

## Part 2

### 1. Proof of circom installation;

```
~/week1V2/week1/Q2 git:(master) (0.065s)
circom --help
circom compiler 2.0.4
IDEN3
Compiler for the circom programming language

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

FLAGS:
--verbose Shows logs during compilation
-h, --help Prints help information
--inspect Does an additional check over the constraints produced
--O0 No simplification is applied
-c, --c Compiles the circuit to c
--json Outputs the constraints in json format
--rics Outputs the constraints in rics format
--sym Outputs the constraints in sym format
--wasm Compiles the circuit to wasm
--wat Compiles the circuit to wat
--O1 Only applies var to var and var to constant simplification
-V, --version Prints version information

OPTIONS:
--O2 <full_simplification> Full constraint simplification [default: full]
-o, --output <output> Path to the directory where the output will be written [default: .]

ARGS:
<input> Path to a circuit with a main component [default: ./circuit.circom]
```

### 2. Running the bash script (editing with chmod to give appropriate permissions, will PR this to include it for future cohorts);

```
~/week1V2/week1/Q2 git:(master) (0.081s)
./scripts/compile-Helloworld.sh
zsh: permission denied: ./scripts/compile-Helloworld.sh

~/week1V2/week1/Q2 git:(master) (0.073s)
chmod 777 ./scripts/compile-Helloworld.sh
```

### Result output (2 pictures);

```
~/week1V2/week1/Q2 git:(master) (4.647s)
./scripts/compile-Helloworld.sh
chmod 777 ./scripts/compile-Helloworld.sh

Downloading powersOfTau28_hez_final_10.ptau
2022-05-09 13:52:14 - https://hermez.s3-eu-west-1.amazonaws.com/powersOfTau28_hez_final_10.ptau
Resolving hermez.s3-eu-west-1.amazonaws.com (hermez.s3-eu-west-1.amazonaws.com)... 52.218.88.56
Connecting to hermez.s3-eu-west-1.amazonaws.com (hermez.s3-eu-west-1.amazonaws.com)|52.218.88.56|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1262744 (1.2M) [application/octet-stream]
Saving to: 'powersOfTau28_hez_final_10.ptau'
Saving: [=====] 100%[=====] 1.29M 4.54MB/s in 0.3s

powersOfTau28_hez_final_10.ptau          100%[=====] 1.29M 4.54MB/s in 0.3s

2022-05-09 13:52:14 (4.54 MB/s) - 'powersOfTau28_hez_final_10.ptau' saved [1262744/1262744]

Compiling HelloWorld.circ...
template instances: 1
non-linear constraints: 1
linear constraints: 0
public inputs: 0
public outputs: 1
private inputs: 2
private outputs: 0
wires: 4
Written successfully: HelloWorld/Helloworld.rics
Written successfully: HelloWorld/Helloworld.sym
Written successfully: HelloWorld/Helloworld_js/Helloworld.wasm
Everything went okay, circom safe
[INFO] snarkJS: Curve: bn-128
[INFO] snarkJS: # of Wires: 4
[INFO] snarkJS: # of Constraints: 1
[INFO] snarkJS: # of Private Inputs: 2
[INFO] snarkJS: # of Public Inputs: 0
[INFO] snarkJS: # of Labels: 4
[INFO] snarkJS: # of Outputs: 1
[INFO] snarkJS: Reading rics
[INFO] snarkJS: Reading tauG1
[INFO] snarkJS: Reading tauG2
[INFO] snarkJS: Reading alphatauG1
[INFO] snarkJS: Reading betatauG1

[INFO] snarkJS: Circuit hash:
4289918f b00cc352 b426119d 49059905
382c440e 3439767d b5835b9 6ce102b4
ea11be67 46375b98 b59f3f6e d5d5b730
122e33c7 65c9a1e1 5e9480aa cbba9b5d
[DEBUG] snarkJS: Applying key: H Section: 0/2
[DEBUG] snarkJS: Applying key: H Section: 0/4
[INFO] snarkJS: Circuit Hash:
4289918f b00cc352 b426119d 49059905
382c440e 3439767d b5835b9 6ce102b4
ea11be67 46375b98 b59f3f6e d5d5b730
122e33c7 65c9a1e1 5e9480aa cbba9b5d
[INFO] snarkJS: Contribution Hash:
b0abbff54 5b979263 43424de1 24df6824
33b1dbd6 d0b03902 c83defa1 d4ea9a962
9d3c27d8 9b17e817 c4669151 d5f569ac
c82e9ffc d49b9b2b fe5a020a 5eb0711c
```

2.

**1. What does the circuit in HelloWorld.circom do?**

This circuit generates a proof that checks that the product (multiplication) of two inputs (a & b) is equal to an output c. That is to say;

$$C == A * B$$

**2. Lines 7-12 of compile-HelloWorld.sh download a file called powersOfTau28\_hez\_final\_10.ptau for Phase 1 trusted setup. Read more about how this is generated [here](#). What is a Powers of Tau ceremony?**

**Explain why this is important in the setup of zk-SNARK applications.**

zkSNARKs rely on a CRS (common reference string) in order to verify and prove to make the SNARK function. This CRS is generated by people contributing randomly to it. This CRS has to be generated securely in a trusted setup to ensure people do not use the generation information to create false proofs. This security can be ensured by either identifying “trusted” participants ahead of time or by randomly requesting input from lots of parties. The second option, which adds more decentralization and is better for scaling is dictated by the MMORPG scheme. This generation is done in two phases, the first phase being a “powers of Tau ceremony”. The powers of Tau ceremony generates generic setup parameters that can be used across all circuits using this scheme. It is important for zk-SNARKs because it allows for a more efficient, secure and decentralized generation of the CRS.

**3. Line 24 of compile-HelloWorld.sh makes a random entropy contribution as a Phase 2 trusted setup. How are Phase 1 and Phase 2 trusted setup ceremonies different from each other?**

Phase 1 ceremonies are perpetual and generates the initial generic setup that can be used by any circuit, phase 2 is circuit specific (each circuit will have a unique phase 2) where participants will take turns to add to a the CRS (adding randomness) and the file will be passed along through a co-ordinator to different participants so that everyone can contribute without having direct communication between any 2 contributors to ensure each participants contributions are kept as hidden as possible.

3.

- 1. In the empty scripts/compile-Multiplier3-groth16.sh, create a script to compile contracts/circuits/Multiplier3.circom and create a verifier contract modeling after compile-HelloWorld.sh.**

Bash needs to be gain additional permission to execute

```
~/week1V2/week1/Q2 git:(master) (0.06s)
chmod 777 ./scripts/compile-Multiplier3-groth16.sh

~/week1V2/week1/Q2 git:(master)
```

Running the bash to try and compile the circuit;

```
~/week1V2/week1/Q2 git:(master) (0.062s)
./scripts/compile-Multiplier3-groth16.sh
powersoffau28_hez_final_10.ptau already exists. Skipping.
Compiling Multiplier3.circom...
error[T3001]: Non quadratic constraints are not allowed!
  "Multiplier3.circom":14:4
    14   d <= a * b * c;
          ^~~~~~ found here
      = call trace:
        ->Multiplier3

previous errors were found
[ERROR] snarkJS: [Error: ENOENT: no such file or directory, open 'Multiplier3/Multiplier3.r1cs'] {
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: 'Multiplier3/Multiplier3.r1cs'
}
[ERROR] snarkJS: [Error: ENOENT: no such file or directory, open 'Multiplier3/Multiplier3.r1cs'] {
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: 'Multiplier3/Multiplier3.r1cs'
}
[ERROR] snarkJS: [Error: ENOENT: no such file or directory, open 'Multiplier3/circuit_0000.zkey'] {
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: 'Multiplier3/circuit_0000.zkey'
}
[ERROR] snarkJS: [Error: ENOENT: no such file or directory, open 'Multiplier3/circuit_final.zkey'] {
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: 'Multiplier3/circuit_final.zkey'
}
[ERROR] snarkJS: [Error: ENOENT: no such file or directory, open 'Multiplier3/circuit_final.zkey'] {
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: 'Multiplier3/circuit_final.zkey'
```

- 2. Try to run compile-Multiplier3-groth16.sh. You should encounter an error with the circuit as is. Explain what the error means and how it arises.**

The error specifies that non-quadratic constraints are not allowed meaning that any function defined in the restraint has to have a complexity of at most a quadratic polynomial (2 unknowns multiplied together).

### 3. Modify Multiplier3.circom to perform a multiplication of three input signals under the restrictions of circom.

```
./scripts/compile-Multiplier3-groth16.s
mkdir: Multiplier3: File exists
powersOfTau28_hez_final_10.ptau already exists. Skipping.
Compiling Multiplier3.circom...
template instances: 2
non-linear constraints: 2
linear constraints: 0
public inputs: 0
public outputs: 1
private inputs: 3
private outputs: 0
wires: 6
labels: 11
Written successfully: Multiplier3/Multiplier3.r1cs
Written successfully: Multiplier3/Multiplier3.sym
Written successfully: Multiplier3/Multiplier3_js/Multiplier3.wasm
Everything went okay, circom safe
[INFO] snarkJS: Curve: bn-128
[INFO] snarkJS: # of Wires: 6
[INFO] snarkJS: # of Constraints: 2
[INFO] snarkJS: # of Private Inputs: 3
[INFO] snarkJS: # of Public Inputs: 0
[INFO] snarkJS: # of Labels: 11
[INFO] snarkJS: # of Outputs: 1
[INFO] snarkJS: Reading r1cs
[INFO] snarkJS: Reading tauG1
[INFO] snarkJS: Reading tauG2
[INFO] snarkJS: Reading alphatauG1
[INFO] snarkJS: Reading betatauG1
[INFO] snarkJS: Circuit hash:
d450100e 589c0685 76e9e3ce 2e0e71e6
f90f89fa 1f8a17ca d07b0ad4 4c044ecd
ff825273 9185368e 1ba6de7a 349f2472
2b974d2d 2dd40cc2 343a6a01 b6d826e6
[DEBUG] snarkJS: Applying key: L Section: 0/4
[DEBUG] snarkJS: Applying key: H Section: 0/4
[INFO] snarkJS: Circuit Hash:
d450100e 589c0685 76e9e3ce 2e0e71e6
f90f89fa 1f8a17ca d07b0ad4 4c044ecd
ff825273 9185368e 1ba6de7a 349f2472
2b974d2d 2dd40cc2 343a6a01 b6d826e6
[INFO] snarkJS: Contribution Hash:
ca309b79 97085ada 39dc51ca f6755aa6
5a66db15 667c554f b4025cc3 c6b8fd03
71c2d3b8 7f912cc2 0dd49d17 2f6a1f57
d8b20c55 8ccda60a ad81ea73 13c2002f
```

**4. In the empty scripts/compile-Multiplier3-plonk.sh, create a script to compile circuit/Multiplier3.circom using PLOUD in snarkjs. Add a \_plonk prefix to the build folder and the output contract to distinguish the two sets of output.**

1. You will encounter an error if you just change snarkjs groth16 setup to snarkjs plonk setup. Resolve this error and answer the following question - How is the process of compiling with PLOUD different from compiling with Groth16?

(with error)

```
~/zku/week1/Q2 git:(working-branch) (1.89s)
./scripts/compile-Multiplier3-plonk.sh
powersOfTau28_hez_final_10.ptau already exists. Skipping.
Compiling Multiplier3.circom with plonk...
template instances: 2
non-linear constraints: 2
linear constraints: 0
public inputs: 0
public outputs: 1
private inputs: 3
private outputs: 0
wires: 6
labels: 11
Written successfully: Multiplier3_plonk/Multiplier3.r1cs
Written successfully: Multiplier3_plonk/Multiplier3.sym
Written successfully: Multiplier3_plonk/Multiplier3_js/Multiplier3.wasm
Everything went okay, circom safe
[INFO] snarkJS: Curve: bn-128
[INFO] snarkJS: # of Wires: 6
[INFO] snarkJS: # of Constraints: 2
[INFO] snarkJS: # of Private Inputs: 3
[INFO] snarkJS: # of Public Inputs: 0
[INFO] snarkJS: # of Labels: 11
[INFO] snarkJS: # of Outputs: 1
[INFO] snarkJS: Reading r1cs
[INFO] snarkJS: Plonk constraints: 3
[INFO] snarkJS: Setup Finished
[ERROR] snarkJS: Error: zkey file is not groth16
    at phase2contribute (/opt/homebrew/lib/node_modules/snarkjs/build/cli.cjs:4881:15)
    at async c1Processor (/opt/homebrew/lib/node_modules/snarkjs/build/cli.cjs:304:27)
[ERROR] snarkJS: [Error: ENOENT: no such file or directory, open 'Multiplier3_plonk/circuit_final.zkey' ] {
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: 'Multiplier3_plonk/circuit_final.zkey'
}
[ERROR] snarkJS: [Error: ENOENT: no such file or directory, open 'Multiplier3_plonk/circuit_final.zkey' ] {
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: 'Multiplier3_plonk/circuit_final.zkey'
}
```

(without error)

```
~/zku/week1/Q2 git:(working-branch) (2.732s)
./scripts/compile-Multiplier3-plonk.sh
mkdir: Multiplier3_plonk: File exists
powersOfTau28_hez_final_10.ptau already exists. Skipping.
Compiling Multiplier3.circom with plonk...
template instances: 2
non-linear constraints: 2
public inputs: 0
public outputs: 1
private inputs: 3
private outputs: 0
wires: 6
labels: 11
Written successfully: Multiplier3_plonk/Multiplier3.r1cs
Written successfully: Multiplier3_plonk/Multiplier3.sym
Written successfully: Multiplier3_plonk/Multiplier3_js/Multiplier3.wasm
Everything went okay, circom safe
[INFO] snarkJS: Curve: bn-128
[INFO] snarkJS: # of Wires: 6
[INFO] snarkJS: # of Constraints: 2
[INFO] snarkJS: # of Private Inputs: 3
[INFO] snarkJS: # of Public Inputs: 0
[INFO] snarkJS: # of Labels: 11
[INFO] snarkJS: # of Outputs: 1
[INFO] snarkJS: Reading r1cs
[INFO] snarkJS: Plonk constraints: 3
[INFO] snarkJS: Setup Finished
```

2. **What are the practical differences between Groth16 and PONK? Hint: compare and contrast the resulted contracts and running time of unit tests (see Q5 below) from the two protocols.**

The difference between the plonk and groth16 CRS generation is that plonk CRS is universally across a set of circuits while groth16 is specific to the circuit that it is generated for. In addition, specific to our case, plonk does not require contribution and to fix the error what was needed was to remove the line that included the logic to contribute to the CRS. Groth16 also verifies more quickly than plonk.

## 5.

### 1&2&3

Running the command (this goes over 3 screenshots due to size)

```
npm run test

> test
> node scripts/bump-solidity.js && npx hardhat test

(node:4872) ExperimentalWarning: The Fetch API is an experimental feature. This feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
You are using a version of Node.js that is not supported by Hardhat, and it may work incorrectly, or not work at all.

Please, make sure you are using a supported version of Node.js.

To learn more about which versions of Node.js are supported go to https://hardhat.org/nodejs-versions
```

```
HelloWorld
x = 2
callData = ["0x1719899814297711233436649438912702165472954463302487039641257733748752995",
    "0x1348613068708854426790272397176850241961034893404240116940542862364589128424",
    "0x1259366293412964208051638231846934479110041618527254535485771562420405921",
    "0x8681181383841763958521787581628089149681346942829921511796928952146164184",
    "0x342587345634557489563623612297767789222996822696750251887285297128765459264211",
    "0x12948454891115516963505210524942415938405087353967241918064647910044540888143",
    "0x1053836663241015481212666800102729803924757608970294287828649357910255724468",
    "0x107882752044613082510498562179284216539653467663772814681101201465611265883",
    "0x2"]

a = [
    "0x1719899814297711233436649438912702165472954463302487039641257733748752995",
    "0x1348613068708854426790272397176850241961034893404240116940542862364589128424"
]

b = [
    "0x12593662934129642080516382318469344791100418527254535485771562420405921",
    "0x8681181383841763958521787581628089149681346942829921511796928952146164184",
    "0x342587345634557489563623612297767789222996822696750251887285297128765459264211",
    "0x12948454891115516963505210524942415938405087353967241918064647910044540888143"
]

c = [
    "0x1053836663241015481212666800102729803924757608970294287828649357910255724468",
    "0x107882752044613082510498562179284216539653467663772814681101201465611265883"
]

Input = [ '2' ]
    ✓ Should return true for correct proof (131ms)
    ✓ Should return false for invalid proof (176ms)
```

```
Multiplier3 with Groth16
x = 6
callData = ["0x10393627602855686177597634499169376916219885121217306978517315426320239318",
    "0x186439598416337576842624115806402927624256961639728318757923110574096876991",
    "0x186439598416337576842624115806402927624256961639728318757923110574096876991"]

b = [
    "0x10393627602855686177597634499169376916219885121217306978517315426320239318",
    "0x186439598416337576842624115806402927624256961639728318757923110574096876991",
    "0x186439598416337576842624115806402927624256961639728318757923110574096876991",
    "0x186439598416337576842624115806402927624256961639728318757923110574096876991",
    "0x186439598416337576842624115806402927624256961639728318757923110574096876991",
    "0x186439598416337576842624115806402927624256961639728318757923110574096876991",
    "0x186439598416337576842624115806402927624256961639728318757923110574096876991",
    "0x186439598416337576842624115806402927624256961639728318757923110574096876991"]

c = [
    "0x1027181245003956100533181282496732024258088532205644579774859703217482490333",
    "0x1027181245003956100533181282496732024258088532205644579774859703217482490333"]

Input = [ '6' ]
    ✓ Should return true for correct proof (88ms)
    ✓ Should return false for invalid proof (162ms)
```



3.

1. contracts/circuits/LessThan10.circom implements a circuit that verifies an input is less than 10 using the LessThan template. Study how the template is used in this circuit. What does the 32 in Line 9 stand for?

```
component lt = LessThan(32);

//documentation in the library for the lessThan function says;
// N is the number of bits the input have.
// The MSF is the sign bit.

//not super clear but seems that n is the number of bits n should have
//output appears to be 0 or 1 denoting false and true (if the number is less than the input or not) Yo
```

2. What are the possible outputs for the LessThan template and what do they mean respectively? (If you cannot figure this out by reading the code alone, feel free to compile the circuit and test with different input values.)

As dictated above, the possible outputs would be 0 or 1

### 3. Included in code (picture below);

```
template RangeProof(n) {
    assert(n <= 252);
    signal in; // this is the number to be proved inside the range
    signal range[2]; // the two elements should be the range, i.e. [lower bound, upper bound]
    signal output out;

    component low = LessEqThan(n);
    component high = GreaterEqThan(n);

    // [assignment] insert your code here
    low.in[0] <== in
    //low first input is the number we want to check is equal to or less than some number (in[1]) (so in for us)
    low.in[1] <== range[1]
    //next we input the number we want to check the input is less than (so if (input[0] <= input[1]) then it is true)
    //so for us we want to check that the input is less than our max range (or range[1])

    //same concept for these lines below just in reverse
    high.in[0] <== in
    high.in[1] <== range[0]

    //use the null rule to make an or statement (since anything multiplied by 0 is 0 both of these have to be 1 to return 1 (or true) if the
    low.in <== low.out * high.out      You, 1 minute ago * Uncommitted changes
}
```

2.

1. Done in github
2. The issue is that the number of constraints was too much for the current powers of tau file. Changing to another powers of tau fixes the issue
3. Done in github
4. The advantage is of the algorithmic implementation is that is more efficient and does less operations in order to verify that the solution is viable.

4.

Another good library to be created could be centered around proving that some model or output from a neural network did in fact come from using that neural network. This could be used to prove that data that is posted on chain but was processed off chain was processed properly and not just a random answer. This could allow for gas efficient and secure data processing on chain.