Lab6 zkSNARK

2112209 张佳璐 计算机科学与技术

2112060 孙蕗 信息安全

了解circom

阅读TUTORIAL，学习circom，回答writeup.md中相关问题（详见 artifacts/writeup.md）

使用SmallOddFactorization电路为7×17×19=2261创建一个证明，证明过程截图如下：

```
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/circuits$ circom example.circom -o example.json
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/circuits$ snarkjs setup -c example.json
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/circuits$ snarkjs calculatewitness
Error: ENOENT: no such file or directory, open 'circuit.json'
    at Object.openSync (fs.js:443:3)
    at Object.readFileSync (fs.js:343:35)
    at Object.<anonymous> (/home/zjl/Ex6_包含全部依赖库/Ex6/snarkjs-0.1.11/cli.js:282:38)
    at Module._compile (internal/modules/cjs/loader.js:778:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:789:10)
    at Module.load (internal/modules/cjs/loader.js:653:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:593:12)
    at Function.Module._load (internal/modules/cjs/loader.js:585:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:831:12)
    at startup (internal/bootstrap/node.js:283:19)
ERROR: Error: ENOENT: no such file or directory, open 'circuit.json'
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/circuits$ snarkjs calculatewitness -c example.json
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/circuits$ snarkjs proof
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/circuits$ snarkjs verify
OK
```

验证密钥（verifier key）保存在artifacts/verifier_key_factor.json 中，证明保存在artifacts/proof_factor.json中。

核心代码部分

IFThenElse

根据输入的条件选择输出真值或假值。条件信号必须是0或1。模块通过约束条件确保了逻辑的正确性。

```
//约束条件：条件必须是0或1
    condition * (1 - condition) === 0;

    //中间信号值，约束形式为 ab + c = 0
    signal diff <-- true_value - false_value;

    //约束输出信号
    //
    // 条件为1
    //  out = 1 * (true_value - false_value) + false_value = true_value
    //
    // 条件为0
    //  out = 0 * (true_value - false_value) + false_value = false_value
    out <== condition * diff + false_value;
```

SelectiveSwitch

`SelectiveSwitch` 表示一个选择性开关。该开关有两个输入（`in0` 和 `in1`），一个选择信号（`s`），以及两个输出（`out0` 和 `out1`）。根据选择信号 `s` 的值，开关可以选择输出其中一个输入信号。

```
    // 约束条件：确保选择信号s为0或1
    s * (1 - s) === 0;

    // 使用两个 If 语句确定输出值。

    // 如果(s==1)则为in1，否则为in0
    component firstOutput = IfThenElse();
    firstOutput.condition <== s;
    firstOutput.true_value <== in1;
    firstOutput.false_value <== in0;

    // 如果（s==1）则为in0，否则为in1
    component secondOutput = IfThenElse();
    secondOutput.condition <== s;
    secondOutput.true_value <== in0;
    secondOutput.false_value <== in1;

    // 输出信号必须等于if语句的结果
    out0 <== firstOutput.out;
    out1 <== secondOutput.out;
```

消费电路

这一部分在一个名为 `Spend` 的模板，实现了一个 Merkle path 验证的逻辑。该逻辑用于验证一个给定深度（`depth`）的 Merkle 树中的路径是否有效。

```
template Spend(depth) {
    signal input digest;
    signal input nullifier;
    signal private input nonce;
    signal private input sibling[depth];
    signal private input direction[depth];

    // TODO
    //这个数组存储我们在每个层级计算的证明哈希，+1用于存储根节点。
    component computed_hash[depth + 1];

    //第0层只是H（'nullifier','digest'）
    computed_hash[0] = Mimc2();
    computed_hash[0].in0 <== nullifier;
    computed_hash[0].in1 <== nonce;

    //存储路径上的开关
    component switches[depth];

    //设置沿着证明路径的约束。
    for (var i = 0; i < depth; ++i) {
        switches[i] = SelectiveSwitch();
        //如果directions[i]为true，将计算H(sibling[i], computed_hash[i])。
        //如果为false，不交换，计算H(computed_hash[i], sibling[i])。
        switches[i].in0 <== computed_hash[i].out;
        switches[i].in1 <== sibling[i];
        switches[i].s <== direction[i];
```

```
        //计算下一层级的哈希。
        computed_hash[i + 1] = Mimc2();
        computed_hash[i + 1].in0 <== switches[i].out0;
        computed_hash[i + 1].in1 <== switches[i].out1;
    }

    //验证摘要是否匹配最终哈希。
    computed_hash[depth].out === digest;
}
```

使用了 Merkle 树的概念，通过计算哈希值逐层验证给定路径上的节点。具体而言，它使用了 `Mimc2` 哈希函数和 `SelectiveSwitch` 模块，以及一些辅助的计算哈希和选择性切换的数组，来构建 Merkle path 的验证逻辑。

计算花费电路的输入

在这段代码中，我们需要通过给定的 `transcript`（包含了 Merkle 树中某些节点的信息）和 `nullifier` 来生成一个验证对象。

```
function computeInput(depth, transcript, nullifier) {
    // TODO
    // 创建一个深度为 depth 的 SparseMerkleTree 实例
    const tree = new SparseMerkleTree(depth);

    //生成验证的给定空化器对应的承诺。
    let input_commitment, input_nonce = [null, null];

    //将transcript中的信息添加到Merkle树中。
    for (let i = 0; i < transcript.length; i++) {
      const commitment_or_info = transcript[i];
      let commitment = null;
      if (commitment_or_info.length == 1) {
        commitment = commitment_or_info[0];
      } else if (commitment_or_info.length == 2) {
        const [t_nullifier, nonce] = commitment_or_info;
        commitment = mimc2(t_nullifier, nonce);
        if (nullifier == t_nullifier) {
          if (input_commitment != null) {
            throw "不应重复，出现问题";
          }
          [input_commitment, input_nonce] = [commitment, nonce];
        }
      } else {
        throw "无效 + " + str(transcript);
      }
      if (commitment == null) {
        throw "承诺为空";
      }
      //将commitment插入Merkle树中
      tree.insert(commitment);
    }

    //获取我们的项目的证明。
    if (input_commitment == null) {
```

```
        throw "nullifier not found in our transcript";
      }
      const path = tree.path(input_commitment);
      //构建输出对象
      const output = {
        digest: tree.digest,
        nullifier: nullifier,
        nonce: input_nonce,
      };
      for (let i = 0; i < depth; i++) {
        let [s, d] = path[i];
        output['sibling[' + i + ']'] = s.toString();
        output['direction[' + i + ']'] = (d) ? "1" : "0";
      }
      return output;
  }
```

在代码中，首先创建了一个指定深度的 `SparseMerkleTree` 实例，并通过遍历 `transcript` 中的信息来逐步构建 Merkle 树。然后，它找到与给定 `nullifier` 相关的承诺，并获取与之相关的 Merkle 路径证明。最后，函数返回一个包含 Merkle 树根哈希、nullifier、随机数、兄弟节点和方向信息的对象。

赎回证明

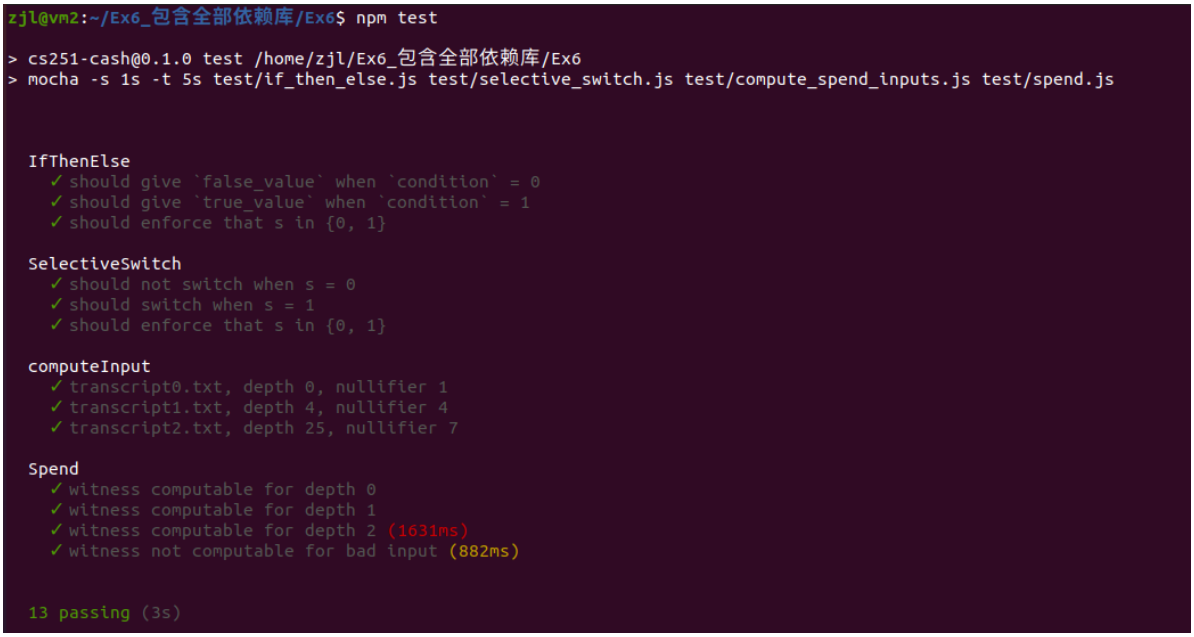
使用circom和snarkjs创建一个 SNARK 用来证明深度为10的Merkle树中存在与 test/compute_spend_input/transcript3.txt 相对应的 nullifier“10137284576094”。使用深度为10（你将在 test/circircuits/spend10.circom 中找到 Spend 电路的 depth-10 实例化）。

首先，需要先通过src/compute_spend_input.js生成一个input.json，将它用于transcript3的测试。对test/circircuits/spend10.circom，进行电路的编译，并将其传递给snarkjs。为电路运行setup，再利用之前生成好的input.json计算见证。最后创建证明，验证证明。具体流程截图如下

```
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/test/circuits$ circom spend10.circom -o spend10.json
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/test/circuits$ snarkjs setup spend10.json
Error: ENOENT: no such file or directory, open 'circuit.json'
    at Object.openSync (fs.js:443:3)
    at Object.readFileSync (fs.js:343:35)
    at Object.<anonymous> (/home/zjl/Ex6_包含全部依赖库/Ex6/snarkjs-0.1.11/cli.js:272:38)
    at Module._compile (internal/modules/cjs/loader.js:778:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:789:10)
    at Module.load (internal/modules/cjs/loader.js:653:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:593:12)
    at Function.Module._load (internal/modules/cjs/loader.js:585:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:831:12)
    at startup (internal/bootstrap/node.js:283:19)
ERROR: Error: ENOENT: no such file or directory, open 'circuit.json'
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/test/circuits$ snarkjs setup -c spend10.json
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/test/circuits$ snarkjs calculatewitness -c spend10.json
Error: ENOENT: no such file or directory, open 'input.json'
    at Object.openSync (fs.js:443:3)
    at Object.readFileSync (fs.js:343:35)
    at Object.<anonymous> (/home/zjl/Ex6_包含全部依赖库/Ex6/snarkjs-0.1.11/cli.js:284:56)
    at Module._compile (internal/modules/cjs/loader.js:778:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:789:10)
    at Module.load (internal/modules/cjs/loader.js:653:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:593:12)
    at Function.Module._load (internal/modules/cjs/loader.js:585:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:831:12)
    at startup (internal/bootstrap/node.js:283:19)
ERROR: Error: ENOENT: no such file or directory, open 'input.json'
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/test/circuits$ snarkjs calculatewitness -c spend10.json
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/test/circuits$ snarkjs proof
zjl@vm2:~/Ex6_包含全部依赖库/Ex6/test/circuits$ snarkjs verify
OK
```

测试

使用npm test来进行测试



发现所有的节点都已经将通过