

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 the `Wasm` module generated by `circom` that helps to do this job. It can also be done with the `C++` code in a similar way (see below).

Let us start with the `Wasm` code. Using the generated `Wasm` 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 called [witness](#).

In our case, we want to prove that we are able to factor the number 33. So, we assign $a = 3$ and $b = 11$.

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 named `input.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 file `witness.wtns` containing it in a format accepted by `snarkjs`.

After calling the `circom` compiler with the flag `--wasm` and the circuit `multiplier2.circom` we can find a `multiplier2_js` folder that contains the `Wasm` code in `multiplier2.wasm` and all the needed JavaScript files.

Computing the witness with WebAssembly

Enter in the directory `multiplier2_js`, add the input in a file `input.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 file `input.json`. This directory is created when using the `circom` compiler with the flag `--c`. In our example, the compiler creates a `multiplier2_cpp` folder that contains all the `C++` code needed to compute the witness and a `Makefile` to easily generate the corresponding executable program.

To do so, enter the directory `multiplier2_cpp` and execute:

`make` The previous command creates an executable called `multiplier2`.

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

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 same `witness.wtns` file. This file is encoded in a binary format compatible with `snarkjs`, 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.