# Problem 1

- **Certificate Generation：**

| 步驟 | 目錄 | 檔案名稱 | 檔案類型 | 說明 |
|---|---|---|---|---|
| 1 | root | root.key.pem | Private Key | Root CA 的私鑰 |
| 2 | root | root.cert.pem | Certificate | Root CA 自簽憑證 |
| 4 | intermediate | intermediate.key.pem | Private Key | Intermediate CA 的私鑰 |
| 5 | intermediate | intermediate.csr.pem | CSR | Intermediate CA 的 CSR，供 Root CA 簽發 |
| 6 | intermediate | intermediate.cert.pem | Certificate | 由 Root CA 簽發的 Intermediate 憑證 |
| 8 | server | server.key.pem | Private Key | Server 的私鑰 |
| 9 | server | server.csr.pem | CSR | Server 的 CSR，供 Intermediate CA 簽發 |
| 10 | server | server.cert.pem | Certificate | 由 Intermediate 簽發的 Server 憑證 |
| 12 | client | client.key.pem | Private Key | Client 的私鑰 |
| 13 | client | client.csr.pem | CSR | Client 的 CSR，供 Intermediate CA 簽發 |
| 14 | client | client.cert.pem | Certificate | 由 Intermediate 簽發的 Client 憑證 |

- **Description of Commands：private key -> csr -> cert**

```
./cert-go create private-key --out ./root/root.key.pem
./cert-go create cert --type root --yaml ./cfg.yml

./cert-go create private-key --out ./intermediate/intermediate.key.pem
./cert-go create csr --type intermediate --yaml ./cfg.yml
./cert-go create cert --type intermediate --yaml ./cfg.yml

./cert-go create private-key --out ./server/server.key.pem
./cert-go create csr --type server --yaml ./cfg.yml
./cert-go create cert --type server --yaml ./cfg.yml

./cert-go create private-key --out ./client/client.key.pem
./cert-go create csr --type client --yaml ./cfg.yml
./cert-go create cert --type client --yaml ./cfg.yml
```

- **Certificate Chain Description：**

  Root CA 自簽並簽發 Intermediate CA；Intermediate 再分別簽 Server 與 Client 憑證。完整鏈條中，Client 與 Server 均由 Intermediate 所簽發，而 Intermediate 則由 Root 所信任。

  [Client / Server Certificate]

  ↓

  [Intermediate CA Certificate]

  ↓

  [Root CA Certificate]

- **Repository Improvement：**

  新增 --force 支援

  - 增加 CLI 參數 --force 於 cert, csr, private-key 指令
  - 若檔案已存在，只有在 --force 啟用下才會刪除並重新建立；否則回傳錯誤訊息 certificate already exists
  - PR: https://github.com/Alonza0314/cert-go/pull/5

## Problem 2

- **Simulation Implementation:**

```
Naive Shuffle:
[1, 2, 3, 4]: 39184
[1, 2, 4, 3]: 39263
[1, 3, 2, 4]: 39315
[1, 3, 4, 2]: 54716
[1, 4, 2, 3]: 43058
[1, 4, 3, 2]: 34975
[2, 1, 3, 4]: 39202
[2, 1, 4, 3]: 58484
[2, 3, 1, 4]: 54996
[2, 3, 4, 1]: 54490
[2, 4, 1, 3]: 42936
[2, 4, 3, 1]: 42805
[3, 1, 2, 4]: 43187
[3, 1, 4, 2]: 42872
[3, 2, 1, 4]: 35074
[3, 2, 4, 1]: 42815
[3, 4, 1, 2]: 42709
[3, 4, 2, 1]: 39068
[4, 1, 2, 3]: 31249
[4, 1, 3, 2]: 35306
[4, 2, 1, 3]: 34989
[4, 2, 3, 1]: 31348
[4, 3, 1, 2]: 38875
[4, 3, 2, 1]: 39084
```

```
Fisher-Yates Shuffle:
[1, 2, 3, 4]: 42154
[1, 2, 4, 3]: 42213
[1, 3, 2, 4]: 41467
[1, 3, 4, 2]: 41811
[1, 4, 2, 3]: 41545
[1, 4, 3, 2]: 41438
[2, 1, 3, 4]: 41480
[2, 1, 4, 3]: 41878
[2, 3, 1, 4]: 41630
[2, 3, 4, 1]: 41640
[2, 4, 1, 3]: 41633
[2, 4, 3, 1]: 41565
[3, 1, 2, 4]: 41377
[3, 1, 4, 2]: 41613
[3, 2, 1, 4]: 41499
[3, 2, 4, 1]: 41365
[3, 4, 1, 2]: 41332
[3, 4, 2, 1]: 41776
[4, 1, 2, 3]: 41758
[4, 1, 3, 2]: 41740
[4, 2, 1, 3]: 41771
[4, 2, 3, 1]: 41439
[4, 3, 1, 2]: 42216
[4, 3, 2, 1]: 41660
```

- **Algorithm Comparison:**
  - Naive Shuffle：
    - 每一張牌與整個牌組中任意一張隨機交換。
    - 輸出結果：有些排列組合明顯出現次數偏高，有些偏低，不同排列的機率並不均等，說明存在 bias。
  - Fisher–Yates shuffle：目前標準化的洗牌演算法
    - 從後往前，每一張牌只與尚未處理過的牌進行交換。
    - 輸出結果：每一種排列出現次數幾乎相等，表示機率接近均勻分布。
  - Which one is better? Fisher–Yates Shuffle
    - 保證每一種排列都只會被生成一次，且機率相同，每個排列機率皆為 1/n! → uniform random permutation。
    - 它能保證生成結果的機率均勻，避免 bias。
    - 適合應用在加密、亂數生成、卡牌遊戲等對隨機性要求高的場景。
    - 相比之下，Naive Shuffle 容易產生偏態分布，存在安全性與統計偏差問題。

- **Drawback Analysis:** Naive Shuffle 的缺點
  - 非均勻（Non-uniform permutation distribution）：每一張牌與整副牌中的任一張交換，這會導致有些排列出現機率過高，有些過低。
  - 產生重複交換序列：某些牌可能在洗牌過程中被重複交換兩次，甚至回到原點，這些冗餘交換導致整體 permutation space 並不完整。
  - Bias 導致無法收斂，即使重複非常多次（例如模擬 1,000,000 次），Naive Shuffle 的分布仍然偏斜，代表本質上設計就有偏差，不是單靠次數可以改善的。
  - 不適合密碼學使用：在安全性敏感的場景（如金鑰初始化、隨機 nonce、密碼隨機序列），非均勻分布會造成嚴重安全風險，因為攻擊者可以透過偏斜的排列統計做側信道或預測性攻擊。

- **RC4 Improvement:**
  - 主要缺陷

    a. KSA 初期 bias 明顯：前 256 個 S[] 值偏向低位元的 key byte（如 S[0] 容易等於 key[0]），易被攻擊者透過統計分析還原部分 key 結構
    b. PRGA 初期密鑰流偏態：第一個輸出的幾個位元非均勻，容易被已知明文攻擊（known-plaintext attack）
    c. 未使用的 S[] 索引會累積偏移誤差：在 PRGA 中，i 與 j 的改變是線性疊加，過程缺乏重新混合（re-mixing）機制

  - 改進建議

    a. 加強 KSA 的亂度（Enhanced Key Scheduling）：在 KSA 結束後，額外進行多輪 Fisher–Yates 洗牌，這會強化初期 S[] 的亂度，破除 key-byte 對初始輸出的控制影響。
    b. 丟棄 PRGA 的前幾個 output（Output Drop）：這能有效避開前幾位元偏態區段，提高密鑰流不可預測性。
    c. 重設 j 值的更新方式（Re-mix Index）：加入更多 index 混合，增加 S[] 位置訪問的不可預測性。

# Problem 3

- **Miller-Rabin on RSA Moduli:**

  Miller-Rabin test 是用來檢測一個數是否為質數的機率型演算法。

  但 RSA 的模數 $n = p \times q$ 是兩個質數相乘，因此 RSA 的模數 n 永遠不是質數，它一定是 Composite number

  所以當我們對 RSA 的模數 n 執行 Miller-Rabin test 會檢測出 none of the above conditions are met, n is composite.

- **RSA Security Analysis:**

  Can we break RSA with it?  NO

  - Miller-Rabin test 無法分解合數，只能判斷是否質數，因為它的設計目的是「過濾非質數」，但對於像 RSA 模數這種 $n = p \times q$ 的合數，它不會提供任何關於 p 和 q 的資訊
  - RSA 的安全性不建立在「質數檢測」上，而是「質因數分解困難性」上，想要破解 RSA，攻擊者需要分解 n 得到 p 和 q ，因此即使知道 n 是合數也沒用。
  - Miller-Rabin 無法區分是否為兩個大質數相乘的特定結構：在設計公鑰時我們會用它來檢查候選的 p 和 q 是否是質數，但它不會反推 n 的因數