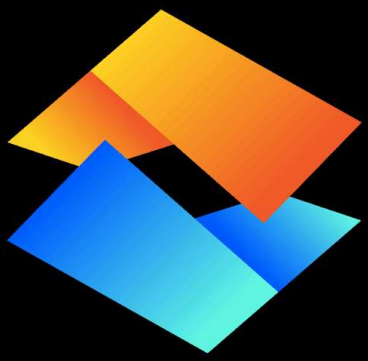

zkFold

zkFold: Zero-Knowledge Prover Backend

zkfold.io

Name of project and Project URL on IdeaScale/Fund

ZKFold: Zero-Knowledge Prover Backend

https://milestones.projectcatalyst.io/projects/1100298

Project Number

Project ID: 1100298

Name of project manager

zkFold team

Date project started

28 April 2025

Date project completed

01 May 2025

Objectives and status of the project

Objective: Develop and deliver an open-source backend application in collaboration with Maestro, to serve as a core infrastructure component for zkFold's suite of zero-knowledge (ZK) products on the Cardano blockchain. The ZK-prover will generate zero-knowledge proofs for smart contract transactions, support critical stages in the ZK transaction lifecycle - specifically input/output balancing, proof generation, and transaction assembly - and be accessible both as a standalone backend and as a hosted service provided by Maestro. The ultimate goal is to enable seamless deployment of ZK-enabled smart contracts through zkFold Symbolic and accelerate adoption of privacy-preserving applications on Cardano.

Project Status: The project was completed within the timeline and delivered all expected outputs.

List of challenge KPIs and how the project addressed them

1. Plonk Prover Integration and Performance

KPI challenge:
Successful implementation of the Plonk zero-knowledge prover algorithm optimized for performance, modularity, and developer accessibility, enabling the backend to generate valid ZK proofs for zkFold Symbolic smart contracts on Cardano.

How the KPI Was Addressed:

- **Performance:** Critical cryptographic operations such as FFT and multi-scalar multiplication were implemented in Rust to ensure computational efficiency.

- **Modularity:** The architecture separates performance-heavy logic (Rust) from high-level protocol flow (Haskell), supporting maintainability and ease of extension.

- **Functionality:** The complete Plonk prover algorithm was implemented in Haskell and integrated with Rust bindings, enabling reliable proof generation.

- **API-readiness:** This milestone provides the foundational functionality that will be exposed via the public API for developers building ZK-enabled smart contracts.

2. API Server Deployment and Developer Accessibility

KPI challenge:
Design and deployment of a backend server component that exposes the Plonk prover via a public API, includes robust logging for observability, and provides clear setup documentation to support integration and use by developers and platforms like Maestro.

How the KPI Was Addressed:

- **API Availability:** An API server component was implemented in Haskell, providing endpoints for calculating Plonk proofs, making it accessible to external applications.

- **Operational Monitoring:** Logging modules were developed to capture key events and system behavior, supporting debugging, monitoring, and maintenance.

- **Developer Enablement:** A comprehensive setup guide was published on GitHub, enabling developers to install, configure, and run the Prover backend locally or in production environments.

- **Platform Integration:** API access was made available and managed through the Maestro platform, ensuring smooth onboarding and usage for dApp developers targeting Cardano.

3. Protostar prover algorithm

KPI challenge:
Implementation of the Protostar prover algorithm and a custom transaction balancing algorithm to enhance Prover’s functionality and support the full zkFold Symbolic smart contract lifecycle.

How the KPI Was Addressed:

- Implemented the Protostar prover algorithm in Haskell and exposed it via API endpoint 2.

- Developed a custom transaction balancing algorithm as API endpoint 3, tailored for zero-knowledge constraints.

- Expanded Prover's API to support multiple proof systems and ZK-specific transaction flows.

4. Scalable Proof Computation via Load Distribution

KPI challenge:
Implementation of a load distribution algorithm to enable parallel proof generation across multiple backend instances, ensuring Prover can handle high throughput and scale with demand.

How the KPI Was Addressed:

- Built load distribution modules in Haskell to split proving tasks across multiple servers.
- Enabled parallel proof computation, reducing latency and improving scalability.
- Integrated with Maestro's managed API, ensuring reliable access under load.

List of project KPIs and how the project addressed them

Quality of Implementation:

- Delivered well-documented, modular Haskell code for all specified tasks.

Testing and Verification:

- Verified outputs against predefined benchmarks to ensure accuracy.

Scalability and Usability:

- The scalability of the zkProver project ensures it can efficiently handle increasing workloads, including a growing number of users, larger proof-generation tasks, and higher transaction volumes. The system achieves this by implementing a load distribution algorithm that allows proof calculations to be split across multiple backend servers, enabling parallel processing and reducing latency. This design ensures that zkProver can scale horizontally to meet demand without compromising performance or reliability, allowing it to support the future growth of zkFold Symbolic smart contracts on Cardano.
- The usability of zkProver focuses on making the system easily accessible and practical for developers. The project provides a well-designed public API for

seamless integration with external applications. It also includes comprehensive setup guides and logging modules to help developers troubleshoot and monitor their interactions with the system. Integration with Maestro further simplifies API access management, enhancing the overall developer experience and enabling smooth deployment and maintenance of zkProver-powered ZK applications.

Collaboration and Open Access:

- Ensured community involvement by open-sourcing key modules and collecting feedback throughout the process.

Key achievements (in particular around collaboration and engagement)

Technical Excellence:

- Successful implementation of core functionalities

Collaboration:

- Partnered effectively with the Maestro team, the Cardano community to refine specifications and receive iterative feedback.

Engagement:

- Engaged with developers through open-source contributions, testnet demonstrations, and documentation workshops.

Community Enablement:

- Empowered the Cardano developer ecosystem with tools and templates to create zkFold Symbolic smart contracts.

Key learnings

Technical Insights

- **Optimizing Performance:** Combining Rust for computational tasks and Haskell for protocol logic resulted in a highly efficient and modular solution.

- **Zero-Knowledge Proofs:** Implementing both Plonk and Protostar provers deepened our expertise in tailoring proof systems for specific use cases.

- **Scalability:** The load distribution algorithm enabled parallel processing, improving throughput and reducing latency.

Community Feedback

- **Developer-Friendly Design:** Clear documentation, making integration smoother for developers.

- **Maestro Integration:** The seamless API management through Maestro simplified onboarding and scaling for users.

Operational Challenges

- **Transaction Balancing:** Ensuring ZK-aware transaction constraints while maintaining proof validity was a complex design challenge.

- **Load Distribution:** Managing parallel computation across servers required careful coordination to optimize performance and consistency.

Next steps for the product or service developed

Enhance and Expand:

- Extend functionality with additional features.
- Provide consistent updates and patches to improve the system's performance, security, and feature set.

Community Building:

- **Developer Support:** Continue to improve documentation, provide tutorials, and create more use case examples to help developers quickly adopt zkProver.

Monitoring and Feedback:

Establish metrics to monitor usage and effectiveness.
Implement regular updates based on user feedback and ecosystem needs.

Final thoughts/comments

The project represents a critical advancement in the adoption of ZK smart contracts on the Cardano blockchain. By providing a backend solution capable of generating zero-knowledge proofs, Prover enables blockchain scaling, unlocks new use cases, and enhances the user experience. This component is pivotal in bridging the gap for zkFold-based smart contracts and facilitating privacy-preserving, scalable solutions on Cardano.

While designed to support zkFold Symbolic smart contracts, the Prover and its corresponding API have far-reaching potential beyond our initial use cases. They offer valuable functionality for other blockchain applications, including identity solutions and confidential data sharing, both of which are currently in active development within the Cardano ecosystem.

The success of zkProver lays a foundation for further innovation, enabling developers to create more advanced, secure, and private applications. As the ecosystem grows, this project will contribute significantly to realizing the full potential of zero-knowledge protocols on Cardano, offering scalable, privacy-preserving solutions that could transform blockchain technology across industries.

Links to other relevant project sources or documents.

https://github.com/zkFold/project-catalyst-reports/tree/main/ZK%20Prover%20Backend
An API server component (Haskell):
https://github.com/zkFold/zkfold-prover-api/blob/main/src/Server.hs

Modules for event logging (Haskell):
https://github.com/zkFold/zkfold-prover-api/blob/8194c4736fad3f0089860e7596972e49d0d1b8a2/app/Main.hs#L16

A guide to setup and run the server:
https://github.com/zkFold/zkfold-prover-api/blob/main/README.md

Managed API access through the Maestro platform:
https://docs.gomaestro.org/Cardano/ZKProver/Introduction

Link to Close-out video

Close out video: https://youtu.be/Fa3zXcFA1ro