|  |
| --- |
|  |
| **Zero-Knowledge Proof：以 Schnorr Protocol 為核心的可驗證身份驗證系統** |
| 日期：114年6月5日  課程名稱：密碼學  系所：電機工程研究所  組別：資訊安全組  組員：r13921a13 林伯叡  d13921c02黃杬霆 |

****

**目錄**

[前言 1](#_Toc198246556)

[一、 背景介紹與動機說明 2](#_Toc198246557)

[(一) 背景介紹 2](#_Toc198246558)

[(二) 動機說明 3](#_Toc198246559)

[二、 專題目標 4](#_Toc198246560)

[(一) 傳統ZKP（互動式驗證機制） 4](#_Toc198246561)

[(二) Schnorr Protocol（簡化版互動式驗證機制）[2] 4](#_Toc198246562)

[(三) Fiat–Shamir Heuristic（非互動式驗證機制）[3] 4](#_Toc198246563)

[三、 系統設計 6](#_Toc198246564)

[(一) 傳統ZKP — 以Graph 3-colorability（簡稱G3C）為例子 6](#_Toc198246565)

[(二) Schnorr Protocol 7](#_Toc198246566)

[(三) Fiat–Shamir Heuristic（待研擬） 8](#_Toc198246567)

[四、 專題成果 9](#_Toc198246568)

[(一) 傳統ZKP — 以Graph 3-colorability（簡稱G3C）為例子 9](#_Toc198246569)

[(二) Schnorr Protocol 11](#_Toc198246570)

[(三) Fiat–Shamir Heuristic（待研擬） 12](#_Toc198246571)

[五、 結論（待研擬） 13](#_Toc198246572)

[六、 未來展望（待研擬） 14](#_Toc198246573)

[七、 參考文獻 15](#_Toc198246574)

[八、 附錄 16](#_Toc198246575)

[(一) 傳統ZKP — 以Graph 3-colorability（簡稱G3C）為例子 16](#_Toc198246576)

[(二) Schnorr Protocol 24](#_Toc198246577)

[(三) Fiat–Shamir Heuristic（待研擬） 29](#_Toc198246578)

**前言**

在資訊安全日益受到重視的當下，強調的安全性標準為「**零日攻擊**（zero - day）」，如何在遭遇未知攻擊，透過不洩漏關鍵個人資訊的前提下，完成身分驗證與機密確認，已成為密碼學中的重要課題。

傳統的驗證機制仰賴密碼、個人資訊、生物特徵或私鑰的直接揭露，然而這類方法存在極高的風險因子，可能導致機密外洩、遭遇重放攻擊，或被第三方竊取…等。為了解決此問題，「**零知識證明**（**Zero-Knowledge Proof**, **ZKP**）」技術於1985年應運而生，提供了一種創新方式，允許「證明者」（Prover）在不洩漏秘密的情況下，說服「驗證者」（Verifier）其確實擁有該秘密。

本專題以 ZKP 為核心出發，從傳統互動式的驗證模型著手研究，實作了以「**Graph 3-colorability**（**G3C**）」為基礎的零知識協定，觀察其完整性、可靠性與零知識性。接著，我們進一步導入 **Schnorr** **Protocol**，以**離散對數問題**為數學基礎，設計一套簡化版的互動式 ZKP，在保有原本ZKP安全性之下，同時大幅降低訊息傳輸成本。最終因應資訊時代的資安需求，我們導入 **Fiat–Shamir Heuristic**，實現「**非互動式**」的零知識證明模型，進一步提升其在真實應用場景中的彈性與可行性，亦為區塊鏈、加密貨幣等無法即時互動的場景，奠定了可行且安全的實作基礎。

1. **背景介紹與動機說明**
2. **背景介紹**

在早期的驗證系統中欲達成登入驗證，通常只能透過使用端直接揭露部分或全部資訊來讓目標端驗證，例如：輸入密碼、提交文件、或洩露原始資料…等方式，雖能達到驗證目的，但容易造成機密洩漏、重放攻擊，或讓未經授權的第三方竊取重要資訊。在高度依賴網路傳輸與重視資訊安全的現代，這類型作法已難以滿足對安全性及隱私性的雙重需求。為了解決這些問題，密碼學家 Shafi Goldwasser、Silvio Micali 與 Charles Rackoff 在 1985 年提出了**零知識證明**（**Zero-Knowledge Proof**， **ZKP**）的概念 [1]。他們設計了一種互動式的證明方法，讓「證明者」能在不揭露秘密本身的情況下，說服「驗證者」其確實知道某個秘密。這項技術徹底顛覆了傳統「揭露才能驗證」的模式，為密碼學、身分驗證與區塊鏈等未來領域帶來了全新的解決方案，以下為相關數學理論基礎及基本性質：

* + - * 數學理論基礎：
    1. 選用的一個數學難題：以Schnorr Protocol為例，其採數學上難解的問題—**離散對數問題**（**Discrete Logarithm Problem**，**DLP**）作為整個方法的核心基礎，駭客若想入侵則必須破解該議題。
    2. 數學背景：在一個有限域中 ，給定一個生成元（generator） 及任意元素 ，欲找到一個，使得：，則稱為相對於 的離散對數**（）**。
       - 基本性質：

1. **可靠性**（**Soundness**）：若陳述為假，不存在一個可證明策略，使證明者（Prover）說服驗證者（Verifier）相信。

例如：假設有人提出證明：「1 + 1 = 3」，證明系統應要求這種錯誤的證明不該存在。

1. **完整性**（**Completeness**）：若陳述為真，證明者（Prover）應該能找到一個可證明策略使驗證者（Verifier）相信。

例如：如果「1 + 1 = 2」是真的，那證明系統應該可以證明，而非找不到證明。

1. **零知識性**（**Zero-Knowledge**）：證明者（Prover）欲向驗證者（Verifier）證明知道「秘密（Security）」，必須在不直接揭露該秘密的情況下，僅能透過一些「陳述（Statement）」的交換，讓驗證者相信其知道該秘密，例如：證明者可以向驗證者證明自己知道密碼，而不需要直接告訴驗證者密碼。
2. **動機說明**

依據前述傳統驗證機制（無論是密碼式或生物特徵式）普遍仰賴資訊的直接揭露，導致使用者在追求安全驗證的同時，卻又必須承擔機密洩漏、重放攻擊、甚至嚴重的隱私外洩風險…等，此外，ZKP所提供的「不揭露秘密亦能驗證」，正好回應了當今資訊安全與隱私保護的雙重需求。本專題正是基於這樣的基礎，進一步探究 ZKP 的相關原理及成效。

1. **專題目標**

本專題旨在循序探討ZKP技術的發展脈絡，從基本架構出發，逐步導入實作效率與交互性的改善技術，藉由改善互動傳輸成本，最終達成兼顧安全性與可實用性的「非互動式驗證機制」，目標分為以下三階段：

1. **傳統ZKP（互動式驗證機制）**

初期的零知識證明為互動式機制，由證明者與驗證者透過多次隨機挑戰與回應的方式達成驗證。在理論上這種架構能確保完整性、可靠性與零知識性；然而，在實務應用上此類 ZKP 通常需多輪互動、傳輸成本高，不利於一對多、非同步或網路不穩定的環境中。為解決該問題，本專題進一步探討如何降低相關傳輸所造成的不穩定。

1. **Schnorr Protocol（簡化版互動式驗證機制）[2]**

Schnorr在結構上延續傳統互動式 ZKP 的安全性設計，但引入更簡潔的數學結構：**承諾（commit）**、**挑戰（challenge）**及**回應（response）**，大幅降低計算與通訊成本，並具有實作上的簡便性，並使其成為多種密碼學應用（如電子簽章）中的基礎原型。然而，Schnorr 協定仍屬於互動式協定，需證明者與驗證者同時在線互動，這在分散式系統或無法同步傳輸的場景中仍有侷限。因此，我們接續探討如何將其轉化為非互動式形式。

1. **Fiat–Shamir Heuristic（非互動式驗證機制）[3]**

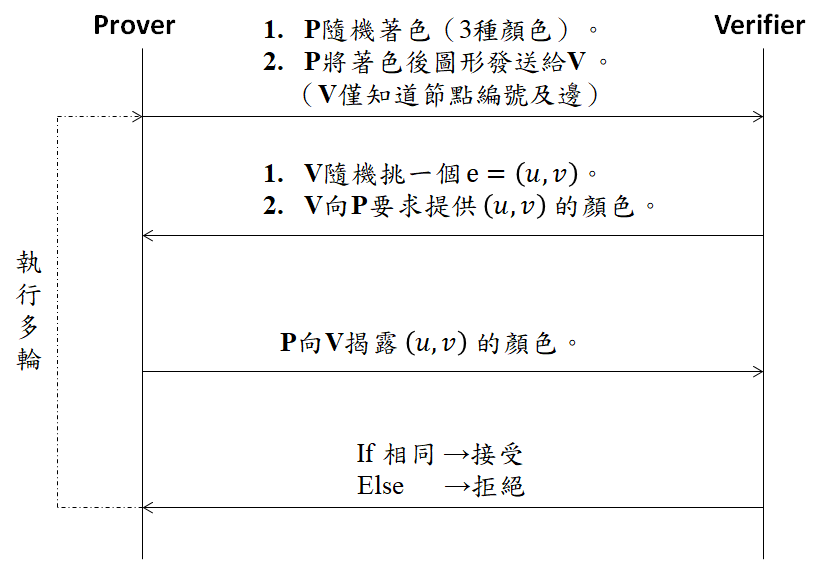
Fiat–Shamir 方法利用 hash 函數將互動式協定中的隨機挑戰值，改為由證明者本地端自行計算產生，實現 ZKP 的「非互動化」。它將原本需由驗證者提出的挑戰，改為自行產生 **hash** 值： ，消除雙方互動需求。

上述變更成功解決 Schnorr 協定的互動性限制，使得零知識證明可應用於區塊鏈（需支援**非互動式證明**的去中心化架構）、電子身份（如一次性提交的**非互動式驗證**流程）、無人監督驗證等場景，這些應用場域皆對**非互動性有高度依賴**，因此Fiat–Shamir 方法成為現代 ZKP 實作中的關鍵技術。

1. **系統設計**
2. **傳統ZKP — 以Graph 3-colorability（簡稱G3C）為例子**
   * + - **G3C定義**：假設有一G = ( V , E )，若存在一個映射（mapping），使得任一個邊，其兩個端點為，則稱G為「3-colorable」。
       - **前提假設**：
3. ：為G的一個3-coloring。
4. ：為 的隨機排列。
5. 定義 ：為一隨機的3-coloring。
6. 將每個放入一個標註為的箱子，並用金鑰（Key*v*）上鎖。
   * + - **ZKP過程**：
7. **P→V**：將前述箱子發送給驗證者，驗證端僅知道節點編號及邊，看不到點的著色。
8. **V→P**：隨機選擇一個邊，並且要求證明者提供兩端點和的顏色。
9. **P→V**：傳送存放和箱子的Key*u*及Key*v*給驗證者。
10. **V→P**：驗證端開啟箱子，若和顏色不同則「接受」，否則「拒絕」。

（重複多次，增加準確率）

* **可靠性（Soundness）：**假設G不為3-colorable，則至少有一個邊的兩個端點會是同色的（即），若P試圖作弊，驗證者會有 的機率選到不合規則的邊，且隨著重複多次執行，該機率將擴增至 以上，故V可以識破。
* **完整性（Completeness）：**若G真的為3-colorable，P及V依據前述協議進行交流後，可符合V任挑兩個端點的顏色皆不同，故V會接受。
* **零知識性（Zero-Knowledge）：**P可在不揭露「**節點顏色**」的情況下，向V證明圖G確實為3-colorable。
  + - * 互動圖：



1. **Schnorr Protocol**

屬於**簡化版互動式零知識證明**，以**離散對數之難度**作為核心，Prover 知道一個秘密私鑰，計算，並提供 Verifier 公鑰，Prover 欲在不透漏的情況下，讓 Verifier 相信其知道秘密 ，以下為參數設定：

質數**：**：隨機挑選的大質數

：

生成元（）**：**

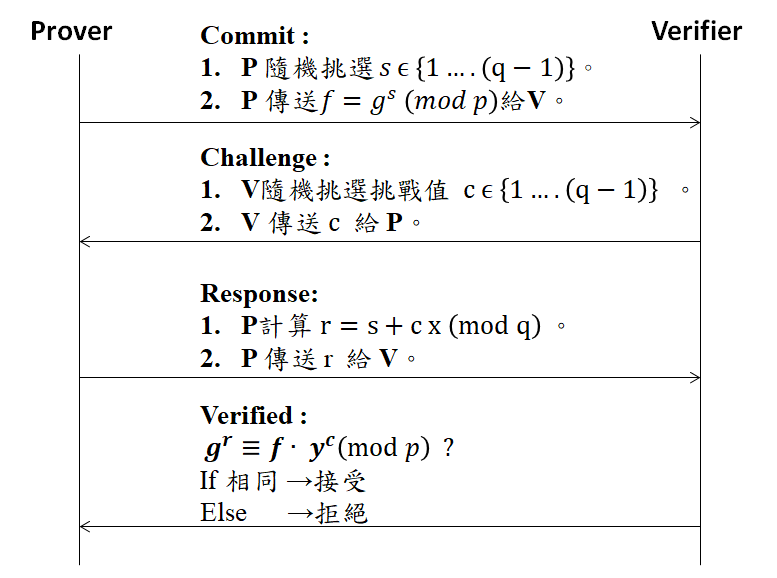
私鑰**： （隨機挑選）**

公鑰**：**

1. **承諾（commit）：**
2. Prover 隨機選擇一個 ephemeral key ，使得
3. 計算承諾值 ： 。
4. 傳送 給 verifier。
5. **挑戰（Challenge）：**
6. Verifier 產生一個隨機的挑戰值 ，使得
7. 傳送 給 Prover。
8. **回應（Response）：**
9. Prover 計算： 。
10. 傳送 給 Verifier。
11. Verifier 驗證：

（成立 → 接受；不成立 → 拒絕）

* **可靠性（Soundness）：**Prover在不知道 的情況下，其不可能找出所有 可以對應到有效的 值，故其透過亂猜的方式能夠回應滿足協議過程的 值機率僅為 ，故Prover難以欺騙Verifier。
* **完整性（Completeness）：**在前述協議執行一輪之後，Verifier將可確信Prover知道，首先 Prover 先隨機挑選 ，其後續只需傳送給Verifier，可使不知道的Verifier驗證，原因如下：
* **零知識性**（**Zero-Knowledge**）：Schnorr 協定滿足零知識性，即 Verifier 即使成功驗證 Prover 知道秘密 ，也無法從中得到任何的資訊。
  + - * 互動圖：

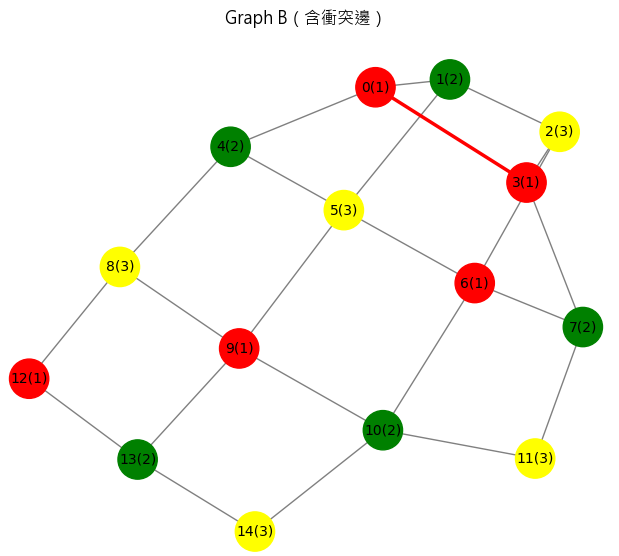
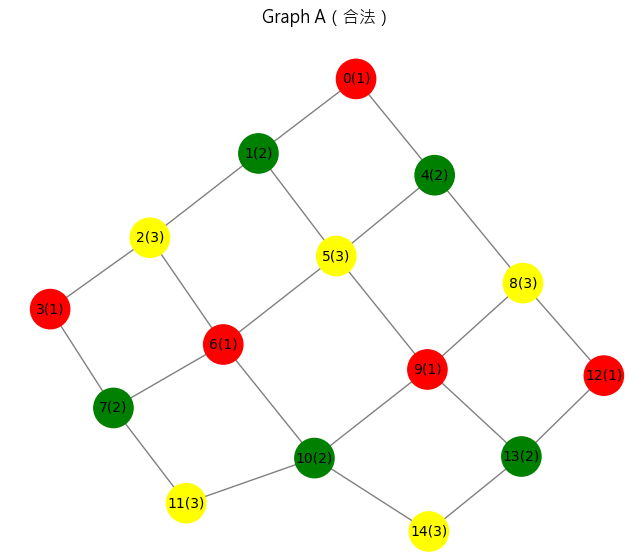


1. **Fiat–Shamir Heuristic（待研擬）**

**待研擬…**

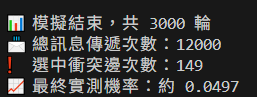
1. **專題成果**
2. **傳統ZKP — 以Graph 3-colorability（簡稱G3C）為例子**

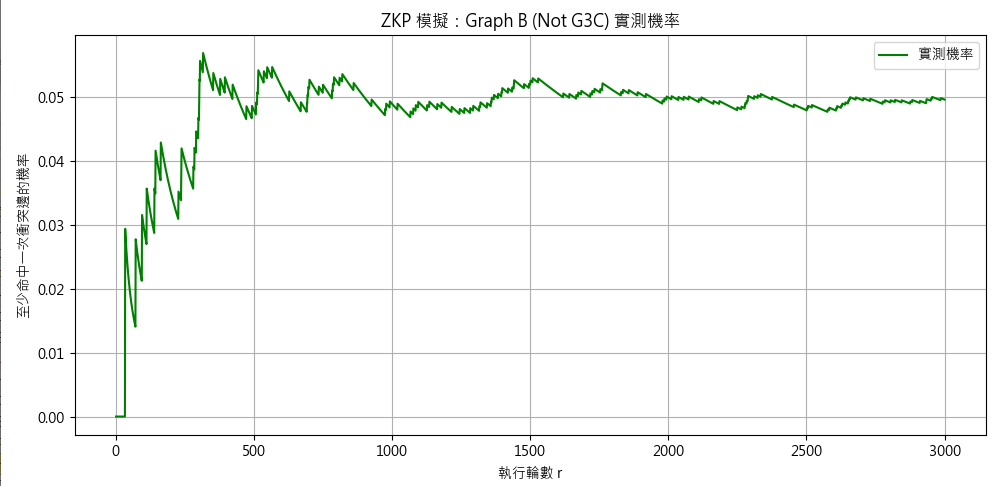
在本次模擬中，Graph A為3-coloring，Graph B為非3-coloring，針對Graph A模擬了20次，詳附錄八之（一）。



在Graph B模擬中，我們針對只有一個衝突邊（即不合法邊）的**非 3-colorable** 的圖（Graph B）進行 **3000 輪**的 ZKP 驗證實驗，並記錄每一輪中是否命中圖中唯一的一條不合法邊。如圖所示，橫軸為執行輪數 ，縱軸則為截至第 輪的「**至少命中一次衝突邊**」的累積實測機率：

* + - * 初始皆為0（未挑到衝突邊），後來逐漸變化劇烈。
      * 隨著「驗證次數」提高，挑到衝突邊機率快速上升，最終穩定於 **5%**，約等同於理論值 **。**
      * 執行多輪之後累積訊息傳達次數高達12000次。





1. **Schnorr Protocol**

前述模擬，我們針對傳統互動式零信任機制採用 Graph 3-colorability（G3C） 問題作為理論基礎，實作並模擬 3000 輪互動，驗證 Prover 是否擁有一組合法著色，而不洩漏實際配色資訊。此方法雖具備完整性與零知識性，但在每一輪中需傳遞 4 筆訊息（承諾、挑邊…等），並且必須透過多輪的執行才能使可信任度提高，使得整體訊息傳輸成本隨輪數成長，總傳遞次數達 12000 次（見附錄八之一）。

為改善此問題，我們進一步研讀相關文獻並實作 Schnorr 協定 [2]，其核心為基於離散對數困難度建立的互動式 ZKP。相較於 G3C，Schnorr 僅需交換一次承諾 ，收到挑戰 後，回傳 ，Verifier 再驗證 是否成立。整體來看，每輪僅需傳遞 3 筆訊息（承諾、挑戰、回應），大幅減少通訊負擔。

實驗結果顯示，模擬執行 100 輪 Schnorr 驗證，成功率達 100%（如附錄八之（二）），總訊息傳遞次數僅為 300 次，明顯低於傳統 ZKP 的 12000 次，在不犧牲安全性的前提下，顯示 Schnorr 協定在通訊效率上具有實質改善。

1. **Fiat–Shamir Heuristic（待研擬）**

**待研擬…**

1. **結論（待研擬）**

**待研擬…**

1. **未來展望（待研擬）**

**待研擬…**

1. **參考文獻**
2. Goldwasser, S., Micali, S., & Rackoff, C. (1985). The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, *18*(1), 186–208. https://doi.org/10.1137/0218012
3. Schnorr, C. P. (1990). Efficient identification and signatures for smart cards. In G. Brassard (Ed.), Advances in Cryptology – CRYPTO ’89 (Lecture Notes in Computer Science, Vol. 435, pp. 239–252). Springer. https://doi.org/10.1007/0-387-34805-0\_21
4. Fiat, A., & Shamir, A. (1987). How to prove yourself: Practical solutions to identification and signature problems. In A. M. Odlyzko (Ed.), Advances in Cryptology – CRYPTO ’86 (Lecture Notes in Computer Science, Vol. 263, pp. 186–194). Springer. https://doi.org/10.1007/3-540-47721-7\_12
5. **附錄**
6. **傳統ZKP — 以Graph 3-colorability（簡稱G3C）為例子**
   * + - **執行結果**

====== Graph A（合法）ZKP 模擬 ======

🔁 第 1 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (0, 1)（1 次）

- Prover 解鎖節點 0 色 1，節點 1 色 2（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 2 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (13, 14)（1 次）

- Prover 解鎖節點 13 色 2，節點 14 色 3（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 3 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (8, 9)（1 次）

- Prover 解鎖節點 8 色 3，節點 9 色 1（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 4 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (2, 3)（1 次）

- Prover 解鎖節點 2 色 3，節點 3 色 1（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 5 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (6, 10)（1 次）

- Prover 解鎖節點 6 色 1，節點 10 色 2（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 6 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (9, 10)（1 次）

- Prover 解鎖節點 9 色 1，節點 10 色 2（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 7 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (10, 14)（1 次）

- Prover 解鎖節點 10 色 2，節點 14 色 3（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 8 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (5, 9)（1 次）

- Prover 解鎖節點 5 色 3，節點 9 色 1（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 9 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (5, 6)（1 次）

- Prover 解鎖節點 5 色 3，節點 6 色 1（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 10 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (10, 11)（1 次）

- Prover 解鎖節點 10 色 2，節點 11 色 3（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 11 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (4, 8)（1 次）

- Prover 解鎖節點 4 色 2，節點 8 色 3（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 12 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (6, 7)（1 次）

- Prover 解鎖節點 6 色 1，節點 7 色 2（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 13 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (2, 6)（1 次）

- Prover 解鎖節點 2 色 3，節點 6 色 1（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 14 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (10, 14)（1 次）

- Prover 解鎖節點 10 色 2，節點 14 色 3（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 15 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (5, 9)（1 次）

- Prover 解鎖節點 5 色 3，節點 9 色 1（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 16 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (2, 6)（1 次）

- Prover 解鎖節點 2 色 3，節點 6 色 1（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 17 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (1, 5)（1 次）

- Prover 解鎖節點 1 色 2，節點 5 色 3（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 18 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (12, 13)（1 次）

- Prover 解鎖節點 12 色 1，節點 13 色 2（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 19 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (8, 12)（1 次）

- Prover 解鎖節點 8 色 3，節點 12 色 1（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

🔁 第 20 輪：

- Prover 傳送承諾（1 次）

- Verifier 挑邊 (8, 9)（1 次）

- Prover 解鎖節點 8 色 3，節點 9 色 1（2 次）

✅ 驗證通過

📦 本輪傳遞訊息總數：4

📊 模擬結束，共 20 輪

📨 總訊息傳遞次數：80

* + - * **程式碼**

#Traditional ZKP with G3C Problem

#Author: 林伯叡、黃杬霆

#Date: 2025/05/15

import random

import copy

import networkx as nx

import matplotlib

import matplotlib.pyplot as plt

matplotlib.rcParams['font.family'] = 'Microsoft JhengHei'  # 微軟正黑體

matplotlib.rcParams['axes.unicode\_minus'] = False  # 避免負號變成亂碼

# === 圖與著色 ===

graph\_A = {

    0: [1, 4], 1: [0, 2, 5], 2: [1, 3, 6], 3: [2, 7],

    4: [0, 5, 8], 5: [1, 4, 6, 9], 6: [2, 5, 7, 10], 7: [3, 6, 11],

    8: [4, 9, 12], 9: [5, 8, 10, 13], 10: [6, 9, 11, 14], 11: [7, 10],

    12: [8, 13], 13: [9, 12, 14], 14: [10, 13]

}

colors\_A = {

    0: 1, 1: 2, 2: 3, 3: 1, 4: 2, 5: 3,

    6: 1, 7: 2, 8: 3, 9: 1, 10: 2, 11: 3,

    12: 1, 13: 2, 14: 3

}

graph\_B = copy.deepcopy(graph\_A)

graph\_B[0].append(3)

graph\_B[3].append(0)  # 非法邊

# === ZKP 模擬函式 ===

def simulate\_zkp\_rounds(graph, colors, is\_valid=True, rounds=20):

    total\_messages = 0

    conflict\_hits = 0

    conflict\_history = []

    log = []

    edge\_set = set()

    for u in graph:

        for v in graph[u]:

            if (v, u) not in edge\_set:

                edge\_set.add((u, v))

    edge\_list = list(edge\_set)

    for r in range(1, rounds + 1):

        messages = 0

        log.append(f"🔁 第 {r} 輪：")

        messages += 1

        log.append(f"  - Prover 傳送承諾（1 次）")

        u, v = random.choice(edge\_list)

        messages += 1

        log.append(f"  - Verifier 挑邊 ({u}, {v})（1 次）")

        messages += 2

        log.append(f"  - Prover 解鎖節點 {u} 色 {colors[u]}，節點 {v} 色 {colors[v]}（2 次）")

        if colors[u] == colors[v]:

            log.append(f"  ❌ 顏色衝突，驗證失敗")

            conflict\_hits += 1

            conflict\_history.append(1)

        else:

            log.append(f"  ✅ 驗證通過")

            conflict\_history.append(0)

        total\_messages += messages

        log.append(f"  📦 本輪傳遞訊息總數：{messages}")

        if not is\_valid:

            empirical\_p = sum(conflict\_history) / r

            theoretical\_p = 1 - (1 - (1 / len(edge\_list))) \*\* r

            log.append(f"  📈 累積實測機率：{empirical\_p:.4f}，理論機率：約 {theoretical\_p:.4f}")

    log.append(f"\n📊 模擬結束，共 {rounds} 輪")

    log.append(f"📨 總訊息傳遞次數：{total\_messages}")

    if not is\_valid:

        log.append(f"❗ 選中衝突邊次數：{conflict\_hits}")

        final\_empirical = sum(conflict\_history) / rounds

        final\_theoretical = 1 - (1 - (1 / len(edge\_list))) \*\* rounds

        log.append(f"📈 最終實測機率：約 {final\_empirical:.4f}")

        log.append(f"📈 最終理論機率：約 {final\_theoretical:.4f}")

    return log

# === 繪圖函式 ===

def draw\_single\_graph(graph, colors, title, highlight\_conflict=False):

    G = nx.Graph()

    for u in graph:

        for v in graph[u]:

            G.add\_edge(u, v)

    pos = nx.spring\_layout(G, seed=42)

    color\_map = {1: 'red', 2: 'green', 3: 'yellow'}

    node\_colors = [color\_map[colors[n]] for n in G.nodes()]

    conflict\_edges = []

    normal\_edges = []

    for u, v in G.edges():

        if colors[u] == colors[v] and highlight\_conflict:

            conflict\_edges.append((u, v))

        else:

            normal\_edges.append((u, v))

    plt.figure(figsize=(7, 6))

    nx.draw\_networkx\_nodes(G, pos, node\_color=node\_colors, node\_size=800)

    nx.draw\_networkx\_edges(G, pos, edgelist=normal\_edges, edge\_color='gray')

    if highlight\_conflict:

        nx.draw\_networkx\_edges(G, pos, edgelist=conflict\_edges, edge\_color='red', width=2.5)

    nx.draw\_networkx\_labels(G, pos, labels={n: f"{n}({colors[n]})" for n in G.nodes()}, font\_size=10)

    plt.title(title)

    plt.axis("off")

    plt.tight\_layout()

    plt.show()

# === 合法性檢查（推薦搭配） ===

def check\_graph\_validity(graph, colors):

    for u in graph:

        for v in graph[u]:

            if u < v and colors[u] == colors[v]:

                print(f"❌ 發現衝突邊 ({u}, {v})，同為色 {colors[u]}")

                return False

    print("✅ 著色合法，無衝突邊")

    return True

enable\_check = "TRUE"

assert enable\_check in [

    "TRUE",

    "FALSE",

], f"Unsupported enable\_check: {enable\_check}"

enable\_plotcurve = "FALSE"

assert enable\_plotcurve in [

    "TRUE",

    "FALSE",

], f"Unsupported enable\_plotcurve: {enable\_plotcurve}"

if enable\_check == "TRUE":

    # === 執行檢查與單圖繪製 ===

    print("\n🧪 檢查 Graph A 合法性：")

    check\_graph\_validity(graph\_A, colors\_A)

    print("\n🧪 檢查 Graph B 合法性：")

    check\_graph\_validity(graph\_B, colors\_A)

    draw\_single\_graph(graph\_A, colors\_A, "Graph A（合法）", highlight\_conflict=True)

    draw\_single\_graph(graph\_B, colors\_A, "Graph B（含衝突邊）", highlight\_conflict=True)

# === 執行模擬 ===

rounds\_A = 20

rounds\_B = 3000

log\_A = simulate\_zkp\_rounds(graph\_A, colors\_A, is\_valid=True, rounds=rounds\_A)

log\_B = simulate\_zkp\_rounds(graph\_B, colors\_A, is\_valid=False, rounds=rounds\_B)

print("\n====== Graph A（合法）ZKP 模擬 ======")

for line in log\_A:

    print(line)

print("\n====== Graph B（非法）ZKP 模擬 ======")

for line in log\_B:

    print(line)

if enable\_plotcurve == "TRUE":

    def extract\_conflict\_history(log):

        return [1 if "❌ 顏色衝突" in line else 0 for line in log if "✅ 驗證通過" in line or "❌ 顏色衝突" in line]

    conflict\_history = extract\_conflict\_history(log\_B)

    total\_edges = sum(len(neigh) for neigh in graph\_B.values()) // 2

    empirical\_probs = []

    conflict\_sum = 0

    for r in range(1, len(conflict\_history)+1):

        conflict\_sum += conflict\_history[r - 1]

        empirical\_probs.append(conflict\_sum / r)

    theoretical\_probs = [1 - (1 - 1 / total\_edges) \*\* r for r in range(1, len(conflict\_history)+1)]

    # 畫圖

    plt.figure(figsize=(10, 5))

    plt.plot(range(1, rounds\_B + 1), empirical\_probs, label="實測機率", color='Green')

    # plt.plot(range(1, rounds\_B + 1), theoretical\_probs, label="理論機率", linestyle='--', color='Red')

    plt.xlabel("執行輪數 r")

    plt.ylabel("至少命中一次衝突邊的機率")

    plt.title("ZKP 模擬：Graph B (Not G3C) 實測機率")

    # plt.title("ZKP 模擬：Graph B 實測機率 vs 理論機率")

    plt.grid(True)

    plt.legend()

    plt.tight\_layout()

    plt.show()

1. **Schnorr Protocol**
2. **執行結果：**

**Round 1: s=11294, f=82303, c=12474, r=8884, g^r=13231, f\*y^c=13231, pass=True**

**Round 2: s=3666, f=119192, c=10491, r=8754, g^r=58879, f\*y^c=58879, pass=True**

**Round 3: s=4276, f=104912, c=1036, r=2636, g^r=34056, f\*y^c=34056, pass=True**

**Round 4: s=1610, f=86478, c=12127, r=9854, g^r=70401, f\*y^c=70401, pass=True**

**Round 5: s=4157, f=79508, c=1726, r=335, g^r=45427, f\*y^c=45427, pass=True**

**Round 6: s=6553, f=121269, c=6996, r=382, g^r=14069, f\*y^c=14069, pass=True**

**Round 7: s=7086, f=38727, c=11672, r=3568, g^r=103889, f\*y^c=103889, pass=True**

**Round 8: s=655, f=68841, c=5684, r=7260, g^r=84377, f\*y^c=84377, pass=True**

**Round 9: s=2770, f=109447, c=3240, r=8142, g^r=30015, f\*y^c=30015, pass=True**

**Round 10: s=10430, f=57522, c=4245, r=2026, g^r=108736, f\*y^c=108736, pass=True**

**Round 11: s=5736, f=56896, c=393, r=6055, g^r=4820, f\*y^c=4820, pass=True**

**Round 12: s=10504, f=120676, c=4938, r=4817, g^r=87842, f\*y^c=87842, pass=True**

**Round 13: s=5537, f=112438, c=1408, r=8410, g^r=23062, f\*y^c=23062, pass=True**

**Round 14: s=696, f=56081, c=148, r=11458, g^r=92572, f\*y^c=92572, pass=True**

**Round 15: s=10626, f=66735, c=1612, r=248, g^r=83203, f\*y^c=83203, pass=True**

**Round 16: s=11583, f=67961, c=5321, r=2714, g^r=51054, f\*y^c=51054, pass=True**

**Round 17: s=3653, f=19745, c=5142, r=10373, g^r=103832, f\*y^c=103832, pass=True**

**Round 18: s=1264, f=53355, c=12466, r=11448, g^r=113707, f\*y^c=113707, pass=True**

**Round 19: s=9213, f=58585, c=10371, r=10776, g^r=111756, f\*y^c=111756, pass=True**

**Round 20: s=3969, f=118967, c=2653, r=116, g^r=11447, f\*y^c=11447, pass=True**

**Round 21: s=12075, f=62898, c=8756, r=6288, g^r=79147, f\*y^c=79147, pass=True**

**Round 22: s=2090, f=2780, c=432, r=1951, g^r=91468, f\*y^c=91468, pass=True**

**Round 23: s=11204, f=24079, c=8139, r=11347, g^r=40649, f\*y^c=40649, pass=True**

**Round 24: s=8094, f=55320, c=8217, r=7321, g^r=7743, f\*y^c=7743, pass=True**

**Round 25: s=6011, f=43424, c=2184, r=6021, g^r=77782, f\*y^c=77782, pass=True**

**Round 26: s=794, f=67329, c=3876, r=5605, g^r=4100, f\*y^c=4100, pass=True**

**Round 27: s=3650, f=41758, c=9862, r=7901, g^r=127997, f\*y^c=127997, pass=True**

**Round 28: s=7931, f=53649, c=574, r=8756, g^r=81619, f\*y^c=81619, pass=True**

**Round 29: s=4910, f=10268, c=3196, r=2575, g^r=97285, f\*y^c=97285, pass=True**

**Round 30: s=10504, f=120676, c=8215, r=6465, g^r=25724, f\*y^c=25724, pass=True**

**Round 31: s=9452, f=108841, c=6925, r=2799, g^r=98778, f\*y^c=98778, pass=True**

**Round 32: s=2849, f=6163, c=705, r=12333, g^r=6123, f\*y^c=6123, pass=True**

**Round 33: s=5990, f=114081, c=6374, r=10413, g^r=24426, f\*y^c=24426, pass=True**

**Round 34: s=5222, f=106102, c=5160, r=2849, g^r=6163, f\*y^c=6163, pass=True**

**Round 35: s=12257, f=75210, c=7180, r=11491, g^r=117991, f\*y^c=117991, pass=True**

**Round 36: s=10050, f=71888, c=7616, r=2848, g^r=121107, f\*y^c=121107, pass=True**

**Round 37: s=2814, f=36528, c=5126, r=9064, g^r=68508, f\*y^c=68508, pass=True**

**Round 38: s=8294, f=59121, c=5446, r=11115, g^r=75938, f\*y^c=75938, pass=True**

**Round 39: s=3247, f=114048, c=9514, r=3690, g^r=104512, f\*y^c=104512, pass=True**

**Round 40: s=216, f=116519, c=11631, r=6719, g^r=103052, f\*y^c=103052, pass=True**

**Round 41: s=11292, f=40237, c=793, r=10532, g^r=116226, f\*y^c=116226, pass=True**

**Round 42: s=4793, f=123965, c=1716, r=10299, g^r=18089, f\*y^c=18089, pass=True**

**Round 43: s=7134, f=115178, c=7696, r=2282, g^r=72263, f\*y^c=72263, pass=True**

**Round 44: s=4586, f=75491, c=7938, r=10050, g^r=71888, f\*y^c=71888, pass=True**

**Round 45: s=2154, f=105426, c=11675, r=3535, g^r=118625, f\*y^c=118625, pass=True**

**Round 46: s=638, f=109871, c=4667, r=1423, g^r=107246, f\*y^c=107246, pass=True**

**Round 47: s=9598, f=23809, c=2603, r=1069, g^r=116398, f\*y^c=116398, pass=True**

**Round 48: s=1467, f=86000, c=12409, r=8373, g^r=55832, f\*y^c=55832, pass=True**

**Round 49: s=2079, f=20942, c=6576, r=2814, g^r=36528, f\*y^c=36528, pass=True**

**Round 50: s=6912, f=107013, c=6325, r=8292, g^r=80109, f\*y^c=80109, pass=True**

**Round 51: s=11501, f=49123, c=1251, r=1744, g^r=64619, f\*y^c=64619, pass=True**

**Round 52: s=11870, f=47145, c=5823, r=1711, g^r=50075, f\*y^c=50075, pass=True**

**Round 53: s=363, f=22190, c=6458, r=839, g^r=14179, f\*y^c=14179, pass=True**

**Round 54: s=5172, f=32630, c=4560, r=10832, g^r=32493, f\*y^c=32493, pass=True**

**Round 55: s=5361, f=16841, c=12198, r=1258, g^r=69362, f\*y^c=69362, pass=True**

**Round 56: s=9548, f=72291, c=9353, r=3658, g^r=29108, f\*y^c=29108, pass=True**

**Round 57: s=1901, f=115448, c=4560, r=7561, g^r=36589, f\*y^c=36589, pass=True**

**Round 58: s=11989, f=96984, c=8451, r=8468, g^r=41097, f\*y^c=41097, pass=True**

**Round 59: s=1786, f=118981, c=957, r=12258, g^r=123951, f\*y^c=123951, pass=True**

**Round 60: s=7239, f=35181, c=3929, r=8796, g^r=457, f\*y^c=457, pass=True**

**Round 61: s=10597, f=29552, c=6282, r=5903, g^r=19145, f\*y^c=19145, pass=True**

**Round 62: s=4775, f=40138, c=3881, r=4922, g^r=94785, f\*y^c=94785, pass=True**

**Round 63: s=8472, f=36832, c=10492, r=2364, g^r=114647, f\*y^c=114647, pass=True**

**Round 64: s=4800, f=6630, c=12613, r=11284, g^r=20904, f\*y^c=20904, pass=True**

**Round 65: s=5822, f=40836, c=296, r=1688, g^r=41356, f\*y^c=41356, pass=True**

**Round 66: s=10880, f=78888, c=6663, r=12567, g^r=78524, f\*y^c=78524, pass=True**

**Round 67: s=2722, f=95804, c=12720, r=4331, g^r=107641, f\*y^c=107641, pass=True**

**Round 68: s=5810, f=127026, c=2534, r=65, g^r=115922, f\*y^c=115922, pass=True**

**Round 69: s=10347, f=89579, c=1989, r=12647, g^r=2887, f\*y^c=2887, pass=True**

**Round 70: s=9844, f=105786, c=8334, r=7697, g^r=22284, f\*y^c=22284, pass=True**

**Round 71: s=6614, f=79608, c=7592, r=11536, g^r=80824, f\*y^c=80824, pass=True**

**Round 72: s=7587, f=44125, c=4622, r=11861, g^r=50570, f\*y^c=50570, pass=True**

**Round 73: s=1674, f=47866, c=4088, r=6298, g^r=73305, f\*y^c=73305, pass=True**

**Round 74: s=1923, f=75808, c=5548, r=4533, g^r=61212, f\*y^c=61212, pass=True**

**Round 75: s=10171, f=57860, c=7277, r=1029, g^r=28816, f\*y^c=28816, pass=True**

**Round 76: s=8161, f=18481, c=1034, r=3255, g^r=39240, f\*y^c=39240, pass=True**

**Round 77: s=3604, f=77919, c=2072, r=324, g^r=122819, f\*y^c=122819, pass=True**

**Round 78: s=6923, f=55152, c=8776, r=8138, g^r=88674, f\*y^c=88674, pass=True**

**Round 79: s=11988, f=20798, c=7788, r=3424, g^r=92840, f\*y^c=92840, pass=True**

**Round 80: s=7454, f=81401, c=4939, r=3400, g^r=97553, f\*y^c=97553, pass=True**

**Round 81: s=6357, f=78973, c=1064, r=11954, g^r=106830, f\*y^c=106830, pass=True**

**Round 82: s=7841, f=27650, c=5975, r=2147, g^r=103736, f\*y^c=103736, pass=True**

**Round 83: s=39, f=111641, c=6819, r=12723, g^r=97684, f\*y^c=97684, pass=True**

**Round 84: s=2402, f=118935, c=1595, r=2750, g^r=122391, f\*y^c=122391, pass=True**

**Round 85: s=3504, f=8746, c=4673, r=1258, g^r=69362, f\*y^c=69362, pass=True**

**Round 86: s=9655, f=28978, c=4567, r=1088, g^r=25281, f\*y^c=25281, pass=True**

**Round 87: s=1142, f=98788, c=7604, r=2, g^r=289, f\*y^c=289, pass=True**

**Round 88: s=7128, f=60584, c=1423, r=8838, g^r=93295, f\*y^c=93295, pass=True**

**Round 89: s=677, f=77510, c=2461, r=4013, g^r=92418, f\*y^c=92418, pass=True**

**Round 90: s=4947, f=102716, c=2156, r=10549, g^r=30360, f\*y^c=30360, pass=True**

**Round 91: s=53, f=69816, c=10936, r=573, g^r=48449, f\*y^c=48449, pass=True**

**Round 92: s=8574, f=91777, c=7047, r=8712, g^r=49252, f\*y^c=49252, pass=True**

**Round 93: s=5249, f=97459, c=9681, r=8994, g^r=75097, f\*y^c=75097, pass=True**

**Round 94: s=3915, f=63471, c=9099, r=6600, g^r=57222, f\*y^c=57222, pass=True**

**Round 95: s=8843, f=97966, c=11572, r=8802, g^r=23980, f\*y^c=23980, pass=True**

**Round 96: s=3771, f=69230, c=4197, r=6786, g^r=103322, f\*y^c=103322, pass=True**

**Round 97: s=3949, f=105356, c=5649, r=4715, g^r=31083, f\*y^c=31083, pass=True**

**Round 98: s=2277, f=3847, c=7531, r=10218, g^r=105902, f\*y^c=105902, pass=True**

**Round 99: s=4691, f=43040, c=9487, r=12359, g^r=16823, f\*y^c=16823, pass=True**

**Round 100: s=630, f=55923, c=2465, r=10498, g^r=104205, f\*y^c=104205, pass=True**

**Total successful rounds: 100/100**

1. **程式碼：**
2. #ZKP with Schnorr
3. #Author: 林伯叡、黃杬霆
4. #Date: 2025/05/08
5. from sympy import isprime
6. import random
7. import os
8. # ===== 公開參數設定 =====
9. def generate\_safe\_prime(bits=8):
10. """
11. 產生一組安全質數 p, q，使得 p = q \* r + 1，且 q, p 都是質數
12. bits: q 的位元長度
13. """
14. while True:
15. q = random.getrandbits(bits)
16. q |= 1  # 確保是奇數
17. if isprime(q):
18. for r in range(2, 20):
19. p = q \* r + 1
20. if isprime(p):
21. return p, q
22. # 產生參數 p, q
23. # 5 → p為2~3位數、8 → p為3~4位數、16 → p為5~6位數、32+ → 高安全性測試用
24. p, q = generate\_safe\_prime(bits=16)
25. def find\_generator(p, q):
26. """找一個生成元 g，使得 g^q ≡ 1 mod p"""
27. for g in range(2, p):
28. if pow(g, q, p) == 1:
29. return g
30. raise Exception("找不到生成元")
31. g = find\_generator(p, q)
32. x = random.randint(1, q - 1)
33. y = pow(g, x, p)
34. # ===== Schnorr 協定主程式 =====
35. def schnorr\_proof(rounds=100):
36. """執行 Schnorr 協定共 rounds 回合，顯示每輪資訊與統計成功次數"""
37. logs = []
38. success\_count = 0
39. for i in range(1, rounds + 1):
40. s = random.randint(1, q - 1)
41. f = pow(g, s, p)
42. c = random.randint(1, q - 1)
43. r = (s + c \* x) % q
44. left = pow(g, r, p)
45. right = (f \* pow(y, c, p)) % p
46. passed = left == right
47. if passed:
48. success\_count += 1
49. logs.append(f"Round {i}: s={s}, f={f}, c={c}, r={r}, g^r={left}, f\*y^c={right}, pass={passed}")
50. logs.append(f"\nTotal successful rounds: {success\_count}/{rounds}")
51. return logs
52. def write\_result(logs, filename="result.txt"):
53. """將結果寫入與此 .py 程式同一個資料夾"""
54. script\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  # 此 .py 檔案的所在資料夾
55. filepath = os.path.join(script\_dir, filename)
56. with open(filepath, "w") as f:
57. f.write("\n".join(logs))
58. print(f"結果已寫入：{filepath}")
59. # ===== 主程式執行區塊 =====
60. if \_\_name\_\_ == "\_\_main\_\_":
61. result\_logs = schnorr\_proof(100)
62. write\_result(result\_logs)
63. **Fiat–Shamir Heuristic（待研擬）**

**待研擬…**