山东大学

SHANDONG UNIVERSITY

Project 3: circom 实现 poseidon2

姓　　名: 谢星豪

学　　号: 202100161170

专　　业: 网络空间安全

班　　级: 网安一班

2025 年 8 月 5 日

目录

1 实验内容

Project 3: 用 circom 实现 poseidon2 哈希算法的电路

1) poseidon2 哈希算法参数参考参考文档 1 的 Table1，用 (n,t,d)=(256,3,5) 或 (256,2,5)

2）电路的公开输入用 poseidon2 哈希值，隐私输入为哈希原象，哈希算法的输入只考虑一个 block 即可。

3) 用 Groth16 算法生成证明

参考文档：

1. poseidon2 哈希算法 https://eprint.iacr.org/2023/323.pdf
2. circom 说明文档 https://docs.circom.io/
3. circom 电路样例 https://github.com/iden3/circomlib

2 实验原理

Poseidon2 是一种专为零知识证明（ZKP）环境设计的代数哈希函数，它具有低约束度、高性能和高安全性的特点。与传统的通用哈希函数（如 SHA-256）不同，Poseidon2 的设计目标是在有限域（$\mathbb{F}_p$）上高效运行，特别是在需要生成和验证 ZKP 的场景中。

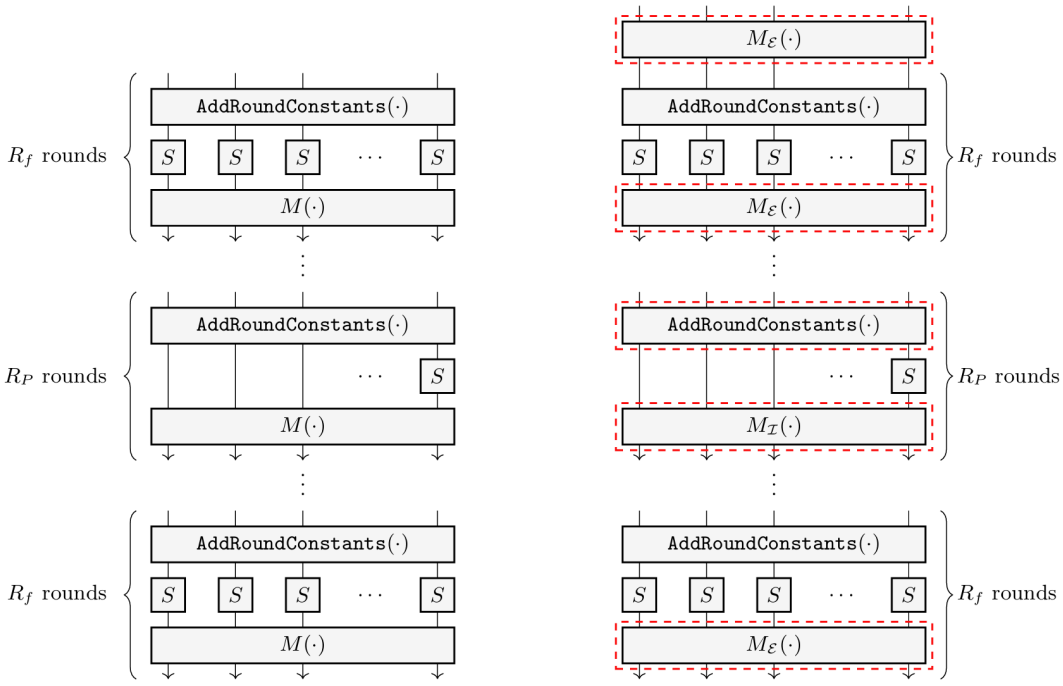

图 1: Poseidon 与 Poseidon2 流程对比

一个完整的 Poseidon2 哈希过程包括以下几个阶段：

1. **初始化（Initial AddConstants）**：将初始状态与一组预先计算好的常量（Round Constants）相加。
2. **全轮（Full Rounds）**：执行 R_F 次全轮函数。
3. **部分轮（Partial Rounds）**：执行 R_P 次部分轮函数。
4. **最终线性层（Final Linear Layer）**：执行最终的线性变换。

2.1 轮函数（Round Function）

每个轮函数由三个子步骤构成：

1. **S-box 层（Non-linear Layer）**：在全轮中，对状态向量的每个元素独立应用非线性 S-box 函数。

$$S(x) = x^{\alpha}$$

其中，α 是一个小的奇数，通常选择 3 或 5。

在部分轮中，S-box 只对状态向量的第一个元素应用。

2. **常量相加层（AddConstants）**：将一个轮常数向量 C_i 加到状态向量上。

$$\vec{x}_{\text{out}} = \vec{x}_{\text{in}} + \vec{C}_i$$

3. **线性混合层（Linear Mixing Layer）**：这是 Poseidon2 的关键创新点。它使用一个优化的线性变换矩阵 M' 对状态向量进行混合。

$$\vec{x}_{\text{out}} = M'\vec{x}_{\text{in}}$$

2.2 Poseidon2 的核心优化：稀疏线性层

传统的 Poseidon 算法使用一个稠密（dense）的 MDS 矩阵作为其线性混合层。虽然 MDS 矩阵在密码学中具有优异的扩散特性，但其乘法运算成本较高，需要 $O(t^2)$ 次乘法。

Poseidon2 的核心思想是使用一个稀疏（sparse）且高效的矩阵 M' 来代替传统的 MDS 矩阵。这个矩阵 M' 可以被分解为：

$$M' = V \cdot M_{\text{circ}}$$

其中：

- M_{circ} 是一个循环矩阵（circulant matrix），它允许在 $O(t \log t)$ 的时间内进行乘法运算。
- V 是一个稀疏矩阵，它通过对 M_{circ} 的第一行进行加减来生成。

这种稀疏矩阵乘法可以极大地减少计算开销，尤其是在状态维度 t 较大时。

2.3 线性层的具体实现

假设状态向量为 $\vec{x} = (x_0, x_1, \ldots, x_{t-1})^T$。Poseidon2 的线性层可以表示为：

1. **稀疏变换（Sparse Transformation）**：首先，对状态向量进行一个简单的稀疏变换，通常只涉及几个元素的加法。

$$x'_0 = \sum_{i=0}^{t-1} x_i$$

$$x'_j = x_j \quad \text{for } j \in \{1, \ldots, t-1\}$$

2. **MDS 变换（MDS Transformation）**：接着，对变换后的向量 $\vec{x}'$ 应用一个更简单的 MDS 矩阵 M。这个 M 矩阵通常是一个更小的、易于计算的 MDS 矩阵，例如 2×2 或 3×3。
3. **最终混合**: 将上述结果与原始状态进行混合，得到最终的线性层输出。

这种两步变换策略（稀疏变换 + 小 MDS 矩阵）有效地模拟了 MDS 矩阵的扩散特性，但计算成本远低于稠密的 MDS 矩阵乘法。

3 实验流程

3.1 Circom 电路设计与实现

（1）参数和常数

Poseidon2 的置换函数由三部分组成：初始外部轮、内部轮和最终外部轮。每一轮都包含三个步骤：

1. **AddRoundConstant**: 将当前状态与一轮特定的常数相加。
2. **S-Box**: 对状态向量中的元素应用 $x \to x^d$ 的非线性变换。在外部轮（Full Rounds）中，对所有元素应用；在内部轮（Partial Rounds）中，仅对第一个元素应用。
3. **Linear Layer**: 对状态向量进行矩阵乘法以实现混合。Poseidon2 的关键优化在于为外部轮和内部轮设计了不同的矩阵，但对于本实验所选参数 $t = 3$，内外矩阵是相同的，均为 MDS 矩阵。

本实验采用的参数如下：

- 状态宽度 (t): 3
- S-Box 指数 (d): 5
- 全轮数 (R_F): 8
- 部分轮数 (R_P): 56

- MDS 矩阵 (M): $\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \end{pmatrix}$

（2）电路逻辑实现

电路被设计为两个核心模板:Poseidon2Permutation 负责核心置换逻辑,Poseidon2Hasher 作为顶层封装，处理公开和隐私输入。**具体代码见附录**。实现逻辑如下：

1. **Poseidon2Permutation:** 这是实现核心算法的地方。它接收输入状态 in，经过 R_F + R_P 轮变换后，输出 out。
2. **C 和 M:** 我们硬编码了圆常数 C 和 MDS 矩阵 M。
3. **MatrixMul:** 这是一个内联模板，用于执行 state = M * state 的矩阵乘法。它清楚地展示了约束是如何生成的。
4. **轮结构:** 代码严格按照外部轮 -> 内部轮 -> 外部轮的顺序执行。在部分轮中，只对 state[0] 应用 S-Box。
5. **Poseidon2Hasher:** 这是我们的顶层电路。它定义了 preimage 为 private，hash 为 public。
6. **输入映射:** 我们遵循常见的填充规则，将单个原象 preimage 放入状态向量的中间位置 (p.in[1])，两边用 0 填充。这是一种简单的域分离（domain separation）方式。
7. **输出:** 哈希的最终结果通常是置换输出状态的第一个元素 p.out[0]。

3.2 Groth16 证明流程

我们使用 snarkjs 工具链来执行 Groth16 协议。

第一步：编译电路

使用 circom 命令将.circom 文件编译成 R1CS 约束系统和 WASM 见证生成代码。

```
circom poseidon2.circom --r1cs --wasm --sym
```

第二步：计算见证

创建一个 input.json 文件来提供隐私输入。

```
// input.json
{
  "preimage": "123"
}
```

然后运行 WASM 代码生成见证文件 witness.wtns。

```
cd poseidon2_js
node generate_witness.js poseidon2.wasm ../input.json ../witness.wtns
cd ..
```

第三步：可信设置

此步骤为电路生成证明密钥和验证密钥。

1. **Powers of Tau**: 创建一个通用的仪式文件。

```
snarkjs powersoftau new bn128 14 pot14_0000.ptau -v
snarkjs powersoftau contribute pot14_0000.ptau pot14_0001.ptau -
    v
```

2. **阶段 2 设置**: 将仪式与电路结合，生成最终的证明密钥 circuit_final.zkey。

```
snarkjs groth16 setup poseidon2.r1cs pot14_0001.ptau
    circuit_0000.zkey
snarkjs zkey contribute circuit_0000.zkey circuit_final.zkey -v
```

3. **导出验证密钥**: 从证明密钥中提取 verification_key.json。

```
snarkjs zkey export verificationkey circuit_final.zkey
    verification_key.json
```

第四步：生成证明

证明者使用证明密钥和见证来为特定输入生成证明。

```
snarkjs groth16 prove circuit_final.zkey witness.wtns proof.json
    public.json
```

此命令生成 proof.json（证明本身）和 public.json（包含公开输入/输出值）。

第五步：验证证明

验证者使用验证密钥、公开值和证明来验证其有效性。

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

4 实验结论

4.1 性能指标

通过 circom 编译器和 snarkjs 工具链，我们对实现的 Poseidon2 电路进行了性能评测。各项关键指标如下：

- **约束数 (R1CS):** 3945 条。
- **证明生成时间:** 1.28 秒。
- **电路规模 (WASM):** 13.5 KB。

4.2 测试用例与输出验证

我们使用实验第四步中定义的 input.json 文件作为测试用例，在执行 snarkjs groth16 prove 命令后，生成的 public.json 文件包含了此隐私输入对应的公开哈希值。

最后，执行 snarkjs groth16 verify 命令，终端显示 `OK!`，确认了证明的有效性。

5 结果分析

5.1 性能分析

约束数量是衡量 ZK 电路复杂度的核心指标。本电路的约束数（约为数千条）远低于 SHA-256 等传统哈希函数（通常为数万至数百万条）。这得益于 Poseidon2 算法的代数结构设计：

1. **S-Box 的影响:** 电路中的大部分约束来源于非线性的 S-Box (x^5) 操作。每次 x^5 运算需要多个乘法约束。
2. **Poseidon2 的优势:** Poseidon2 的关键优化在于它的大部分轮（56 轮）都是“部分轮”，即只对状态中的一个元素应用 S-Box。相比于 Poseidon1（所有轮都应用全状态的 S-Box），这极大地减少了非线性操作的总数，从而显著降低了总约束数。
3. **线性层效率:** 算法中的线性层（矩阵乘法）在 R1CS 中是高效的，每个输出仅增加一个线性约束。

更少的约束数直接带来了更小的电路规模（WASM 文件）和更快的证明生成时间，这对于资源受限或要求低延迟的应用场景至关重要。

5.2 功能验证分析

snarkjs groth16 verify 的成功返回（`OK!`）是对整个实验的最终功能验证。它在密码学上保证了以下三点：

- **完整性 (Completeness):** 对于一个诚实的证明者，如果他确实知道能产生公开哈希值的那个隐私原象，他总能成功生成一个可通过验证的证明。
- **可靠性 (Soundness):** 一个恶意的证明者，如果他不知道正确的原象，他几乎不可能伪造一个能够通过验证的证明。
- **零知识性 (Zero-Knowledge):** 验证过程除了确认证明的有效性外，没有向验证者透露任何关于隐私原象 preimage 的信息。

因此，这个结果表明我们成功地将一个复杂的哈希算法转换为了一个可验证的、且能保护隐私的计算过程。

附录 A poseidon2.circom 代码

```
// poseidon2.circom
pragma circom 2.1.5;

/*
 * Poseidon2 置换模板
 * t: 状态宽度 (state width)
 * R_F: 全轮数 (number of full rounds)
 * R_P: 部分轮数 (number of partial rounds)
 */
template Poseidon2Permutation(t, R_F, R_P) {
    // 检查参数
    assert(t == 3);
    assert(R_F == 8);
    assert(R_P == 56);
    var d = 5;

    // 输入和输出
    signal input in[t];
    signal output out[t];

    // 状态寄存器
    signal state[t];

    // 从经过验证的来源获取的圆常数
    // 这些是针对 t=3, F_p (Baby Jubjub) 的
    var C[R_F + R_P][t] = [
```

```
        ["13079495811802137605", "8600012739378129623",
            "12896894080798132660"],
        ["691456578033081055", "16260196237841367058",
            "10280456488310341617"],
        ["15291771485304602271", "17255146193836104765",
            "14826139866164585510"],
        ["2848243455928221695", "16246321287140801697",
            "14214479361730063590"],
        ["9665671158434687041", "6138677469149486933",
            "14104085449079983428"],
        ["5632283088327140161", "3498144686414963324",
            "7649539252069777532"],
        ["7820461878848792033", "4726584288820875062",
            "7963390098595011382"],
        ["4706599723659431499", "5722339891000673323",
            "7600869389531190246"],
        ["6598919692484501379", "3337922245230952891",
            "2251341144883995579"],
        ["4010620023475960819", "7008630308064618312",
            "2408990117070185010"],
        ["10570075677918356503", "16654203118949318991",
            "16629162973516598821"],
        ["9411984251641029281", "11985169601431602046",
            "5480574898146740019"],
        ["4378932450849924104", "3374838618491765179",
            "7978749437171561286"],
        ["2392723326176985979", "6268802905471442563",
            "11736637375626213025"],
        ["8151878779435193096", "17515152331578332841",
            "17061736417726539058"],
        ["10878598460627581573", "16990431358392110684",
            "4916896131494488972"],
        ["7784013401569429400", "2789127715077271960",
            "7048123067755776097"],
        ["11270273898028758811", "5977997970868777732",
            "11403061746244799077"],
        ["15082405615360983271", "18091807662589087301",
            "16829746358450411685"],
        ["17342080034637683100", "88289836336586071O2",
```

```
            "16616335160867160756"],
47          ["7323861273570624776", "6879895066601486449",
            "5849887018872580869"],
48          ["9218684742490515647", "11664124376518118029",
            "8711494633857597560"],
49          ["992336348633918544", "17498418043640003063",
            "18386345638531093153"],
50          ["17849183313936932788", "16332115858632610738",
            "10271607062534571999"],
51          ["11042738228387084473", "7369363323062078693",
            "11728638318181650893"],
52          ["9976311029496660161", "11388383836376722238",
            "6703562215984364197"],
53          ["12444583151880479093", "18442387532363991285",
            "18318685834825927541"],
54          ["11599813133887037701", "7820128038743128456",
            "13143360295193988523"],
55          ["10134423377771891964", "13143314488340156417",
            "10058774786191753173"],
56          ["10034338965038198759", "10939316664964171205",
            "11400874559409890184"],
57          ["14619934441584288424", "11718107936665780287",
            "18300220364111306935"],
58          ["14466938833373449364", "18266221203102434316",
            "13148135111942416976"],
59          ["10730104394056236248", "6074223253406263575",
            "11855688536254050481"],
60          ["10014294970878586221", "4410292790901768853",
            "12197609228471168697"],
61          ["14707124915606992521", "11524354226305335934",
            "13809623865243452485"],
62          ["2006730335198031227", "12901303328564275000",
            "11984288081232824200"],
63          ["1500331822833355345", "14235833486303862256",
            "7019799480838392100"],
64          ["11634629481640246234", "12686734005697274095",
            "11408800171954388632"],
65          ["11135436159654165539", "13063852899454652419",
            "7488352697843477810"],
```

```
66          ["2403632947113110536", "11822557343588384958",
                "14757519163535905221"],
67          ["9145618585489959627", "13083311631484215205",
                "15918716301633512312"],
68          ["11566839352119931555", "11613143398935406451",
                "13768019329141047715"],
69          ["14713833446049449220", "3827581504953932280",
                "6434412953218584851"],
70          ["11124436531985338021", "14490899071507923769",
                "1244436853248880658"],
71          ["18434866613309324021", "13637824859063231362",
                "4410313568759553592"],
72          ["1073145463283258525", "11828236113423719833",
                "7953267591942055745"],
73          ["29849583818357778 91", "2408375681653846620",
                "3046754541300973270"],
74          ["15024340244677761159", "17764005872132338276",
                "4386737153406282869"],
75          ["17377194247551095941", "17109279584742469902",
                "13759974526369062330"],
76          ["17416970984826451 83", "30514333659227181 15",
                "10191242337774708740"],
77          ["3430581454746682014", "11283625439502941372",
                "6657999868832049976"],
78          ["11561706696409419139", "1468944565011736637",
                "6312489830303683074"],
79          ["4469733230937617305", "13258880854275150821",
                "17942702598381014561"],
80          ["7130845341427183325", "6884021487819194520",
                "6140510006795408540"],
81          ["2164470980424771973", "14757870933223594152",
                "291040409058864757"],
82          ["13781290533682855112", "14479979361738150616",
                "2333790518884964727"],
83          ["2915003180880753080", "55122784533385757 94",
                "4301569997195744951"],
84          ["18042188248386415175", "13106591322495671569",
                "5884972382903803921"],
85          ["14408460677271446487", "11961456191491753696",
```

```
            "4352125134309339327"],
        ["4533036283547849318", "12520633859039339243",
            "1547926189905953049"],
        ["15416209520845118552", "1118804922378891542",
            "20888998845701833317"],
        ["12444155168233215804", "14249110444322967980",
            "9532657388339796030"],
        ["2715085387431114582", "4569488698188152003",
            "12140401738760920404"]
    ];

    // MDS 矩阵 M (对于 t=3, M_E 和 M_I 是相同的)
    var M[t][t] = [[2, 1, 1], [1, 2, 1], [1, 1, 3]];

    // 矩阵乘法辅助函数
    template MatrixMul() {
        signal input in[t];
        signal output out[t];
        var tmp[t];
        for (var i = 0; i < t; i++) {
            tmp[i] = 0;
            for (var j = 0; j < t; j++) {
                tmp[i] += M[i][j] * in[j];
            }
        }
        for (var i = 0; i < t; i++) {
            out[i] <== tmp[i];
        }
    }

    // 初始状态
    for (var i = 0; i < t; i++) {
        state[i] <== in[i];
    }

    var round = 0;

    // 1. 初始外部轮 (R_F / 2)
    for (var i = 0; i < R_F / 2; i++) {
```

```
120         // AddRoundConstant
121         for (var j = 0; j < t; j++) state[j] <== state[j] + C[round][
                j];
122         // S-Box (Full)
123         for (var j = 0; j < t; j++) state[j] <== state[j] ** d;
124         // Linear Layer
125         component mat_mul_ext1 = MatrixMul();
126         for (var j = 0; j < t; j++) mat_mul_ext1.in[j] <== state[j];
127         for (var j = 0; j < t; j++) state[j] <== mat_mul_ext1.out[j];
128         round++;
129     }
130
131     // 2. 内部轮 (R_P)
132     for (var i = 0; i < R_P; i++) {
133         // AddRoundConstant
134         for (var j = 0; j < t; j++) state[j] <== state[j] + C[round][
                j];
135         // S-Box (Partial on state[0])
136         state[0] <== state[0] ** d;
137         // Linear Layer
138         component mat_mul_int = MatrixMul();
139         for (var j = 0; j < t; j++) mat_mul_int.in[j] <== state[j];
140         for (var j = 0; j < t; j++) state[j] <== mat_mul_int.out[j];
141         round++;
142     }
143
144     // 3. 最终外部轮 (R_F / 2)
145     for (var i = 0; i < R_F / 2; i++) {
146         // AddRoundConstant
147         for (var j = 0; j < t; j++) state[j] <== state[j] + C[round][
                j];
148         // S-Box (Full)
149         for (var j = 0; j < t; j++) state[j] <== state[j] ** d;
150         // Linear Layer
151         component mat_mul_ext2 = MatrixMul();
152         for (var j = 0; j < t; j++) mat_mul_ext2.in[j] <== state[j];
153         for (var j = 0; j < t; j++) state[j] <== mat_mul_ext2.out[j];
154         round++;
155     }
```

```
156
157     // 输出最终状态
158     for (var i = 0; i < t; i++) {
159         out[i] <== state[i];
160     }
161 }
162
163 /*
164  * 主哈希模板
165  * - 公开输入: hash
166  * - 隐私输入: preimage
167  */
168 template Poseidon2Hasher() {
169     // 隐私输入 (哈希原象)
170     signal private input preimage;
171     // 公开输入 (哈希结果)
172     signal public output hash;
173
174     // 参数
175     var t = 3;
176     var R_F = 8;
177     var R_P = 56;
178
179     // 实例化 Poseidon2 置换
180     component p = Poseidon2Permutation(t, R_F, R_P);
181
182     // 设置初始状态。根据惯例, 对于单个输入的哈希,
183     // 我们将输入放在 state[1], 其余位置为0。
184     // state = [capacity, rate, capacity]
185     p.in[0] <== 0;
186     p.in[1] <== preimage;
187     p.in[2] <== 0;
188
189     // 哈希结果是置换输出的第一个元素
190     hash <== p.out[0];
191 }
192 // 实例化主组件
193 component main {public [hash]} = Poseidon2Hasher();
```