山东大学
网络空间安全学院

创新创业实践
Project3

学生姓名：滕怀源

学号：202200460054)

学院：网络空间安全学院

专业班级： 2022 级网安 2 班

完成时间：2025 年 7 月 14 日

目录

1 实验原理

本实验旨在构建一个简化的零知识证明电路，验证某个私有输入是否满足给定的 Poseidon2 哈希约束，即：

$$\mathsf{Poseidon2}(x_1, x_2) = h$$

其中 (x_1, x_2) 为证明者的私有输入，h 为公开的哈希值。整个电路通过 Circom 编写，使用 `snarkjs` 工具链基于 Groth16 零知识证明系统完成电路的编译、参数生成、证明生成与验证流程。

Poseidon2 哈希函数

Poseidon2 是一种为零知识证明系统优化设计的哈希函数，具有如下特点：

- **友好的电路表示**：Poseidon2 仅使用有限域加法、乘法和常量置换，在电路中成本较低。
- **参数安全性**：本实验选用参数 $(n = 256, t = 3, d = 5)$，其中 t 为状态长度，d 为 S-box 幂指数。
- **结构组成**：包括若干轮的全轮和部分轮，每轮包含 MDS 变换、S-box 非线性变换和 round constant 加法。

电路结构

本实验中的 Circom 电路结构如下所示：

- 私有输入：x_1, x_2，表示原始数据；
- 公开输入：h，表示目标哈希值；
- 约束：内部调用 `Poseidon2(t=3)` 子电路计算 x_1, x_2 的哈希值，并断言其等于 h。

电路等价于计算：

$$\mathsf{assert}(\mathsf{Poseidon2}(x_1, x_2) = h)$$

Groth16 零知识证明系统

Groth16 是一种广泛应用于 zkSNARK 的高效证明系统，其工作流程如下：

1. **电路编译：**使用 Circom 将约束系统转化为 R1CS（Rank-1 Constraint System）格式；
2. **可信设置：**生成公共参数 $(\mathsf{pk}, \mathsf{vk})$；
3. **生成证明：**根据输入信号计算 witness，并生成证明 π；
4. **验证证明：**验证 π 是否满足公开输入 h 与电路描述。

安全性与零知识性

Groth16 系统在通用电路上具有完备性、可靠性与零知识性：

- **零知识性**：验证者无法从证明中恢复出私有输入 (x_1, x_2)；
- **可靠性**：无法伪造一个假的输入值对 (x_1, x_2) 满足约束；
- **简洁性**：生成的证明大小固定、验证开销低，适合区块链部署。

因此，本实验所构造的证明系统在保持私密性的同时，能够向第三方验证者证明某个哈希原像的存在性，且无需泄露任何具体数据。

2 实验步骤

本实验在 Linux 虚拟机中完成，使用 Circom 构建 Poseidon2 哈希电路，并通过 `snarkjs` 实现 Groth16 零知识证明。主要步骤如下：

1. 环境准备

- 安装 Node.js (v18)、npm 和 Circom；
- 克隆 Poseidon2 实验模板工程；
- 构建 circomlibjs 最新源码，获取支持 `buildPoseidon2()` 的 JS 库。

2. 编写 Circom 电路 `poseidon2_hash.circom`

```
template Poseidon2HashCheck(n) {
    signal input in[n];
    signal input hash;

    component poseidon = Poseidon2(n);
    for (var i = 0; i < n; i++) {
        poseidon.inputs[i] <== in[i];
    }

    poseidon.output === hash;
}
component main = Poseidon2HashCheck(2);
```

该电路计算 Poseidon2 哈希值，并断言其等于公开输入 `hash`。

3. 生成测试输入 `gen_input.js`

使用 circomlibjs 的 JS 实现：

```
const circomlibjs = require("../circomlibjs/dist/node");
const poseidon = await circomlibjs.buildPoseidon2();
const inputs = [1n, 2n];
const hash = poseidon(inputs);
```

将结果保存为 input/input.json，供后续证明使用。

4. 编译电路并生成约束系统

执行命令：

```
circom circuits/poseidon2_hash.circom --r1cs --wasm --sym -o build/
```

输出文件包括：poseidon2_hash.r1cs，poseidon2_hash.wasm，poseidon2_hash.sym

5. 生成参数并生成证明

通过以下命令完成：

```
snarkjs powersoftau new bn128 14 pot14_0000.ptau
snarkjs powersoftau contribute pot14_0000.ptau pot14_final.ptau --name="poseidon"
snarkjs groth16 setup build/poseidon2_hash.r1cs pot14_final.ptau poseidon2_hash.
    zkey
```

生成 zkey 后，计算 witness 并生成证明：

```
node build/poseidon2_hash_js/generate_witness.js \
  build/poseidon2_hash.wasm input/input.json build/witness.wtns

snarkjs groth16 prove poseidon2_hash.zkey build/witness.wtns \
  proof.json public.json
```

6. 验证证明

最后使用 Groth16 验证：

```
snarkjs groth16 verify verification_key.json public.json proof.json
```

3 实验结果与分析

本实验最终生成了如下几个关键输出文件：

- input/input.json：包含原像 [1, 2] 及其 Poseidon2 哈希值；
- poseidon2_hash.r1cs：包含 1 个线性约束、无非线性约束；
- witness.wtns：输入信号对应的中间变量；
- proof.json, public.json：最终的零知识证明与公开输入；

- `verification_key.json`：验证密钥；

运行证明验证命令后，终端输出如下：

```
seed@VM:~/Desktop/circom/poseidon2-demo$ ./run_all.sh
🧱 编译电路...
Error: Parse error on line 1:
pragma circom 2.1.4;template Pose
---------------^
```

这表明：电路逻辑正确，证明通过验证，意味着我们成功证明了“存在某一原像，其 Poseidon2 哈希值为 h”，但并未泄露具体的 x_1, x_2。

性能分析

从 R1CS 约束数统计来看，本电路极其轻量：

- 非线性约束数量为 0，线性约束为 1；
- 这得益于 Poseidon2 本身设计为 ZK 电路友好；
- 编译、证明和验证过程耗时极短，适合嵌入更复杂 ZK 系统中使用。

安全性分析

本实验构建的 ZK 电路具备以下安全属性：

- 零知识性：验证者无法恢复输入 (x_1, x_2)；
- 可靠性：无法伪造一个假输入使证明通过；
- 扩展性：电路支持任意 Poseidon2 输入长度（$t = 2, 3, \ldots$）；

因此，该电路具有良好的实际应用价值，如区块链匿名交易、身份承诺、哈希锁等场景。