

# 区块链基础及应用 2025 Exercise 6

在这个实验里，你会学到

- `circom`, 一个描述算术电路的工具
- `snarkjs`, 一种用于生成和验证电路满意度的 zk-SNARKs 的工具。

本实验建议使用 Ubuntu 20.04 以上的版本运行。

你将使用以下知识来探索私有事务（private transactions）的实现：

- 制作一个简单的版本的花费 Tornado 的电路，和
- 生成赎回 Tornado 的有效性证明。

## 1 安装程序

原始版本的实验需要配置 `github` 环境并在 `github` 上下载大量代码，此操作成功率不高。为方便同学们进行实验，本次实验将所需依赖库下载并打包进实验文件夹中。同学们直接安装即可。经测试本操作在 Ubuntu 20.04 版本下运行正常，Ubuntu 18.04 版本下会报错。

1.配置 `ubuntu` 上的 `github` 环境

[https://blog.csdn.net/weixin\\_41011452/article/details/133803139](https://blog.csdn.net/weixin_41011452/article/details/133803139)

2.安装 `nodejs` 和 `npm`.

3.安装 `snarkjs` (`sudo npm install -g snarkjs@0.1.11`)

4.安装 `circom` (`sudo npm install -g alex-ozdemir/circom#cs251`)

5.安装 `mocha test runner` (`sudo npm install -g mocha`)

6.在 `Ex6` 文件夹中运行 `npm install`

7.运行 `npm test`，并验证是否大多数测试失败。如果大部分测试失败、少部分测试通过，那么说明环境配置成功。如果全部失败，说明环境配置有问题。

## 2 了解 `circom`

项目文件夹中给出了 `TUTORIAL.md` 文件，该文件为本项目教程。然后，阅读 `circuits/example.circom` 中的电路示例。并回答 `artifacts/writeup.md` 中的相关问题。

应完成的内容：`artifacts/writeup.md`

之后根据你对 `circom` 和 `snarkjs` 的理解，使用 `SmallOddFactorization` 电路为

$7 \times 17 \times 19 = 2261$  创建一个证明。并将验证密钥（verifier key）保存到 artifacts/verifier\_key\_factor.json 中，将证明保存到 artifacts/proof\_factor.json 中。

应完成的内容：

artifacts/verifier\_key\_factor.json, artifacts/proof\_factor.json

## 3 开关电路

### 3.1 IfThenElse

IfThenElse 电路（位于 circuits/spend.circom）验证了条件表达式的正确求值。

它有 1 个输出，和 3 个输入：

condition: 应该是 0 或 1

true\_value: 如果 condition 是 1，那么输出 true\_value

false\_value: 如果 condition 是 0，那么输出 false\_value

IfThenElse 需要设置额外的条件保证 condition 为 0 或 1。请实现 IfThenElse。

应完成的内容：circuits/spend.circom 中 IfThenElse 函数

### 3.2 SelectiveSwitch

SelectiveSwitch 接受 2 个输入并产生 2 个输出，如果第三个输入为 1，则打开顺序。

利用 IfThenElse 电路来实现 SelectiveSwitch 电路。

应完成的内容：circuits/spend.circom 中 SelectiveSwitch 函数

## 4 消费电路

用户有一对(nullifier, nonce)，这一对定义一枚硬币

$\text{coin} = H(\text{nullifier}, \text{nonce})$

其中， $H$  是一个哈希函数。每枚这样的硬币占据 Merkle 树里面的一片叶子。第一枚硬币被放在 Merkle 树的最左边的叶子上，每一枚新硬币都被立即放在前一枚硬币的右边的叶子上。

当取出硬币时，硬币所有者使用它的(nullifier, nonce)来在不透露  $\text{coin} = H(\text{nullifier}, \text{nonce})$

是哪片叶子的情况下证明其是 Merkle 树的一片叶子。

在这个任务中，你将做的是制作一个算术电路来验证一个(nullifier, nonce)对应于 Merkle 树中的一个硬币。然后，你将公开显示 nullifier(允许每个人验证这个 nullifier 还没有被使用)，并且使用 SNARK 来证明 nonce 的存在使得对应的硬币在零知识的 Merkle 树中。电路的输入如下：

- digest: Merkle tree 根节点摘要（公开），
- nullifier: the nullifier（公开），
- nonce: the nonce（私有），
- Merkle path: a list of (direction, hash) pairs（私有）。

电路应该验证  $H(\text{nullifier}, \text{nonce})$  是 Merkle 树中的一片叶子，其根节点哈希值是所提供的摘要 (digest)。具体而言，电路应验证所提供的 (私有) Merkle 路径是硬币  $H(\text{nullifier}, \text{nonce})$  的有效 Merkle 证明。

对于用于确定硬币和 Merkle 树的哈希函数  $H$ ，使用哈希函数 Mimc2，它的电路已包含在文件中。你应该使用你的选择开关电路来正确地处理 direction。

应完成的内容：circuits/spend.circom 中 Spend 函数

## 5 计算花费电路的输入

下面的任务是编写一个程序，计算给定的 nullifier/coin 的 Merkle 路径。

通过在 src/compute\_spend\_input.js 中实现 computeInput 函数来实现。此函数接受以下输入：

- depth: Merkle 树的深度。
- coins: 一个硬币的清单。有些是由你创建的，并且是由两个元素组成的数组 (nullifier 和 nonce)。其他的并不是你创造的，而是一个单一的价值——coin。
- nullifier: 计算电路输入的 nullifier。

该函数应该返回一个适合作为 Spend 电路输入的 JSON 对象。

为了帮助你，我们提供了 SparseMerkleTree 类，你可以在 src/sparse\_merkle\_tree.js 中找

到。对于承诺哈希函数，请使用文件中包含的 Mimc2。

应完成的内容：`src/compute_spend_input.js`

## 6 痞回证明

最后，使用 `circom` 和 `snarkjs` 创建一个 SNARK 用来证明深度为 10 的 Merkle 树中存在与 `test/compute_spend_input/transcript3.txt` 相对应的 nullifier“10137284576094”。使用深度为 10（你将在 `test/circuits/spend10.circom` 中找到 Spend 电路的 depth-10 实例化），并将你的验证密钥放在 `artifacts/verifier_key_spend.json` 中，将你的证明放在 `artifacts/proof_spend.json` 中。

应完成的内容：`artifacts/verifier_key_spend.json`, `artifacts/proof_spend.json`

## 7 测试

当然，你可以用 `snarkjs` 来检查你的证明。

我们还为系统的各个组件提供了一些单元测试，它们可以使用 `npm test` 来运行。

## 8 调试提示

你的 `circom` 版本支持 `log` (1 参数) 函数，它将打印其参数。

## 9 提交内容

①代码文件：将除了 `node_modules` 文件夹之外的所有文件、文件夹打包为一个压缩文件提交。

②简要的实验报告：包含程序核心代码、实验结果截图和代码的简要说明。