ZKU Week 6 Assignment

Email: qadir@xord.com
Discord Id: AbdulQadir#0432

Part 1 WorldCoin

Question 1:

There are currently two challenges when using Semaphore for such user verifications and their possible solutions:

1. Identity Insertion:

   Inserting every leaf individually into the Merkle tree and updating the root inside the smart contract is cost-effective as every Merkle proof is verified inside the Zero-knowledge Proof. This happened due to using Keccak inside the Smart contract while inserting the leaf, which consumes a lot of gas.

   Solution: A sequence or rollup which batches the insertion of the leaf updates in ZKP that aggregate multiple Merkle root updates will eventually drive down the cost of identity insertion significantly

2. Proof Verification:

   In semaphore, every proof is submitted and verified, and verifying every proof takes around 300k gas. We can reduce this cost by aggregating the proofs and verifying them at once to reduce the cost of verifying. Still, semaphore does not support as it uses the (Groth16) and Groth16 does not support the multiple proof aggregation and verifying.

   Solution: Using recursive snarks instead of normal snarks can solve this problem. (A recursive SNARK enables producing proofs that prove statements about prior proofs)

Question 2:

Hubble is a minimal, application-specific rollup. It allows a highly efficient, non-custodial airdrop at a one billion people scale. It is an optimistic L2 solution but has higher throughput than the zk rollups and the optimistic rollups as it supports the BLS signature aggregation to reduce the size to store data on-chain and optimizes for simple token transfers. Another reason for improving the throughput of Hubble is that the contract does not validate the balance updates or the authenticity of the sender.

In Hubble, transactions are in canonical order. Therefore, they can be saved back and forth, due to which if a fraud-proof is detected, it reverts the whole batch. However, since the transactions are not canonical, they are confirmed only after seven days.

What else can Hubble do?

- Users can migrate their tokens to other L2 solutions without withdrawing and depositing from Layer 1 (L1).
- Users can onboard accounts to Hubble without going through L1. The coordinator can register their public keys, and then they can acquire tokens from holders who are already in the Hubble L2.

Part 2 Polygon NightFall

Question 1:

Polygon Nightfall is an application for transferring ERC20, ERC721, and ERC1155 applications under Zero-Knowledge. It is a Layer 2 solution, an Optimistic rollup with on-chain data availability.

Polygon Nightfall achieves the private transfer of ERC20 tokens by a shielded transaction on layer 2. They are using a nullifier pattern allowing users to create a commitment on L2 and spend it.

Polygon Nightfall does not have any privacy on the L1, but on L2. the user deposits funds on layer 1 Shielded contracts to access them on layer 2. In short, users can use the commitment made on layer 1 to access funds on layer 2. Hence this approach reveals the user's identity while moving funds from L1 to L2 and L2 to L1. But the main privacy is that the L2 user can spend tokens by creating the commitment associated with the new keys, which can then be used to spend the tokens. While to withdraw the tokens, the user has to spend the nullifier.

Question 2:

Private NFT marketplaces are one of the use cases of the Polygon Nightfall. Private NFT marketplace can use the polygon nightfall by allowing users to deposit their NFTs on L1 shielded contracts on layer 2. Users can create the commitment of the NFT with the new keys and use it to trade the NFT on the L2 by transferring the ownership. Hence the trade remained private.

Part 3 Tooling

Question 1:

Foundry is a blazing fast, portable and modular toolkit for Ethereum application development written in Rust. (by foundry-rs).

Features:

- **Traces:** The foundry provides the feature to see the paths and what happens during a function call and shows which function is called between the function call and with what parameters.
- **Fuzz Testing:** Fuzzing allows users to test the functions with all the possibilities. The foundry supports testing the Functions with random values to cover every possible use case.
- **Forge:** Forge is the Ethereum development and testing framework. Gas cost snapshots and assertions are possible by using forge.
- **Fast:** Foundry is blazingly fast compared to hardhat due to core implementation in the rust. ~26.22s vs. ~39.33s.
- **Tests:** Foundry allows to write in Solidity.

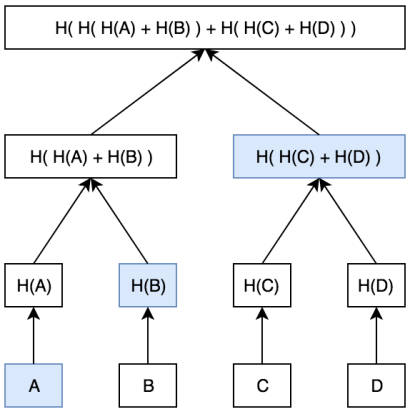
Drawbacks:

- Hardhat offers better tooling for switching between and deployment to various networks.
- Hardhat supports different plugins to work with, which provides additional functionality to the project.
- Hardhat is user-friendly and has a better community.

Question 2:

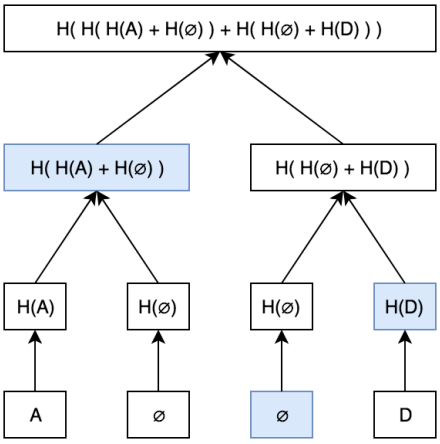
Merkle trees allow us to commit to a data set cryptographically. We start by hashing each piece of data in the set and then keep hashing our way up the tree until we get to the root node.

Finally, an Incremental Markle tree is used to prove inclusion to check whether the data exists in the Merkle tree or not.



But what if we want to prove that something is not part of the Merkle tree. Prove non-inclusion. Sparse Merkle tree help in this case.

A Sparse Merkle tree is like a standard Merkle tree, except the contained data is indexed and stored in a key/value pair map, and each data point is placed at the leaf corresponding to that datapoint's index. So, for example, let's take a Merkle tree with four leaves with an A leaf in the first position and a D leaf in the last. So the rest leaves have a null value in them. So now, if we want to prove that something is not part of the Merkle tree, we prove that a specific position contains null instead of any value.



Part 4 Final Project:

Project Name: ZkPoolTogether
Project Manager: Abdul Qadir
Report Date: 06/13/2022

Overall Project Status ⏳
On-track / At-risk / Off-track

Notable Changes 📋
None.

Timelines & Targets 📅

- Core Development - 18th of June
- Front-end - 21st of June
- Launch on testnet - 24th of June

Top Level Summary 📝
Setup and finalization of the circuits and associated Smart-Contracts. 50% completion of lottery (DrawManeger) contract and wrote high-level unit tests.

Project Components Overview	Question	Comments
Schedule	Is the project running on schedule?	Yes, the project is on track.
Resources	Are you low or have excess resources?	Yes, Resources are enough for the V1 launch.
Quality	Is the project's quality being jeopardized?	No,
Roadblocks	Potential risks & roadblocks that could affect the project timeline.	No, Currently, no roadblocks have been faced in the Development.

Last Week's Tasks:

1. Setting up the ZKPT-Core repo: [link]
    a. Setting up project structure
    b. Dependencies and package setup
    c. Hardhat support setup

    d. Linter, Prettier, and config setup

2. Circom circuits:
    a. Merkle Tree inclusion circuit completion
    b. Withdraw circuit completion
    c. Compilation and setups

3. Smart contract development:
    a. MerkleTreeWithHistory contract
    b. Pool contract (allowing withdrawal and deposit)
    c. DrawManager (50% completed)
    d. Mock contract for yield generation

4. Unit test and interaction test:
    a. Basic Smart contract compilation
    b. Verification test for Poseidon hash function
    c. MerkleTreeWithHistory insertion tests
    d. Deposit and withdrawal from pool and verification of the proof in verifier contract
    e. New draw creation test and bare assertions

Next Week's Tasks:

1. Completion of DrawManagers contract and Vrf integration.
2. Finding flaws in the current circuits and fixing them.
3. Completion of relayer server.
4. Integration test of circuits and Smart contracts.
5. Basic UI sampling.