

Problem 1 表示答案, 表示關鍵字

Ans:

(a)

令  $P(x) = x^4 + x + 1$  作為不可約多項式

$$f_1(x) = x^2 + 1, \quad g_1(x) = x^3 + x^2 + 1$$

$$f_2(x) = x^3 + 1, \quad g_2(x) = x + 1 \quad \rightarrow \text{求 } f_i + g_i \& f_i \cdot g_i \bmod P(x), i \in \{1, 2\}$$

首先先看  $i=1$  的部分:

$$f_1 + g_1 \Rightarrow (x^2 + 1) \oplus x^3 + x^2 + 1 \rightarrow \begin{array}{r} x^3 & x^2 & x & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 \end{array} \rightarrow f_1 + g_1 \Rightarrow x^3$$

至於  $f_1 - g_1 \Rightarrow 1 + 1 \Rightarrow 0 \rightarrow 1 = -1 \rightarrow$  減掉某數同義於再加一次同一數

$\therefore f_1 - g_1$  可視為  $f_1 + g_1$ , 故  $f_1 - g_1 = x^3$

$$\text{首先求 } f_1 \cdot g_1 \Rightarrow (x^2 + 1)(x^3 + x^2 + 1) = x^5 + x^4 + x^3 + x^3 + x^2 + 1 \rightarrow x^5 + x^4 + x^3 + (1+1)x^2 + 1$$

$$\text{接著 } f_1 \cdot g_1 \bmod P(x) = (x^5 + x^4 + x^3 + 1) \bmod (x^4 + x + 1) \rightarrow x^5 + x^4 + x^3 + 1$$

$\rightarrow$  將  $P(x)$  的  $x^4 + x + 1$  替成  $x^4 = x + 1$  再代入  $f_1 \cdot g_1$  之後

$$(f_1 \cdot g_1) \bmod P(x) \equiv x^5 + x^4 + x^3 + 1 \Rightarrow x \cdot (x+1) + (x+1) + x^3 + 1$$

$$\Rightarrow x^2 + (1+1)x + (1+1) + x^3 \Rightarrow x^3 + x^2 \bmod P(x)$$

接著, 看  $i=2$  的部分,

$$f_2 + g_2 \Rightarrow (x^3 + 1) + (x + 1) \rightarrow x^3 + x + (1+1)$$

$$f_2 - g_2 \Rightarrow \text{同 } f_2 + g_2 \rightarrow x^2 + x$$

$$f_2 \cdot g_2 \bmod P(x) \Rightarrow (x^3 + 1)(x + 1) \bmod P(x) \rightarrow x^3 + x^2 + x + 1 \bmod P(x)$$

六次方人少  
不必約簡.

(b)

① 証明  $P(x) = x^4 + x + 1$  在  $\mathbb{F}_2$  上是不可約的

(1) 因為找不到一次因子  $\Rightarrow P(0) = 1 \neq 0, P(1) = 1 \neq 0$

(2) 檢查是否可以被唯一不可約的二次式  $x^2 + x + 1$  整除

$$\frac{x^4 + x + 1}{x^2 + x + 1} \Rightarrow x^2 + 1 \text{ 不能整除 } x^3 \text{ 无法整除}$$

$\because$  (1), (2) 兩原因，故該  $P(x)$  在  $\mathbb{F}_2$  上是不可約的

② 求  $\alpha$  在  $\mathbb{F}_{2^4}^\times$  中的 Multiplicative Order

$\mathbb{F}_{2^4}^\times$  為循環群  $\hookrightarrow$  階  $|\mathbb{F}_{2^4}^\times| = 2^4 - 1 = 15$

因此  $\text{order}(\alpha) \leq 15$ . 只會發生在  $\alpha \in \{1, 3, 5, 15\}$  中

$$\alpha^3 = \alpha^3 \neq 1;$$

$$\alpha^5 = \alpha^4 \cdot \alpha = (\alpha + 1) \alpha = \alpha^2 + \alpha \neq 1 \Rightarrow (\alpha^4 \equiv \alpha + 1 \text{ 之義})$$

$\because 3, 5$  都不是階  $\hookrightarrow$  剩下 15 這個可能

$\forall i: \alpha^{15} \text{ 於任一个 15 階族皆會成立} \Rightarrow \text{ord}(\alpha) = 15$

③ 判定  $P(x)$  是否為 primitive

$P(x)$  在  $\mathbb{GF}(2)$  上的 primitive  $\leftrightarrow$  其根  $\alpha$  在  $\mathbb{GF}_{2^4}^\times$  中的階  $= 2^4 - 1 = 15$

$\forall 15 = 3 \times 5 \Rightarrow$  若  $\alpha^3 \neq 1$  且  $\alpha^5 \neq 1$  則  $\text{order}(\alpha) = 15$

$\forall i: \text{order}(\alpha) = 15$  且能生成  $\mathbb{GF}_{2^4}^\times$  故  $P(x) = x^4 + x + 1$  為 primitive

## Problem 2

本題將使用流程+程式碼作答。

[Ans]

(a) pf that  $p(x) = x^4 + x + 1$  on  $F_2[x]$  is irreducible

①一次因式檢查:  $p(0) = 1$ ,  $p(1) = 1 + 1 + 1 - 1 \neq 0$

並無  $x, x+1$  的因式

②二次因式檢查:  $F_2$  上全部首一且常數 1 的二次式只有

$q(x) = x^2 + x + 1$  經長除法得  $p(x) = q(x)(x^2 + x) + x$

餘式  $\neq 0$

∴無一次、二次因式  $\Rightarrow$  故本多項式不可約

(b) 証明  $p(x)$  is primitive polynomial

設  $\alpha$  為  $p(x)$  在  $GF(16)$  的一根  $\Rightarrow$  需證明  $ord(\alpha) = 2^4 - 1 = 15$

在本題我們使用 python code 進行撰寫可知

$\because p(x)$  不可約且  $deg = 4$ ,  $\alpha \in GF(16)$ , 又有  $\alpha^{15} = 1$ , 故  $ord(\alpha) | 15$ ,

又程式可知,  $ord(\alpha) \neq 1, 3, 5 \Rightarrow ord(\alpha) = 15 \nmid 15 \Rightarrow$  故為 primitive polynomial

```

1  # -*- coding: utf-8 -*-
2  from sympy import Poly, symbols, GF
3
4  # Define symbol and field
5  x = symbols('x')
6  F2 = GF(2)
7
8  # (a) Irreducibility test
9  p = Poly(x**4 + x + 1, x, domain=F2)
10 print("(a) Is p(x) = x^4 + x + 1 irreducible over F2? ", p.is_irreducible)
11 print()
12
13 # (b) Primitivity test: find the smallest n such that p(x) | (x^n - 1)
14 orders = []
15 for n in range(1, 2**4):          # search 1 ... 15
16     if Poly(x**n + 1, x, domain=F2).rem(p).is_zero:    # +1 == -1 in GF(2)
17         orders.append(n)
18 print("(b) Values of n with p(x) | (x^n - 1):", orders,
19       "\n  → smallest n =", orders[0])
20 print()
21

```

## (c) primitive polynomials 和 longest LFSR 的關係

LFSR period: m-<sup>必</sup>若 LFSR 的特徵若 polynomials 是 primitive 的，則其根之階數為  $2^m - 1$ ；非 0 狀態和  $\text{GF}(2^m)$  同構

$$\Rightarrow \text{period} = 2^m - 1 \Rightarrow m\text{-sequence.}$$

應用於  $p(x) = x^4 + x + 1 \rightsquigarrow m=4; \text{period}=15$

Feedback:  $S_{t+4} = S_{t+1} \oplus S_t$  ( $x^4 = x + 1$ ) 並用 python code 寫

把 15 個非 0 狀態輸完  $\Rightarrow$  最長序列為：

$[0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1]$

```

22 # (c) Simulate a 4-stage LFSR with feedback polynomial p(x)
23 def lfsr(taps, init_state, length):
24     """
25         taps: list of tap exponents (excluding the highest degree 4);
26             [1, 0] means  $x^1$  and constant term.
27         init_state: list of 4 bits (MSB ... LSB), not all zero.
28     """
29     state = init_state[:]
30     seq = []
31     for _ in range(length):
32         output = state[-1]           # take the LSB as output
33         seq.append(output)
34         new_bit = 0
35         for tap in taps:
36             new_bit ^= state[-1 - tap]
37         state = [new_bit] + state[:-1]
38     return seq
39
40 taps = [1, 0]           #  $x^4 + x + 1 \Rightarrow$  taps at degrees 1 and 0
41 init = [1, 0, 0, 0]      # any non-zero initial state
42 m_seq = lfsr(taps, init, 15)
43 print("(c) 15-bit maximal-length sequence:", m_seq)
44 print()

```

## (d) polynomial Identification

step ①: 係數 8 係:  $x^4 + a_3x^3 + a_2x^2 + a_1x + 1$ ,  $a_i \in \{0, 1\}$

②: 使用 python 寫並且發現僅有 3 係為不可約。

$$\text{P}_1 = x^4 + x + 1$$

$$\text{P}_2 = x^4 + x^3 + 1$$

→ 接下來作 primitive check

$$\text{P}_3 = x^4 + x^3 + x^2 + x + 1$$

③  $x^4 - 1 = (x - 1)(x^3 + x^2 + x + 1) = (x - 1)(x^2 + 1)(x + 1)$  為因數其中, 3, 5 為質因數

拿  $\text{P}_1, \text{P}_2, \text{P}_3$  去作  $x^5 \bmod p$ .  $x^3 \bmod p \Rightarrow$  發現  $\text{P}_3$  並非 primitive

故, 只有  $x^4 + x + 1$ ,  $x^4 + x^3 + 1$  既不可約又是 primitive.

```

46 # (d) List all monic, degree-4, irreducible polys over F2
47 # and identify which of them are primitive
48 irreducible_polys, primitive_polys = [], []
49
50 for coeffs in range(16): # enumerate middle 3 bits of the polynomial
51     q = Poly(x**4 + sum(((coeffs >> i) & 1) * x**i for i in range(4)),
52               x, domain=F2)
53     if q.is_irreducible:
54         irreducible_polys.append(q)
55         # determine the order of x mod q(x)
56         for n in range(1, 2**4):
57             if Poly(x**n + 1, x, domain=F2).rem(q).is_zero:
58                 if n == 2**4 - 1:
59                     primitive_polys.append(q)
60                     break
61
62 print("(d) All monic, irreducible degree-4 polynomials over F2:", irreducible_polys)
63 print("    Primitive polynomials among them:", primitive_polys)

```

problem ↗ code 執行結果截圖:

(a) Is  $p(x) = x^4 + x + 1$  irreducible over F2? → True

(b) Values of n with  $p(x) | (x^n - 1)$ : [15] → smallest n = 15

(c) 15-bit maximal-length sequence: [0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1]

(d) All monic, irreducible degree-4 polynomials over F2: [Poly( $x^4 + x + 1$ , x, modulus=2), Poly( $x^4 + x^3 + 1$ , x, modulus=2), Poly( $x^4 + x^3 + x^2 + x + 1$ , x, modulus=2)]  
 Primitive polynomials among them: [Poly( $x^4 + x + 1$ , x, modulus=2), Poly( $x^4 + x^3 + 1$ , x, modulus=2)]

PS C:\Users\蘇柏叡\Desktop\|

### problem 3

Ans

本題將會針對 Berlekamp-Massey Algo . EEA Algo 解題流程作說明，並且於 problem 3 / main.py 中詳細以 python 實寫 2 個 Algos 的實作。

#### (a) Berlekamp - Massey Algo 解題流程

① 定義 Discrepancy:

對 sequence  $s_0, s_1, s_2, \dots$  在第  $N$  步計算  $d_N = s_N + \sum_{i=1}^L c_i s_{N-i} \pmod{x}$

\* 其中， $L$  是當前估計的線性複雜度， $\{c_i\}$  則是現行連接 polynomial 系數。

② 更新規則：if  $\{d_N = 0\}$  則僅會將  $N \rightarrow N+1$

$d_N = 1$ ，則以上次更新時的多項式  $B(x)$  做「移位 + XOR」：

$$C(x) \leftarrow C(x) - x^{N-m} B(x)$$

且 when  $2L \leq N$  時，同步更新

$$L \leftarrow N+1-L, B(x) \leftarrow \text{old } C(x), m \leftarrow N$$

③ 終止條件：當  $N$  遍歷了全部 sequence 且不再更改  $L$  時，

$$\text{得 } C_{BM}(x) = 1 + c_1 x + \dots + c_L x^L$$

→ 此為最小多項式

#### (b) EEA Algo 解題流程

① 首先，先以 Berlekamp - Massey Algo 得到  $L$

② 構造多項式對

$$A(x) = x^{2L}, S(x) = \sum_{i=0}^{N-1} s_i x^i$$

承上題 b) 之流程

③ EEA iteration:

以  $r_0 = A, r_1 = S, u_0 = 1, u_1 = 0, v_0 = 0, v_1 = 1$

套用 polynomial 除法:  $q_k = \lfloor \frac{r_{k-2}}{r_{k-1}} \rfloor, r_k = r_{k-2} \bmod r_{k-1}$

並且更新  $u_k = u_{k-2} - q_k u_{k-1}, v_k = v_{k-2} - q_k v_{k-1}$

當  $\deg r_k < L$  時停止。

④ 取出最小 polynomial: 正規化  $v_k$  (leading coefficient = 1)

即得  $C_{EEA}(x) = x^4 + x^3 + 1$

(a), (b) 之結果圖如下: 即為  $x^4 + x^3 + 1$

```
PS C:\Users\蘇柏叡\Desktop\problem3> python main.py
【BM minimal polynomial】 x**4 + x**3 + 1
【EEA minimal polynomial】 x**4 + x**3 + 1
```

(c) 比較結果:

① 以結果而言,  $C_{BM}(x) = C_{EEA}(x) = x^4 + x^3 + 1$

② 球價性原因: BM, EEA 均是在  $GF(x)$  上求最短線性關係  
使 sequence 符合 LFSR feedback relationship

③ BM 是以 Discrepancy 和多項式調整

EEA 則是以多項式 gcd, 逆元計算

兩者均是在找使  $s(x)$  可整除 feedback polynomial 的單位多項式

## problem 4.

**Ans:** (a) Definition of Affine Matrix M and prove that it is invertible

$$V = [1, 1, 1, 1, 0, 0, 0, 0]^T$$

並且以 cyclic right-shift 方式生成  $8 \times 8$  的 circulant matrix M

$$M = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

\* 可逆性證明：

① 行列式計算：在  $\mathbb{F}_2$  上，circulant Matrix 的行列式等於產生多項式。

$$V(z) = V_0 + V_1 z + V_2 z^2 + V_3 z^3 + \dots + V_7 z^7$$

在所有  $z^8=1$  根上的值的乘積

$$\text{代入 } V(z) = 1 + z + z^2 + z^3$$

可以驗証  $z$  在  $-1$  次單位根時， $V(z) \neq 0$

因此在  $\det(M) \neq 0 \pmod{2}$ ，M 在  $\mathbb{F}_2$  上必定可逆。

引用：  
 [1] Joan Daemen and Vincent Rijmen.  
 The Design of Rijndael: AES — The Advanced Encryption Standard.  
 Springer-Verlag, Berlin / Heidelberg / New York, Nov. 26 2001.

[2] Bosung Sun, Meicheng Liu, Jun Guo, Lei Qu, and Vincent Rijmen.  
 "New Insights on AES-Like SPN Ciphers."  
 In Advances in Cryptology — CRYPTO 2016, Lecture Notes in Computer Science, vol. 9814, pp. 605–624, 2016.

(b) S-Box 的程式碼附於 problem 4 / main.py

(c) 我們將程式碼加入檢驗以下 7 個 criterion 發現在這 7 個標準下，

我們實作的程式均有符合預期目標。詳介網將於 README.MD 上進行說明。

#	Criterion	Target	Result
1	Bijectivity	256 unique values	256 unique
2	Non-linearity	≥112	112
3	Strict Avalanche (SAC)	~50% bit flips	50.6%
4	Differential Uniformity	≤4	4
5	Linear-approx. bias	≤16	16
6	Algebraic degree	7	7
7	Fixed-point count	0	0

◎ 在下一頁！

(d) Cyclic Rotation 1111000 作為 secure Affine layer

\* pros and cons:

- \* pros:
  - ①擴散性: circular 結構可使任一輸入位在 Affine 計算後影響多達 4 個 output bits, 並且在後續級性 layer 中擴散至全字元。
  - ②對稱性: 每列是同一個基向量的循環移位, 確保每個輸出位有相同結構複雜度。
  - ③易於實作: 只要存一個 8 bits 向量, 便可由位移動態產生整個 Matrix

\* cons:

- ①基向量選擇不恰當: 若是連續 1 的長度太短, 可能導致行列式值 = 0 或是矩陣退化、擴散能力不足...等。

②需要嚴格驗証所選向量之產生的多項式在  $\mathbb{F}_2$  上不含任何 8 次單位根的 root 才可保證可逆性、分支數量最優。

引用:[1] Joan Daemen and Vincent Rijmen.  
The Design of Rijndael: AES — The Advanced Encryption Standard.  
Springer-Verlag, Berlin / Heidelberg / New York, Nov. 26 2001.

[2] Bosung Sun, Meicheng Liu, Jun Guo, Lei Qu, and Vincent Rijmen.  
"New Insights on AES-Like SPN Ciphers."  
In Advances in Cryptology — CRYPTO 2016, Lecture Notes in Computer Science, vol. 9814, pp. 605 - 624, 2016.

## problem 5.

Ans:

(1) Four-table T-box 可以 merge SubBytes, ShiftRows, MixColumns

(1) SubBytes: 對每一 byte  $x \in [0, 255]$  作  $GF(2^8)$  反轉 & Affine 變換產生一個 8bit 值

(2) ShiftRows: 僅會改變 byte 在狀態陣列中的 index, 而不是內容。

(3) MixColumns: 對 ShiftRows 後相鄰四個 byte 作矩陣乘法, 運算可表示為對單一 byte 的 4 種  $(x, 3, 1, D)$  線性組合。

(4) 把上述 256 種可能的 input  $x$  預先計算  
且存於 4 張  $256 \times 32$  bits 查表之中, 可對應不同 ShiftRows 偏移,

一次查表便可完成 3 次運算。

(b) Design a constant-time LUT scheme  $\Rightarrow$  Avoid cache-timing leakage on Modern CPU

(1) 固定存取次數: 不論 index 為何, 皆會對整張 256 元素的查表執行相同 (256-R) 的存取, 消除依 input 分支或是快取行變化之可能。

(2) Bit Mask 的選擇: 對每次掃描, 去計算  $mask = -(i == idx)$

其中, 該結果為全 1 或是全 0  $\Rightarrow$  再作位運算

result  $\oplus$  table[i] & mask, 將 target element 累加至 result  
其餘 elements 對最終值不會產生影響。

(3) 恒定 path: 整個查表過程無條件執行相同讀取、AND / XOR 操作,  
以確保時間不受到 index 影響。

(4) Python code 實作 constant-time LUT 的 code 將附於 problem 5/main.py

## (C) Implementation Comparison

### (I) Constant-time LUT

(i) Memory Footprint: ① 4个T-Box, 每个 $25b \times 4\text{bytes} \Rightarrow 1\text{KB}$

$$\textcircled{2} 16\text{KB} + 1\text{KB} = 17\text{KB} \quad \text{共 } 4\text{KB} \times 4 = 16\text{KB}$$

↳ code, Mask operations Logistic

(ii) Latency Delay: ① 每 byte 掃描 4 張 table  $\times 25b = 2(\text{AND/XOR})$   
共  $1024 = 2$

② 設每  $= 2$  AND/XOR 要 1 cycle  $\Rightarrow$  单 byte  $\Rightarrow 1024$  cycles

③ 16 byte block  $\Rightarrow 16 \times 1024$  cycles = 16384 cycles

### (II) Bit-sliced

(i) Memory Footprint: ① 只要少數常數、 $< 8$  个 128 bits 的 AVX2 register  
② 無需要大型查表, 近乎無 cache 壓力

(ii) Latency Delay: ① 1x 128-bit vector 同步處理 bit-plane  
② MixColumns bit-slice 計算約 60 = 2 vector  
AND/XOR/shift command  
③ loading, storing 約 20 cycles  
⇒ 合計約 80 cycles 遠小於 constant-time LUT

\* summary:

(1) Constant-time LUT 雖然可以有效抑制 Side-Channel attack  
但是, 其 Memory, Latency 之成本过高!

(2) Bit-sliced 實作 Memory 佔用低, 且具有低延迟, 且同樣能有效抑制  
Side-channel attack

## Problem b.

Ans:

(a) operation commutativity explanation

為使AES加密及解密流程更加對稱，將最後一輪MixColumns省略

(1) InvSubBytes 和 InvShiftRows 可以互換

以定義而言，InvSubBytes 是逐 byte 的 S-box 反查

至於，InvShiftRows 則是僅改變 byte 的位置

因為，其2種操作互相不干擾，因此可以滿足「可交換性」

換言之， $\text{InvSubBytes} \circ \text{InvShiftRows} = \text{InvShiftRows} \circ \text{InvSubBytes}$   
所以這2个modules，是可以共用相同邏輯單元！

(2) AddRoundKey 和 InvMixColumns 可以交換 (前提是修改 Round Key)

\* AES解密中原本操作為：InvMixColumns (State  $\oplus K$ )

利用GF(2<sup>8</sup>)線性性質

$$\text{InvMixColumns}(A \oplus B) = \text{InvMixColumns}(A) \oplus \text{InvMixColumns}(B)$$

因此原AES解密操作之 InvMixColumns (state  $\oplus K$ ) 可以被改寫成

$$\Rightarrow \text{InvMixColumns}(\text{State}) \oplus \text{InvMixColumns}(K)$$

此時，令  $K' = \text{InvMixColumns}(K)$  即可實現順序互換：

$$\text{AddRoundKey}(\text{InvMixColumns}(\text{State}), K')$$

因此，透過預處理 Round Key，可以共用 InvMixColumns 以及 AddRoundKey Modules.



## (b) Gate-Count savings

① software: 每輪 MixColumns  $\approx 200 \sim 250$  個 instructions

若是移除最後一輪 MixColumns 能節省約 10%

i) AES 原本共 10 輪，省去最後一 round  $\Rightarrow 10\%$

☆ 每輪 MixColumns 對 16 個 byte 作  $GF(2^8)$ , XOR

而每個 byte 要查表並作 3 次 XOR

## ② Hardware:

每輪 MixColumn 要 16 次  $GF(2^8)$  乘法，每次邏輯 gate 約需要 200~300 個

i) 共需要 3200~4800 個 gates

另外 XOR 共作 12 次，每次約 4 gates，合計 48 gates

因此若搭配共用設計，可省下約 3248~4848 個 gates

# Problem 7

Ans:

(1) Provide Formal Definition

(1) Definition of Irreducible Polynomial over  $F_2$ :

一个多项式  $f(x) \in F_2[x]$  是不可约的, 若且唯若它的次数  $> 1$ , 且无法在  $F_2$  上分解为次数更低的 2 个多项式乘积, 换言之, 若是  $f(x) = g(x)h(x)$ , 则必有  $\deg(g) = 0$  或  $\deg(h) = 0$

(2) Definition of Primitive Polynomial over  $F_2$ :

一个不可约多项式  $f(x)$  的根  $\alpha$  可生成  $F_{2^n}$  的所有非 0 元素, 则称此多项式为 primitive polynomial, 换言之, 若  $\alpha$  是  $F_2$  中的 primitive element, 则其极小多项式  $f(x)$  为 primitive polynomial。

\* primitive polynomial 必是 irreducible, 但是 irreducible polynomial 并非一定是 primitive

i) Irreducible 成为 primitive 的必要条件, 但非充分条件。

(1)  $p(x) = x^4 + x^3 + 1$

(1) Irreducible testing:

先试一次多项式  $x, x+1$

$p(0) = 0 + 0 + 1 = 1 \neq 0$  ] 无一次因式  
 $p(1) = 1 + 1 + 1 = 1 \neq 0$  ] 无一次因式

再试二次多项式:  $x^2+x+1, x^2+1$

i) 用  $(x^2+x+1)$ 去除  $p(x)$

$(x^2+x+1)$  去除  $p(x)$  均发现余数非 0

故  $p(x)$  为 irreducible

(2) 检查  $p(x)$  是否为 primitive

定义域为  $F_4$ , 其中非 0 元素构成 1 个阶是 15 的循环群, 因此要确认其根  $\alpha$  的最小正整数  $m$  使得  $\alpha^m = 1$  为 15  $\Rightarrow \text{order}(\alpha) = 15$

测试  $\alpha^m \neq 1$  for  $m \in \{1, 3, 5\}$  若均不为 1, 则阶为 15  
即为 primitive element

$\alpha^{15} = 1$ , 但因先前皆非 1  $\Rightarrow$  为 primitive polynomial

(c)  $q(x) = x^4 + x + 1$  testing

(1) Irreducible testing

先測  $x, x+1$

$$q(0) = 1 \neq 0, q(1) = 1+1+1 = 1 \neq 0$$

接著, 测  $x^2+x+1; x^3+1$

發現餘數皆  $\neq 0 \Rightarrow q(x)$  為 irreducible

(2) 測試是否 primitive

如同如我們一樣要看  $x$  的最小正整數  $d$ ,

使得  $x^d = 1$ , 若  $x^m = 1$  for some  $m < d$  則非 primitive

→ 反之, 則代表 primitive.

而在  $F_2[x]$  中, 所有非 0 元素皆是  $x^{15} = 1$  的根

∴ 所有不是 irreducible polynomial 的根皆滿足於  $x^{15} - 1 = 0$

→ proof that  $q(x) \mid x^{15} - 1$

接著, 要證明  $q(x) \nmid x^d - 1$  for  $d = 1, 3, 5$



$$d=1; x-1=x+1 \in F_2[x]$$

而  $q(x)$  為四次式  $\Rightarrow q(1) = 1+1+1 \neq 0 \Rightarrow q(x) \nmid x+1$

至於  $d=3; q(x) \mid x^3 - 1$ , 則其根  $x$  滿足  $x^3 = 1$

代表  $x$  階為 3  $\Rightarrow$  和 primitive 定義矛盾!

至於  $d=5; q(x) \mid x^5 - 1$ , 則  $x^5 = 1$

階為 5, 並非 primitive element

exp

$\therefore q(x) \mid x^{15} - 1$

且  $q(x) \mid x^d - 1$  for all  $d < 15$  且  $d \mid 15$

因此,  $q(x) \nmid x^d - 1$  for all  $d \in \{1, 3, 5\}$

by primitive polynomial 定義  $\Rightarrow q(x) = x^4 + x + 1$  為 primitive polynomial

(d) primitive polynomial 和 LFSR 的關係

若 LFSR 使用 degree- $n$  的 primitive polynomial 作為 connection polynomial

則其輸出 sequence 的週期為  $2^n - 1 \rightarrow$  Maximum-Length sequence

又: primitive polynomial 的 root 在  $F_{2^n}$  中具有最大乘法階, 能遍歷所有非 0 狀態

∴ 若是使用的 polynomial 僅為 irreducible 但並非 primitive, 則其根的階為

$d < 2^n - 1$ , 又 LFSR 的週期為  $d$ , 故無法生成 Maximum-Length sequence

## Problem 8

Ans.:

(a) State Shannon's definition of perfect secrecy

一个密碼 system 達到 perfect secrecy, 若对任何明文  $m \in M$  和密文  $c \in C$  有  $P(M=m | C=c) = P(M=m)$

換言之, 即使知道了密文, 也无法对明文的机率分布有任何額外資訊。

(b) (proof) 若一密碼 system 達成 perfect secrecy, 則  $|K| \geq |M|$

pf. 由 Shannon 的理論得知:

每組明文  $m \in M$  為均勻選取的, 且每個  $k$  定義一個 bijective function

此 function  $E_k: M \rightarrow C$ , 才能確保密文分布為均勻, 進而使得

$P(M=m | C=c) = P(M=m)$  因此至少需要不同的 key 對應到每個  $m$

意即  $|K| \geq |M|$

(c) Explain why one-time pad achieves perfect secrecy but is impractical for most real-world applications.

OTP 使用和明文等長隨機選取的 key  $\Rightarrow$  確保在統計上和明文是無關的。

→ 以符合 Shannon 定義

但因為<sup>(1)</sup> OTP 需要有和明文等長的 key  $\Rightarrow$  儲存成本高。

<sup>(2)</sup> key 只能使用一次

<sup>(3)</sup> key 的生成、分發極為困難 (需要安全通道)

∴ OTP 僅適用於極少數安全需求高的環境, 並不適合大量部署。

## (b) LCG的計算與漏洞分析：

(1) 我們將 generator 使用 python code 協助計算  
結果如下， $X_0$ 為一開始設定之 12345

```
a = 75
c = 74
m = 2**16
X = [12345]

# 計算前五項輸出，並append到X0後面
for _ in range(5):
    next_x = (a * X[-1] + c) % m
    X.append(next_x)

print("X0~X5: ", X[0:])
```

```
X0~X5: [12345, 8445, 43625, 60685, 29465, 47261]
```

另外， $X_1 = 8445$

$X_2 = 43625$

$X_3 = 60685$

$X_4 = 29465$

$X_5 = 47261$

(2) 作為 keystream 時的 2 項安全漏洞分析：

① Predictability：若 LCG 是線性函數，Attacker 只需要觀察數字 output，即可根據線性關係解出  $a, c \Rightarrow$  即可預測所有未來值。

p.f. 紿定  $X_0, X_1, X_2$ ，可解聯立方程：

$$X_1 = aX_0 + c \bmod m \quad \text{①}$$

$$X_2 = aX_1 + c \bmod m \quad \text{②}$$

解出  $a$  後可反推  $c$   
完全破壞隨機性。

$$\textcircled{2} - \textcircled{1} \Rightarrow X_2 - X_1 = a(X_1 - X_0) \bmod m \Rightarrow a = \frac{X_2 - X_1}{X_1 - X_0} \bmod m$$

② short period：  
若  $m = 2^n$ ，且  $a, c$  未設計為和  $m$  互質，則 period 會大幅度下降  $\Rightarrow$  最多為  $m$ ，不堪！

# (C) LFSR polynomial Analysis

$$p(x) = x^8 + x^4 + x^3 + x^2 + 1 \quad | \text{ 將利用 python code 檢驗是否 primitive / irreducible}$$

```

from sympy import Poly
from sympy.abc import x

# 定義多項式 p(x) = x^8 + x^4 + x^3 + x^2 + 1 over F2
p = Poly(x**8 + x**4 + x**3 + x**2 + 1, x, modulus=2)

# 檢驗不可約性
irreducible = p.is_irreducible

# 若不可約，則再檢驗是否為 primitive
primitive = False
if irreducible:
    # 將 sympy Poly 轉為 GF2 下的整數表示 (bitmask)
    def poly_to_int(poly):
        coeffs = poly.all_coeffs()
        deg = poly.degree()
        mask = 0
        for i, c in enumerate(coeffs):
            if c % 2:
                mask |= 1 << (deg - i)
        return mask

    f_int = poly_to_int(p)

    # GF(2) 多項式乘法
    def gf2_mul(a, b):
        res = 0
        while b:
            if b & 1: res ^= a
            b >>= 1
            a <<= 1
        return res

    # GF(2) 多項式取模
    def gf2_mod(a, mod):
        deg_mod = mod.bit_length() - 1
        while a.bit_length() - 1 >= deg_mod:
            a ^= mod << (a.bit_length() - 1 - deg_mod)
        return a

    # 快速幕模
    def gf2_pow(base, exp, mod):
        result = 1
        while exp:
            if exp & 1:
                result = gf2_mod(gf2_mul(result, base), mod)
            base = gf2_mod(gf2_mul(base, base), mod)
            exp >>= 1
        return result

    # primitive 檢測: 2^8 - 1 的所有真因數均不能做出 1
    order = 2**8 - 1
    divisors = [d for d in range(1, order) if order % d == 0]
    primitive = True
    for d in divisors:
        if gf2_pow(2, d, f_int) == 1: # 2 代表 x
            primitive = False
            break

    # 輸出結果
    print("Irreducible over F2:", irreducible)
    print("Primitive?", primitive)


```

Irreducible over F2: True  
Primitive? True

經程式驗証後，發現  $x^8 + x^4 + x^3 + x^2 + 1$  為不可約且 primitive。

(2) Berlekamp-Massey 可还原整條 sequence

若 Attacker 觀察到連續 2t 個輸出 bits, 即可使用 Berlekamp-Massey Algo 求出最小生成 polynomial, 而在本題中的 t 為 8, 已知 20 個連續位元  $> 2 \times 8$

故本題可以完全还原初始狀態及 keystream  $\xrightarrow{L} \text{LFSR 長度}$

⇒ 提出抗 Berlekamp-Massey Attack 方法以 + 安全性。

改進方法：① 加入非線性組合函數

把多個 LFSR 的輸出經過非線性 Boolean Function  
以提升線性複雜度以抵抗 Berlekamp-Massey Attack

② 使用 NFSR 中把 feedback 項改成非線性函數

e.g.: 把 feedback function 改成含 XOR, AND, OR ...

在適量的加入下, 可以在提升線性複雜度之下  
也能抵抗 BM, 又不會使硬體實現複雜度太高。

## Problem 9

Ans:

(a)  $\mathbb{Z}_p$  上 2 次方程求解

$$f(x) = ax^2 + bx + c \in \mathbb{Z}_p, \text{ 其中 } a \neq 0 \pmod{p}$$

$$\text{故, 欲解 } f(x) = 0 \text{ 等價於 } x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \pmod{p}$$

判別式為:  $\Delta = (b^2 - 4ac) \pmod{p}$

具有 3 種 case:

case 1:  $\Delta \equiv 0 \pmod{p}$ ; 只有一重根

case 2:  $(\frac{\Delta}{p}) = 1$ ; 有 2 个 distinct sols.

case 3:  $(\frac{\Delta}{p}) = -1$  無平方根, 無解

$\Rightarrow \mathbb{Z}_p$  上的二次方程  
最多只有 2 個解, 至於  
有幾個解倚賴於  
判別式。

(b) group  $G$  中元素  $ab$  的 order

給定條件:  $a^5 = e$ ;  $b^3 = e$ ;  $ab = ba$

又;  $ab$  以交換,  $\Rightarrow (ab)^n = a^n b^n \Rightarrow$  為使  $(ab)^n = e$

則需要  $a^n = e$  且  $b^n = e \Rightarrow n \equiv 0 \pmod{5}, n \equiv 0 \pmod{3} \Rightarrow n = \text{lcm}(3, 5) = 15$

$\therefore ab$  的 order 為 15

(c) Quadratic Residue Protocol Analysis

(i) Alice 如何確認  $y$  為 Quadratic Residue

可計算:  $y^{\frac{p-1}{2}} \pmod{p} \Rightarrow \begin{cases} 1 & \text{if } y \in QR_p \\ -1 & \text{if } y \notin QR_p \end{cases}$

此為拉格朗日符號  $(\frac{y}{p})$  之判定方法  $\Rightarrow$  確定性 Algorithm

(ii) Bob 能否作弊傳送 non-residue?

若 Alice 驗証  $y$  是否為 Quadratic Residue, 則 Bob 傳送 Non-Residue  
會使檢查失敗, 因此在此驗証步驟下, Bob 無法!

(iii) 和 Cryptography 之間的關聯：

此機制關聯到 Quadratic Residuosity Problem

即在不知道平方根的情況下，判斷一個元素是否為 Quadratic Residue 是困難問題。

因此，此問題為許多加密系統的安全性基礎。

(d) RSA 中使用小  $e$  的安全性分析

設公開指數為小質數  $e = 3$  or  $65537$ ，且 private key  $d$  滿足  $ed \equiv 1 \pmod{\varphi(n)}$

\* 選擇小  $e$  的理由：(1) 快速指數運算

(2)  $e$  的選擇，並不會影響 private key  $d$  的隨機性、長度。

\* 安全性分析：(1) 從公開的  $e$  推出  $d \rightarrow$  需計算  $\varphi(n)$

(2)  $\varphi(n) = (p-1)(q-1)$ ，計算需分解  $n$

(3) 若  $n$  為 2 大質數的乘積，分解極為困難

\* 只要  $d$  是隨機的，縱使  $e$  很小，仍可維持 RSA 安全性。

## Problem 10

Ans: (a) key revocation process

當原CTO解雇後，其持有的加密金鑰應立即抹除  
其流程如下：

① 金鑰標記狀態應要改成「Compromised」或是「Retired」，  
並且要停止所有用途並移除。

② 中止使用：停止該金鑰在所有應用系統中的使用

③ 金鑰儲存 or 銷毀：分為 2 種 case,

case 1：若仍需要存取歷史加密資訊，該金鑰需要  
安全地儲存在 HSM 或是受控備份中。

case 2：已明確不再需要該金鑰，應依照 NIST 800-57 7.1.3 中  
提到的把金鑰進行銷毀 eg: cryptoerase

△：CTO 是被炒掉的，因此在 step ③ 部分理應選 case 2。

(b) New key Deployment:

部署新的金鑰需要符合“保密性”、“可用性原則”，流程如下：

① 新金鑰的產生：在可信任環境 (eg: HSM) 下，產生新的金鑰，  
此金鑰 (可以是對稱 or 非對稱)，並且記錄其金鑰 ID、用途、產生時間。

② 金鑰派發：[對稱金鑰：透過 TLS + Mutual Authentication 分派至授權系統]

[非對稱金鑰：安全公開發布 public key，而 private key 由特定管理人員  
保管。]

③ 系統更新&驗証：確保所有 user 和服務已更新為使用新金鑰。  
並且已驗証其加解密功能正常。

④ 移除前任金鑰存取權限：從所有金鑰存取控制清單中移除前 CTO，確保其無法  
再利用任何方式取得金鑰資

## (C) Incident Response Timeline and Key Destruction

根據 NIST SP 800-61 事件回應流程與 SP 800-57 銷毀規範，應實施以下時間線

時間點	事件階段	說明
T <sub>0</sub>	偵測及通報	發現 CTO 解雇，啟動 key recovery 並通報資安單位。
T <sub>1</sub>	緊急隔離	將舊金鑰從所有系統移除、撤銷 其存取憑証、API 金鑰。
T <sub>2</sub>	啟動調查	檢視該金鑰是否於任職內有 濫用紀錄，並記錄處置過程。
T <sub>3</sub>	新金鑰部署	安全產生及派發新金鑰，更新金鑰 資料庫及使用端系統。
T <sub>4</sub>	安全銷毀	若不保留舊金鑰，應依據 NIST SP 800-57 使用 Crypto Erase or 擂壞寫以達不可逆銷毀。
T <sub>5</sub>	稽核與封存	建立完整處置紀錄，更新資產盤查報告，並且 封存此事件紀錄供未來查驗。

\* NIST SP 800-57 定義的金鑰銷毀條件

含 <sup>(1)</sup> Non-Recoverable

<sup>(2)</sup> Verifiable

<sup>(3)</sup> Audit-able

# Problem 1

Ans

(a) Explain How Alice's laziness might get her in trouble

以下列出可能 troubles:

(1) Modulus Reuse:

RSA 的 private key 為  $d = e^{-1} \bmod \varphi(n)$ , 但當  $n$  是固定時, 若 Attacker 可以利用任意一組  $(n, e_i)$  去破解  $\varphi(n) \Rightarrow$  就能还原出所有份的 private key。比如說, 若是  $e = 3$ , 且明文  $< n^{\frac{1}{3}}$  會遭受 Broadcast Attack

⇒ key recovery by single Factorization

因為  $n$  是由 2 个安全質數相乘而得, 故若是任一月份被破解, 則意謂著, 所有月份的 private key 全部都會被破解。

→ Attacker 只要攻擊一次  $n \Rightarrow$  能算出多組 private key  $d_i$   
甚至不管  $e$  是否一樣。

(b) pf. a concrete lattice attack

題設 Jan:  $N = p \cdot q$

$s_p, s_q$  皆為小的整數

eg:  $p \sim 10$  的質數差。

$$\text{Feb: } N' = (p + s_p)(q + s_q) = N + s_p \cdot q + s_q \cdot p + s_p s_q$$

step 1: 建立關係式:  $N' - N = s_p \cdot q + s_q \cdot p + \boxed{s_p s_q}$  → 可忽略不計

$$s_p \cdot q + s_q \cdot p \approx \Delta := N' - N$$

step 2: 改寫成線性近似式

$$\text{設 } X = p, Y = q \text{ 得 } s_q X + s_p Y \approx \Delta$$

step 4: 使用 LLL Reduction 得

$$\vec{z} = (X, Y) \approx (p, q)$$

若得  $p \rightarrow$  能計算  $q = \frac{N}{p}$

並驗證  $pq = N$

step 3: 建構 Lattice

$$\text{令 } \vec{v}_1 = (2^k, 0), \vec{v}_2 = (s_p, s_q), \vec{t} = (0, \Delta)$$

其中,  $2^k \approx N^{\frac{1}{2}}$  為 scaling factor

i. 因 Modulus 間差是極小, 所以, Attacker 能利用 Lattice Reduction 復原前一個月的  $p, q$  以完成 RSA 金鑰破解。

$$\text{建構 Lattice: } B = \begin{bmatrix} 2^k & 0 \\ s_q & s_p \end{bmatrix}$$

尋找  $\vec{z} = (X, Y) \in \mathbb{Z}^2$  使  $\|B \cdot \vec{z} - \vec{t}\|$  最小

## Problem 12

Ans:

### (a) Matrix Multiplication in AES Inverse MixColumns step

在 AES 的解密中, Inverse Mixcolumns 会把每个 column 和題目提供的固定矩阵进行乘法运算, 该运算并非標準整數矩阵乘法, 而是基於  $GF(2^8)$  的矩阵乘法  $\Rightarrow$  换言之, 此處的每个元素加法都是使用 XOR 乘法則是根據 AES 使用不可約  $m(x) = x^8 + x^4 + x^3 + x + 1$  進行 Modulo Polynomial Multiplication

以下 summarize 和標準矩阵的差異

i) 加法: 使用 XOR, 而非一般加法

ii) 乘法: 使用 Modulo Polynomial Multiplication 下的 **有限域乘法**

iii) 為單位元算  $\Rightarrow$  無進位, 故适合硬体實現

### (b) Multiplication by 9, 11, 13, 14 in $GF(2^8)$ using simple operations

在  $GF(2^8)$  中, 可以把乘上 9, 11, 13, 14 透过乘以 2(Xtime) 和 XOR 递推

$$\text{令 } \text{xtime}(a) \text{ 代表 } a \times 2 \Rightarrow \text{xtime}(a) = \begin{cases} (a \ll 1), & \text{if } a < 0x80 \\ ((a \ll 1) \oplus 0x1B), & \text{if } a \geq 0x80 \end{cases}$$

$$a \cdot 9 = \text{xtime}(\text{xtime}(\text{xtime}(a))) \oplus a$$

$$a \cdot 11 = \text{xtime}(\text{xtime}(\text{xtime}(a))) \oplus \text{xtime}(a) \oplus a$$

$$a \cdot 13 = \text{xtime}(\text{xtime}(\text{xtime}(a))) \oplus \text{xtime}(\text{xtime}(a)) \oplus a$$

$$a \cdot 14 = \text{xtime}(\text{xtime}(\text{xtime}(a))) \oplus \text{xtime}(\text{xtime}(a)) \oplus \text{xtime}(a)$$

※

### (c) Use of lookup tables (LUTs) in hardware Implementation

在AES的Inverse Mix Columns中，實現  $GF(2^8)$ 乘法所需的邏輯 complexity 較高，因此常用 LUTs 以降低 complexity 進行優化。 $\rightarrow$  LUTs 通常以 ROM、cache 形式儲存

pros and cons: \* pros: (1) constant time: 查找時的執行時間固定，不因 input 而所更動  
 (2) 高效率運算: 可以省去傳統要移位作 XOR，利用查表可以減少邏輯開銷，以提升效能。

(3) 易於驗證: 結果固定，可方便進行邏輯模擬。

\* cons: (1) Memory 使用量大: 因為每個常數皆需要 256 庫結果  $\Rightarrow$  共要  $4 \times 256\text{ bits}$   
 (2) 可能存在 cache timing attack  $\Rightarrow$  Attacker 利用觀察 cache hit / miss time side-channel  $\rightarrow$  推測出 密鑰

### (d) Logical Depth and Area for shared sub-multiplication Network

我們將 b) 的結果拿出來用

$$0 \times 09 \Rightarrow xtime^3(a) \oplus a$$

xtime 次數      XOR = 次數

3                    1

$$0 \times 11 \Rightarrow xtime^3(a) \oplus xtime(a) \oplus a$$

3                    2

$$0 \times 13 \Rightarrow xtime^3(a) \oplus xtime^2(a) \oplus a$$

3                    2

$$0 \times 14 \Rightarrow xtime^3(a) \oplus xtime^2(a) \oplus xtime(a)$$

3                    2

最大邏輯深度  $\Rightarrow 3(xtime) + 2(XOR) = 5$

\* xtime module: 3 layers

若獨立設計  $\Rightarrow$  4組 xtime module + 4組 XOR (2 layers)

共用設計  $\Rightarrow$  1組 xtime module + 1組 XOR (2 layers)

由上並可知若採共用設計，能明顯在面積方面大幅節省空間達到硬體優化。

## Problem 13

Ans:

### (a) Attack Demonstration:

設 plaintext Block Chaining 的加密方式為

$$C_0 = E_K(IV \oplus m_0)$$

$$C_1 = E_K(C_0 \oplus m_1)$$

$$C_2 = E_K(C_1 \oplus m_2)$$

而根據題意，已知  $m_1 = m_2 = X$ ，因此 Attacker 可以進行以下推導

首先，對  $C_2$  解密  $\rightarrow D_K(C_2) = C_1 \oplus X$

$$\text{故 } C_1 = D_K(C_2) \oplus X$$

接著，對  $C_1$  解密  $\rightarrow D_K(C_1) = C_0 \oplus X$

$$\text{故 } C_0 = D_K(C_1) \oplus X$$

最後，對  $C_0$  解密  $\rightarrow D_K(C_0) = IV \oplus m_0$

$$\text{故 } m_0 = D_K(C_0) \oplus IV$$

$$\left. \begin{array}{l} C_1 = D_K(C_2) \oplus X \\ C_0 = D_K(C_1) \oplus X \\ m_0 = D_K(C_0) \oplus X \end{array} \right\} *$$

\*由上述過程可知，此攻擊手法，可以單純依賴公開資訊、已知的  $X$ ，就可以順利回推出原始明文  $m_0$ ，甚至完全不用金鑰  $K$

### (b) Distinguishing Advantage

在理想狀況下，對隨機  $n$ -bit 的密文，其差值為某固定值的機率為  $(\frac{1}{2^n})$

不過，因為明文若是滿足了  $m_1 = m_2 = X$  時，則加密結構就會導致

$$D_K(C_2) \oplus D_K(C_1) = (C_1 \oplus X) \oplus (C_0 \oplus X) = C_1 \oplus C_0$$

∴ 若 Attacker 發現  $D_K(C_2) \oplus D_K(C_1) = C_1 \oplus C_0$  時，便可以懷疑其就是重複明文加密  $\Rightarrow$  成功區分出加密方式。

此時， $E(n)$  可以定義為： $E(n) = |P(\text{成功判定其加密方式}) - P(\text{隨機情形猜測成功})|$

$$E(n) = \left| 1 - \frac{1}{2^n} \right| \Rightarrow 1 - 2^{-n}$$

其是否為重複明文  
輸入所加密的密文

∴ 當  $n$  大時， $E(n)$  即趨近於 1  $\Rightarrow$  Attacker 可以在近乎 99.9...% 確定

# Problem 14

[Ans]:

## (a) Existential Forgery Attack

CBC-MAC: 給定 key K, 以及輸入訊息  $M = (m_1, m_2, m_3, \dots, m_t)$

而 CBC-MAC 的計算為:  $V_0 = IV$ ,  $V_i = E_K(m_i \oplus V_{i-1})$  for  $i = 1 \rightarrow t$   
 $MAC(M) = V_t$

攻擊步驟:

若能查詢 MAC oracle 並獲得 2 個合法訊息的 tag

①  $\Rightarrow M_1 = (m_1, m_2) \Rightarrow T_1 = MAC(M_1) = E_K(E_K(m_1) \oplus m_2)$

$M_2 = (m_3) \Rightarrow T_2 = MAC(M_2) = E_K(m_3)$

② 定中間值  $X = E_K(m_1)$

構建新訊息為  $M_3 = (m_1, m_2 \oplus m_3)$

③ 計算 MAC:

$$V_1 = E_K(m_1) = X$$

$$\begin{aligned} V_2 &= E_K((m_2 \oplus m_3) \oplus X) = E_K(m_2 \oplus X \oplus m_3) \\ &= E_K((m_2 \oplus X) \oplus m_3) \end{aligned}$$

而若是原本  $T_1 = E_K(m_2 \oplus X)$ , 而  $T_2 = E_K(m_3)$  則構造會使 MAC 等於  $T_2$

只要是 Attacker 提交訊息  $M_3$ , 與已知合法 MAC  $T_2$ , 並且使驗証通過  
 則完成 Existential Forgery

## (b) Variable-length generalization

為了證明在 variable-length messages 下 CBC-MAC 是不安全的，我們構建了一個 chosen-message existential forgery Attack，以證明允許長度變長下的情況下不具安全性。

設 CBC-MAC 使用密鑰  $K$ ，而初始向量  $IV = 0$

此為 Attack 的 background

以及一些前提及假設

給定任意訊息  $M = (m_1, m_2, m_3 \dots m_t)$ ，

其 MAC 為  $T = E_K(\dots E_K(E_K(m_1) \oplus m_2) \dots \oplus m_t)$

且 Attack 可以查詢 oracle 並且獲得 tag

攻擊步驟：

① 查詢首筆合法訊息：

→ Attacker 查詢  $M_1 = (m_1, m_2, m_3, \dots m_t)$  得  $T_1 = CBC\text{-MAC}(M_1)$

② 查詢第二筆單區塊訊息

→ Attacker 查詢  $M_2 = (m')$  得  $T_2 = CBC\text{-MAC}(M_2) = E_K(m')$

③ 製作出偽造的訊息  $M^*$

$$使 M^* = M_1 \parallel (m' \oplus T_1)$$

$$\rightarrow VT = T_1$$

$$VT = E_K((m' \oplus T_1) \oplus T_1) = E_K(m') = T_2$$

$$\Rightarrow MAC(M^*) = T_2 \rightarrow \text{合法 tag}$$

\*: Attacker 利用  $T_1$  去控制下一個 block 輸入，並偽造出 MAC 為  $T_2$  合法訊息  $M^*$

由此可證，CBC-MAC 在 variable-length 訊息下不具備 Existential Unforgeability under chosen-message Attack

→ 因為 chaining structure 會允許以 MAC 值作為中間輸入，如圖

## Problem 15

[Ans]: a) Approach to Broadcast Attack

在 RSA 加密中，若 3 位收件者使用相同的公鑰指數  $e$ ，且分別擁有不同且互質的 Modulo 分別是  $n_A, n_B, n_C$  且同時接收到加密相同明文  $m$  的密文

$$Y_A = m^e \bmod n_A$$

$$Y_B = m^e \bmod n_B$$

$$Y_C = m^e \bmod n_C$$

Attacker 看到這 3 輯密文和 public Modulo 之後可以推斷出以下：

(1) 明文  $m$  經過 3 次方再取  $\bmod$  各自不同的  $n_i$  所得的  $m^3 \bmod n_i$

或者是  $m^3 < N = n_A \cdot n_B \cdot n_C$ ，則這 3 輙  $\bmod n_i$  的餘數唯一對應到真央值  $m^3$ 。能利用 CRT 進行還原。

(2) 只要可以取得  $m^3$ ，Eve 就可以再利用立方根解出明文  $m$  且完全不用先取得私鑰！

此為 Hastad's Broadcast Attack，這充分

說明若是 RSA 未使用 padding 會過於脆弱！

b) CRT Application and Proof

令： $Y_A, Y_B, Y_C$  為 3 個密文，而  $N = n_A \cdot n_B \cdot n_C$ ， $n_A = \frac{N}{n_A} : n_B = \frac{N}{n_B} ; n_C = \frac{N}{n_C}$

$$\text{X } u_A = N^{-1} \bmod n_A, u_B = N^{-1} \bmod n_B, u_C = N^{-1} \bmod n_C$$

by CRT：

$$M = (Y_A \cdot n_A \cdot u_A + Y_B \cdot n_B \cdot u_B + Y_C \cdot n_C \cdot u_C) \bmod N$$

此  $M \equiv m^3 \bmod N$ ，在  $m^3 < N$  的條件下， $M$  值僅會有  $M = m^3$  這個組合

∴ Attacker Eve 就能夠利用這個  $m^3$  值去開 3 次方根求  $m$ 。

c) Plaintext Extraction Procedure

由上面所述得出  $M = m^3 \Rightarrow m = \sqrt[3]{M} \Rightarrow$  可以使用 Binary Search 尋找

滿足  $m^3 = M$  的最小整數  $m$ 。又  $\because m^3 < N$ ，所以  $m$  值唯一

→ 直接還原出明文  $m$

## Problem 1b

Ans:

(a) PFS protocol spec:

① 系統假設 公開的參數：

\* 假定已選定某安全質數  $p$  以及生成元  $g \rightarrow$  所有人都知！

\* Alice 與 Bob 各自有擁長期金鑰對 Alice 為  $(a, A = g^a \bmod p)$

Bob 為  $(b, B = g^b \bmod p)$

(b) communication 的協定流程為 ① 生成 temp 金鑰  $\rightarrow$  ② 交換 temp 金鑰  $\rightarrow$  ③ 計算 share 金鑰  
 $\rightarrow$  ④ 指派通訊金鑰並加密  $\rightarrow$  ⑤ 銷毀 temp 金鑰

① 首先，Alice 隨機選 temp 金鑰  $a' \rightarrow$  計算  $A' = g^{a'} \bmod p$

Bob 隨機選 temp 金鑰  $b' \rightarrow$  計算  $B' = g^{b'} \bmod p$

② Alice 傳  $A'$  給 Bob，而 Bob 把  $b'$  傳送給 Alice

③ Alice 開始計算共享金鑰  $K = (B')^{a'} \bmod p$

Bob 也計算共享金鑰  $K = (A')^{b'} \bmod p \rightarrow$  by DH  $K = g^{a'b'} \bmod p$   
且  $K$  值雙方應相同！

④ 把  $K$  輸入至 Key Derivation Function  $\rightarrow$  生成出對稱金鑰 SK  
用來加密訊息

⑤ 在通訊結束時，會立即刪除  $a', b'$  2 個 temp 金鑰

根據上述 ① ~ ⑤ 步驟  $\rightarrow$  將 PFS specification 定義完成

## b) Forward Secrecy Proof:

首先，我們已知 Eve 擁有 Alice 與 Bob 的 temp public key ( $A'$ ,  $B'$ )  
並且也獲得 Alice, Bob 的長期私鑰  $a'$ ,  $b'$

接着，Eve 想要復原共享金鑰  $K = g^{a' b'}$  能嗎？當然不能！

已知  $A' = g^{a'}$ ，但無法算出  $a'$  基於無法克服離散對數 problem  
相同的，Eve 也無法從  $B'$  去回推  $b'$

→ 基於上述事實，縱使 Eve 擁有双方長期私鑰  $a'$ ,  $b'$  也是無法  
还原出  $K$  → 主因為無法求得 temp private key  $a'$ ,  $b'$   
∴ 此 protocol 有達成 Perfect Forward Secrecy！

## (c) Real-world Implementation Evaluation

① TLS 協定：在 real-world 中，Perfect Forward Secrecy 已廣泛實作於 TLS protocol 中，  
eg. TLS 1.2, TLS 1.3, ..., TLS 1.2: 需要 server 有支援 DHE-SHA or ECDHE-RSA 才具備提供 PFS

⇒ TLS 1.3: 強調 Ephemeral Diffie-Hellman 金鑰交換  
且所有連線皆具 Perfect Forward Secrecy

② Signal protocol 採用 Double Ratchet Algorithm 不但結合 temp key 还有對稱 ratchet  
可以實現 post-compromise security ⇒ 每次訊息皆會用不同的会話金鑰，使歷史通訊就算  
private key 外漏也無法还原。

③ OpenSSL 有支援 ECDHE-RSA 此種套件，部署於 Apache, Nginx 以確保 HTTPS 也能具備  
前向保密性。

## Extra Credit 1:

Ans:

(a) Perfect Secrecy proof

pf that  $C = m \cdot k \bmod p$  具有 perfect secrecy

令明文  $m \in \mathbb{Z}_p^*$ , 密鑰  $k \in \mathbb{Z}_p^*$ , 密文  $C = m \cdot k \bmod p$ , 其中為 uniform 隨機數  
並且在  $\mathbb{Z}_p^*$  上均勻分布, 共有  $\varphi(p) = p-1$  個可能值

對任意固定  $m$ , 計算:  $\Pr[C=c | M=m] = \Pr[m \cdot k \equiv c \bmod p]$

由於  $m \in \mathbb{Z}_p^*$  可逆, 存在唯一一個  $k = c \cdot m^{-1} \bmod p$ ,

因此,  $\Pr[C=c | M=m] = \frac{1}{p-1}, \forall m, c \in \mathbb{Z}_p^*$

所以, 對任意 2 個明文  $m_1, m_2$ , 皆有:

$$\Pr[C=c | M=m_1] = \Pr[C=c | M=m_2] \quad \text{根据 Shannon} \Rightarrow \text{此加密方案已達 perfect secrecy}$$

by combining problem 1's theory

→ 本加密方案需要在  $\bmod p$  下執行乘法和反元素的運算  
等同 problem 1 中在有限域  $\mathbb{F}_p$  上  $\bmod$  irreducible 的  $p(x)$  的乘、加法一樣  
此類結構保證了運算封閉性、可逆性  $\Rightarrow$  使其達成 perfect secrecy

(b) 小題在下一頁!

## b) LCG cryptanalysis

要攻破 LCG  $x_i = ax_{i-1} + b \bmod p$  並且回推 key

設觀察到連續 3 個 keystream 輸出  $x_0, x_1, x_2$  有：

$$x_1 = ax_0 + b \bmod p$$

$$x_2 = ax_1 + b \bmod p$$

$$\hookrightarrow \text{兩式相減} \Rightarrow x_2 - x_1 = a(x_1 - x_0)$$

$$\text{而 } a = (x_2 - x_1)(x_1 - x_0)^{-1} \bmod p \quad \text{條件: } x_1 \neq x_0, \text{ 故 } x_1 - x_0 \in \mathbb{Z}_p^*$$

有 inverse element

接著，再代入 & 解出以下：

$$b = x_1 - ax_0 \bmod p$$

\* 只要得出  $a, b \Rightarrow$  即可重建整個 keystream

$$\boxed{\begin{array}{l} x_3 = ax_2 + b \bmod p \\ x_4 = ax_3 + b \bmod p \end{array}}$$

故此 Attack 需要 3 個連續輸出，即可还原 LCG 結構

而不用窮舉 search。因此，此方法並不適合作為安全 keystream 來源

by combining problem 2:

primitive polynomial 可以生成最大週期序列，但非 primitive  $\Rightarrow$  短

因此，使用 LCG 若無良好的參數設計，eg:  $a$  本身並非 primitive element

其產生的 sequence 將具有可預測性、短週期性。

另外，LCG 本身非線性不可約  $\Rightarrow$  結構更易遭到線性逆推

縱使你取  $a, b$  不要，但只要少量輸出被洩漏  $\rightarrow$  會被反推 key、內部狀態。

## Extra credit 2

Ans: (a) First Non-linear Design

設計一個 Non-linear Boolean Function 以處理 LFSR 狀態中的多位元並產生 keystream 位元

而採用的 Function 如:  $Z_t = X_t \oplus (X_{t-1} \cdot X_{t-3}) \oplus (X_{t-2} \cdot X_{t-4})$

此三階 Boolean Function 結合 AND、XOR \* 非線性

安全性分析: (1) 因為 Non-linear Boolean Filter Function 可以提升非線性意即破壞 LFSR 的線性結構, 使 BM 難以正確推導線性流逝關係

(2) 若使用非線性度高的 Boolean Function, 能使輸出序列之線性複雜度可以逼近 LFSR 長度  $\times$  函數階數  $\rightarrow$  遠超出 BM Algo 可破解之範疇

gate count 分析: 需要 2 個 AND gates + 3 個 XOR gates 共 5 個邏輯閘!

(b) second design & comparison

實作 clock-controlled LFSR

設計 3 個 LFSR: 分別是 LFSR-A(控制器)、LFSR-B(主輸出)、LFSR-C(子機器)

設計 mechanism: ① 每輪由 LFSR-A 的輸出  $C_t$  決定是否讓 LFSR-B 更新 & 輸出 keystream

② 若 A 的輸出為  
[ 0  $\rightarrow$  skip ]  
[ 1  $\rightarrow$  B 正常輸出 ]

③ LFSR-C 動態控制 A or B 的 Feedback taps 例如:

若  $C_t = 0$ :  $B_{t+1} = B_{t-1} \oplus B_{t-4}$

$C_t = 1$ :  $B_{t+1} = B_{t-2} \oplus B_{t-3}$

安全性分析: ① keystream 時序變動, 使輸出 bit 和內部狀態關係不連續 BM 無法觀察序列。

② 時脈將引入隨機性, 使結構為非線性, 難以線性追推。

○ 承 extra credit 的由

- △ gate count 分析：
- ① 每个 LFSR 需要  $P_n$  gates,  $n$  為 LFSR 長度。
  - ② 控制邏輯需要額外的 MUX, clock gating  
估計 gate 數量可能約  $50 \sim 100$  個

\* Comparison 其中, stop-and-go filter  $\equiv$  clock-controlled LFSR

Item	NLF Boolean Filter	Stop-and-go Filter
抗 BM 能力	中等偏上	較 NLF 強
線性複雜度	取決於函數階及選取參數	極高, 且較不易建模
需要 gate 數量	較少	較多
資源效率	較高	中等偏上

\* 若是資源受限之情況，建議可使用 NLF Boolean Filter 或是可以將 NLF Boolean - Stop-and-go 合併應用。

## Extra Credit 3

[Ans]:

(a) Irreducibility proofs

(1)  $f_1(x) = x^4 + x + 1$  部分，在  $F_2$  中，先檢查一次因子，再看二次因子

一次因子： $x=0 \Rightarrow f_1(0) = 1 \neq 0$ ;  $x=1 \Rightarrow f_1(1) = 1 \neq 0$

二次因子： $x^2+x+1$ ,  $x^2+1$ ,  $x^2+x$

將 3 個二次因子嘗試相除  $\Rightarrow$  皆不可以整除  $\Rightarrow f_1(x)$  在  $F_2$  上不可約

(2)  $f_2(x) = x^4 + x^3 + 1$

一次因子： $x=0 \Rightarrow f_2(0) = 1 \neq 0$ ;  $x=1 \Rightarrow 1+1+1 = 1 \neq 0$

二次因子： $x^2+x+1$ ,  $x^2+1$ ,  $x^2+x$

將 3 個二次因子嘗試相除  $\Rightarrow$  皆不可以整除  $\Rightarrow f_2(x)$  在  $F_2$  上不可約

(b) order computations in  $F_{2^4}$

設  $\alpha$  為  $F_{2^4}$  中由不可約多項式生成的元素，其乘法群  $F_{2^4}^*$  為循環群階： $2^4 - 1 = 15$

對  $f_1(x)$ ：找出  $\alpha$  的最小階，使  $\alpha^n = 1$ ，則  $\text{ord}(\alpha) = 15$ ，若達到  $\Rightarrow \alpha$  是生成元

if  $\alpha^{15} \equiv 1 \pmod{f_1(x)}$ ，但  $\alpha^d \not\equiv 1 \pmod{f_1(x)}$  對於任  $d < 15$ ，則  $f_1(x)$  是 primitive

以  $\pmod{f_1(x)}$  逐次計算  $\alpha^1, \alpha^2, \alpha^3, \dots, \alpha^{15} \pmod{f_1(x)}$ ，確認是否為 15

若 minimum 滿足條件的  $n$  是 15  $\Rightarrow$  primitive

→ 此 step 會使用 python code

的輸出進行回應！

下一页接着  $f_2(x)$

對  $f_2(x)$ , 一樣 check  $x^n \equiv 1 \pmod{f_2(x)}$  的最小  $n$  是否為 15

若取小  $n < 15 \Rightarrow$  則並非 primitive

以下結果截圖可知不論 mod 的是  $f_1(x)$  或  $f_2(x) \Rightarrow n$  值均是 15 (最特殊)

Powers of  $x$  modulo  $f_1(x)$ :

```
x^1 ≡ 2  
x^2 ≡ 4  
x^3 ≡ 8  
x^4 ≡ 3  
x^5 ≡ 6  
x^6 ≡ 12  
x^7 ≡ 11  
x^8 ≡ 5  
x^9 ≡ 10  
x^10 ≡ 7  
x^11 ≡ 14  
x^12 ≡ 15  
x^13 ≡ 13  
x^14 ≡ 9  
x^15 ≡ 1
```

Powers of  $x$  modulo  $f_2(x)$ :

```
x^1 ≡ 2  
x^2 ≡ 4  
x^3 ≡ 8  
x^4 ≡ 9  
x^5 ≡ 11  
x^6 ≡ 15  
x^7 ≡ 7  
x^8 ≡ 14  
x^9 ≡ 5  
x^10 ≡ 10  
x^11 ≡ 13  
x^12 ≡ 3  
x^13 ≡ 6  
x^14 ≡ 12  
x^15 ≡ 1
```

c) primitive analysis, 我們使用以下 python code 進行判別) [放在 bonus/extracredits]

```
1 import galois  
2  
3 # 1. 建立 GF(2) 與兩個四次多項式  
4 F2 = galois.GF(2)  
5 f1 = galois.Poly([1, 0, 0, 1, 1], field=F2)      #  $x^4 + x + 1$   
6 f2 = galois.Poly([1, 1, 0, 0, 1], field=F2)      #  $x^4 + x^3 + 1$   
7  
8 print("Is f1(x) irreducible? ", f1.is_irreducible())  
9 print("Is f2(x) irreducible? ", f2.is_irreducible())  
10  
11 # 2. 以 f1 · f2 分別建立 GF(2^4)  
12 GF1 = galois.GF(2**4, irreducible_poly=f1)  
13 GF2 = galois.GF(2**4, irreducible_poly=f2)  
14  
15 # x 在各擴域中的表示 (0b0010 = 2)  
16 x1 = GF1(2)  
17 x2 = GF2(2)  
18  
19 print("Order of x modulo f1(x) =", x1.multiplicative_order())  
20 print("Is f1(x) primitive? ", x1.multiplicative_order() == 15)  
21 print("Order of x modulo f2(x) =", x2.multiplicative_order())  
22 print("Is f2(x) primitive? ", x2.multiplicative_order() == 15)  
23  
24 # 3. 列印  $x^k$  ( $k = 1 \dots 15$ ) 在兩個體中的值  
25 print("\nPowers of x modulo f1(x):")  
26 for k in range(1, 16):  
27     print(f"x^{k} ≡ {x1 ** k}")  
28  
29 print("\nPowers of x modulo f2(x):")  
30 for k in range(1, 16):  
31     print(f"x^{k} ≡ {x2 ** k}")
```

由先前(b)得之結果，並與 code  
進行驗証  $\Rightarrow f_1(x), f_2(x)$

均為 irreducible 且 primitive  
的 polynomial

```
PS C:\Users\user\Desktop> python ex  
Is f1(x) irreducible? True  
Is f2(x) irreducible? True  
Order of x modulo f1(x) = 15  
Is f1(x) primitive? True  
Order of x modulo f2(x) = 15  
Is f2(x) primitive? True
```

#### (d) Real-World Application

primitive polynomial 可使  $n$ -stage LFSR 週期達  $2^n - 1$

以實例來看，GSM A5/1 其採用的是 19-stage LFSR  
其 polynomial 為： $x^{19} + x^5 + x^2 + x + 1$  且為 primitive

primitive 保證了週期為  $2^{19} - 1$  約  $5.2 \times 10^5 - 1 \Rightarrow$  約  $5.2 \times 10^5$

而 LFSR 在任何非 0 初始狀態下遍歷全部可能狀態

以生成出最長 m-sequence，以確保 keystream 無週期重複

並且具有統計性，若採用非 primitive polynomial  $\Rightarrow$  序列重複 +  
以致於安全性較差，不足！