Overview

Lido Accounting Oracle contract is trustful - it relies on an oracle committee (9 members at the time of writing) of trusted third-parties and quorum mechanism (5 out of 9) to maintain its state. This constitutes a potential attack vector - if the majority of the oracle committee members are hacked or malicious, an attacker will be able to submit a false value. Such an attack will directly endanger funds in DeFI pools (such as ~$1B Curve pool), undermine trust in Lido protocol security, and cause other reputational and financial losses.

In the past, our team has shown it is possible to compute one of the key parts of the accounting oracle report - total value locked - in a trustless manner, secured by a ZK-proof. Compared to legacy Lido Oracle, Accounting oracle carries on multiple additional responsibilities, including reporting active and exited validator counts, withdrawal and rewards balances, and more.

Proposal

We propose using zkLLVM to build an additional sanity/correctness check for the Accounting oracle, thus mitigating the risk outlined above.

Short-term (3-4 weeks):

- Report Lido's total value locked, active and exited validator counts to a dedicated Execution Layer contract in a verifiable, trustless manner.
- Use the new contract as an additional correctness check for Accounting oracle report.

Long-term (4+ months):

- Expand the solution to cover other parts of the Accounting oracle report.

We are asking the following commitments from Lido DAO:

- Grant. We estimate the overall costs of developing the first production iteration of the solution ("mid-term" above) to be around to be around 50K USD (to be transferred to this address).
- Support necessary audits for the smart-contract(s) and oracle(s) to be built.
- Cover the additional expenses for oracle operators - tentatively <$1K a month

Solution

zkLLVM technology overview

At a high level, zkLLVM technology stack used in this proposal consists of the following components

zkLLVM compiler and development toolchain

- the compiler allows building zk-SNARK or zk-STARK circuits using mainstream programming languages - currently supporting C++ and (partially) Rust, with more languages coming in the future. Other toolchain utilities enable a range of other activities - such as generating the proofs locally, or transpiling the circuit into verification gates deployable to Ethereum Execution Layer.

Proof market

- while it is possible to generate ZK-proofs locally, it is a resource- and time-demanding process.Proof market allows delegating producing the proof to 3rd parties, without compromising correctness or security of the proof.

Proof verifier contract

- proof verification happens completely on-chain, viaverifier contract + verification gates generated from a circuit. The applications using zkLLVM stack can use a shared instance deployed and operated by =nil; Foundation team, however this is not mandatory. The verifier is completely standalone - does not depend on any external data sources, oracles, libraries, etc. - which allows deploying a dedicated instance for additional security.

General approach

This is a high-level description aimed to provide a birds-eye view on the process end-to-end and intentionally omits lengthy

and complex explanations. For concrete details, please refer to a separate Detailed spec document.

The work is split between three components: oracle

, proof producer

and contract

.

- Oracle

obtains necessary information from Consensus and Execution layers - such as Beacon Block Header, Beacon State, Lido contract addresses, etc.

- Oracle

computes the report - total locked value and validator counts for all Lido validators. * While “bulletproof” check if a validator belongs to Lido requires obtaining Lido validator keys from Lido Node Operator Keys Service, we found a simple heuristic that can substitute this check with 100% accuracy.

- While “bulletproof” check if a validator belongs to Lido requires obtaining Lido validator keys fromLido Node Operator Keys Service, we found a simple heuristic that can substitute this check with 100% accuracy.

- Oracle

produces additional data necessary to produce and verify the proof, and passes it to a proof producer

- Additional data includes Beacon Block hash, Beacon State Merkle root, validators’ inclusion witness, and more.

- Additional data includes Beacon Block hash, Beacon State Merkle root, validators’ inclusion witness, and more.

- Proof producer

accepts the input and “runs” it through a zk circuit, producing a zk-proof. * At a high level, the circuit repeats the computations performed in the oracle (including additional verification witness), producing a verifiable “trail” of operations performed.

- At a high level, the circuit repeats the computations performed in the oracle (including additional verification witness), producing a verifiable “trail” of operations performed.

- Oracle

fetches the proof from the proof-producer

, and submits the report, additional witness and zk-proof to a contract

.

- Contract

performs necessary checks - verifies zk-proof with zkLLVM verifier contract, checks Beacon Block hash against an expected value, etc.

- If all checks pass, contract

stores the report for future retrieval, otherwise rejects the report.

Note

: until EIP-4788 is available, we would rely on an auxiliary oracle+contract to deliver BeaconBlock hashes to the Execution layer. This oracle+contract will use the oracle committee and consensus mechanism similar to the Accounting module.

FAQ

Q

: How is a trustless solution more secure than consensus-based?

A

: With a trustless solution, the correctness of the oracle report will be guaranteed by verifying a ZK-proof, and “anchoring” both report and proof to the actual blockchain state (via block hash and Merkle inclusion proofs). This allows making no

assumptions about the oracle operator, proof producer and other involved parties - as long as the report passes all checks, it is known to be legitimate, even if the sender is compromised. In short, anyone

can submit the report, and the correctness of the report can be verified against blockchain state

.

Q

: What exactly does ZK check verify?

A

: Oracle report is produced from two “ingredients”: data (validators and balances from Beacon State) and algorithm (computations to perform on them). Proof is essentially an execution trace of a computation performed on some input data. The circuit encodes the expected algorithm, so verifying the provided proof against the circuit ensures that the correct computations

were performed on some data

. Additional witness (in particular, BeaconBlock hash and Merkle inclusion witness) is passed to the contract to check if the oracle used the correct data

. Lastly, to prevent tampering with the data after proof was produced, but before the report is submitted, the circuit includes computing the witness from the raw data. Put together, these ensure that a correct computation

was performed on the correct data

.

Q

: Why build a supplementary check and not a complete replacement for the accounting oracle?

A

: While building a complete replacement is possible, we believe gradual development and rollout is more beneficial, compared to a “big leap” approach. Having the trustless solution act as an additional check for a battle-tested existing oracle will gradually build confidence and expertise operating trustless solutions (especially in terms of development speed, correctness, operational cost, etc.), and simultaneously expand the trustless solution to cover the full range of accounting oracle responsibilities.

Q

: Why zkLLVM and not (something else)?

A

: The answer will likely be different for each other technology, but here are a few unique advantages zkLLVM stack provides:

- Proof verification happens completely

in the Execution Layer - i.e. Layer 1 solution with no external dependencies, as secure and reliable as Ethereum blockchain itself.

- Verifier contract is open-sourced, so dedicated copies can be deployed for additional security, if necessary.
- “Immediate” verification - the proofs can be securely verified on-chain immediately after they are produced.
- Proofs can be generated via a centralized on-premises or cloud setup, or delegated to a decentralized proof generation system (proof market). Both options achieve the same security and correctness guarantees.
- Speed of development, ecosystem and talent pool - ZK-circuits are developed using mainstream programming languages, as opposed to vendor-specific DSLs.
- zkLLVM is powered by a Placeholder proof system which features no trusted setup. This eliminates last proof system-related attack vectors many other systems (e.g. Halo2 or Groth16-based ones) are subject to.
- zkLLVM was chosen by Ethereum, Mina and Solana Foundations for zkBridging usages with an audit by ABDK in progress.
- Proof Market applies market-dynamics to proof generation which means performance optimizations become a market-driven metric incentivized by applications, which means further optimizations do not require additional financing.

Demo

Video: ZKLLVM_Oracle_Demo.webm - Google Drive

Description: ZKLLVM Oracle Demo transcript - Google Docs

Implementation timeline

Phase 1: Initial implementation (main logic)

What

: building an initial implementation

- zkLLVM circuit covering computing main report and Merkle inclusion proofs.
- Reading BeaconState and BeaconBlockHeader from Consensus Layer
- Computing SSZ hash tree roots and inclusion proofs.

When

: 2-3 weeks from now.

Outcome

: “happy path” works end-to-end in a devnet/forked mainnet

Details

:

- ZK circuit proving validity of computations.
- Oracle implementation - fetching data from Consensus Layer, computing the report, generating the proof and sending it to Ethereum contract.
- Contract to perform validity and correctness check (incl. proof verification).
- The solution runs in a production-like environment (devnet/forked mainnet), controlled by “scenario script(s)” - end-to-end tests, emulating different real-world scenarios.

Phase 2: Productionization

What

: Productionization

When

: 3-6 weeks from Phase 1

Outcome

: “code complete” - all parts of the solution are finalized and ready for security audit.

Details

:

- All components of the final solution are lifted to Lido production quality (monitoring, logging, ACL, dockerization, etc.), finalized (no code changes expected) and ready for the audit.
- Oracle is fully productionized (logging+monitoring+dockerization), documented and open-sourced.
- Contract is expanded to include administrative functionality (ACLs, redeployment, DAO capabilities, etc.), where necessary.
- Contracts pass necessary audit(s)
- Oracle is fully productionized (logging+monitoring+dockerization), documented and open-sourced.

- Contract is expanded to include administrative functionality (ACLs, redeployment, DAO capabilities, etc.), where necessary.
- Contracts pass necessary audit(s)

Phase 3: Audit and final polishes

What

: Audit

When

: End of Phase 2 + time for audit

Outcome

: ready for testnet and mainnet deployment ceremony

Details

:

- Testnet deployment and user-acceptance testing
- Security audit and fixes