區塊鏈作業一報告

利用 Pollard’s Rho 方法解離散對數問題 (DLP)

姓名：胡樂麒　　學號：B1228005

長庚大學人工智慧學系大三

2025 年 10 月

摘要

本報告以兩種方式完成離散對數問題 (Discrete Logarithm Problem, DLP)：(1) 使用 Python 自行實作 Floyd–Pollard Rho 演算法；(2) 以 PARI/GP 套件進行快速驗證。最終兩種方法所得結果一致，成功求得正確的離散對數值。

1　作業目標

本作業的使用 Pollard’s Rho 方法解離散對數問題，其數學形式如下：

$$h \equiv g^x \pmod{p}$$

其中給定質數模數 p、生成元 g 及元素 h，要求出未知整數 x。

2　方法一：使用 Python 實作 Floyd–Pollard Rho 演算法

2.1　原理概述

Pollard’s Rho 方法藉由定義一個擬隨機函數 $f(x)$：

$$x_{i+1} = f(x_i),$$

不斷運算直到出現「碰撞」(collision)：$x_i = x_j$，此時有：

$$g^{a_i} h^{b_i} = g^{a_j} h^{b_j} \Rightarrow x = -\frac{a_i - a_j}{b_i - b_j} \pmod{n}.$$

演算法 average-case time complexity $O(\sqrt{n})$。

本程式運用 **Floyd cycle detection**（烏龜與兔子法）尋找循環。

2.2 Python 程式碼

Listing 1: Floyd–Pollard Rho 演算法實作 (Python)

```python
import random
from math import gcd

def pollards_rho_floyd(g, h, p, n, max_attempts=5):

    def partition(x):
        if x % 3 == 0:
            return 0
        elif x % 3 == 1:
            return 1
        else:
            return 2

    def f(x, a, b):
        part = partition(x)
        if part == 0:
            x = (h * x) % p
            b = (b + 1) % n
        elif part == 1:
            x = (x * x) % p
            a = (2 * a) % n
            b = (2 * b) % n
        else:
            x = (g * x) % p
            a = (a + 1) % n
        return x, a, b

    for attempt in range(max_attempts):
        x_t, a_t, b_t = g, 1, 0
        x_h, a_h, b_h = g, 1, 0

        for i in range(1, 2 * n):
            x_t, a_t, b_t = f(x_t, a_t, b_t)
            x_h, a_h, b_h = f(*f(x_h, a_h, b_h))

            if x_t == x_h:
                r = (a_t - a_h) % n
                s = (b_h - b_t) % n
                if s == 0 or gcd(s, n) != 1:
                    print(f"[嘗試 {attempt+1}] s 不可逆，重試")
                    break
```

```
                try:
                    s_inv = pow(s, -1, n)
                    x_result = (r * s_inv) % n
                    return x_result
                except ValueError:
                    print(f"[嘗試 {attempt+1}] 模反元素錯誤，重試")
                    break
    raise ValueError("超過最大嘗試次數，仍未找到解")

if __name__ == "__main__":
    import time

    p = 58002118547
    n = 29001059273
    g = 13513236970
    h = 39818986975

    print("開始執行 Floyd-Pollard rho DLP")

    try:
        x = pollards_rho_floyd(g, h, p, n)
        print(f"找到的 x = {x}")
        print(f"驗證: {g}^{x} mod {p} = {pow(g, x, p)}")
    except ValueError as e:
        print(e)
```

2.3 實驗結果

$$p = 58002118547, \quad n = 29001059273, \quad g = 13513236970, \quad h = 39818986975$$

程式輸出結果如下:

```
開始執行 Floyd-Pollard rho DLP
找到的 x = 10802198852
驗證: 13513236970^10802198852 mod 58002118547 = 39818986975
```

3 方法二: 使用 PARI/GP 套件

3.1 操作步驟

```
$ gp
p = 404859789103686130573;
n = 337383157586405108 81;
g = 184835890044895767 62;
h = 541288209845519354 52;
default(parisize, 100000000)
x = znlog(Mod(h, p), Mod(g, p), n)
x
```

執行結果：

$$x = 4949827726406875460$$

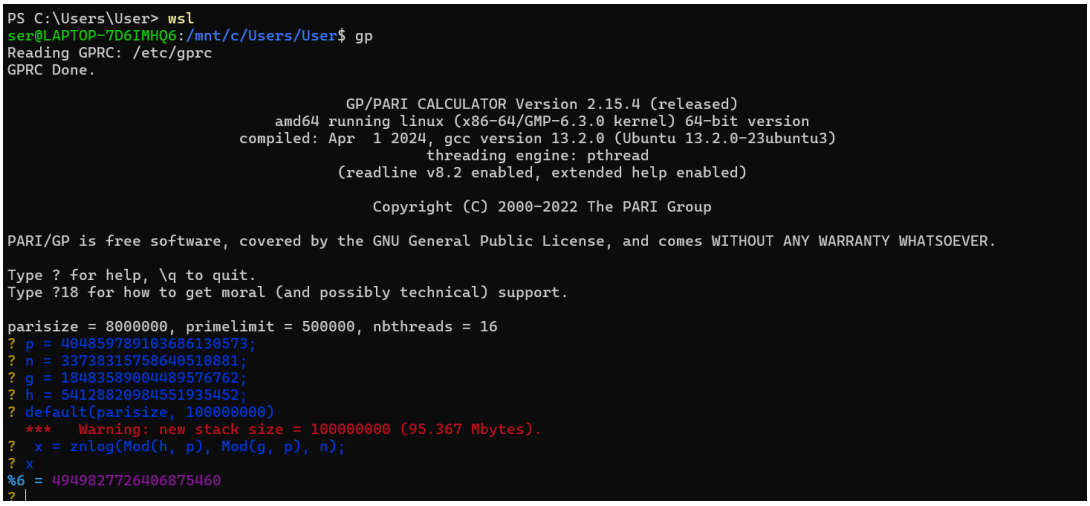

图 1: 於 WSL 下使用 PARI/GP 計算離散對數的執行畫面

4 比較與結論

方法	工具	優點	缺點
Python Pollard’s Rho	自行實作	可以看到運算流程	速度較慢
PARI/GP	電腦代數系統	運算快速且精確	黑箱運算

我在本作業以這兩種方式求解 DLP，且兩種方法結果一致。我覺得 PARI/GP 比 Python 實作方便且迅速，就算大數字也能很快得出結果，只是要先在終端機下載 wsl 環境與套件。