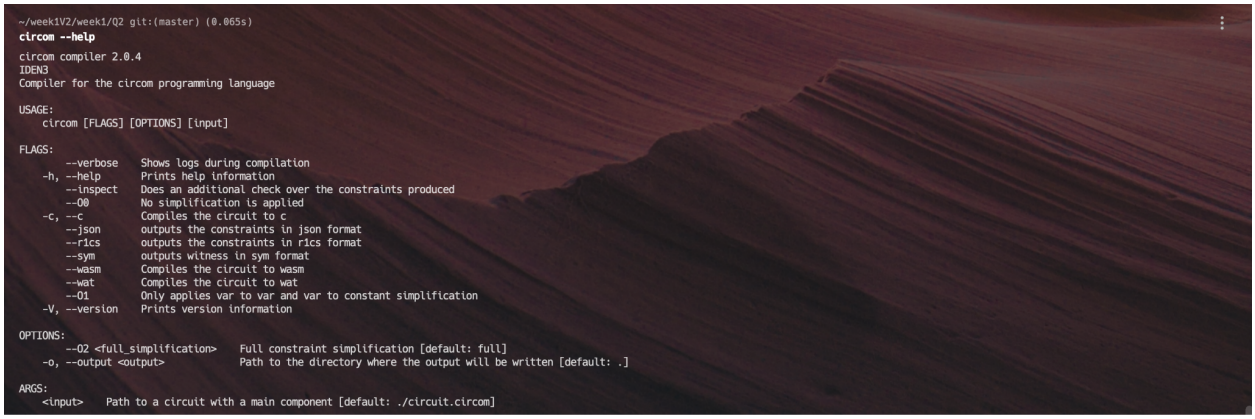
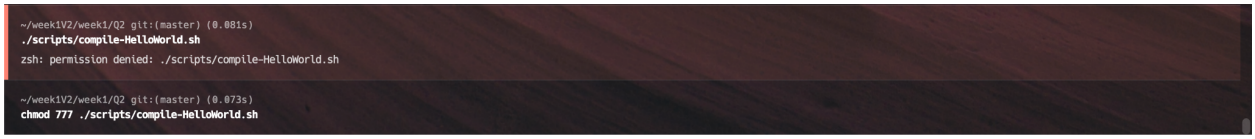
Part 2

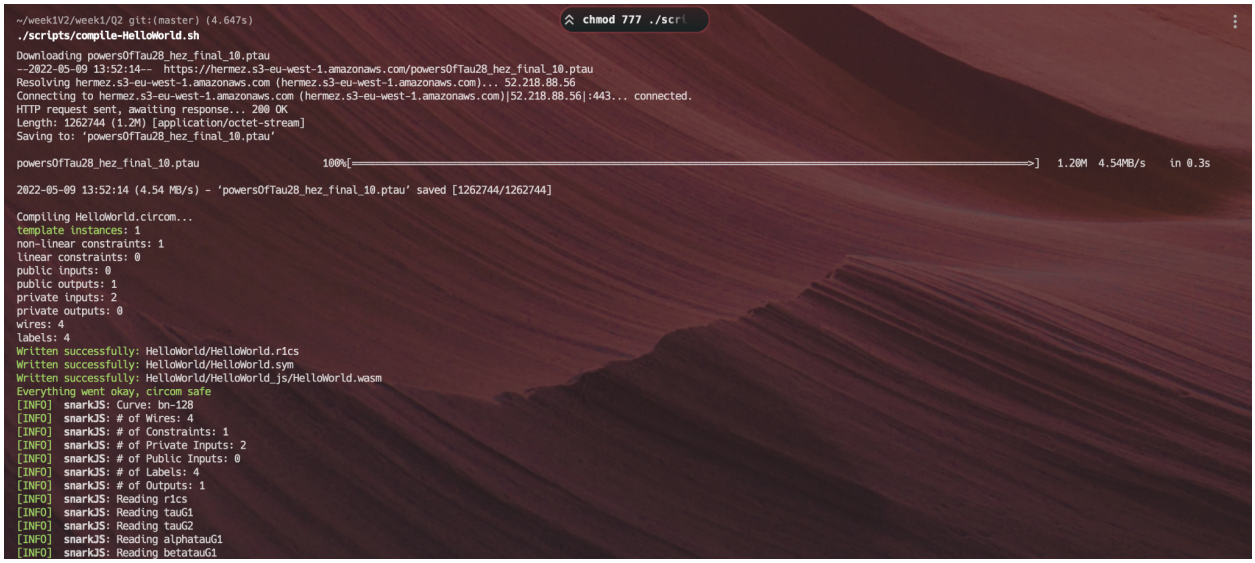
1. Proof of circom installation;

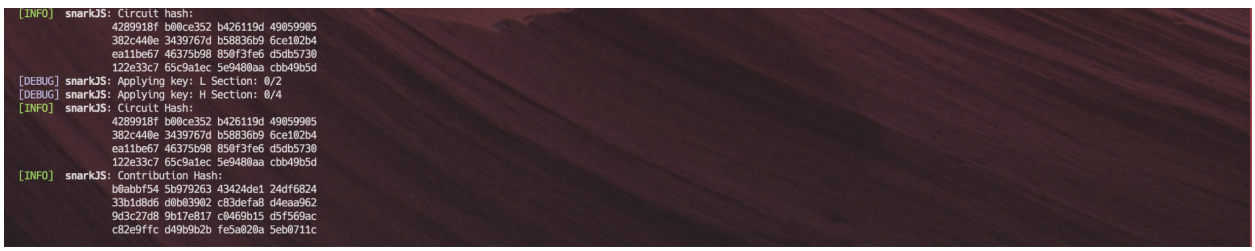


2. Running the bash script (editing with chmod to give appropriate permissions, will PR this to include it for future cohorts);



Result output (2 pictures);





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;

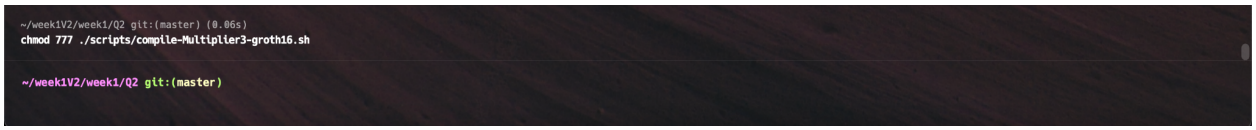
   $C == A * B$

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.

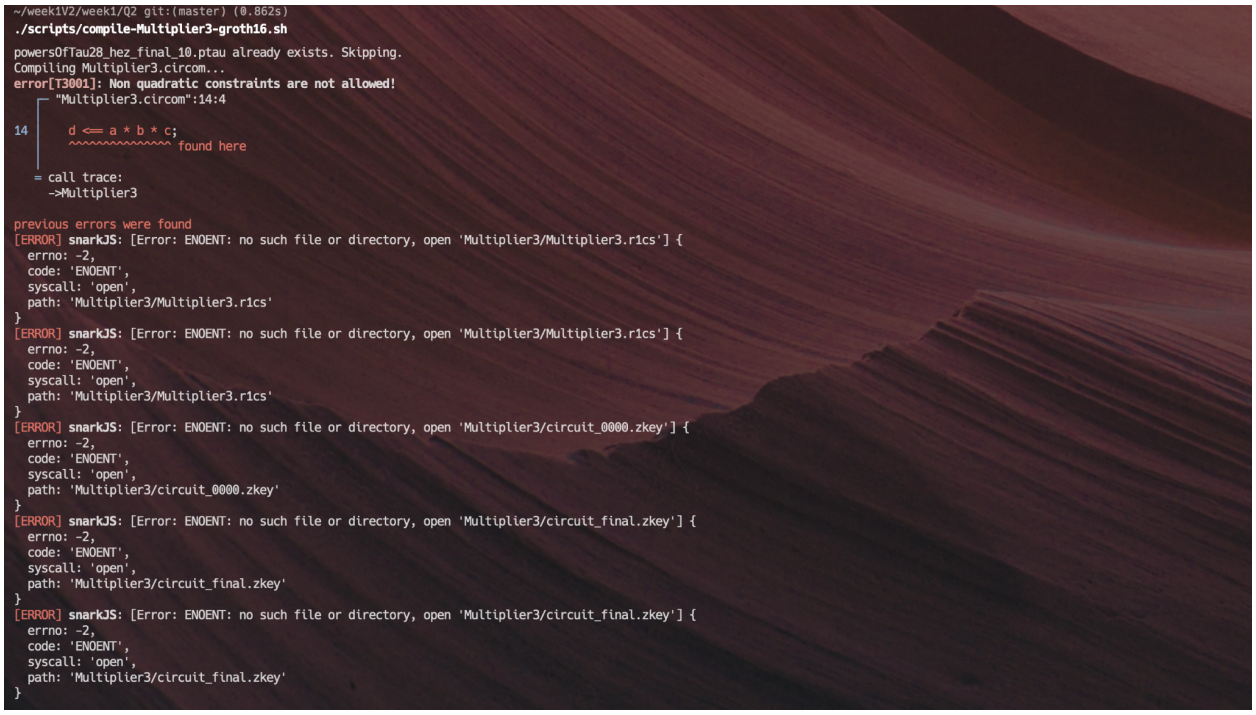
3. **Line 24 of compile-HelloWorld.sh makes a random entropy contribution as a Phase 2 trusted setup. How are Phase 1 and Phase 2 trusted setup ceremonies different from each other?**
   Phase 1 ceremonies are perpetual and generates the initial generic setup that can be used by any circuit, phase 2 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

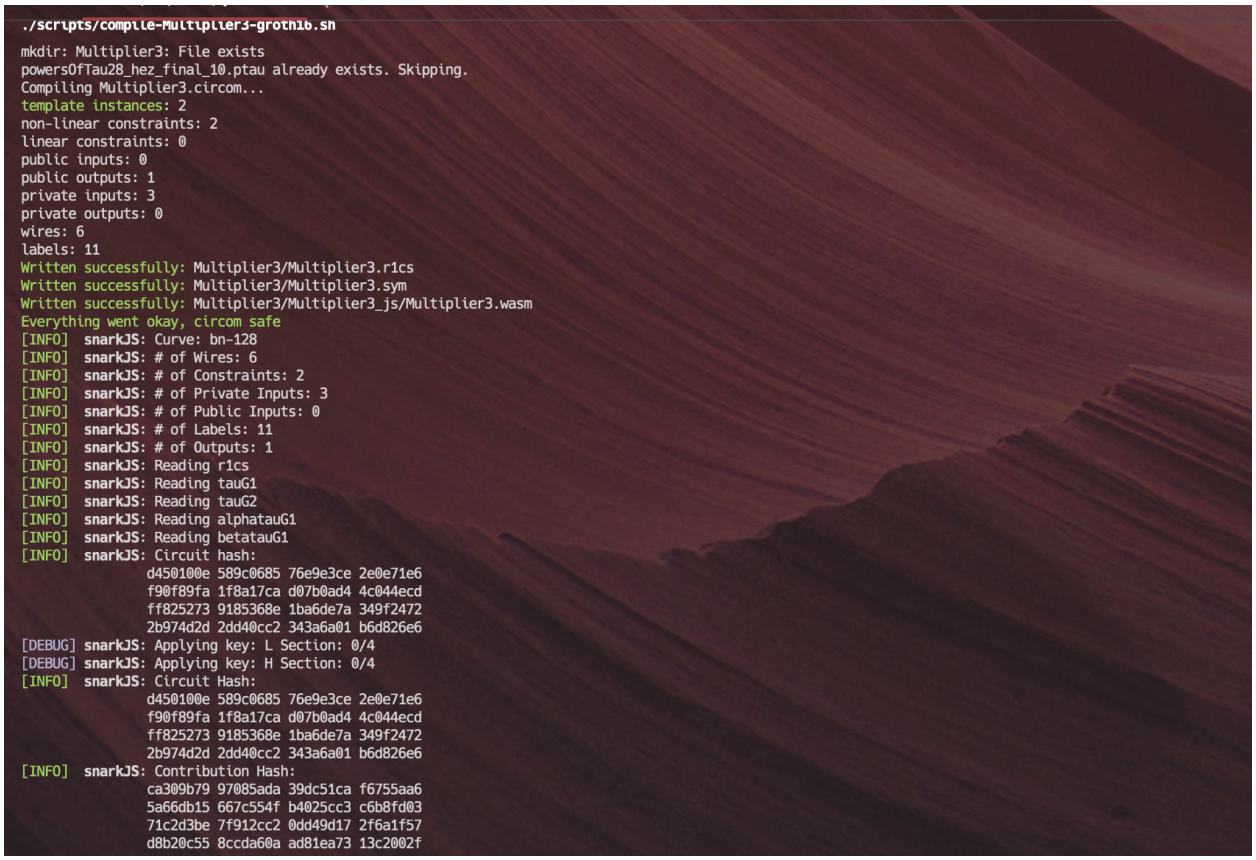


   Running the bash to try and compile the circuit;



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).

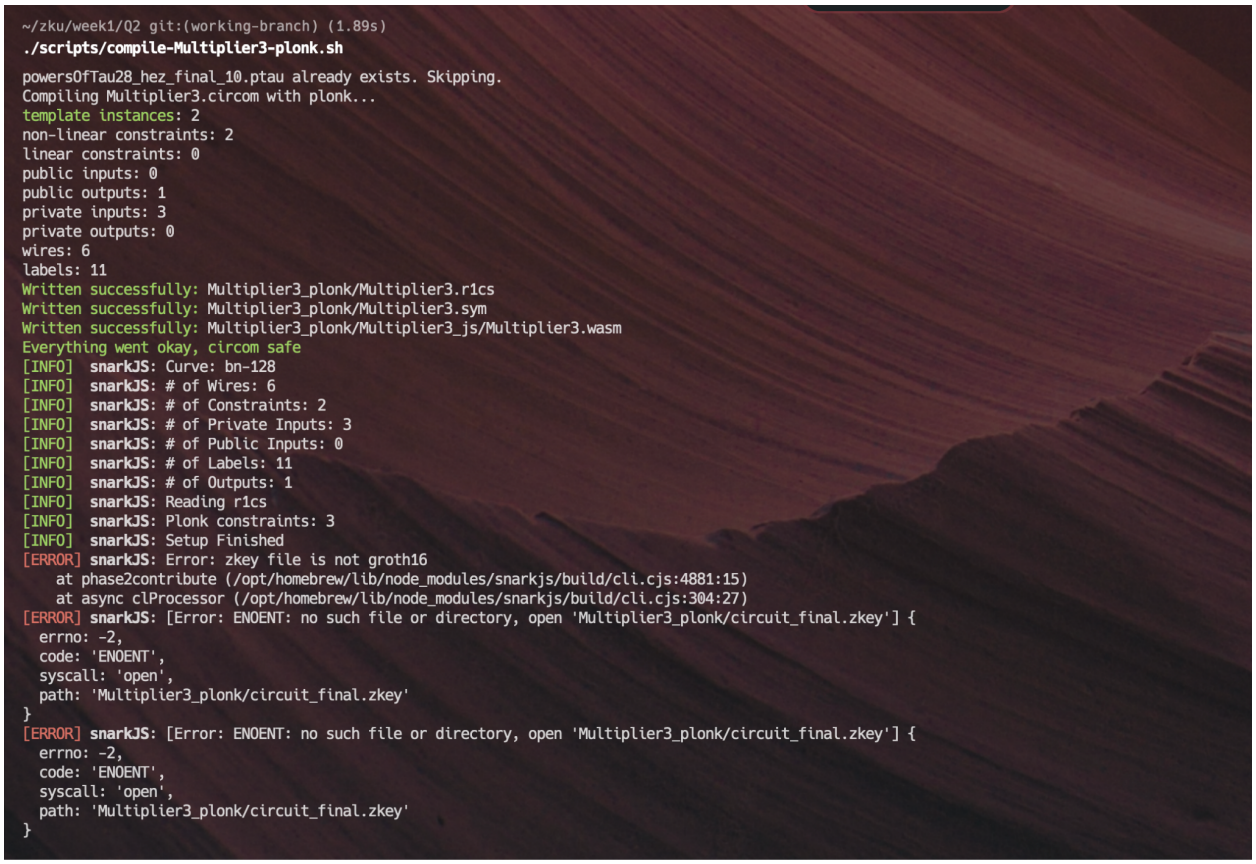
3. **Modify Multiplier3.circom to perform a multiplication of three input signals under the restrictions of circom.**



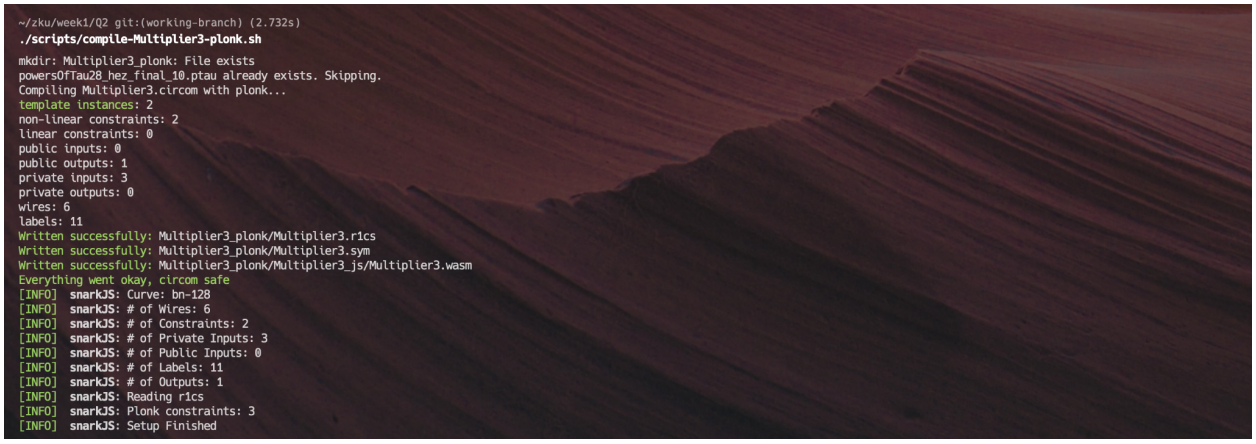
4. In the empty scripts/compile-Multiplier3-plonk.sh, create a script to compile circuit/Multiplier3.circom using PLONK in snarkjs. Add a _plonk prefix to the build folder and the output contract to distinguish the two sets of output.

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 PLONK different from compiling with Groth16?**

   (with error)



(without error)

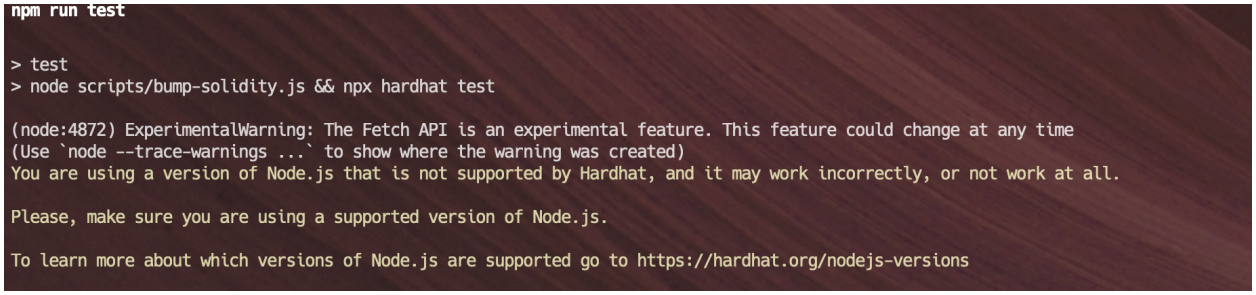


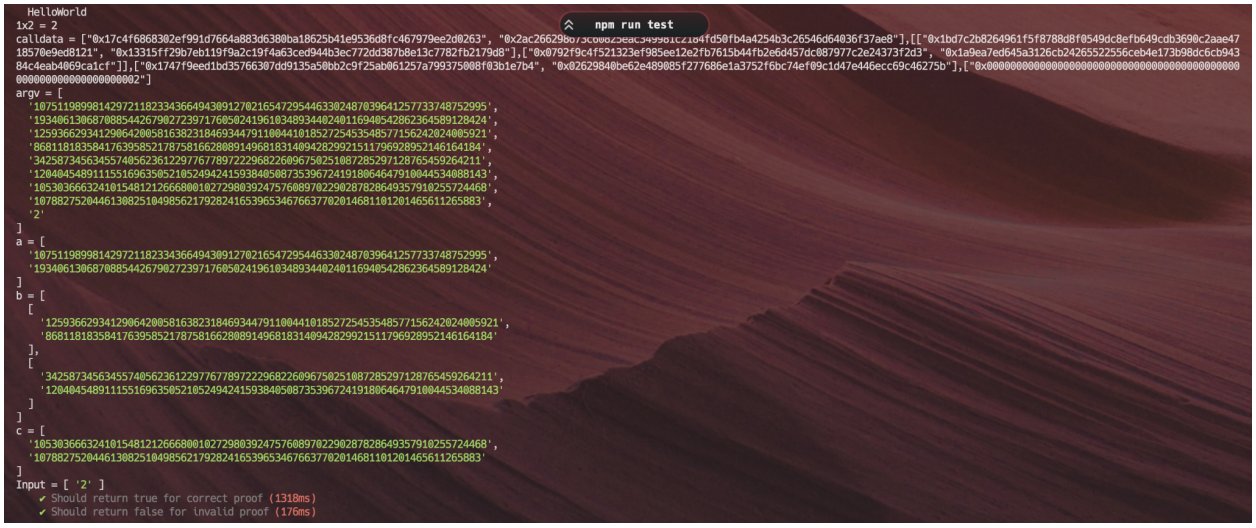
2. **What are the practical differences between Groth16 and PLONK? Hint: compare and contrast the resulted contracts and running time of unit tests (see Q5 below) from the two protocols.**
   The difference between the plonk and groth16 CRS generation is that plonk CRS is universally across a set of 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.

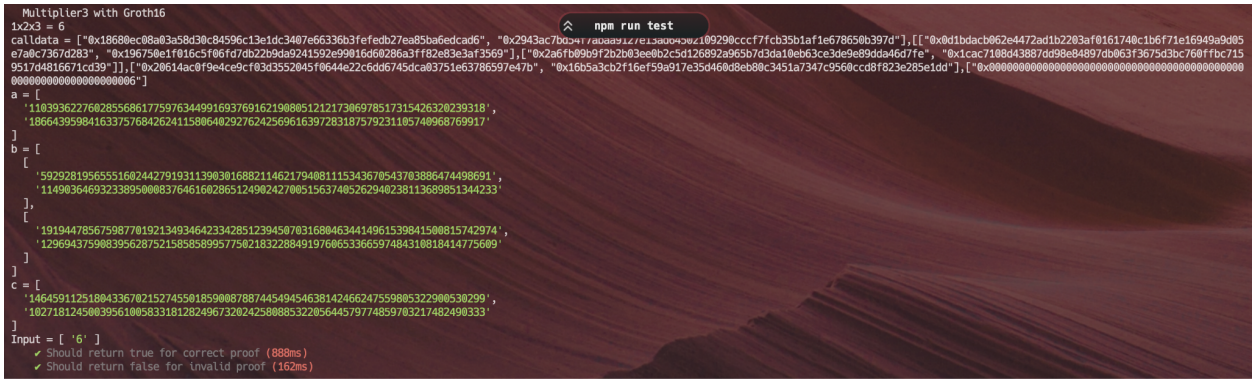
5.

1&2&3

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







```
  Multiplier3 with PLONK
1x2x3 = 6
calldata = 0x10c182d488942e234423741354d4a7ec0e6bc1ab754f93b9ce115d1b7cec6e0703703c6ced5c0d1d0a189cdb3f2185abfe0fc448a367d1ad938397cc98ec45bb0532b860ce9b0902e783628c96911b81c1b7a5b6ee1886fb2525e14a9c43f3d2215
b558f699ff8128e0e8c977c7125aab1445ad0bf3f582ea4430a56d3bd4c861d35197af95bc793b0b536cf31de4f8733d1c7299abcf10240a3a4371190f9432bf9f05b3e7b0cddf619342ac8562e3d4cc9f71689d26ec1860e2481341c34be163bd354c9db7fbac92
38afc2091e54974e011f80a6e63d16e451c5adc05fe492b7d005c7ca66d70efcafbef1ccea7e4c00fe4e17120e3658f58839819ce75f40818d42927643b62e676e49a1621ab5192ad301b7107d6507dc42a1b887775d91d5970c738edbb681257419cadc1c35c642
6b604c26b2ae03a4d4f961795d762118c4830648b4ca710b6b0b1f69172608f9e6172241c7945b9a9170a3f046e8e2076222b6506ed1a920372086d6a59772637960211df56cc182c54bd470f4b5f0e0acc77697659b48168c81accf4771239b06dcc8e210a37113
8e19115f00acd005befe985176fa1579e65e7fec12fa6079527cf5bcc76eed00aecd5c5971a6f263b4c847261221bc5f57a858113d860bf1837c1cbf60deb94bb5acdeb599ba21481add1db3405afc94a35e75f505add6d26e8260e2823148fadb0a8a5e12d4a1d6
ca4fd2e4397300bd21507ae21d8d87cb23897b9bfbdaa8a42d8a0e7a1e9f80e80a768717b08484699c0859303fd500ddef094d8948c8dbddb9a399e29920c17b1f8dccf54b93376265a21c6853497b7acd99e5423e8aa0759cf247cc58acd28d683111713be75a0e
c1053a9fe18d028a9add7a80b671775661d33ec5c37281ceff7e294e1558b2a82011c39510c459bfc2016618d8ac32cd9c973d7bcc66d169d12c889eea37154e0c3144d86375b991f5d54951364237b9ad241760c091d04b1324dda6bc40ae5cf750f8bdabef85bf
9daeb1b63179b07435da318ce433d15f04f50463b1224e53544d3a3f33e7953d4e7ec7c971768089362758O8e0ec41fba63fdddb86454bd2656993015822dd7c4a6eb628e9e9016a41d5433c42506,["0x0000000000000000000000000000000000000000000000000000000000000006"]
bytesforinput = 0x10c182d488942e234423741354d4a7ec0e6bc1ab754f93b9ce115d1b7cec6e0703703c6ced5c0d1d0a189cdb3f2185abfe0fc448a367d1ad938397cc98ec45bb0532b860ce9b0902e783628c96911b81c1b7a5b6ee1886fb2525e14a9c43f3
d2215b558f699ff8128e0e8c977c7125aab1445ad0bf3f582ea4430a56d3bd4c861d35197af95bc793b0b536cf31de4f8733d1c7299abcf10240a3a4371190f9432bf9f05b3e7b0cddf619342ac8562e3d4cc9f71689d26ec1860e2481341c34be163bd354c9db7f
bac9238afc2091e54974e011f80a6e63d16e451c5adc05fe492b7d005c7ca66d70efcafbef1ccea7e4c00fe4e17120e3658f58839819ce75f40818d42927643b62e676e49a1621ab5192ad301b7107d6507dc42a1b887775d91d5970c738edbb681257419cadc1c3
5c6426b604c26b2ae03a4d4f961795d762118c4830648b4ca710b6b0b1f69172608f9e6172241c7945b9a9170a3f046e8e2076222b6506ed1a920372086d6a59772637960211df56cc182c54bd470f4b5f0e0acc77697659b48168c81accf4771239b06dcc8e210a
371138e19115f00acd005befe985176fa1579e65e7fec12fa6079527cf5bcc76eed00aecd5c5971a6f263b4c847261221bc5f57a858113d860bf1837c1cbf60deb94bb5acdeb599ba21481add1db3405afc94a35e75f505add6d26e8260e2823148fadb0a8a5e12d
4a1d6ca4fd2e4397300bd21507ae21d8d87cb23897b9bfbdaa8a42d8a0e7a1e9f80e80a768717b08484699c0859303fd500ddef094d8948c8dbddb9a399e29920c17b1f8dccf54b93376265a21c6853497b7acd99e5423e8aa0759cf247cc58acd28d683111713be
75a0ec1053a9fe18d028a9add7a80b671775661d33ec5c37281ceff7e294e1558b2a82011c39510c459bfc2016618d8ac32cd9c973d7bcc66d169d12c889eea37154e0c3144d86375b991f5d54951364237b9ad241760c091d04b1324dda6bc40ae5cf750f8bdabe
f85bf9daeb1b63179b07435da318ce433d15f04f50463b1224e53544d3a3f33e7953d4e7ec7c971768089362758O8e0ec41fba63fdddb86454bd2656993015822dd7c4a6eb628e9e9016a41d5433c42506
uintarrayforinput = [
  '0x0000000000000000000000000000000000000000000000000000000000000006'
]
    ✔ Should return true for correct proof (1105ms)
    ✔ Should return false for invalid proof (222ms)


  6 passing (4s)
```

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)
```

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 input in; // this is the number to be proved inside the range
    signal input 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 th
    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.