ANASTASIA LABS

Project Plan

Version 1.4 – October 2024

Contents

Project Plan

1. Introduction

The objective of this project is to design and implement a bridging contract for the Cardano blockchain that leverages Mithril signatures and zero-knowledge proofs (ZKPs). This bridge aims to enhance transparency, improve security, and reduce costs associated with cross-chain transactions and the development of blockchain bridges on Cardano.

2. Project Objectives

2.1. Develop and Validate Zero-Knowledge Circuits for Verifying Mithril Signatures

- Design and implement arithmetic circuits for Mithril certificate verification.
- Produce a technical report detailing the design and performance of these circuits.
- Create a prototype demonstration of the circuits.

2.2. Develop a Prototype of the Bridge Smart Contract Using zkFold Symbolic

- Create an initial version of the bridge smart contract.
- Document the functionality and architecture of the smart contract.
- Deploy a prototype on a test network.

2.3. Fork and Modify the Mithril Signer to Produce Required Certificates

- Modify the Mithril signer to generate the necessary certificates for the bridge.
- Document all modifications made to the signer.
- Develop test cases and present results validating the modifications.

2.4. Develop the Signature Aggregation Backend Component and Finalize the Bridge Smart Contract

- Create a signature aggregation backend capable of handling Mithril signers.
- Complete the development of the bridge smart contract.
- Provide integration testing results demonstrating the end-to-end functionality of the bridge.

3. Project Scope

The project encompasses the following components:

- Zero-Knowledge Proof Circuits: Designing and implementing ZK circuits for verifying Mithril signatures.

- Bridge Smart Contract: Developing a secure and efficient smart contract using zkFold Symbolic.
- Mithril Signer Modification: Adjusting the existing Mithril signer to suit bridge requirements.
- Signature Aggregation Backend: Building a backend system for aggregating signatures from Mithril signers.
- Documentation: Producing comprehensive technical documentation, reports, and user/developer guides.
- Testing and Deployment: Conducting thorough testing and deploying prototypes on test networks.

4. Milestones and Timeline

4.1. Phase 1: Develop and Validate Zero-Knowledge Circuits for Verifying Mithril Signatures

Duration: Weeks 1-4

Tasks:

- Design Arithmetic Circuits
  - Analyze the requirements for Mithril certificate verification.
  - Design appropriate arithmetic circuits compatible with ZKPs.
- Implement Circuits
  - Develop the designed circuits using suitable ZKP frameworks (e.g., zk-SNARKs).
  - Optimize for performance and security.
- Technical Report
  - Document the design choices, methodologies, and performance metrics.
  - Include diagrams, mathematical formulations, and benchmarks.
- Prototype Demonstration
- Create a demonstrative application showcasing circuit functionality.
- Prepare a presentation or video demonstration.

Outputs:

- Implemented arithmetic circuits for Mithril certificate verification.
- Technical report on the design and performance of the circuits.
- Prototype demonstration of the circuits.

4.2. Phase 2: Develop a Prototype of the Bridge Smart Contract Using zkFold Symbolic

Duration: Weeks 5-8

Tasks:

- Create Initial Smart Contract
  - Develop the core functionalities of the bridge contract.
  - Ensure compatibility with Cardano's Plutus platform and zkFold Symbolic.

- Documentation
  - Detail the smart contract's functionality and architecture.
  - Provide code comments, UML diagrams, and workflow explanations.
- Deploy Prototype on Test Network
  - Set up a Cardano test network environment.
  - Deploy the smart contract and perform initial testing.

Outputs:

- Initial version of the bridge smart contract.
- Documentation detailing functionality and architecture.
- Prototype deployment on a test network.

4.3. Phase 3: Fork and Modify the Mithril Signer to Produce Required Certificates

Duration: Weeks 9-10

Tasks:

- Modify Mithril Signer
  - Fork the existing Mithril signer codebase.
  - Implement modifications to produce certificates compatible with the bridge.
- Documentation of Modifications
  - Clearly document all changes made to the codebase.
  - Explain the rationale behind each modification.
- Testing and Validation
  - Develop test cases to validate the modified signer.
  - Record results and verify correctness.

Outputs:

- Modified Mithril signer capable of producing required certificates.
- Detailed documentation of modifications made.
- Test cases and validation results.

4.4. Phase 4: Develop the Signature Aggregation Backend Component and Finalize the Bridge Smart Contract

Duration: Weeks 11-14

Tasks:

- Develop Signature Aggregation Backend
  - Design the backend architecture for aggregating signatures.
  - Implement functionalities to handle communications with Mithril signers.
- Finalize Bridge Smart Contract

  - Integrate the smart contract with the signature aggregation backend.
  - Enhance security features and optimize performance.

- Integration Testing
  - Conduct end-to-end testing of the entire bridge system.
  - Identify and fix any issues or bugs.

- Documentation
  - Update all technical documents to reflect the final system.
  - Prepare user and developer guides.

Outputs:

- Signature aggregation backend capable of handling Mithril signers.
- Complete version of the bridge smart contract.
- Integration testing results demonstrating end-to-end functionality.
- Updated documentation, including user and developer guides.

5. Tools and Technologies

- Development Tools
  - Haskell and Plutus programming environments.
  - zkFold Symbolic framework.
  - Mithril libraries and SDKs.
  - Zero-knowledge proof libraries (e.g., libsnark, Bellman).

- Testing Tools
  - Automated testing frameworks (e.g., QuickCheck for Haskell).
  - Cardano testnet environments.

- Documentation Tools
  - Markdown editors, LaTeX for technical reports.
  - Diagramming tools (e.g., draw.io, excalidraw).

- Version Control

  Git and repository hosting (e.g., GitHub).

6. Risk Management

6.1. Technical Risks

- Complexity of ZK Circuits
  - Mitigation: Allocate additional time for research and consult with experts.

- Integration Challenges
  - Mitigation: Conduct iterative testing during development phases.

- Security Vulnerabilities

  - Mitigation: Perform regular security audits and code reviews.

6.2. Project Risks

- Timeline Delays
  - Mitigation: Implement agile methodologies for flexibility.
- Resource Constraints
  - Mitigation: Ensure adequate staffing and consider outsourcing if necessary.