zokrada
Bringing zk-SNARKS to Cardano

Niels Mündler

July 1, 2023

Abstract

zk-SNARKs are a highly complex tool to allow zero-knowledge proofs to be checked in an on-chain validator setting. Being already deployed on Ethereum, this technology is highly promising. zk-SNARKS fit the Cardano validator setup on eUTXO very naturally. In this work, we propose a system that allows using zk-SNARK setup intended for Solidity contracts to be used on Cardano in Plutus-based validator contracts. Specifically we present feasibility of this approach by extending the smart contract language the tool zokrates to add the Cardano Smart Contract language OpShin as output. This allows for a significant jumpstart in the development of zero-knowledge tooling on Cardano by leveraging existing technology.

1 Introduction

zero knowledge (zk) proofs are a promising new technology to build useful and novel tools on blockchain. in particular the ability to not having to reveal secret data prevents many attacks in an entirely transparent system, such as frontrunning or idea theft.

zk snarks are a variant of this that are applicable well in blockchain settings. ethereum already has plenty of tooling and provides helper functions on-chain to support zk operations.

In this document we quickly outline what is necessary to bring zk to cardano and how we can leverage the existing tooling on ethereum to jumpstart development of zk applications on cardano. we provide a proof of concept tool for this and compile an example smart contract that runs on Plutus V3 with zk tooling.

first we give a short introduction on zero knowledge proofs and existing approaches on ethereum

1.1 Zero Knowledge Proofs

In zero knowledge proofs, users can provide proof about knowledge of some information to a verifier, without having to reveal the actual piece of information. for example, when asked whether the user knows an answer to the question "what is the hash preimage of Y?", the user is able to provide proof X that they know the answer without the verifier or any third party being able to derive the answer from X.

1.2 Mathematical perspective on Zero Knowledge Proofs

On the mathematical level, a zero knowledge proof boils down to a number of arithmetic expressions on an algebraic curve. the exact implementation and functioning of the curve is not so important to understand if we just want to enable zk tooling. What we can observe is that some zero knowledge challenge (i.e. "what is the hash preimage of Y?") can be formulated in some higher level language to tools like zokrates. In turn zokrates compiles this challenge down to a number of (very large) integers which are input to a program that will verify a proof of a solution to the challenge (presented as another large integer).

1.3 Tooling on Ethereum

As mentioned before there are various different tools available to transform higher level challenge formulations into a low level formulation in form of numbers which can be validated on chain. Among them are tools like zokrates, which is very basic to tools like arkworks and circom that support a larger variety of tools at the cost of being more complex to handle. we present our proof of concept work based on zokrates. The tools come with built in support for Solidity.

Moreover, because zero knowledge proofs generally require expensive curve operations, the ethereum blockchain provides precompiled smart contracts that circumvent traditional costing and perform these operations using feasible amounts of gas. The gas is paid to cover resources spent by node to verify the transaction.

1.4 Smart Contracts on Cardano

In Cardanos eUTXO model, smart contracts are functions that are run to validate permission of executing some operation in place of the usage of public keys. They can hence be used to lock funds at an address and release them conditionally. Cardano Smart Contracts boil down to functions that either pass or fail, where a pass is interpreted by the cardano ledger as a permission to execute the operation (i.e. spending of funds) and a fail indicates that the smart contract does not approve this operation.

Smart Contracts on Cardano can be written in a variety of languages. Next to the official language provided by IOG called Plutus, there are emerging alternatives such as aiken, OpShin, helios and many more. We focus in our approach on the support of OpShin, as it is based on python and has hence the largest possible adoption. Our approach is not limited to OpShin however and can easily be transferred to other languages.

Smart Contracts on Cardano have a relatively tight limit on CPU and Memory consumption. Similar to the way it is done on Ethereum, additional transaction fee can be spent to allow consumption of more resources, however there are tight upper limits on the overall consumption. In order to assess the feasibility of our approach we need to evaluate the consumption of resources by the generated smart contracts.

2 The zokrada pipeline

zokrada is a tool to transform an abstract zero knowledge challenge formulation into a Cardano Smart Contract verifier that checks precisely the solution to the challenge and lets the user unlock funds from an address if they provide the solution. This sections presents the different components of the zokrada pipeline.

2.1 Transforming zero knowledge challenges

We use zokrates to transform an abstract zero knowledge challenge into a low level mathematical representation as outlined in section 1.2. As an example we provide the following sample program to zokrates:

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

This is a simple challenge that checks whether for a number b the user can provide a number a that will square to b. a is private here, so it is a parameter that will later be encoded by the user and does not have to be revealed to prove knowledge of a. Using zokrates, we transform this challenge into a set of integers that contains a number of parameters called α, β etc and define the challenge mathematically.

This provides us with a number of parameters that we need to apply to a verifier written in a Plutus on-chain language.

2.2 Transforming zero knowledge proofs

To verify that we know of a solution to the provided challenge (i.e. generate a proof), we also use zokrates to compute a proof based on specific inputs.

2.3 Verifying proofs on-chain

We write a smart contract in the language OpShin that verifies zero knowledge proofs as generated by zokrates. For this we adapt the solidity implementation of the verifier included in the zokrates distribution. The OpShin verifier has a fixed structure and is parameterizable with the output of zokrates produced in **??**. The parameterized version based on parameters provided by zokrates create a dedicated smart contract for every challenge defined in the zokrates language. Public inputs are provided as datums to the verifier and can thus be used to dynamically create new challenges for a known topic (i.e. provide different numbers that all require a factorization). The resulting contracts takes one remaining parameter from the user: A proof computed by zokrates based on the witness (=solution) to the presented challenge. It computes on chain that the proof is valid for the presented challenge and unlocks funds on success.

3 Evaluation and Experiments

We use the challenge presented in section 2.1 to evaluate the correctness and performance of our appraoch.

We first evaluate the check of the given proof based on the implementation in python rather than on-chain. The on-chain compilation will add overhead to the code so we can assume that the python evaluation is faster than the on-chain evaluation and gives us an estimate on the feasibility of the generated code.

Our experiments show that the evaluation of a proof in the implementation in python takes roughly 10s. This is clearly highly expensive. Usual Plutus Smart Contracts reach the execution limit already much early, even when execution takes only a few milliseconds. Most of the time is spent on expensive curve operations. Hence we conclude that without specialized built-ins for curve operations in the way that it is done on Ethereum.

Fortunately, there are already proposals to add curve builtins to Cardano in the upcoming Plutus V3 release. We propose additional curve operations to the already proposed ones as they do not coincide with the curve used on Ethereum. However for both curves tooling in zokrates exists, so this is not a limitation to the approach.

4 Conclusion and outlook

we showed a proof of concept implementation of zero knowledge verifiers on Cardano. for this we extend the tool zokrates by applying its output to custom written zero knowledge verifiers on Cardano. in future work, this tool can be extended in two directions.

zokrada as a library for more complex setups zokrada support for other languages

References