XCS251 Assignment 6: Using SNARKs

Due Sunday, November 6 at 11:59pm PT.

Guidelines

1. If you have a question about this assignment, we encourage you to post your question on slack channel.
2. Familiarize yourself with the collaboration and honor code policy before starting work.

Submission Instructions

You should submit a PDF with your solutions online in Gradescope. As long as the PDF is legible and organized, the course staff has no preference between a handwritten and a typeset LaTeX submission. If you wish to typeset your submission and are new to LaTeX, you can get started with the following:

- Download and install Tex Live or try Overleaf.
- Submit the compiled PDF.

Honor code

We strongly encourage students to form study groups. Students may discuss and work on assignment in groups. However, each student must write down the solutions independently, and without referring to written notes from the joint session. In other words, each student must understand the solution well enough in order to reconstruct it by him/herself. In addition, each student should write on the assignment the set of people with whom s/he collaborated. Further, because we occasionally reuse problem set questions from previous years, we expect students not to copy, refer to, or look at the solutions in preparing their answers. It is an honor code violation to intentionally refer to a previous year's solutions. More information regarding the Stanford honor code can be found at https://communitystandards.stanford.edu/policies-and-guidance/honor-code.

Introduction

In this project, you'll learn about circom, a tool for describing arithmetic circuits, and snarkjs, a tool for generating and verifying zk-SNARKs of circuit satisfaction. You'll use this knowledge to explore the implementation of private transactions by crafting a simple version of the Tornado spend circuit and generating a proof of validity for a Tornado withdrawal. Details on the instructions of setup the environment for the started code, how to learn circom, and deliverables will be provided in the handout. By the end of this assignment, you should have a better understanding of zk-SNARKs and Tornado cash.

1 Setup

To get your environment set up, do the following:

1. Install `nodejs` and `npm`. Note that you should have installed `nodejs` and `npm` during Assignment 4.
2. Install `snarkjs` (`npm install -g snarkjs@0.1.11`). Note that you can ignore the warning of vulnerabilities during install.
3. Install our fork of `circom` (`npm install -g alex-ozdemir/circom#cs251`). Note that you can ignore the warning of vulnerabilities during install.
4. Install the `mocha test runner` (`npm install -g mocha`).
5. Download and unzip the starter code.
6. Run `npm install` within the `Starter Code` folder. Make sure that inside the `node_modules` folder, you have a folder called `snarkjs` and a folder called `circom` after the install.
7. Run `npm test` and verify that most of the tests fail, but not because of missing dependencies. (Note if tests fail because of dependency issues, you might be using an incompatible version of `snarkjs`.) Please use version `0.1.11` as specified in the package.json file. You can use `npm list -g snarkjs` to check `snarkjs` version.

2 Learning About circom

First, follow the Iden3 circom tutorial at https://iden3-docs.readthedocs.io/en/latest/iden3_repos/circom/TUTORIAL.html#circom-and-snarkjs-tutorial. Skip section "1.1 Pre-requisites" and "1.2 Install circom and snarkjs".

We have already installed `nodejs`, `npm`, our fork of `circom` and specific version of `snarkjs`. Also note that you can always specify a different circuit file by adding `-c <circuit JSON file name>` for the command `snarkjs calculatewitness` and `snarkjs setup` if you choose to use a name different from `circuit.json` for the circuit file. You can stop after the "Verifying the proof" section.

Then, read our example circuits in `circuits/example.circom` and answer the questions in `artifacts/writeup.md`. You can find the introduction of `circom` language here. You may find it useful to understand the keywords `template`, `component`, and `signal`. In short, `template` is the mechanism to create generic circuits in `circom`. The instantiation of a `template` is made using the keyword `component` and by providing the necessary parameters. The arithmetic circuits built using `circom` operate on `signals` like `signal input` and `signal output`. You may also find the introduction on "constraints generation" useful. See https://docs.circom.io/circom-language/constraint-generation/.

Deliverable: `artifacts/writeup.md`

Then, demonstrate your knowledge of `circom` and `snarkjs` by creating a proof that $7 \times 17 \times 19 = 2261$ (using the `SmallOddFactors` circuit).

Store the verifier key in `artifacts/verification_key_factor.json` and the proof in `artifacts/proof_factor.json`.

Deliverable: `artifacts/verification_key_factor.json, artifacts/proof_factor.json`

3 A Switching Circuit

3.1 IfThenElse

The `IfThenElse` circuit (located in `circuits/spend.circom`) verifies the correct evaluation of a conditional expression. It has a single output, `out` and 3 inputs:

- `condition`, which should be 0 or 1,
- `true_value`, which `out` will be equal to if condition is 1, and
- `false_value`, which `out` will be equal to if condition is 0.

`IfThenElse` additionally enforces that `condition` is indeed 0 or 1.

Implement `IfThenElse`.

Deliverable: `IfThenElse` in file `circuits/spend.circom`

3.2 SelectiveSwitch

The `SelectiveSwitch` takes two inputs and produces two outputs, flipping the order if a third input is 1.

Implement `SelectiveSwitch`, making use of your `IfThenElse` circuit.

Deliverable: `SelectiveSwitch` in file `circuits/spend.circom`

4 A Spend Circuit

We're going to implement a simplified Tornado withdrawal as discussed in class. The user has a pair `(nullifier, nonce)` and this pair defines a coin as

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

where H is a hash function. Each such coin occupies one leaf of a Merkle tree. The first coin is placed in the left-most leaf of the Merkle tree, and every new coin is placed in the leaf immediately to the right of the previous coin.

When withdrawing a coin, the coin owner uses its `(nullifier, nonce)` pair to prove to the Tornado contract that the corresponding `coin` $= H$`(nullifier, nonce)` is one of the leaves of the Merkle tree, without revealing which one.

What you will do in this assignment is craft an arithmetic circuit for verifying that a `(nullifier, nonce)` pair corresponds to a coin in the Merkle tree. You will then reveal the nullifier publicly (allowing everyone to verify that this nullifier hasn't been spent already), and use a SNARK to prove the existence of a nonce such that the corresponding coin is in the Merkle tree, in zero-knowledge. The inputs to the circuit are thus:

- `digest`: the Merkle tree root digest (public),
- `nullifier`: the nullifier (public),
- `nonce`: the nonce (private), and
- Merkle path: a list of (direction, hash) pairs (private)

The circuit should verify that `coin` $= H$`(nullifier, nonce)` is a leaf in a Merkle tree whose root hash is the provided digest. Specifically, the circuit should verify that the provided (private) Merkle path is a valid Merkle proof for the `coin` $= H$`(nullifier, nonce)`.

You should implement the verification circuit, `Spend, in circuits/spend.circom`.

For the hash function H used to define the coin and the Merkle tree, use the hash function `Mimc2` whose circuit has been included into the file. You should make use of your `SelectiveSwitch` circuit to handle the `directions` properly.

Deliverable: `Spend` in file `circuits/spend.circom`

5 Computing the Spend Circuit Input

The only task that remains is writing a program that computes the Merkle path for a given nullifier/coin.

Implement this by implementing the `computeInput` function in `src/compute_spend_inputs.js`. This function takes the following inputs:

- `depth`: the depth of the Merkle tree
- `coins`: a list of coins. Some are created by you, and are an array of two elements (the `nullifier` and `nonce`). Others were not created by you and are an array of a single value—the `coin`.
- `nullifier`: the nullifier to compute the circuit inputs for.

The function should return a `JSON` object suitable as input to the `Spend` circuit.

To assist you, we've provided a `SparseMerkleTree` class, which you will find in `src/sparse_merkle_tree.js`. For the commitment hash-function, use `mimc2`, which has been included in the file.

Deliverable: `src/compute_spend_input.js`

6 Proving a Withdrawal

Finally, use your input computer, `circom`, and `snarkjs` to create a SNARK proving the presence of the `nullifier` "10137284576094" in Merkle tree of depth 10 corresponding to the transcript `test/compute_spend_input/transcript3.txt`. Use a depth of 10 (you'll find a depth-10 instantiation of your `Spend` circuit in `test/circuits/spend10.circom`), and place your verifier key in `artifacts/verification_key_spend.json` and your proof in `artifacts/proof_spend.json`.

Deliverable: `artifacts/verification_key_spend.json, artifacts/proof_spend.json`

7 Testing

You can, of course, check your proofs using `snarkjs`.

We've also provided a few unit tests for the various components of your system, which can be run using `npm test`.

8 FAQ and tips

1. If you use linux during setup, you may need to use `sudo` in front of the command to have root privileges to install.
2. If you use Apple M1 computer, you may have an issue with a missing git package. You can install "Xcode-select" developer tools, and after that, all packages should be installed for you to run the test.
3. Your version of `circom` supports the `log` (1 argument) function, which prints its argument.
4. If you encountered this error "ERROR: Error: Invalid signal identifier:" when running `calculatewitness`, you may want to check that the variable names in your input file match the input signals to your circuit.
5. If you have "before all" hook errors, then this maybe because your `snarkjs` is not properly installed. However, not all "before all" hook errors will be due to not having `snarkjs` properly installed. Having errors in your code can also cause this error. If your `snarkjs` version looks all good, it could be an issue with changes you've made to your code.
6. You should not use `if and else` in `IfThenElse` circuit. We want you to practice writing rank-1 constraints using `circom`, which becomes trivial if `if-else` is used.
7. A common question from students in the past is the difference between `<==` and `<--` followed by `===`. You can find detailed explanations in the following discussion thread https://github.com/iden3/circom/issues/23.
8. If you have something along these lines:

```
for(var i = 0; i < num; i++)
{
component my_component = MyTemplate();
        /* do some computation */
}
```

You should change it to

```
component my_components[num];
for(var i = 0; i < num; i++)
{
  my_components[i] = MyTemplate();
  /* Do some computation */
}
```

Same for `signals`. The reason is that if you let the `components` go out of scope, then `circom` can't actually enforce the constraints. Hence the need to create an array of components instead.

9. Figure 1 is the visual summary of the `circom` and `snarkjs` procedure from https://docs.circom.io/

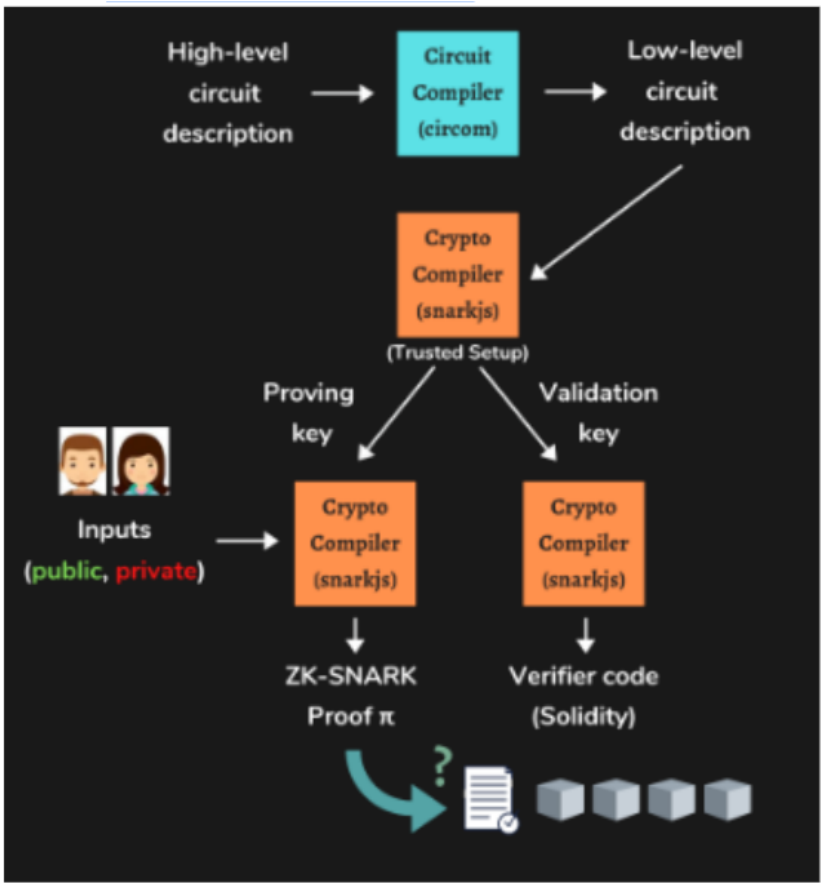

Figure 1:

10. In the past, students were often confused about the meaning of "rank-1" in "rank-1 constraint system (R1CS)" that is required by `circom` (see https://docs.circom.io/getting-started/writing-circuits/). It is probably easier to just accept that the definition of rank-1 constraint is an equation of the following form (see https://docs.circom.io/background/background/#signals-of-a-circuit):

    $(a_1 * s_1 + ... + a_n * s_n) * (b_1 * s_1 + ... + b_n * s_n) + (c_1 * s_1 + ... + c_n * s_n) = 0$

    If you still want to know more, you can also check https://crypto.stackexchange.com/questions/67857/what-is-a-rank-1-constraint-system

9 Submission

Please upload a compressed file that contains all of your code files and write-up document to Gradescope. Do not include your npm modules directory for this project.