Compiling our circuit

Once you have the compiler installed you can see the available options as follows:

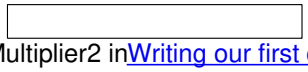
circom --help

circom compiler 2.1.7 IDEN3 Compiler for the Circom programming language

USAGE: circom [FLAGS] [OPTIONS] [input]

FLAGS: --r1cs Outputs the constraints in r1cs format --sym Outputs witness in sym format --wasm Compiles the circuit to wasm --json Outputs the constraints in json format --wat Compiles the circuit to wat -c, --c Compiles the circuit to c --O0 No simplification is applied --O1 Only applies signal to signal and signal to constant simplification --O2 Full constraint simplification --verbose Shows logs during compilation --inspect Does an additional check over the constraints produced --use_old_simplification_heuristics Applies the old version of the heuristics when performing linear simplification --simplification_substitution Outputs the substitution in the simplification phase in json format -h, --help Prints help information -V, --version Prints version information

OPTIONS: -o, --output Path to the directory where the output will be written [default: .] -p, --prime To choose the prime number to use to generate the circuit. Receives the name of the curve (bn128, bls12381, goldilocks, grumpkin, pallas, vesta) [default: bn128] -l ... Adds directory to library search path --O2round Maximum number of rounds of the simplification process

ARGS:  Path to a circuit with a main component [default: ./circuit.circom] We created a template calledMultiplier2 inWriting our first circuit . However, to actually create a circuit, we have to create an instance of this template. To do so, create a file with the following content:

pragma circom 2.0.0;

template Multiplier2() { signal input a; signal input b; signal output c; c <== a*b; }

component main = Multiplier2(); After we write our arithmetic circuit usingcircom , we should save it in a file with the.circom extension. Remember that you can create your own circuits or use the templates from our library of circuitscircomlib .

In our example, we create a file calledmultiplier2.circom . Now is time to compile the circuit to get a system of arithmetic equations representing it. As a result of the compilation we will also obtain programs to compute the witness. We can compile the circuit with the following command:

circom multiplier2.circom --r1cs --wasm --sym --c With these options we generate three types of files:

- --r1cs
- : it generates the filemultiplier2.r1cs
- that contains theR1CS constraint system
- of the circuit in binary format.
- --wasm
- : it generates the directorymultiplier2_js
- that contains theWasm
- code (multiplier2.wasm) and other files needed to generate thewitness
- .
- --sym
- : it generates the filemultiplier2.sym
- , a symbols file required for debugging or for printing the constraint system in an annotated mode.
- --c
- : it generates the directorymultiplier2_cpp
- that contains several files (multiplier2.cpp, multiplier2.dat, and other common files for every compiled program like main.cpp, MakeFile, etc) needed to compile the C code to generate the witness.

We can use the option-o to specify the directory where these files are created.

Since version 2.0.8, we can use the option-l to indicate the directory where the directiveinclude should look for the circuits indicated.