Computing our witness

What is a witness?

Before creating the proof, we need to calculate all the signals of the circuit that match all the constraints of the circuit. For that, we will use theWasm module generated bycircom that helps to do this job. It can also be done with theC++ code in a similar way (see below).

Let us start with theWasm code. Using the generatedWasm binary and three JavaScript files, we simply need to provide a file with the inputs and the module will execute the circuit and calculate all the intermediate signals and the output. The set of inputs, intermediate signals and output is calledwitness .

In our case, we want to prove that we are able to factor the number 33. So, we assigna = 3 andb = 11 .

Note that we could assign the number 1 to one of the inputs and the number 33 to the other. So, our proof does not really show that we are able to factor the number 33.

We need to create a file namedinput.json containing the inputs written in the standard json format.

We use strings instead of numbers because JavaScript does not work accurately with integers larger than 253 .

{"a": "3", "b": "11"} Now, we calculate the witness and generate a binary filewitness.wtns containing it in a format accepted bysnarkjs .

After calling thecircom compiler with the flag--wasm and the circuitmultiplier2.circom we can find amultiplier2_js folder that contains theWasm code in multiplier2.wasm and all the neededJavaScript files.

Computing the witness with WebAssembly

Enter in the directorymultiplier2_js , add the input in a fileinput.json and execute:

node generate_witness.js multiplier2.wasm input.json witness.wtns

Computing the witness with C++

As a faster alternative, we can use the C++ directory to compute the witness using the previous fileinput.json . This directory is created when using thecircom compiler with the flag--c . In our example, the compiler creates amultiplier2_cpp folder that contains all theC++ code needed to compute the witness and a Makefile to easily generate the corresponding executable program.

To do so, enter the directorymultiplier2_cpp and execute:

make The previous command creates an executable calledmultiplier2 .

Note. To compile the C++ source, we rely on some libraries that you need to have installed in your system. In particular, we usenlohmann-json3-dev ,libgmp-dev andnasm .

After the executable is created, we execute it indicating the input file and the name for the witness file:

./multiplier2 input.json witness.wtns

The Witness file

The two programs will generate the samewitness.wtns file. This file is encoded in a binary format compatible withsnarkjs , which is the tool that we use to create the actual proofs.

Note. For big circuits, the C++ witness calculator is significantly faster than the WASM calculator.

g++ -pthread -o circuit-512-32-256-64 -I ../../Fr -I ../../ ../../main.cpp ../../Fr/fr.o ../../Fr/fr.cpp ../../calcwit.cpp ../../utils.cpp circuit-512-32-256-64.cpp -lgmp -O3

g++ -pthread -o circuit-512-32-256-64 -I Fr main.cpp Fr/fr.o Fr/fr.cpp calcwit.cpp utils.cpp circuit-512-32-256-64.cpp -lgmp -O3

g++ -pthread -o circuit-512-32-256-64 main.cpp fr.o fr.cpp calcwit.cpp utils.cpp circuit-512-32-256-64.cpp -lgmp -O3

To run the previous lines we can use make. But first need to install some dependencies:

sudo apt install libgmp3-dev nasm

./aliascheck_test

Usage: ./aliascheck_test

wtns is a binary format for the witness.

fr.asm is a field arithmetic in assembly. fr_asm.o is the compiled with nasm of the previous one.

fr.cpp is the c++ that includes the previous program.