Poseidon2 Circom 数学原理：

一、Poseidon2 哈希函数设计背景

Poseidon2 是 Poseidon 哈希函数的第二代改进版本，专为 零知识证明友好 (ZK-friendly) 而设计。设计目标包括：

- 在 R1CS/PLONK 电路中具有低约束成本
- 提供抗预映像、抗碰撞、抗差分等加密安全性
- 使用 低指数 S-Box（通常 d=5 ） 以降低电路非线性复杂度
- 高扩散性（使用 MDS 矩阵），防止局部输入控制全局输出

Poseidon2 采用 SPN（Substitution–Permutation Network）结构，其设计受传统分组密码启发，并适配现代证明系统。

---

二、参数详解

| 参数 | 含义 | 示例值 |
| --- | --- | --- |
| p | 素域 F_p 的素数模数 | BN254: p = 218882428718392752... |
| t | 状态大小 | 3 |
| r | rate：用于哈希输入的槽数 r = t - c | 2 |
| c | capacity（安全余量）槽数 | 1 |
| d | S-Box 的幂指数，推荐使用 d=5 | 5 |
| R_f | Full rounds（所有 S-Box） | 8 |
| R_p | Partial rounds（部分 S-Box） | 通常依赖安全性要求 |
| M | MDS 矩阵（t × t） | 满秩广义 Hadamard 矩阵 |
| C | 每轮的加常数矩阵 C_{r,i} | 来自 Grain LFSR 生成器 |

说明：

- Poseidon2 不同于 Poseidon，移除了 sparse MDS，在 partial round 仍使用 full MDS，但仅一个元素做非线性（节省约束）；
- 常见实例如 (t=3,d=5,R_f=8,R_p=22) 。

---

三、Poseidon2 哈希轮结构推导

整个过程从状态 x = (x_0, x_1, ⋯., x_t-1) 开始，每一轮执行以下三步：

1. Add-Round Constants（加轮常数）

每一轮 r，每个状态位 x_i 加上常数：

$$x_i \leftarrow x_i + C_{r,i}$$

这些常数来自于伪随机流，如 Grain LFSR。保证轮之间差异化、防止代数攻击。

---

2. S-Box 非线性变换

在 Full round 中：对所有状态 x_i 执行：

$$x_i \leftarrow x_i^d = x_i^5$$

在 Partial round 中：仅对第一个状态执行 S-Box：

$$x_0 \leftarrow x_0^d$$

目的在于 增加代数非线性度，抵抗差分线性攻击。

---

3. MDS Mixing（MDS 矩阵乘法）

令 M 为 t \times t MDS 矩阵（最大距离可分矩阵），则状态向量更新为：

$$\mathbf{x} \leftarrow M \cdot \mathbf{x}$$

MDS 确保输入中任何一位变化，都会扩散影响到所有状态位，提供高度混淆。

---

四、Circom 中对应映射

以下是这些数学操作在 Circom 电路中的逐一映射：

1. 加常数：

stateWithRC[r][i] <== states[r][i] + rc.C[r][i];

2. 幂运算（S-Box）：

使用自定义的 SBox 模块：

sq <== in * in;

quad <== sq * sq;

out <== in * quad; // x^5

注意 Circom 中无法直接写 x**5，需拆解为乘法组合。

3. MDS：

手动写出：

out[0] <== m00*x0 + m01*x1 + m02*x2;

out[1] <== m10*x0 + m11*x1 + m12*x2;

out[2] <== m20*x0 + m21*x1 + m22*x2;

或使用模板：

component mds[r] = MDS3();

for (var i = 0; i < t; i++) {

    mds[r].in[i] <== sbox[r][i].out;

}

---

## 五、输入输出规范

- 输入数据格式：

对于 Poseidon2，状态大小为 t=3 ：

- 前两个：私有输入：in[0], in[1]
- 第三个：capacity 初始设为 0
- 输出：仅输出 out[0]，即 rate 第一个位（最常用于哈希）

- Circom 输入约定：

signal input in[2];  // 私密原像

signal input hash_expected[1];  // 公开哈希值（约束之）

- 输出验证：

circom

复制编辑

assert(states[last][0] == hash_expected[0]); // 实际输出是否等于目标哈希

---

六、零知识证明系统中的使用方式

定义电路后，配合 Groth16 执行：

1. 编译电路（.circom → .r1cs, .wasm）
2. 计算可信参数（powers of tau, .zkey）
3. 提供 witness 输入（input.json → witness.wtns）
4. 生成证明（.proof）
5. 验证证明（比对 public input）

证明目标：

证明：我知道某个输入 in，使得 Poseidon2(in) == hash_expected

这使得 Poseidon2 可用于：

- 零知识资产证明（Zcash、Privacy Pools）
- 匿名投票
- 信任系统中的身份哈希

---

七、参数如何选取

选取依据 NIST 安全级别：

| 安全级别 | 推荐参数 | 示例 |
| --- | --- | --- |
| 80-bit | t=3, Rf=8, Rp=22 | zkSNARKs |
| 128-bit | t=5, Rf=8, Rp=33 | zkRollup |
| 256-bit | t=6, Rf=8, Rp=56 | 高安全应用 |

**注意： t 越大，每轮 MDS 成本越高，但可输入更多数据。**

**Poseidon2 Circom 实现思路**

**一、总体目标**

电路 Poseidon2_3 用来做如下验证：

给定私有输入 in[0], in[1]，以及公开输入 hash_expected[0]，判断是否满足：

Poseidon2(in[0], in[1]) == hash_expected[0]

电路会在哈希过程的最后使用 assert(...) 来确保哈希结果匹配目标值。

---

**二、主要组成模块分析**

**1. 常量 RoundConstants**

```
template RoundConstants() {
    signal output C[8][3];
    ...
}
```

- 该模块定义了 8 轮每轮使用的 t=3 个常数。
- 每一轮的状态值在加 S-Box 前都会加上对应常量：stateWithRC[r][i] = states[r][i] + rc.C[r][i]

**2. MDS 矩阵混合 MDS3**

```
template MDS3() {
```

signal input in[3];

signal output out[3];

...

}

- 实现了一个 3×3 MDS 矩阵乘法（具体数值为 Poseidon2 的预设常数矩阵）。
- 保证状态的非线性扩散，防止碰撞。

---

**3. S-Box 非线性变换**

template QuinticSBox() {

signal input in;

signal output out;

...

}

- 实现 x^5（即 d=5）次幂变换，这是 Poseidon2 指定的 S-box 类型。
- 通过 x → x^2 → x^4 → x^5 的链式操作构建，逻辑高效。

---

**4. 主电路 Poseidon2_3**

template Poseidon2_3() { ... }

**输入输出定义：**

signal input in[2]; // 私有输入，2 个 field 元素

signal input hash_expected[1]; // 公开输入，目标哈希

signal output isValid;

**初始化状态：**

states[0][0] <== in[0];

states[0][1] <== in[1];

states[0][2] <== 0; // capacity 元素

- 输入 [x, y] 和初始 capacity 0 拼成 [x, y, 0]，作为第 0 轮初始状态。

**轮函数：**

- **8 轮迭代处理**（可扩展为 full_round + partial_round 格式）：

```
for (var r = 0; r < 8; r++) {
    // 加常数
    stateWithRC[r][i] <== states[r][i] + rc.C[r][i];

    // S-box: x → x^5
    sbox[i][r].in <== stateWithRC[r][i];

    // MDS: 状态混合
    mds[r].in[i] <== sbox[i][r].out;

    // 更新下一状态
    states[r+1][i] <== mds[r].out[i];
}
```

**输出验证：**

```
assert(states[8][0] == hash_expected[0]);
```

- 用 assert 断言最终状态的第一个元素等于目标哈希，失败则电路验证不通过。

**验证标志：**

```
isValid <== 1;
```

- 恒定输出 1，用于 proof 中做组合判断。

**代码说明：**

**根据官方给出的 python 代码生成轮常数和 mds 矩阵，保存在 round_constants.circom 中**

**测试方式：**

运行 run.bat

```
# 编译电路
circom circuits/poseidon2.circom --r1cs --wasm --sym -o build

# Trusted setup
snarkjs powersoftau new bn128 12 pot12_0000.ptau -v
snarkjs powersoftau prepare phase2 pot12_0000.ptau zkeys/pot12_final.ptau
snarkjs groth16 setup build/poseidon2.r1cs zkeys/pot12_final.ptau zkeys/poseidon2_0000.zkey
snarkjs zkey contribute zkeys/poseidon2_0000.zkey zkeys/poseidon2_final.zkey --name="Contributor"
snarkjs zkey export verificationkey zkeys/poseidon2_final.zkey zkeys/verification_key.json
# 生成 witness
node build/poseidon2_js/generate_witness.js build/poseidon2_js/poseidon2.wasm input/input.json proofs/witness.wtns
# 生成证明
snarkjs groth16 prove zkeys/poseidon2_final.zkey proofs/witness.wtns proofs/proof.json proofs/public.json

 #证明
snarkjs groth16 verify zkeys/verification_key.json proofs/public.json proofs/proof.json
```

结果

```
C:\Users\2555\Desktop\test>circom circuits/poseidon2.circom --r1cs --wasm --sym -o build
template instances: 4
non-linear constraints: 72
linear constraints: 49
public inputs: 0
private inputs: 3 (2 belong to witness)
public outputs: 1
wires: 148
labels: 224
Written successfully: build\poseidon2.r1cs
Written successfully: build\poseidon2.sym
Written successfully: build\poseidon2_js\poseidon2.wasm
Everything went okay

C:\Users\2555\Desktop\test># Trusted setup
'#' 不是内部或外部命令，也不是可运行的程序
或批处理文件。

C:\Users\2555\Desktop\test>snarkjs powersoftau new bn128 12 pot12_0000.ptau -v
[DEBUG] snarkJS: Calculating First Challenge Hash
[DEBUG] snarkJS: Calculate Initial Hash: tauG1
[DEBUG] snarkJS: Calculate Initial Hash: tauG2
[DEBUG] snarkJS: Calculate Initial Hash: alphaTauG1
[DEBUG] snarkJS: Calculate Initial Hash: betaTauG1
[DEBUG] snarkJS: Blank Contribution Hash:
                786a02f7 42015903 c6c6fd85 2552d272
                912f4740 e1584761 8a86e217 f71f5419
                d25e1031 afee5853 13896444 934eb04b
                903a685b 1448b755 d56f701a fe9be2ce
[INFO]  snarkJS: First Contribution Hash:
                9e63a5f6 2b96538d aaed2372 481920d1
                a40b9195 9ea38ef9 f5f6a303 3b886516
                0710d067 c09d0961 5f928ea5 17bcdf49
                ad75abd2 c8340b40 0e3b18e9 68b4ffef
PS C:\Users\2555\Desktop\test>
```