Zero-Knowledge University (ZKU):

Assignment 1

Tal Derei

07 March 2022

Course Registration Email: Tad222@lehigh.edu

Discord Username: Margulus#4273

Github (Personal Repistory): `https://github.com/TalDerei`

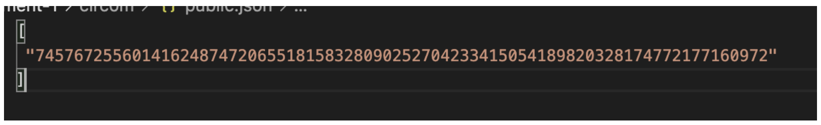
Gitub (ZK Research): `https://github.com/TalDerei/Zero-Knowledge`

<u>Circom and snarkJS:</u>

1. Constructing a zk-circuit that computes the merkle-root hash recursively involved using MiMCsponge, a ZK optimized hash function. Computing the root hash for the input "leaves":[1,2,3,4,5,6,7,8] yielded the following hash, as expected.

Codebase (Git Commits)

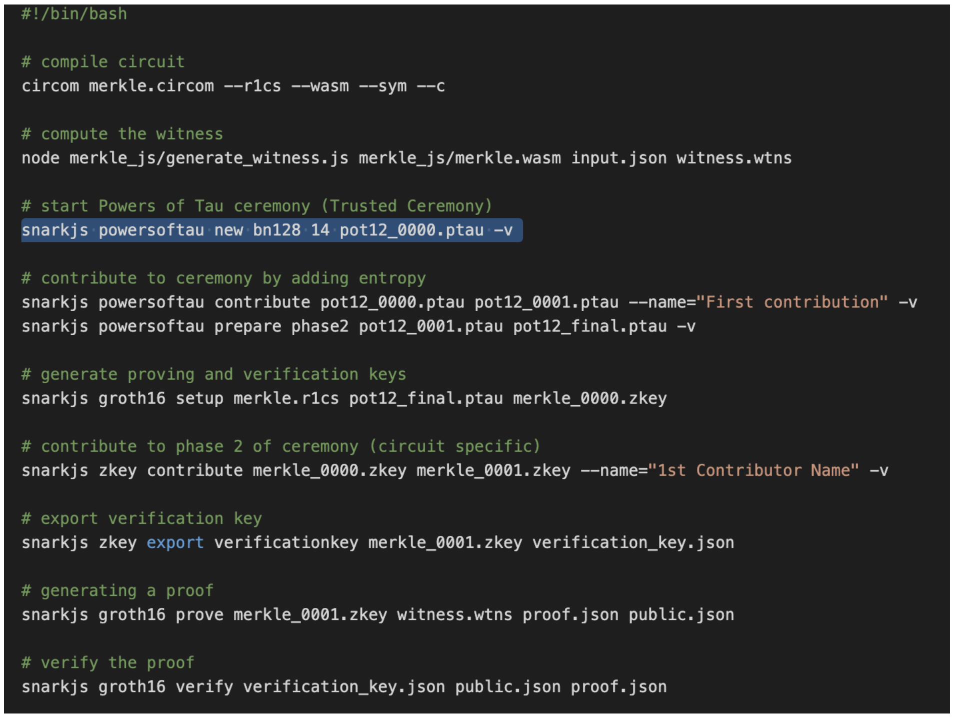
`https://github.com/TalDerei/ZKU/commit/59da0ef7de91b8271301af67973dabe700c4355d`



```
[
 "7457672556014162487472065518158328090252704233415054189820328174772177160972"
]
```

Figure 1: *Merkle Root Hash*

2. When modifying the input to 8 elements, the compiler complained with *"circuit too big for this power of tau ceremony. 5280*2 ¿ 2**12"*. The number of constrains in my circuit exceeded the maximum number of supported constraints that the ceremony can accept. The Powers of Tau ceremony accepts a modifiable constraint range, setting the default to $2^{12} = 4096$ constrains. The maximum value supported here is 2^{12}, which means snarkJS can generate zk-snark parameters for circuits with up 268 million constraints! In order to resolve my issue, modified the CLI command to instruct snarkJS to give the Powers of Tau a larger constrain tolerance. I had to increase the number of constraints to $2^{14} = 16{,}384$ constraints.



```
#!/bin/bash

# compile circuit
circom merkle.circom --r1cs --wasm --sym --c

# compute the witness
node merkle_js/generate_witness.js merkle_js/merkle.wasm input.json witness.wtns

# start Powers of Tau ceremony (Trusted Ceremony)
snarkjs powersoftau new bn128 14 pot12_0000.ptau -v

# contribute to ceremony by adding entropy
snarkjs powersoftau contribute pot12_0000.ptau pot12_0001.ptau --name="First contribution" -v
snarkjs powersoftau prepare phase2 pot12_0001.ptau pot12_final.ptau -v

# generate proving and verification keys
snarkjs groth16 setup merkle.r1cs pot12_final.ptau merkle_0000.zkey

# contribute to phase 2 of ceremony (circuit specific)
snarkjs zkey contribute merkle_0000.zkey merkle_0001.zkey --name="1st Contributor Name" -v

# export verification key
snarkjs zkey export verificationkey merkle_0001.zkey verification_key.json

# generating a proof
snarkjs groth16 prove merkle_0001.zkey witness.wtns proof.json public.json

# verify the proof
snarkjs groth16 verify verification_key.json public.json proof.json
```

Figure 2: *Powers of Tao: Constraints*

3. Zero Knowledge proofs are neccessary for this type of computation, especially in the context of blockchain scalability using layer-2 ZK-Rollups. ZK-Rollups maintain smart

contracts rooted in Ethereum, and move computation and state storage (i.e. the execution) off-chain. In this zero-knowledge environment on ZK-Rollups, validators are tasked with computing a proof of validity for a batch of transactions, which is then posted back on the mainchain where a collection of smart contract precompiles will verify the proof and update the state hash. In order to verifiablly prove the transactions were executed correctly on Layer-2, zero-knowledge computation is used to recursively aggregate many individual proofs (each representation a transaction), into a single proof that's posted on the mainchain. Zero-Knowledge is the core technology that allows layer-2 solutions that are based on zk cryptography to scale computations, by creating proofs that amortize the cost for users to transact off-chain compared to the Ethereum mainchain.

NFT Minting:

Codebase (Git Commits)

`https://github.com/TalDerei/ZKU/commit/fb0eb280f45cfdbf9d6066856a0f8d8dafbd6117`

Zero-Knowledge Technology:

1. SNARKs vs STARKs: It's been long debated about the tradeoffs amongst these zero-knowledge proof technologies. SNARKs have been adopted at by the general ZK community at a faster pace than STARKs, in part because of the better documentation and support. This hasn't changed despit SNARKs requiring a trusted setup (similiar to the Power's of Tau ceremony conducted earlier in the course). Unlike SNARKs on the other hand, STARKs (developed by ZK teams like STARKWARE) rely purely on hash functions which provides two concrete benefits: [1] no trusted setups are required, [2] hash functions provide post-quantum resistance.

2. The trusted setup associated with the Groth16 prover is based on a single-circuit "Trusted-Setup", requiring a new MPC ceremony (Aztec's Ignition Ceremony or Zcash's

Sprout Ceremony) every time the circuit (i.e. the smart contract) is updated. ZK provers like PLONK on the other hand rely on a public (structured) reference string created during a one-time trusted setup, known as a "Universal Setup". This means the setup isn't limited to a single circuit, but can be used for many circuits up to a certain maximum constraint size. It's worth noting both Groth16 and PLONK still produce toxic waste.

3. Zero Knowledge can be applied in the NFT space in regards to privacy-preserving minting transactions. If we extended Aztec's L2 ZK-ZK-Rollup model, we can construct a "Mint" proof that can allow a prover to prover to a verifier that an NFT was minting, without revealing the state (i.e the NFT itself), or the transaction data (i.e. which addresses is minting or which addresses received it). Inherently, Aztec's protocol preserves the user's privacy by replacing public variables / addresses in normal solidity contracts with ZK proofs as input variables, allowing a user to interact with Aztec's API to generate different types of composable proofs depending on their desired business logic / functionality. In that regard, ZK is abstracting the minting transaction itself from the public.

4. Zero-Knowledge is often discussed with respect to privacy in the DAO space, specifically private voting using semaphores. But another important zk application in DAOs are authentication protocols. Specifically, corrupt issuers / certificate authorities may become malicious and try to cancel access to their permissionless DAO, or even try forge certificates using the same input parameters. ZK would allow anonymous and private participation in the DAO (beyond voting) without ever having to use the certificate at all. Imagine if you never had to use your password to login, instead proving some statement that cryptography verifies that password is correct without actually supplying it.

counterfeit a certificate