zkFold

# zkFold x zkPass: Bring zkPass to Cardano

zkfold.io

## Name of project and Project URL on IdeaScale/Fund

zkFold x zkPass: Bring zkPass to Cardano

https://milestones.projectcatalyst.io/projects/1200255

## Project Number

Project ID: 1200255

## Name of project manager

zkFold team

## Date project started

12 August 2024

## Date project completed

28 April 2025

## Objectives and status of the project

**Objective**: The objective of the project was to provide Cardano users with the ability to validate their personal data privately and securely using zkPass's zero-knowledge proof technology, by integrating the necessary smart contracts and tools into the Cardano blockchain. This initiative aimed to leverage zkPass's privacy-focused data oracle capabilities, which enable selective, secure, and private validation of diverse data types - such as legal identity, financial records, healthcare information, and more - from HTTPS-based web sources, without requiring users to disclose sensitive information to third parties. By implementing zkPass proof verification on Cardano, the project sought to provide Cardano users with seamless access to these features directly from their wallets, enhancing privacy and security within the Cardano ecosystem.

**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. Create documentation

**KPI Challenge**

The KPI challenge for Milestone 1 was to produce clear, comprehensive, and developer-friendly documentation that enables seamless integration of zkPass on Cardano. The success of this milestone relied on ensuring that the smart contract specification and API documentation were detailed, accessible, and practical, allowing developers to understand and implement zkPass functionality effectively without ambiguity or confusion.

**How the KPI was Addressed**

- **Clarity and Completeness**: The documentation underwent multiple reviews by both technical and non-technical team members to ensure it was thorough and easy to understand. Feedback was iteratively incorporated to refine explanations, eliminate jargon, and cover all essential aspects of the smart contract and API, such as what the smart contract does and how the API supports developers.
- **Usability**: The API documentation included practical examples, use cases, and step-by-step guides to demonstrate how developers can integrate zkPass into their Cardano projects. This ensured the documentation was actionable and directly supported developers in using zkPass effectively.
- **Accessibility**: Both the smart contract specification and API documentation were provided in PDF format, ensuring they are easily distributable and accessible across different platforms and devices, meeting the needs of a wide developer audience.
- **Version Control**: The documentation was versioned to track updates and changes over time. This allowed developers to reference the most current information and understand the evolution of the smart contract and API, maintaining consistency and reliability.

These measures collectively ensured that the smart contract specification and API documentation met the project's standards for clarity, usability, and developer support, laying a strong foundation for the successful adoption of zkPass on Cardano.

## 2. Implement the smart contract prototype

**KPI Challenge**

The KPI challenge for this milestone is to develop a fully functional and high-quality smart contract prototype in Haskell that effectively implements the zkPass oracle functionality, meeting the project's specifications and the operational requirements of the Cardano blockchain. The success of this milestone hinges on delivering a prototype that is not only operational but also secure, efficient, and well-suited for future development and integration within the Cardano ecosystem.

**How the KPI was Addressed**

- **Functionality**: The smart contract prototype was built to encompass all key features required for the zkPass oracle, including secure data validation, zero-knowledge proof verification, and smooth interoperability with the Cardano blockchain. These components were meticulously designed to align with the project's technical requirements, ensuring the prototype fulfills its intended purpose.
- **Code Quality**: Leveraging Haskell's strengths in functional programming, the code was developed to be clean, modular, and thoroughly documented. This high standard of code quality enhances maintainability, reduces technical debt, and supports seamless collaboration or future enhancements.
- **Testing**: A robust testing strategy was employed, featuring a comprehensive set of unit tests to validate individual functions and integration tests to confirm proper interaction with the Cardano blockchain. This rigorous testing ensured the prototype's reliability and correctness under various conditions.
- **Performance**: The prototype was fine-tuned for optimal performance, ensuring that its operations are efficient and remain within the resource constraints of the Cardano blockchain. This optimization minimized execution costs and maximized responsiveness, critical for real-world deployment.
- **Security**: Security was a top priority, with the implementation incorporating best practices such as input validation and safeguards against common vulnerabilities. A thorough security review by experts further strengthened the prototype, mitigating risks and ensuring robustness.
- **Documentation**: The prototype was delivered with detailed inline comments and external documentation, clearly outlining its functionality and usage. This

comprehensive documentation empowers other developers to understand, utilize, and build upon the codebase with ease.
- **Version Control**: Development was managed using version control systems, enabling meticulous tracking of changes, fostering team collaboration, and preserving a clear history of the codebase. This structured approach supported an organized and efficient workflow.
- **Feedback and Iteration**: The prototype was presented to stakeholders for evaluation, and their insights were integrated through iterative refinements. This feedback loop ensured the final deliverable aligned with both technical goals and stakeholder expectations.

By addressing these KPIs, the smart contract prototype in Haskell was successfully developed to meet the project's standards for functionality, quality, and security, laying a strong foundation for the zkPass oracle's integration into the Cardano blockchain.

## 3. Implement off-chain code prototype

**KPI Challenge**
The challenge for this milestone was to create a robust, efficient, and secure off-chain code prototype in Haskell capable of accurately constructing transactions for the smart contract. The prototype needed to meet technical requirements, integrate effectively with the Cardano blockchain, and align with stakeholder expectations, all while maintaining high standards of quality, performance, and documentation.

**How the KPI was Addressed**

- **Functionality**
  The off-chain code was carefully implemented to construct transactions that meet the smart contract's specific requirements. This ensured that all transaction-building operations, such as data validation and proof verification, were accurately executed, enabling reliable interaction with the Cardano blockchain.
- **Code Quality**
  Written in Haskell, the code leveraged the language's functional programming features to achieve clarity, modularity, and maintainability. It was structured to be clean and well-documented, facilitating collaboration and future enhancements by other developers.
- **Testing**
  A thorough testing framework was established, including unit tests to verify the correctness of individual functions and integration tests to ensure compatibility with

the smart contract and the Cardano blockchain. This approach confirmed the prototype's reliability across various use cases.

- **Performance**
  The code was optimized to operate efficiently, reducing computational overhead and minimizing transaction costs on the Cardano blockchain. This focus on performance ensured the prototype adhered to the blockchain's resource constraints while maintaining responsiveness.
- **Security**
  Security best practices were followed, including rigorous input validation and safeguards against common vulnerabilities. A detailed security review was conducted to mitigate risks, ensuring the integrity and safety of the constructed transactions.
- **Documentation**
  The prototype included comprehensive inline comments and external documentation, detailing its functionality, usage, and integration points. This thorough documentation enabled developers to understand and work with the code effectively.
- **Version Control**
  A version control system was utilized to manage the codebase, providing a clear history of changes, supporting collaboration, and maintaining an organized development process.
- **Feedback and Iteration**
  The off-chain code was reviewed by stakeholders, and their feedback was incorporated through iterative refinements. This collaborative process ensured the prototype met both technical and stakeholder expectations.

## 4. Implement client-side building transactions

**KPI Challenge**

The KPI challenge for this milestone was to successfully adapt the Haskell-based off-chain transaction building code into a browser-compatible version that maintains the same functionality, security, and performance standards. The adapted code needed to be reliable, efficient, and user-friendly while operating within the constraints of a browser environment, ensuring seamless integration with Cardano-based applications and providing a robust solution for developers and end-users.

**How the KPI was Addressed**

- **Functionality**: The adapted code retained the full functionality of the original Haskell-based off-chain code, enabling accurate construction of transactions for the zkPass smart contract. It was meticulously tested to ensure it interacted correctly with the Cardano blockchain, preserving the integrity of transaction-building operations in the browser environment.
- **Browser Compatibility**: The code was transpiled using GHCJS to generate JavaScript, ensuring compatibility with major browsers (Chrome, Firefox, Safari, Edge). Extensive cross-browser testing was conducted to confirm consistent behavior and functionality across different platforms.
- **Performance Optimization**: The code was optimized to minimize computational overhead, leveraging WebAssembly for performance-critical components where necessary. This ensured fast transaction building, even on resource-constrained devices, while adhering to the Cardano blockchain's efficiency requirements.
- **Security**: Robust security measures were implemented, including HTTPS for all blockchain communications and secure handling of sensitive data (e.g., private keys) using browser-based secure storage solutions. A security review was conducted to mitigate risks and protect against vulnerabilities like cross-site scripting (XSS).
- **Code Quality**: The adapted code maintained high standards of modularity, readability, and maintainability. It was structured to be clean and well-documented, facilitating future updates and collaboration among developers.
- **Testing**: A comprehensive testing suite was implemented, including unit tests for individual functions and integration tests to verify compatibility with the Cardano blockchain and front-end systems. Automated tests ensured reliability across various browser environments and use cases.
- **Documentation**: Detailed documentation was provided, including setup instructions, usage examples, integration guides, and troubleshooting tips. This empowered developers to easily adopt and integrate the client-side code into their applications.
- **Version Control**: The development process utilized version control systems to track changes, ensure collaboration, and maintain a clear history of the codebase, supporting an organized and efficient workflow.

By addressing these KPIs, the client-side transaction building code was successfully developed to operate reliably and efficiently in a browser environment, meeting the project's standards for functionality, security, and usability while enabling seamless integration with Cardano-based applications.

## 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.

**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 zkPass team to refine specifications and receive iterative feedback.

**Engagement:**

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

## Key learnings

### Technical Insights

- **Robust Haskell Development**: The use of Haskell enabled the creation of modular, maintainable, and secure codebases across smart contracts, off-chain code, and browser adaptations, leveraging functional programming strengths.

- **Performance and Resource Optimization**: All components - from smart contracts to client-side code were optimized to meet Cardano's resource constraints, incorporating techniques for browser performance.

- **Security-First Design**: Across all layers, security best practices were implemented, including thorough input validation, secure browser storage, HTTPS communications, and expert reviews.

- **Comprehensive Testing Strategy**: Each milestone included extensive unit and integration tests, ensuring reliability across smart contracts, transaction logic, and browser interactions, which is crucial for blockchain deployment.

### Community Feedback

- **Iterative Documentation Improvement**: Developer feedback significantly shaped the documentation quality through continuous updates and simplifications based on usability and clarity.

- **Stakeholder-Informed Refinement**: Smart contract, off-chain, and client-side prototypes were iteratively improved based on reviews and input from technical stakeholders.

- **Open Feedback Channels**: A structured feedback mechanism (e.g., dedicated channel) allowed to contribute insights and suggestions, directly influencing technical and usability improvements.
-

### Operational Challenges

- **Browser Environment Constraints**: Adapting Haskell-based logic for browser compatibility involved overcoming limitations in JavaScript performance, secure data handling, and cross-browser consistency.

- **Resource-Constrained Optimization**: Ensuring all components performed efficiently within Cardano's execution cost limits required careful profiling, optimization, and sometimes redesign.

- **Coordination Across Teams**: Synchronizing progress on on-chain, off-chain, and front-end components, along with documentation and stakeholder reviews, demanded tight project management and version control discipline.

- **Maintaining Documentation Alignment**: Keeping documentation accurate and versioned across evolving codebases required constant synchronization and added overhead to the release process.

## 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 zkPass.

**Monitoring and Feedback:**

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

## Final thoughts/comments

The zkPass integration project on Cardano marks a significant technical and strategic advancement in the blockchain ecosystem. By successfully delivering clear documentation, functional smart contracts, robust off-chain logic, and browser-compatible transaction-building tools, the project has laid a solid foundation for widespread adoption and developer engagement.

One of the most compelling aspects of this initiative is zkPass's unique approach to data oracle functionality. Unlike conventional oracles that depend on multi-signature validation and multiple data sources, zkPass leverages **Multi-party Computation (MPC)** to securely process any HTTPS-based communication. This innovation ensures **uniform data treatment, stronger data integrity**, and a **higher level of decentralization** across all data flows - an essential leap forward in oracle design.

The implications for Cardano are transformative:

- **New categories of dApps** can now emerge, fueled by the ability to securely verify user-owned or web2-sourced data on-chain.
- **Existing decentralized applications** stand to gain enhanced resilience and simplicity, eliminating complex oracle infrastructures without compromising on trust or security.
- **Developers benefit** from a cleaner integration process, backed by comprehensive documentation and reliable prototype implementations across the stack.

Overall, this project not only meets its KPIs but also introduces a **paradigm shift in how data is authenticated and used on-chain**, opening the door to a new wave of secure, decentralized, and user-centric applications on Cardano.

## Links to other relevant project sources or documents.

**Smart contract specification:**
https://docs.zkfold.io/other-zkapps/zkpass/
**API documentation:**
https://docs.zkfold.io/other-zkapps/zkpass/#api

**Smart contract prototype (zkPass oracle):**

https://github.com/zkFold/zkfold-base/blob/main/symbolic-cardano/src/ZkFold/Symbolic/Cardano/Contracts/ZkPass.hs

**A repository containing the frontend (client-side) code of the zkPass integration**:

https://github.com/zkFold/zkpass-client

**The core API functions of the integration are implemented here:**

https://github.com/zkFold/zkpass-client/blob/main/src/components/ZkPassApp.tsx

## Link to Close-out video

Close out video: zkpass demo