Voca Functional Specification

Students: - Date Completed: 17/11/23

Senan Warnock: 20752725 —-- Zak Smith: 20723579

Table of Contents:

1. Introduction

Project Title: VOCA

Purpose: To enhance Ethereum network scalability

Description:

VOCA (Verifiable Off-Chain Computations are Awesome) is a project designed to enhance the scalability of the Ethereum network by implementing a zk rollup mechanism for ether/ERC20 transfers. This approach will leverage the efficiency of zk-SNARKs to process transactions off-chain, significantly improving transaction throughput on the Ethereum network.

2. Project Overview

Goals:

- Create a scalable solution for Ethereum transactions.
- Implement a zero-knowledge rollup system for efficient off-chain transaction processing, Minimizing processing time and costs associated with ether/ERC20 transfers.
- Ensure security and integrity of transactions using zk-SNARKs, through the processing stages

Scope:

The project will focus Primarily on ether and ERC20 token transfers, using zk rollups to streamline transaction processing.

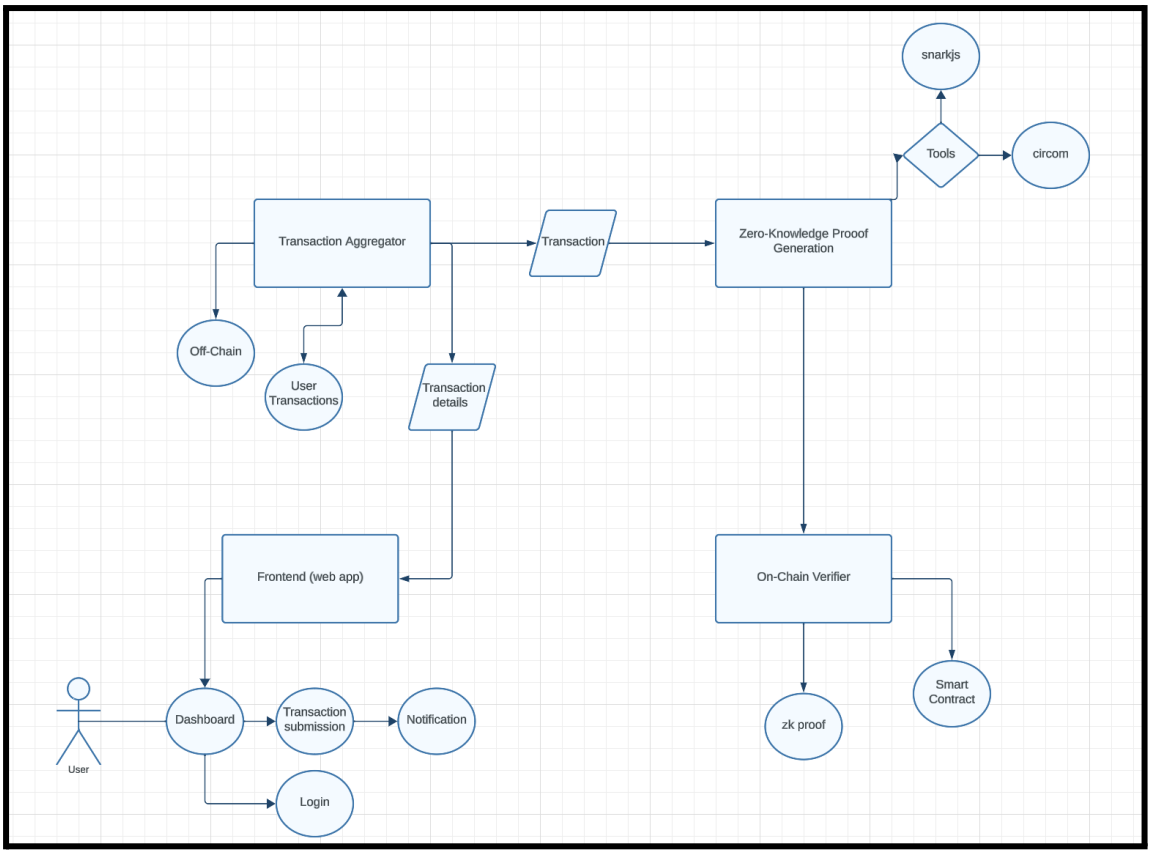
It will involve:

- Creating Off-Chain Transaction Aggregation measures, along with batching transactions, preparing them for off-chain processing.

- Zero-Knowledge Proof Generation: Utilizes zk-SNARKs to create cryptographic proofs that validate the transactions in a privacy-preserving manner.
- On-Chain Verification: A smart contract on the Ethereum blockchain that verifies the zero-knowledge proofs and updates the blockchain state accordingly.

3. System Architecture

Fig1: Basic system architecture diagram



High-Level Description:

- **Transaction Aggregator:** An off-chain system that collects user transactions, batches them, and prepares them for processing.
- **Zero-Knowledge Proof Generation:** A component that uses snarkjs and circom to create and validate zk-SNARK cryptographic proofs for the aggregated transactions, These proofs can confirm

that the transactions are valid without revealing their specific details for privacy.
- **On-Chain Verifier:** A smart contract on Ethereum blockchain that verifies the zero-knowledge proofs and updates the blockchain state to reflect the batched transactions, effectively completing the process.
- **Frontend:** This is the user-facing side of the system. It allows users to interact with the blockchain without needing to understand the complex processes happening behind the scenes. Users can submit transactions, view their status, and receive notifications.

4. Technology Stack

Programming Languages:

- Solidity: Used for writing smart contracts on the Ethereum blockchain, crucial for the On-Chain Verifier component.
- JavaScript: used for scripting in snarkjs and circom, facilitating zero-knowledge proof generation and verification.
- Rust: Considered for performance-critical components, offering safety and speed, potentially enhancing the efficiency of our cryptographic computations.
- Foundry for smart contract testing.

Components:

Transaction Aggregation Mechanism:

- Algorithm for selecting and batching transactions.
- Security measures for transaction data during aggregation.

Zero-Knowledge Proof Generation:

- Use of snarkjs for proof generation and validation.
- Workflow for generating, testing, and optimising zk-SNARK proofs.

On-Chain Verification:

- Smart contract development in Solidity for proof verification.
- Integration with Ethereum's blockchain for updating the account Merkle tree.

Web Application:

- Developed using Next.js/React for a robust and intuitive user interface.

5. Functional Requirements

Transaction Processing: Efficiently process, validate, and batch transactions off-chain.
Details:

- Handling high volumes of transactions with minimal latency.
- Ensuring accuracy and integrity in transaction batching and aggregation.

Proof Generation: Generate zero-knowledge proofs that attest to the validity of transaction batches without revealing individual transaction details.

On-Chain Verification: Ensure that proofs are correctly verified on-chain, leading to the appropriate state updates on the Ethereum network.

6. Non-Functional Requirements

Scalability: The system must handle an increasing number of transactions without significant degradation in performance.

Security: Robust cryptographic practices to ensure transaction integrity and system security.

Usability: User-friendly interfaces for interaction with the system, including transaction submission and tracking.

Maintainability: Ensure the system is easy to update, debug, and extend.

Details:

- Writing clean, well-documented, and modular code.
- Adopting best practices in software development to facilitate future upgrades and maintenance.

7. Testing and Validation

Unit Testing: Testing individual components like smart contracts, circuits, and the aggregator algorithm.
Outline:

- Smart Contracts: Ensuring the solidity contracts behave as expected.
- Cryptographic Circuits: Verifying the correctness of zk-SNARK circuits.
- Transaction Aggregator Algorithm: Checking the efficiency and reliability of transaction batching and aggregation.
- Tools and Methods: Utilizing Foundry for smart contract testing and other relevant unit testing

Integration Testing: Ensuring that the entire system functions correctly when all components are integrated.
Outline:

- Interaction between the Transaction Aggregator and Zero-Knowledge Proof Generation.

- Integration of the Zero-Knowledge Proofs with the On-Chain Verifier.
- Frontend interaction with the backend systems for transaction submission and status tracking.
- Example: Simulate real-world basic scenarios of transactions to evaluate system behavior under typical operating conditions.

Performance Testing: Assessing system throughput, latency, and resource usage under various load conditions.

Outline:

- The number of transactions processed within a specific time frame.
- Latency: The time taken to complete a transaction from submission to confirmation.
- Resource Usage: Assessing the computational and memory resources consumed during transaction processing, if possible.
- Testing Environment: Creating a controlled environment to the best of our abilities, that mimics the Ethereum network to assess performance under various load conditions.