Practical Report

Proof of Innocence in Cardano

Introduction

This document is the practical continuation of the research documented presented in the first milestone of the Catalyst grant “Zero-Knowledge Proof of Innocence on Cardano - Encoins + Módulo P + Eryx”. First we will revisit the protocol design, then the process of implementing the protocol and its features, and the testing made to check the correct implementation of the project. There were no changes in the main code during this stage that seem worth noting to us. We think this is because we developed all the parts of the code using test-driven development, so the code was tested from the beginning.

Proof of Innocence design

The main purpose of the Proof of Innocence project is to anonymously ban interactions from users related to malicious activities on the blockchain. Usually, the more straightforward use case of this feature is in the context of a Mixer, or any other protocol that aims to have privacy on the blockchain using an unlinkable deposit-withdraw scheme. Technically, by using a Zero-Knowledge proof we claim that our deposit or interaction with a protocol is not related to a previous malicious transaction. We prove that we aren't part of the set of banned transactions without specifying exactly which transaction is being proven, this means anonymacy due to the properties of the Zero-Knowledge proof.

We use a proof of exclusion to show that some user is not related to any malicious transaction recorded by an oracle of the mixer's choice. The main data structure used to achieve this is the *sparse Merkle tree*.

We implemented the proof of innocence as a smart contract that could be used as an input for another smart contract, the latter corresponding to a privacy service. This means that we designed the contract to be used in a modular way by different services that use private deposit-withdraw schemes.

Protocol elements

The protocol relies on several components:

Circuit

The mechanism to provide the proof of exclusion with the sparse Merkle tree as a Circom language circuit.

Oracle

An on-chain oracle to publish the collection of banned transactions as the root hash of its sparse Merkle tree representation.

PoI validator (“smart contract”)

Performs the on-chain verification of the proof of exclusion, using the corresponding oracle.

Implementation

The PoI CLI (command line interface)

The major change of this milestone is the addition of a command line interface application that mainly acts as the tool that a user of the protocol would employ, but also allows us to test the functionalities of PoI on-chain. It includes a working example which you can run only using the provided commands, via the **`poi-cli`** program..

1. **`create oracle`** command - You can create an oracle and publish the root hash of the banned transactions. This transaction will create a unique token (thread token) to prevent UTxO spoofing, that UTxO will be owned by the wallet owner that created the Oracle instance.
2. **`update oracle`** command - By appending the signature of the owner of the Oracle, the root hash of transactions could be updated to increase or decrease the collection of banned transactions.
3. **`set-verification-key`** command - The Zero-Knowledge proof relies on a verification key to successfully prove the statement being verified. This command allows you to publish the verification key on the blockchain, which later will be used by the proof of innocence contract to check the proof.
4. **`create poi`** command - You can create a Proof of Innocence validator instance with this command. Again, a thread token will be created to prevent spoofing in the protocol.
5. **`verify`** command - This command allows us to check a proof of Innocence. It runs the code needed to generate the proof, create a transaction to verify the proof and execute the smart contract.

Tests

The testing of the project was achieved in two stages. The off-chain tests, which consist of unit tests to check the functionalities of the circuit and the different smart contracts used in the project. The on-chain testing was performed using the CLI, with which we created a working example.

Aiken unit tests

We performed several aiken tests to check the different functionalities of the smart contracts and edge cases where it could fail. You can find it here and run it with **`aiken check`** from the same folder.

- **`tests/oracle_minting.test`** - These tests prove the correct instantiation of the Oracle also proves the edge cases where this instantiation should fail.
- **`tests/oracle_spending_test`** - These tests prove the update logic of the oracle validator.
- **`tests/proof_of_innocence_minting_test`** - These tests prove the correct instantiation of the PoI contract.
- **`tests/proof_of_innocence_spending_test`** - These tests prove the correct spending of the PoI contract. Both when we aim to update the data present in the contract, also when we want to check the Zero-Knowledge proof as well.

Circom tests

We performed tests to check that the circuit accomplished the assertions needed to prove the protocol statements of innocence. You can find them here in the and run them with **`make test_circuit`** from the main folder. The instructions to run the tests are in this section of the Readme document.

CLI testing

The instructions about how to run the commands of the CLI contract could be found in this section of the Readme document, also one can get help about the use of the application by running:

```
npx tsx --trace-warnings src/index.ts --help
```

There is a record of transactions that shows the different interactions between the protocol and the Cardano testnet:

1. Oracle transactions
2. PoI transactions
3. Vk transactions