By: John Guibas (Succinct)

TLDR; We're excited to share with the Lido community that we have a working POC of correctly calculating total Lido reserves, number of validators, and number of exited validators in ZK across all ~1M validators on Goerli based on Mainnet data. The code is open-source here and we have succesfully verified the proof on Goerli for ~300k gas.

Using the Succinct SDK, we were able to develop the core logic in around ~130 lines of rust and ~100 lines of solidity and quickly deploy and test the system onchain.

In this proposal, we outline a roadmap for the Succinct team to become another ZK oracle provider as part of Lido's initiative to augment/harden the security of the core protocol.

Context

Currently, the Lido Accounting Oracle contract is used for getting key consensus and execution layer information to the Lido protocol, such as the balances of Lido-participating validators. The oracle relies on a third-party quorum set, which is a 5/9 multisig, to attest to this information.

The Lido Community has already funded two similar proposals to explore the feasibility of partially replacing the oracle with more trustless methods:

- DendrETH: A trustless oracle for liquid staking protocols
- ZKLLVM: Trustless ZK-proof TVL oracle

Multiprover

Because ZK technology is new, a ZK-based oracle should be used to augment the security of the existing oracle, rather than completely replacing it. As a result, it makes sense to add several layers of redundancy by having several ZK oracles based on different implementations reporting in parallel.

With this type of approach, if there is a bug in a single implementation, it is unlikely for the same attack to work on the other approaches. Because of this, we think it is in Lido's interest to continually support work that aims to build multiple different implementations of a ZK Lido Oracle.

Succinct SDK

Our solution is implemented on top of the Succinct SDK, an open-source and modular toolkit for rapidly building and deploying production-grade ZK applications for teams that are not ZK experts.

The SDK addresses following problems when using ZK in production systems:

- Reading inputs and writing outputs to and from Ethereum
- Support for EVM types and data structures
- Deploying verifiers on the EVM
- Versioned upgrades and previews on testnets
- Running off-chain actors such as watchers, provers and relayers
- Cloud proving infrastructure
- Optimized proof verification onchain

The support for these features allows developers with limited knowledge of ZK to build end-to-end POCs using ZK, similar to the one we developed for Lido.

With this SDK, we hope that existing contributors to Lido can quickly ramp up to leveraging ZK when building new systems and applications for Lido.

Technical Details

Circuits.

In the current POC, we calculate the following three fields using the plonky2x proving system with the same logic in the official Lido Oracle implementation.

- [clBalanceGwei

](https://github.com/lidofinance/lido-dao/blob/cadffa46a2b8ed6cfa1127fca2468bae1a82d6bf/contracts/0.8.9/oracle/AccountingOracle.sol#L212): cumulative balance of all Lido validators on the consensus layer

- [numValidators

](https://github.com/lidofinance/lido-dao/blob/cadffa46a2b8ed6cfa1127fca2468bae1a82d6bf/contracts/0.8.9/oracle/AccountingOracle.sol#L208): the number of Lido-participating validators on consensus layer that ever appeared (deposited + activated)

- [numExitedValidators

](https://github.com/lidofinance/lido-dao/blob/cadffa46a2b8ed6cfa1127fca2468bae1a82d6bf/contracts/0.8.9/oracle/AccountingOracle.sol#L222): the number of Lido-participating exited validators on the consensus layer that ever appeared

In terms of how the circuit is implemented:

- Core Logic
- First, we use a SSZ proof to prove that the validators root exists within the provided block at a specific gindex.
- Then, we witness all validators and recalculate the validators root in the consensus layer and assert that the roots match.
- After, we iterate over all the validators and compute the aggregate statistics.
- First, we use a SSZ proof to prove that the validators root exists within the provided block at a specific gindex.
- Then, we witness all validators and recalculate the validators root in the consensus layer and assert that the roots match.
- After, we iterate over all the validators and compute the aggregate statistics.
- To handle all (~1M) validators in a single proof, we utilize a “map-reduce” style computational model where we shard proving by batches of validators.
- To know whether the validator is associated, we filter by a withdrawal credential.
- Using cloud proving infrastructure, we parallelize the generation of the proofs.
- The proofs are designed to be verified directly inside the EVM for ~300K gas.

Contracts

.

Our contracts store the three calculated statistics in storage.

- uint256 clBalancesGwei
- uint256 numValidators
- uint256 numExitedValidators

To request and verify proofs they use the FunctionGateway

contract inside the Succinct SDK, which abstracts away the logic of how to communication information from the contract and the circuit and vice-versa.

Demo

We have a fully-working POC of correctly calculating the total Lido reserves, number of active validators, and number of exited validators in ZK across all ~1M validators on Goerli.

Event: Etherscan

The report was delivered for slot 6984000 and the results were verified against the forked Lido Oracle repo from Nil Foundation. The proof took around 40 minutes to generate, but is highly parallelizable so the latency can be reduce with better utilization of elastic compute in the cloud.

Roadmap & Deliverables

We propose that the Succinct team work in collaboration with Lido to develop an MVP on testnet that can demonstrate how the ZK oracle can augment the security of the existing oracle.

Already Completed:

- Report Lido's total value locked, active and exited validator counts to a smart contract on Goerli with 2^20 (~1M) validators.
- On-chain verification for ~300k gas
- On-chain API to request new update.

Phase 1 (Productionization, December 31st, 2023)

:

- Conform the current smart contract to the desired interface of the ZK Lido oracle.
- Deliver an oracle on Goerli with multiple reports with the new interface.
- Create a testing suite for the oracle to ensure soundness / correctness.
- End-to-end tests
- End-to-end tests
- Create cost-effective and highly available offchain actors that can deliver the proof every 225 epochs
- Efficient use of cloud compute
- Monitoring / logging
- Efficient use of cloud compute
- Monitoring / logging
- Make the system easer to administer
- Documentation on how to run the system and how it works
- Open-source
- Documentation on how to run the system and how it works
- Open-source
- Start necessary audits for contracts/circuits (may complete in Q1)

Phase 2 (Testing, Q1 2024)

:

- Latency requirements: less than 30 minutes for 1M validators
- The Succinct team will provide infrastructure to run the oracle on Holesky testnet to emulate Mainnet conditions for up to 3 months.
- Includes generating proofs up to every 225 epochs (~1 day)
- Includes generating proofs up to every 225 epochs (~1 day)

We request that Lido provide us $50K to reward our existing work and to continue our ongoing efforts in this direction.

This grant will be used to cover:

- Development: 1 full time engineer from the Succinct team to complete this project
- Compute: Covering prover/cloud costs during development and testing
- Audits: The cost of auditing the contracts/circuits involved in this system

If this project succeeds, we hope to continue working with Lido on expanding the scope of this project and maintaining a long-term relationship between both teams.

FAQ

Who is the team behind this proposal?

Succinct is a company based in San Francisco, CA, working on applications of zero-knoweldge proofs on Ethereum. Our team has backgrounds from Celo, Google Brain, MIT, Stanford, UC Berkeley, and Caltech. In the past, Succinct built Telepathy, a zkSNARK-based light client for the Ethereum Beacon chain. The light client is one of the few non zkEVM applications of ZK used in production and to this day is used to secure over $40M in TVL with multiple integrations.

How do we get the beacon block root?

Once EIP 4788, is passed, the beacon block root will be trustlessly available on the EVM. Until then, either they must come from a trusted source or they can rely on another ZK oracle (such as Telepathy) to get the block root.

How does it compare to the other proposals?

Our solution is built upon open-source software built/contributed to by Succinct over the past year. In particular, the smart contracts and proving infrastructure is managed with the Succinct SDK and the circuit itself is built using our open-source Plonky2 verifier, our frontend for Plonky2 known as Plonky2x, and our frontend for writing STARK-based accelerators known as Curta.

The zkLLVM and DendrEth teams also take a circuit-based approach, although in different ways. The zkLLVM team uses LLVM to define the circuit while the DendrEth team uses a frontend library more similar to our approach, also using the Plonky2 proving system. The biggest difference between our approach and the DendrEth team is that we are exploring the usage of STARK-based accelerators built using Curta which dominates most of the proving time.

How will the work be maintained after delivery?

In the short-term, we are happy to run the infrastructure to run the oracle for at least a year. We may need another proposal in the future to cover longer-term proving costs.

In the long-term, our goal is to equip Lido contributors with the tools and infrastructure to maintain the system themselves and add functionality as the oracle requires changes.