AbdulQadir#0432 qadir@xord.com

Part 1

Question 1:

Traditionally Zk-Snarks are differentiated in terms of the trusted setup and proof size of zk-snark. A trusted setup is the core property of the Zk-snark.

- **Universal Setup:** A Structured Reference String (SRS) creates **toxic waste** in a universal setup. The string is universal and can be used with multiple types of circuits of a specific size. Furthermore, a Universal setup is more differentiated in terms of upgradable or non-upgradable.

  Example: Plonk, Marlin, SuperSonic-RSA

- **Transparent Setup:** In a transparent setup. A Common Reference String (CRS) is created with transparent arguments. It does not create a toxic waste compared to the universal setup snarks. These are non-upgradable and work for a single circuit.

  Example: Groth16, Halo, Fractal, and SuperSonic

Question 2:

Snarks required a trusted setup to generate a reference string **CRS/SRS** to generate proofs of the private transaction. It is necessary to make the system fraud-proof. Once the system is updated, a new reference string is generated to make it secure.

Starks does not need the trusted setup because they are transparent based on the hash functions. Anyone can become part of the system and become a verifier.

Question 3:

- **Quantum Secured:** Zk-Snarks are not quantum secured, while Zk-Starks are quantum secured as they rely on the hash functions

- **Proof Size:** Zk-Snarks proofs take less time to verify than Zk-Starks since it has a smaller proof size than Zk-Starks.
- **EVM Gas Cost:** Zk-Snarks consume only 24% gas compared to Zk-Starks due to their smaller proof size.

Part 2

Question 2:

Part 2.1:

It is a circuit for simple multiplication that declares two private input signals and the public out signals. It has a single constraint that assigns and checks the multiple of **a** and **b** signals to the output signal **c**.

Part 2.2:

Power of Tau is a multi-party computation ceremony used for generating the generic setup parameters, i.e., CRS. Multiple parties participating in the ceremony contribute value to creating a trusted reference string. Therefore, even if a single person did toxic waste, it will secure the reference string.

This ceremony is important in the setup of zk-snark to secure reference strings by adding randomness while generating the initial parameter. Which inturn ensures that that system parameter cannot be replicated.

Part 2.3:

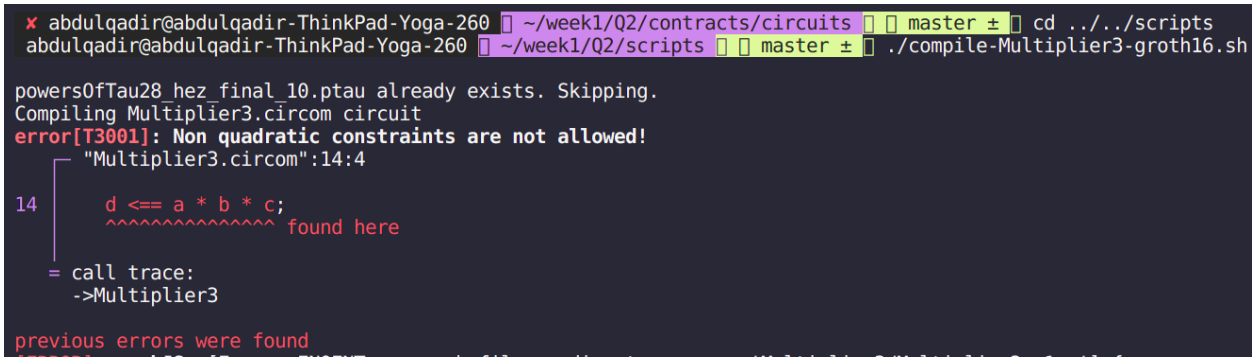
In part one of the trusted setup, we use the predefined CRS and build it with our circuit. While in the second phase, we are adding random entropy as a contribution from our side to make it more secure.

Question Three:

Part 3.2:

The circom constraint system supports only non-quadratic expressions built using only multiplication and addition. This expression ends with the error "Non-quadratic constraints are not allowed!" as we use a single expression holding a non-quadratic equation. This is due to the nature of the r1cs algorithm that contains only quadratic expressions not higher than it.

Remark. Neither the operator === nor <== can be used with signal expressions that are not quadratic.



Question Four:

Part 4.1:

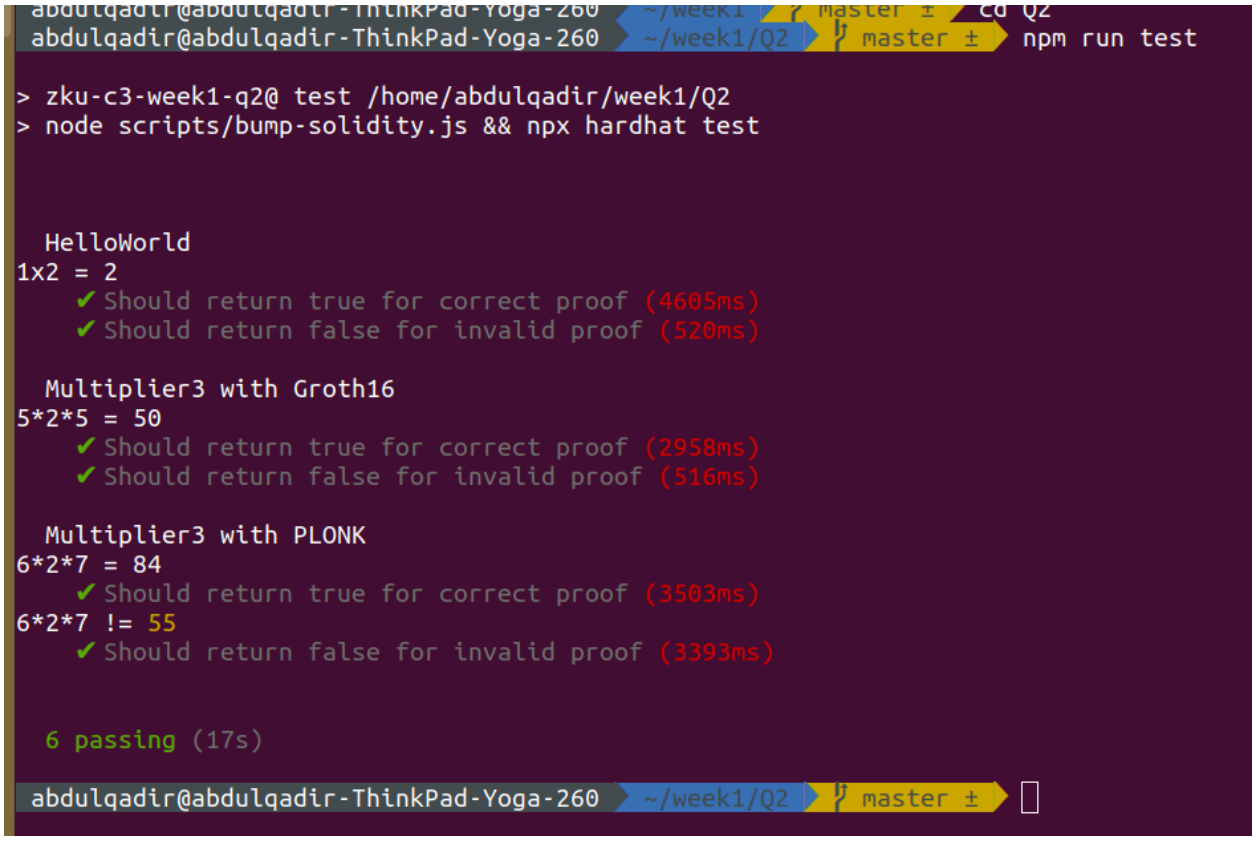
Plonk is universal and updatable; hence it doesn't require a contribution to the ceremony. However, whereas groth16 is not-universal and compiles with circuits, respectively, it requires the contribution.

Part 4.2:

- Verification time of the proof, groth16 takes less time to verify the proof.
- Parameter to verify the proof, plonk takes less parameter in verifyProof.
- Phase 2 of the trusted setup, plonk does not require phase 2 setup of the ceremony.

Question Five:

Part 5.3:



Part 3

Question One:

Part 1.1:

On-Line no 9, 32 stands for the arbitrary number of bits contained by the number we provide to the LessThan template that further pass it to the NumToBits template to convert the integer to the bits of n size. In this case 32 bit size.

Part 1.2:

There are two possible outputs for the LessThan10 circuit **0** and **1 bit.**

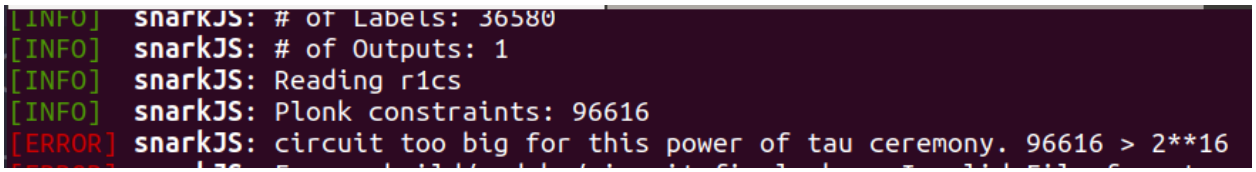
Output 0 ⇒ When the input signal is greater than 10, the circuit will give an output signal of 0.

Output 1 ⇒ When the input signal is less than 10, the circuit will give an output signal of 1.

Question Two:

Part 2.2:

Circuits convert into ".r1cs" upon compilation. This file consists of the constraints which are to be proven. The bigger the circuits, the higher number of constraints there will be. The code has a 2**16 constraints limit, but the number of constraints is higher, so using a bigger file solved the problem.



Part 2.4:

The number of possibilities in brute force is quite large. We check the answers we know are incorrect even in brute force, which is not efficient. Using the sum method reduces the complexity and gives us better verification constraints.

Question Four [Bonus]:

There are multiple libraries that we can create that foster the growth, which includes

- **Message parsing/encryption:** lib that converts user messages into bits and then verifies that the user knows the message or text
- **Merkle tree/ Verkle tree:** lib that covers the Merkle tree, Partritia merkle tree and verkle tree implementation in circom will help in zk-rollups side
- **Arithmetic Equation:** Lib that covers more arithmetic equations and expressions will also help in the scale of circom