zkFold

Zero-Knowledge Proof Trustless P2P Fiat-to-Crypto On-Ramp for Cardano

zkfold.io

Name of project and Project URL on IdeaScale/Fund

ENCOINS x Anastasia Labs: Zero-Knowledge Proof Trustless P2P Fiat-to-Crypto On-Ramp for Cardano

https://milestones.projectcatalyst.io/projects/1100125

Project Number

Project ID: 1100125

Name of project manager

zkFold team

Date project started

11 March 2024

Date project completed

23 October 2025

Objectives and status of the project

Objective: The primary objective of this project is to build a trustless, decentralized P2P fiat-to-crypto on-ramp using Cardano smart contracts and zero-knowledge proofs. The protocol enables two parties - one with ADA and one with fiat - to transact securely without relying on intermediaries, custodians, or oracles. It aims to ensure that cryptocurrency sellers can safely deposit funds into a smart contract, while buyers can unlock those funds only after cryptographically proving fiat payment.

To achieve this, the project introduces a new mechanism where off-chain events (like bank transfers or payment receipts) can be proven on-chain using zero-knowledge circuits, enabling truly decentralized compliance and verification. The system leverages zkFold Symbolic to author smart contracts that are readable, verifiable, and efficient on Cardano. A key goal is to make advanced ZK tooling accessible to everyday developers, removing the need for deep cryptographic expertise.

The project includes the development of ZK circuits to match HTTPS API responses, verify JSON patterns, and validate digital signatures from fiat payment providers. It also delivers a modular on-chain smart contract, a reusable off-chain transaction builder, and a clean, role-based Web UI for both sellers and buyers. Usability, security, and clarity are prioritized to make the user experience smooth and trustworthy.

Another core objective is to ensure modularity and reusability so that the protocol can be extended to support multiple payment providers, identity layers, and potentially a reverse off-ramp flow. Interoperability with the broader Cardano ecosystem (e.g., wallet connectors, UTXO logic, CIP standards) is built in by design. The system is also structured to support testability, auditability, and future integration into SDKs for use by dApps or DeFi platforms.

Finally, the project aims to show that Cardano can support fully decentralized ZK-powered financial applications, leading to increased adoption, reduced custodial risk, and more inclusive access to the crypto economy. By making the protocol open-source

and developer-friendly, it contributes directly to the advancement of Cardano as a privacy-aware, ZK-ready blockchain platform.

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

List of challenge KPIs and how the project addressed them

1. From HTTPS requests to zero knowledge proofs

KPI Challenge

The main challenge of this milestone was to bridge the gap between off-chain HTTPS communication and on-chain zero-knowledge proof verification in a way that is both secure and generalizable. Since HTTPS responses are external, unverifiable data from the perspective of a blockchain, transforming them into reliable zero-knowledge proofs required building logic that could validate the structure and authenticity of such responses within an arithmetic circuit. This involved two core difficulties: (1) matching flexible and potentially complex JSON response patterns inside a constrained ZK circuit environment, and (2) verifying cryptographic signatures on those responses to ensure trustless validation without relying on centralized intermediaries. Ensuring generality and reusability of the modules, while optimizing for circuit performance and developer usability, added further complexity. The implementation also had to align with the zkFold Symbolic language model and be compatible with its symbolic representation of contract logic.

How the KPI was Addressed

- **Modules for Arithmetic Circuit Generation for JSON Pattern Match**
  A dedicated module was implemented in Haskell to translate JSON pattern-matching logic into arithmetic circuits that can be consumed by the zkFold Symbolic system. The pattern matcher was designed to be flexible, allowing developers to define expected structures (e.g., fields, value types, nested keys) that the API response must adhere to. Each pattern is encoded as a series of circuit constraints that evaluate whether a given input matches the expected format. This module enables the creation of zero-knowledge proofs that attest

not only to the presence of a valid API response but also to its structural integrity, a foundational requirement for building trustless off-chain-to-on-chain bridges.

- **Modules for Checking Signatures on HTTPS Responses and Circuit Generation**
  To ensure authenticity, another module was developed to parse and verify digital signatures included in HTTPS responses. This logic transforms public key checks and cryptographic verification steps into arithmetic constraints, enabling the resulting zero-knowledge proofs to assert that the response was signed by a known and trusted party (e.g., a payment provider or API operator). This feature is crucial to establishing trust in the off-chain message without relying on any central authority or oracle. Signature formats (e.g., ED25519, ECDSA) were abstracted for modularity, allowing for future extensibility and broader support. The module integrates tightly with the JSON matcher, completing a trust-minimized pipeline from API call to verifiable, provable input.

- **Composable Design with zkFold Symbolic**
  Both modules were designed to integrate seamlessly with the zkFold Symbolic language and toolchain. This ensures that developers can use these new primitives alongside existing zkFold constructs when building complex smart contracts. The composability of the system allows a wide variety of use cases - such as fiat on-ramps, document verification, or external data attestations - to be expressed simply, using familiar high-level abstractions. By embedding HTTPS interaction into symbolic contract flows, zkFold significantly lowers the barrier for secure off-chain interaction in ZK-enabled Cardano applications.

- **Performance Optimization and Test Coverage**
  During implementation, emphasis was placed on minimizing constraint count and optimizing the way pattern-matching logic was encoded into circuits. The

modules were thoroughly tested with multiple JSON response scenarios and edge cases to ensure that the matcher and signature checker behave predictably and securely. This allowed the team to validate performance metrics and confirm that the logic remains efficient even as the size or complexity of JSON patterns increases. These efforts ensure the modules are production-ready and suitable for integration in real-world smart contract apps like the trustless P2P on-ramp.

.

2. Smart contract specification

KPI Challenge

The key challenge of this milestone was to define a clear and exhaustive specification for a non-trivial smart contract that enables a trustless, proof-based P2P fiat-to-crypto on-ramp. Given that the contract must securely handle user funds, validate off-chain actions via zero-knowledge proofs, and ensure that all possible transaction paths are covered, the specification needed to account for every valid and invalid state transition. A further challenge was aligning symbolic-level contract logic (as used in zkFold) with actual transaction behaviors on Cardano's EUTXO model, ensuring that the specification was precise enough to guide implementation and testing. To support this, the deliverable also required a reliable set of test scripts for constructing and simulating smart contract interactions, validating both the structure and execution logic of the proposed flows.

How the KPI was Addressed

- **Comprehensive Smart Contract Specification Document**

  A technical document was created to formally define the structure and logic of the trustless P2P on-ramp smart contract. It outlines all contract states, roles (seller, buyer), input requirements, and expected behaviors at each stage - including deposit, fiat payment verification, zero-knowledge proof submission, and fund withdrawal. The document also specifies the required data formats, on-chain validators, and the rules governing transitions between contract states. This specification acts as a blueprint for development and ensures consistency across implementation, auditing, and integration.

- **Design of All Contract Behaviors and Edge Cases**

  Special care was taken to document not only the happy path (i.e. successful transactions) but also all edge cases - such as missing proofs, expired time windows, invalid data, or malicious inputs. These scenarios were captured clearly in the specification to ensure the final smart contract can handle unexpected or adversarial behavior without compromising funds or logic. The contract model follows a strict state machine approach, where each transaction must provide valid inputs and satisfy expected predicates, enforced by ZK verification when needed.

- **Development of Test Scripts for Contract Simulation**

  A suite of TypeScript scripts was implemented to simulate smart contract interactions on the Cardano testnet or within a local dev environment. These scripts cover transaction construction, submission, and validation workflows, aligned with the behaviors defined in the specification. The test suite ensures that developers can rapidly test and iterate on contract logic, and also supports regression testing for future updates. Through these scripts, the specification is

not only theoretical but immediately actionable and testable.

- **Alignment with zkFold Symbolic and Future Circuit Integration**
  The specification was designed with zkFold Symbolic in mind, ensuring that future integration of zero-knowledge proofs (e.g., for HTTPS verification or identity proofs) would be supported without requiring a redesign. The structure of transactions, datum, and redeemer logic was made compatible with ZK circuits, making the contract forward-compatible and modular. This foresight reduces future development overhead and ensures that the trustless P2P flow remains extensible and maintainable.

This milestone successfully delivered a detailed smart contract specification and supporting test scripts for a trustless P2P on-ramp. It ensures all behaviors - normal and edge cases - are fully defined, tested, and ready for implementation. The outputs provide a solid foundation for secure, ZK-enhanced contract development on Cardano.

3. The on-chain code implementation

KPI Challenge

The key challenge of this milestone was to implement secure and verifiable on-chain logic for a non-custodial, zero-knowledge-based P2P fiat-to-crypto on-ramp using zkFold Symbolic. The contract needed to coordinate between off-chain actions (fiat payments), cryptographic proof submission, and on-chain fund release - all while ensuring safety for both the buyer and the seller in a trustless environment. Given the expressive power of zkFold Symbolic and the constraints of Cardano's eUTXO model, the on-chain code had to enforce strict validation conditions, preserve clean state transitions, and integrate future-proof mechanisms for zero-knowledge proof verification. It was also essential

that the implementation remain modular and composable to support additional use cases and evolving zkFold language features.

How the KPI was Addressed

- **Implementation of On-Chain Modules in Haskell Using zkFold Symbolic**
  The core smart contract logic was implemented in Haskell using the zkFold Symbolic language. The contract handles deposit validation, proof verification triggers, timeouts, and safe fund unlocking through symbolic state transitions. This modular implementation allows seamless integration with zkFold's off-chain prover and verifier workflows while staying compatible with Cardano's UPLC execution model. By encapsulating each logical phase into clear symbolic constructs, the contract remains readable, auditable, and reusable.

- **Secure Enforcement of Trustless P2P Flow**
  The smart contract enforces a strictly non-custodial model, where the seller locks funds, and the buyer must provide a valid zero-knowledge proof - typically derived from a verified fiat payment - to unlock them. The contract validates these conditions before allowing fund transfers, protecting against premature withdrawal or fraud. It also supports timeout logic to refund the seller if the buyer fails to complete the process in time, ensuring fairness and capital efficiency.

- **Designed for Extensibility and Modularity**
  The codebase was written in a modular fashion to facilitate the future addition of features such as reputation systems, dispute resolution, or multi-asset support. Components such as proof validation, payment confirmation, and contract expiry are separated into reusable building blocks. This modularity also allows the core on-chain logic to be reused for other zero-knowledge contract applications

beyond the P2P on-ramp use case.

- **Tested for Logical Consistency and Integration Readiness**
  The on-chain modules were tested to ensure logical soundness and readiness for integration with off-chain components like HTTPS response verification and ZK proof generation. These tests validate that each contract state behaves as expected and transitions correctly under normal and adversarial scenarios. This ensures that the implementation meets the requirements defined in the previously delivered smart contract specification and is aligned with zkFold's broader proof-generating infrastructure.

This milestone delivered a secure, modular, and forward-compatible on-chain smart contract for the trustless P2P on-ramp using zkFold Symbolic. The implementation enforces all key flow controls - deposits, timeouts, and proof-based unlocking - while maintaining extensibility for future upgrades. It is now ready for integration with the ZK proof pipeline and off-chain verification logic.

4. Implement Web UI

KPI Challenge

The core challenge for this milestone was to design and implement a simple, intuitive, and secure Web UI that enables users to interact with the trustless P2P on-ramp smart contract. The interface needed to support both seller and buyer roles, guiding them through key steps like depositing ADA, submitting fiat payment information, and generating or submitting zero-knowledge proofs. In addition to handling blockchain interactions, the UI had to provide feedback on contract state transitions, simulate transaction flows, and prepare users for steps involving off-chain data (like payment confirmations and HTTPS responses). Ensuring ease of use while interacting with a

zero-knowledge-enabled smart contract system - without overwhelming users with technical complexity - was a crucial aspect of the challenge.

How the KPI was Addressed

- **Development of a Minimal, Functional Web UI**

  A lightweight frontend was developed using modern web technologies to support direct interaction with the P2P on-ramp smart contract. The interface allows sellers to deposit funds and buyers to initiate or complete a transaction, including proof submission when applicable. The UI was designed with clarity and simplicity in mind, presenting each role's actions in a step-by-step flow to reduce user error. All essential logic - such as contract state display, address handling, and transaction feedback - was implemented in a developer-friendly and extensible manner.

- **Smart Contract Integration via Wallet APIs**

  The Web UI integrates with Cardano wallet connectors (e.g., Nami, Eternl) to allow users to interact with the contract from their browser. Users can sign transactions, check wallet balances, and trigger on-chain actions without needing to leave the UI. The app uses a backend or smart contract indexer to track transaction status and present clear success/failure outcomes. This seamless integration ensures a smooth end-to-end experience without requiring users to interact directly with CLI tools or blockchain explorers.

- **Role-Based Interface Logic and State Awareness**

  The frontend distinguishes between the seller and buyer roles, offering tailored views and actions depending on the current stage of the transaction lifecycle. For example, sellers are guided through ADA deposit steps, while buyers are

prompted to submit a zero-knowledge proof after confirming fiat payment. The UI monitors smart contract state and adapts dynamically to guide the user through the correct next action, helping to minimize mistakes or miscommunications in P2P interactions.

- **Designed for Iteration and Scalability**
  While the current implementation is simple, it was built with modular components and scoped styles to allow for easy upgrades. Future additions - such as identity integration, ZK proof generation directly from the browser, or fiat gateway widgets - can be added without rewriting the core interface logic. This ensures that the Web UI will continue to evolve alongside the underlying protocol and feature set.

This milestone successfully delivered a simple yet functional Web UI that enables sellers and buyers to interact with the trustless P2P on-ramp contract. The interface is intuitive, supports both user roles, and integrates seamlessly with Cardano wallets. It forms a strong foundation for more advanced ZK and fiat gateway features in the future.

5. Implement the off-chain code

KPI Challenge

The main challenge of this milestone was to build an off-chain component that reliably handles the construction, balancing, and submission of transactions interacting with the trustless P2P on-ramp smart contract. In Cardano's eUTXO model, off-chain code plays a critical role in preparing correct transactions that conform to the smart contract logic, ensuring valid inputs, datums, redeemers, collateral, and execution units. Since this dApp involves conditional unlocking of funds based on cryptographic proofs, the off-chain code needed to manage dynamic contract states, coordinate user roles, and

prepare precise transactions under changing conditions (e.g. timeouts or proof submissions). Building a reusable, modular off-chain system that integrates smoothly with the UI and CLI tooling - while remaining maintainable and extensible for future upgrades - was essential.

How the KPI was Addressed

- **Development of Transaction Construction Logic**
  The off-chain component was implemented to build valid transactions for both seller and buyer flows. It handles selecting UTXOs, setting appropriate datums and redeemers, attaching scripts, and preparing transactions for submission through wallet APIs or CLI tools. Special logic was included to detect contract state (e.g. awaiting proof, timeout reached) and adapt the transaction accordingly. This ensures that both happy-path and fallback scenarios are fully supported.

- **Transaction Balancing and Fee Calculation**
  The module includes automatic balancing and fee estimation, ensuring that users don’t overpay or encounter failed submissions due to underfunded transactions. ADA and token inputs are dynamically calculated based on the contract’s needs and the user’s wallet state. Outputs are structured to return change properly and maintain compliance with Cardano protocol parameters. This reduces friction for end users and ensures a smoother experience.

- **Integration with ZK Proof Flow and UI**
  The off-chain code was designed to integrate with the zero-knowledge proof generation and submission flow. Once a buyer generates a valid proof (off-chain), the transaction builder constructs the corresponding unlock transaction,

embedding the proof into the correct on-chain format. This flow bridges the HTTPS verification logic and the contract logic, acting as a trustless relay between off-chain computation and on-chain enforcement.

- **Reusable, Extensible Architecture**
  The off-chain logic is modular and written with future use cases in mind, such as multi-party coordination, escrow models, or fiat on-ramp extensions. The components are structured to be reused by both backend systems and Web UI interfaces. This flexibility supports wider ecosystem adoption of the trustless P2P protocol and makes the off-chain code a critical piece of the broader zkFold smart contract toolkit.

This milestone successfully delivered the off-chain logic for building, balancing, and submitting transactions to the trustless P2P on-ramp smart contract. It handles the complexity of dynamic contract states, proof embedding, and fee calculation while integrating smoothly with the broader user experience. The off-chain component ensures secure, predictable execution for both sellers and buyers.

List of project KPIs and how the project addressed them

Quality

KPI: Deliver high-quality, modular, and maintainable smart contract code (on-chain and off-chain) that meets formal specifications.

Addressed:

- The on-chain logic was implemented using zkFold Symbolic in a modular and composable manner, with each contract behavior clearly separated for readability and future extensibility.
- The off-chain code was developed to dynamically handle contract states, build valid transactions, and manage wallet integration with a focus on robustness and code reusability.
- The CLI and Web UI components were built with clean abstractions, ensuring consistency between user interaction layers and smart contract logic.
- Specification documents were written before implementation, ensuring that all code adhered to formally defined behaviors, data formats, and role-based flows.

Testing

KPI: Ensure comprehensive testing coverage of smart contract behavior, off-chain logic, and user interaction flows.

Addressed:

- A set of TypeScript scripts was created to simulate and test all critical transaction flows, including both happy-path and edge-case scenarios.
- On-chain code modules were validated against the smart contract specification to ensure logical consistency and proper state transitions.

- Off-chain transaction construction was tested for multiple contract stages (e.g. deposit, timeout, proof submission), including balancing, fee handling, and redeemer accuracy.
- Integration testing between the Web UI and wallet APIs ensured real users could reliably interact with the smart contract through the frontend interface.

Collaboration

KPI: Align implementation with the broader zkFold ecosystem and ensure interoperability between components through effective team coordination.
Addressed:

- The team collaborated across contract, UI, and cryptographic proof modules to ensure smooth end-to-end flows from HTTPS response to ZK proof to smart contract interaction.
- The off-chain logic was built to interface directly with zkFold Symbolic circuits, supporting the submission of zero-knowledge proofs derived from external events (e.g. fiat payment verification).
- The smart contract specification acted as a shared foundation across the team, ensuring that on-chain, off-chain, and UI developers implemented features in sync.
- Ecosystem alignment with Cardano tooling (wallet connectors, transaction builders, UTXO management) ensured that the solution could be extended and adopted by external developers and partners.

Key achievements

1. **End-to-End Implementation of a Trustless P2P Fiat-to-Crypto Protocol**
   The project successfully delivered a fully functional smart contract-based system that enables buyers and sellers to exchange fiat and cryptocurrency without intermediaries. It integrates zero-knowledge proof logic, smart contracts, off-chain components, and a user-facing Web UI into a cohesive flow. This achievement lays the foundation for a new class of decentralized, compliant on-ramps on Cardano.

2. **Modular Smart Contract Built with zkFold Symbolic**
   The on-chain code was implemented using zkFold Symbolic, enforcing a non-custodial escrow model with provable fiat payment verification. The contract logic was modular and extensible, supporting timeouts, cryptographic proof validation, and state-controlled fund unlocking, making it both auditable and adaptable to future requirements.

3. **Zero-Knowledge Integration with HTTPS Response Verification**
   The team implemented arithmetic circuits capable of matching HTTPS API responses against JSON patterns and verifying their cryptographic signatures inside zero-knowledge circuits. This enabled a novel use case where buyers can prove fiat payments off-chain and unlock funds on-chain in a trust-minimized, privacy-preserving way - without relying on oracles or custodians.

4. **Robust Off-Chain Code and Transaction Builder**
   A Haskell-based off-chain component was delivered to construct, balance, and submit transactions aligned with smart contract logic. It handles dynamic contract states, validates ZK proof conditions, manages fees and collateral, and integrates with user wallets - ensuring secure and efficient execution across

different transaction flows.

5. **User-Friendly Web UI with Role-Based Flow**
   A frontend interface was developed to guide both sellers and buyers through the entire P2P transaction process, from deposit to proof submission. It integrates with Cardano wallets and dynamically adapts to contract state, providing a clean, accessible experience for non-technical users.

6. **Comprehensive Smart Contract Specification and Test Scripts**
   A detailed contract specification was authored, capturing all possible behaviors, data formats, and edge cases. Supporting test scripts were developed in TypeScript to simulate transaction flows and validate logic, ensuring consistency between the specification and the actual on-chain behavior.

7. **ZK Circuit Modules for Future-Proof Extensibility**
   The core cryptographic and pattern-matching logic was implemented in reusable modules, designed to be integrated into other zkFold-based applications beyond this use case. This makes the solution not just a single dApp, but a toolkit for future ZK-enabled Cardano applications.

Key learnings

Technical Insights

- **Zero-knowledge circuits can be meaningfully applied to real-world, user-facing dApps.**

  The integration of HTTPS response pattern matching and digital signature verification into zero-knowledge circuits showed that advanced cryptographic features are now accessible in practical smart contract workflows. This milestone demonstrated that ZK circuits can be used not just for privacy, but also to create trustless bridges between off-chain actions and on-chain outcomes.

- **UPLC constraint limitations require careful circuit optimization.**

  Translating complex logic like JSON parsing and cryptographic checks into arithmetic circuits revealed performance and constraint-size challenges. The team addressed this by designing modular, reusable circuit components optimized for constraint efficiency - critical for scaling ZK applications on Cardano.

- **zkFold Symbolic enables readable and maintainable on-chain code for ZK applications.**

  The use of zkFold Symbolic significantly improved developer productivity and code clarity compared to traditional Plutus approaches. Symbolic state transitions allowed a clean separation of contract phases (e.g., deposit, proof, unlock), making the contract easier to debug, test, and extend.

Community Feedback

- **Developers value ZK functionality but expect plug-and-play experiences.**
  Early feedback from Cardano builders highlighted excitement about ZK capabilities, but also emphasized the need for abstraction and simplicity. The CLI tool, off-chain logic, and UI were all shaped by this expectation - to ensure developers don't need to understand circuit-level cryptography to use the system.

- **Wallet integration and Web UI usability are top adoption drivers.**
  Community members stressed that seamless wallet integration (Nami, Eternl) and intuitive UI flows are essential for adoption beyond the developer audience. This feedback led to improved transaction feedback, role-specific interfaces, and clearer status messaging within the UI.

- **ZK developer tooling is still maturing, and documentation is essential.**
  Contributors requested more visual explanations, example scripts, and usage guides. This prompted the team to document the contract specification thoroughly and provide test scripts and UI flows that make it easier for new teams to integrate or extend the protocol.

Operational Challenges

- **Coordinating cross-component development requires strong architectural alignment.**
  The project required simultaneous development of smart contracts, ZK circuits, off-chain infrastructure, and frontend UI. Early investment in a shared contract specification was critical to keeping all teams aligned and avoiding integration delays.

- **ZK development introduces steep testing complexity.**
  Testing flows that span cryptographic proof generation, off-chain data handling, and on-chain verification introduced new challenges. The team had to build custom test cases and simulate both successful and adversarial scenarios to ensure robustness across the full stack.

- **Balancing rapid iteration with correctness and security is difficult but necessary.**
  The team worked in agile cycles, delivering incremental value while continuously validating against the formal contract specification. This approach helped manage scope and risk, especially given the project's emphasis on financial correctness and trustlessness.

Next steps for the product or service developed

1. Complete End-to-End Proof Integration and Automation

Now that the ZK circuits and on-chain verification logic are implemented, the next step is to fully integrate the proof generation and submission workflow. This includes automating the process of generating a ZK proof from an HTTPS API response (e.g., a fiat payment confirmation), embedding the proof into a transaction, and submitting it through the Web UI. Making this flow seamless and user-friendly is critical for non-technical users. This will mark the transition from functional prototype to complete trustless system.

2. Launch Testnet MVP with Real Users

Deploy the complete application to a Cardano testnet and invite selected users (sellers and buyers) to interact with the full flow - from wallet connection to deposit, proof, and withdrawal. This MVP launch will allow for valuable feedback on usability, edge cases, and potential friction points. Collecting metrics on transaction success rate, proof verification, and time-to-completion will guide product refinements. A public testnet version also signals readiness to ecosystem partners and developers.

3. Integrate Fiat Payment Providers and Signature Schemes

Work with fiat payment services (e.g., SEPA, Revolut, PayPal Business) that offer APIs and digital signature mechanisms to produce verifiable payment receipts. Each provider will require tailored integration logic for parsing their JSON responses and verifying their signatures. Adding multi-provider support increases the flexibility and coverage of the protocol across different markets. This step ensures the on-ramp is adaptable to real-world payment ecosystems.

Final thoughts/comments

The Trustless P2P On-Ramp project demonstrates that advanced cryptographic tools like zero-knowledge proofs can be integrated into real-world smart contract applications in a way that is secure, modular, and user-friendly. Through a carefully designed system of on-chain contracts, off-chain logic, ZK circuit modules, and a clean Web UI, we've proven that it's possible to build a fully decentralized fiat-to-crypto gateway - without intermediaries or custodians. Each milestone contributed to making this vision a reality, from defining a robust contract specification and implementing zkFold Symbolic logic, to building testable transaction flows, circuit-powered proof validation, and intuitive user interactions.

The project also highlighted important learnings: that developer usability is just as critical as protocol correctness, and that cross-component alignment is essential for ZK-based applications to succeed in production. Our modular approach ensures this protocol can evolve to support multiple fiat providers, new identity layers, or even trustless off-ramps in the future. With community feedback already validating demand for this type of tool, and strong alignment with the broader zkFold and Cardano ecosystems, the foundation has been laid for production deployment and wide adoption.

This work positions Cardano as a blockchain capable not only of scaling securely - but also of innovating at the edge of privacy, compliance, and financial sovereignty. The Trustless P2P On-Ramp is more than just a dApp; it's a new ZK-enabled primitive for decentralized finance on Cardano.

Links to other relevant project sources or documents.

Smart contract specification:

https://docs.zkfold.io/introduction/what-is-zkfold-symbolic

Developer documentation for contributors to zkFold Symbolic:
https://hackage.haskell.org/package/zkfold-base-0.1.0.0/candidate.

Modules for Trustless P2P On-ramp smart contract can be found in the repo:

https://github.com/zkFold/p2p-onramp/blob/vlasin-catalyst-m3-version2/src/ZkFold/Cardano/UPLC/OnRamp.hs

The main output of this Milestone is Code for Web UI which can be found at:

https://github.com/Anastasia-Labs/money-kit-ui

At this link we are providing a website showcasing the UI:

https://anastasia-labs.github.io/money-kit-ui/

Link to Close-out video

Close out video: https://youtu.be/Ch6VSv1s_cg?si=GElSOwVTtqIPyhxW