circom 和 snarkjs 教程

本教程将指导您创建第一个零知识 zkSnark 电路，涵盖编写电路的各种技术，并展示如何在链下和以太坊链上创建证明并验证它们。

---

1. 安装工具

1.1 前置条件

如果尚未安装，您需要安装 `Node.js`。

使用最新的稳定版 `Node.js`（如 8.12.0）就足够了，但如果您安装最新版本（如 10.12.0），性能会显著提高。因为新的版本原生支持大整数（Big Integer）库，而 `snarkjs` 会利用该功能提升大约 10 倍的性能。

1.2 安装 circom 和 snarkjs

运行以下命令：

```
shCopy codenpm install -g circom
npm install -g snarkjs
```

---

2. 使用电路

我们将创建一个电路来证明您可以分解一个数字的因数。

2.1 在新目录下创建电路

1. 创建一个名为 `factor` 的空目录，用于存放本教程的所有文件：

   ```
   shCopy codemkdir factor
   cd factor
   ```

   > 在实际项目中，您可能希望创建一个包含 `circuits`（存放电路）和 `test`（存放测试及脚本）的 Git 仓库。

2. 创建名为 `circuit.circom` 的新文件，并添加以下内容：

   ```
   circomCopy codetemplate Multiplier() {
       signal private input a;
       signal private input b;
       signal output c;

       c <== a * b;
   }

   component main = Multiplier();
   ```

   这个电路有两个私有输入信号 `a` 和 `b`，以及一个输出信号 `c`。

电路的功能是强制信号 `c` 等于 `a * b` 的值。

声明完 `Multiplier` 模板后，我们用一个名为 `main` 的组件实例化它。

注意：编译电路时，必须存在一个名为 `main` 的组件。

2.2 编译电路

现在我们可以编译电路，运行以下命令：

```
sh


Copy code
circom circuit.circom -o circuit.json
```

这会将电路编译为 `circuit.json` 文件。

---

3. 使用 `snarkjs` 操作编译后的电路

电路编译完成后，我们继续使用 `snarkjs`。
您可以通过运行以下命令查看帮助信息：

```
sh


Copy code
snarkjs --help
```

3.1 查看电路的信息和统计数据

运行以下命令查看电路的总体统计信息：

```
sh


Copy code
snarkjs info -c circuit.json
```

您也可以打印电路的约束信息：

```
sh


Copy code
snarkjs printconstraints -c circuit.json
```

3.2 使用 `snarkjs` 进行设置

为电路设置参数：

```
sh


Copy code
snarkjs setup
```

> 默认情况下，`snarkjs` 会使用 `circuit.json`。您也可以通过 `-c <文件名>` 指定不同的电路文件。

设置过程将生成两个文件：`proving_key.json` 和 `verification_key.json`。

3.3 计算 Witness

在创建任何证明之前，需要计算符合电路所有约束的信号。

`snarkjs` 可以根据您提供的输入文件计算所有信号，包括中间信号和输出信号。这些信号的集合称为 *witness*。

零知识证明的核心在于证明您知道一组满足约束的信号，而无需泄露这些信号（除了公开的输入和输出）。

例如，假设您想证明您知道如何分解数字 33。这意味着您知道两个数字 `a` 和 `b`，使得它们相乘结果为 33。

> 您可以简单地使用两个相同的数字作为 `a` 和 `b`，我们稍后会处理这个问题。

要证明您知道 3 和 11，可以创建一个名为 `input.json` 的文件：

```
json


Copy code
{"a": 3, "b": 11}
```

接着，计算 witness：

```
sh


Copy code
snarkjs calculatewitness
```

查看生成的 `witness.json` 文件以了解所有信号。

创建证明

生成 witness 后，我们可以创建证明：

```
sh


Copy code
snarkjs proof
```

此命令会默认使用 `proving_key.json` 和 `witness.json` 文件生成 `proof.json` 和 `public.json`。

- `proof.json` 文件包含实际的证明。
- `public.json` 文件包含公开输入和输出的值。

验证证明

运行以下命令验证证明：

```
sh


Copy code
snarkjs verify
```

此命令会使用 `verification_key.json`、`proof.json` 和 `public.json` 验证证明是否有效。

验证成功会显示 `OK`，验证失败会显示 `INVALID`。

生成 Solidity 验证器

```
sh


Copy code
snarkjs generateverifier
```

此命令会基于 `verification_key.json` 生成一个 Solidity 文件 `verifier.sol`。

您可以将 `verifier.sol` 中的代码复制到 Remix IDE 中。

合约中包含两个合约：`Pairings` 和 `Verifier`。您只需部署 `Verifier` 合约。

> 建议在测试网（如 Rinkeby、Kovan 或 Ropsten）上部署，或者使用 Remix 的 Javascript VM。

在链上验证证明

部署的验证器合约包含一个 `view` 函数 `verifyProof`，它会在证明和输入有效时返回 `true`。

为了方便调用，您可以使用以下命令生成调用参数：

```
sh


Copy code
snarkjs generatecall
```

将生成的输出复制并粘贴到 Remix 的 `verifyProof` 方法的参数字段中。如果一切正常，该方法应返回 `true`。

如果修改任意一个参数位，结果将验证为 `false`。

额外补充

可以通过增加约束，修改电路使其不接受 `1` 作为输入：

```
circomCopy codetemplate Multiplier() {
    signal private input a;
    signal private input b;
    signal output c;
    signal inva;
    signal invb;

    inva <-- 1 / (a - 1);
    (a - 1) * inva === 1;

    invb <-- 1 / (b - 1);
    (b - 1) * invb === 1;

    c <== a * b;
}

component main = Multiplier();
```

`<--` 和 `===` 是 Circom 的特性，用于分离值赋予和约束操作。

后续学习

- 阅读 circom 的 README 了解更多功能。
- 查看基本电路库 circomlib。

Enjoy zero-knowledge proving!