zokrada
Bringing zk-SNARKS to Cardano

Niels Mündler

July 1, 2023

Abstract

zk-SNARKs represent an advanced tool utilized for verifying zero-knowledge proofs in an on-chain validator setting. This technology, currently deployed on Ethereum, holds immense potential and is ideally suited to the Cardano validator setup on eUTXO. This paper proposes a system for undertaking zk-SNARK set up designated for Solidity contracts, with its application on Cardano in Plutus-based validator contracts. We demonstrate the feasibility of this approach through the expansion of the smart contract language - the zokrates tool, to incorporate the Cardano Smart Contract language OpShin as output. Such an approach facilitates a substantial leap forward in the development of zero-knowledge tooling on Cardano, by leveraging pre-existing technology.

1 Introduction

Zero Knowledge (ZK) proofs present a promising technology platform to create innovative and beneficial tools on blockchain. Their attractive feature – the ability to avert the need to reveal secret data, serves as a protective barrier to many potential security breaches in a fully transparent system, including frontrunning and idea theft.

One such variant of ZK proofs applicable in blockchain environments is ZK-SNARKs. Notably, Ethereum is robustly equipped with comprehensive tooling and provides helper functions on-chain to support ZK operations.

This document provides a brief overview, outlining the necessities in bringing ZK to Cardano, and demonstrates how the pre-existing tooling in Ethereum can catalyze the development of ZK applications on Cardano. We demonstrate a proof of concept tool for facilitating this process, alongside an example smart contract that successfully runs on Plutus V3 embedded with ZK tooling.

Initially, we provide a concise introduction to Zero Knowledge proofs, and existing approaches on Ethereum.

1.1 Zero Knowledge Proofs

In Zero-Knowledge Proofs, users are able to provide proof to a verifier about their knowledge of certain information without revealing the actual piece of information. As an example, if the user is asked about their knowledge on the answer to the question, "What is the hash preimage of Y?", they can provide a proof, X, with confidence that they know the answer. In providing such proof, neither the verifier nor any third party can derive the answer from X.

On the mathematical level, a zero-knowledge proof is reduced to a series of arithmetic expressions on an algebraic curve. The exact functioning and implementation of the curve aren't vital to understanding, particularly if our objective is merely to enable zero-knowledge tooling. Interestingly, we can note that some zero-knowledge challenge, such as "What is the hash preimage of Y?", can be formulated into a higher-level language for tools like zokrates. In return, zokrates converts this challenge into several (quite large) integers, which are then used as input to a program. This program will consequently verify a proof of a solution to the challenge, which is presented as another large integer.

1.2 Tooling on Ethereum

As previously noted, there exist numerous tools that can convert high-level challenge formulations into lower-level numerical formulations. These lower-level formulations can then be validated on-chain. These tools range from the simple 'ZoKrates', to more complex options like 'Arkworks' and 'Circom'. These latter options offer a broader variety of tools but are correspondingly more intricate to manage. Here, we present our proof-of-concept work which is based on ZoKrates. These tools inherently support Solidity.

Zero-knowledge proofs require labor-intensive curve operations. To help, the Ethereum blockchain offers precompiled smart contracts that bypass traditional costing structures. These contracts execute curve operations using manageable amounts of gas. This gas is typically used to cover the resources spent by a node to verify a transaction.

1.3 Smart Contracts on Cardano

In Cardano's eUTXO model, smart contracts function as validators that grant permission for the execution of certain operations, rather than relying on the use of public keys. They serve a crucial role, locking funds at an address and conditionally releasing them. Essentially, Cardano Smart Contracts are functions that either pass or fail. A pass permits the execution of an operation, such as the spending of funds, on Cardano's ledger, while a fail indicates the smart contract's disapproval of the operation.

Smart contracts on Cardano can be crafted in a variety of programming languages. While the official language provided by IOG is Plutus, there are alternative emerging languages such as Aiken, OpShin, Helios, and many others. Our focus is primarily on the support of OpShin, largely due to its Python base, which allows for the broadest possible adoption. That said, our approach is not exclusive to OpShin and can be adapted for use with other languages.

Notably, Cardano's smart contracts come with tight restrictions on CPU and memory consumption. In comparison to Ethereum, Cardano allows for additional transaction fees to increase resource consumption, albeit within strict upper limits. To determine the viability of our approach, it's necessary to evaluate the consumption of resources by generated smart contracts during verification of a proof.

2 The zokrada pipeline

Zokrada is a tool designed to transform an abstract zero-knowledge challenge formulation into a Cardano Smart Contract verifier. This verifier checks the exact solution to the challenge, allowing the user to unlock funds from an address if they can provide the solution. This section presents the various components of the Zokrada pipeline.

2.1 Transforming zero knowledge challenges

We employ the use of Zokrates to metamorphose an abstract zero-knowledge challenge into a fundamental mathematical illustration, as detailed in section 1.1. To exemplify this, we provide a sample program to Zokrates:

```
def main(private field a, field b) {
assert(a * a == b);
return;
```

This challenge is simplistic in nature, requiring the user to present a number a such that its square equals a given number b (i.e., $a * a = b$). The quantity a is classified as private; thus, it will be encoded discretely by the user and its revelation won't be demanded to substantiate knowledge of a. Leaning on Zokrates, we convert this challenge into a series of integers, comprising a set of components denoted as α, β, etc., and consequently establish the challenge in mathematical terms.

The conversion grants us with several necessary parameters needed to be applied to a verifier written in Plutus on-chain language.

2.2 Transforming zero knowledge proofs

To verify that we have a solution to the provided challenge - which implies generating a proof - we utilize Zokrates. This proof is computed based on specific inputs. During this process, Zokrates keeps public inputs intact, while transforming the private inputs into points located on the curve that is used in the zero-knowledge setup.

2.3 Verifying proofs on-chain

We create a smart contract using OpShin . The smart contract verifies zero-knowledge proofs, which are generated by zokrates. To do this, we modify the Solidity implementation of the verifier that is included in the zokrates distribution.

The OpShin verifier has a rigid structure and is parameterizable with the output of zokrates as shown in **??**. By utilizing the parameters provided by zokrates, we are able to create a unique smart contract for every challenge defined in the zokrates language. Public inputs are given as datums to the verifier, which then can be used to dynamically create new challenges for a known topic. For instance, they can provide different numbers, all requiring a factorization.

The final smart contract accepts one remaining parameter from the user: a proof computed by zokrates as described in section 2.2, based on the solution to the presented challenge. Afterward, the contract computes on-chain to ensure the proof is valid for the given challenge, and subsequently unlocks the funds upon successful validation.

3 Evaluation and Experiments

The challenge depicted in Figure 1 (section 2.1) is employed to assess the accuracy and efficiency of our approach.

Initially, we assess the verification of the provided proof premised on the Python implementation rather than the on-chain one. The on-chain compilation will, inevitably, contribute additional overhead to the code, hence we can presume that the Python evaluation is more accelerated than the on-chain evaluation. This provides us with an estimated feasibility of the code generated.

Our experiments confirm that the evaluation of a proof utilizing the Python implementation takes approximately 10 seconds. Evidently, this is exceedingly expensive. Commonly, Plutus Smart Con-

tracts exceed the execution limit much earlier, even when the execution only takes a few milliseconds. Most of the time is spent on resource-intensive curve operations. Consequently, we infer that without specialized built-ins for curve operations, as executed on Ethereum, the system would be inadequate.

Fortunately, there are current proposals to incorporate curve built-ins to Cardano in the forthcoming Plutus V3 release. We further propose additional curve operations to the ones that have already been suggested, as they do not align with the curve utilized on Ethereum. However, tools for both curves exist in Zokrates, indicating that this does not pose a restriction to the approach.

4 Conclusion and outlook

We presented a proof-of-concept implementation of zero-knowledge verifiers on Cardano. This was realized by extending the tool ZoKrates, applying its output to our custom-written zero-knowledge verifiers on Cardano. In future work, this tool can be broadened in two directions:

4.1 Zokrada as a Library for More Complex Setups

The utilization of ZoKrada in a library format could significantly enhance the ability to implement more complex setups. Such an approach could provide a streamlined methodology while preserving optimum functionality. This would allow for better management of resources and potentially improve the efficiency of the zero-knowledge verifiers.

4.2 Zokrada Support for Other Languages

A further extension for this tool could include the development of ZoKrada support for other languages. This would establish a broader platform for cross-language utilization, thereby promoting more diverse application. With such versatility in its system support, we can shape a globally comprehensive zero-knowledge verification platform on Cardano.