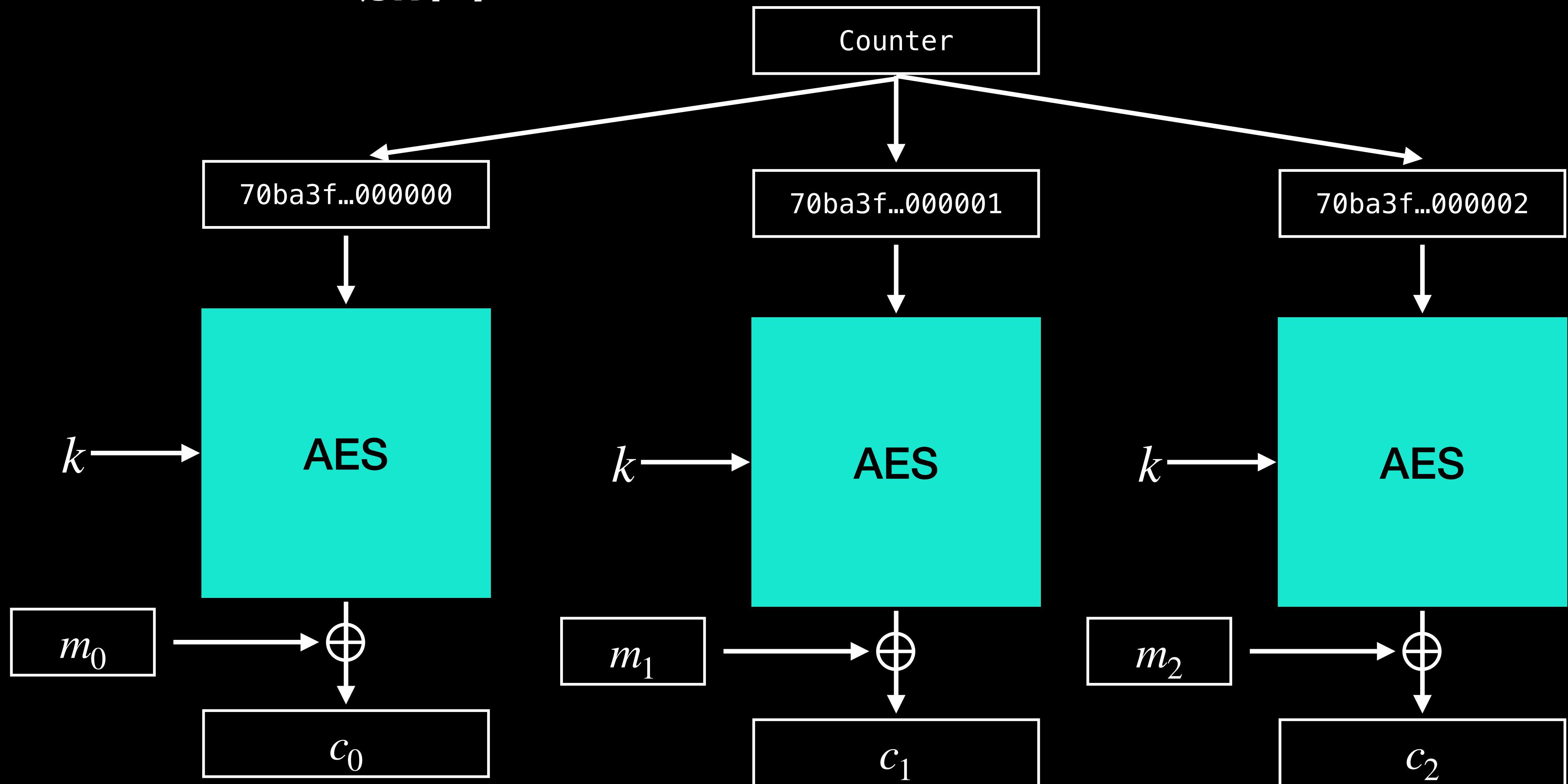
- AES 本身尚未發現比暴力破解更好的攻擊方式
- 上述攻擊是針對工作模式而非加密演算法
  - AES 本身沒有問題，而是使用的工作模式有弱點
  - 其他區塊加密演算法使用相同工作模式時，也可能有相同弱點

# 計數器工作模式 CTR Mode

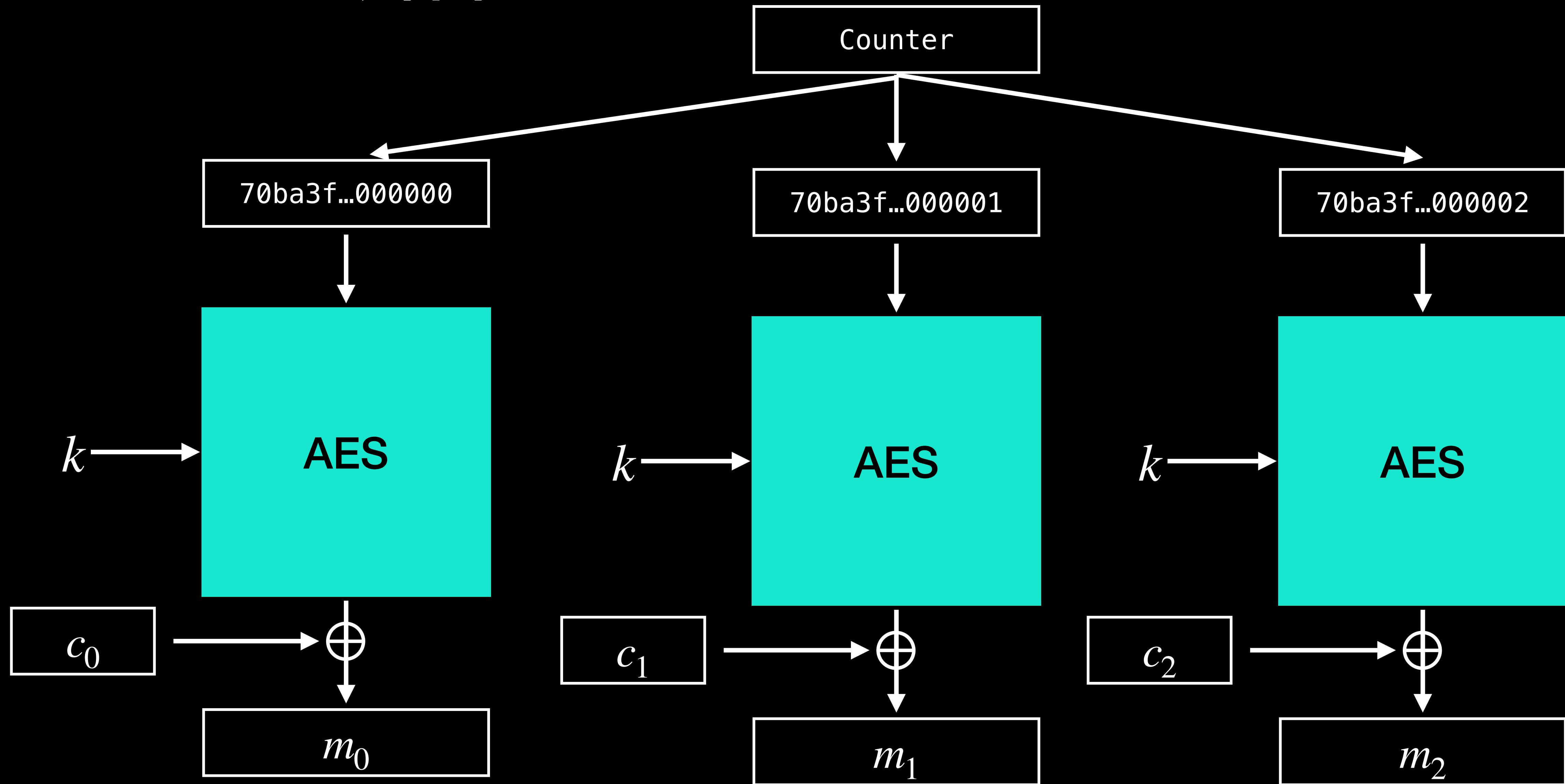
## Counter Mode

- 把區塊加密變成串流加密
  - counter：隨機數生成初始值 nonce 後，每次加 1
- 優點
  - 加密平行化
  - 解密平行化
- 缺點
  - Nonce 必須隨機
  - Nonce 不可重複

# CTR Mode 加密



# CTR Mode 解密



# 計數器工作模式 CTR Mode

## Counter Mode

- 把區塊加密變成串流加密
  - 區塊加密：視為偽隨機數產生器
  - 將區塊加密輸出與明文做 XOR
  - 不需要做 padding

# 總結：區塊加密

- 基於乘積密碼 (Product Cipher) 概念設計的加密方式
- Padding : input 為固定長度  $b$ ，需要將資料長度補齊至整除  $b$
- Mode : 使單一密鑰可用於加密多個區塊的方法，包含 ECB、CBC、CTR 等
- 加密 :  $Pad(m) = m_{pad}$  ,  $E_{mode}(m_{pad}, k) = c$
- 解密 :  $D_{mode}(c, k) = m_{pad}$  ,  $Pad^{-1}(m_{pad}) = m$
- 區塊加密加密演算法 : AES

# Reference

- [oalieno/Crypto-Course](#)
- [Wiki - Advanced Encryption Standard](#)
- [Wiki - Block cipher mode of operation](#)
- [Communication Theory of Secrecy Systems](#)