Proof of Validator: A simple anonymous credential scheme for Ethereum’s DHT

Authors: George Kadianakis, Mary Maller, Andrija Novakovic, Suphanat Chunhapanya

Introduction

Ethereum’s roadmap incorporates a scaling tech called Data Availability Sampling (DAS). DAS introduces new requirements to Ethereum’s networking stack, necessitating the implementation of specialized networking protocols. One prominent protocol proposal uses a Distributed Hash Table (DHT) based on Kademlia to store and retrieve the samples of the data.

However, DHTs are susceptible to Sybil attacks: An attacker who controls a large number of DHT nodes can make DAS samples unavailable. To counteract this threat, a high-trust networking layer can be established, consisting solely of beacon chain validators

. Such a security measure significantly raises the barrier for attackers, as they must now stake their own ETH to attack the DHT.

In this post, we introduce a proof of validator

protocol, which enables DHT participants to demonstrate, in zero-knowledge, that they are an Ethereum validator.

Motivation: “Sample hiding” attack on DAS

In this section, we motivate further the proof of validator

protocol by describing a Sybil attack against Data Availability Sampling.

The DAS protocol revolves around the block builder ensuring that block data is made available so that clients can fetch them. Present approaches involve partitioning data into samples, and network participants only fetch samples that pertain to their interests.

Consider a scenario where a Sybil attacker wants to prevent network participants from fetching samples from a victim node, which is resposible for providing the sample. As depicted in the figure above, the attacker generates many node IDs which are close to the victim’s node ID. By surrounding the victim’s node with their own nodes, the attacker hinders clients from discovering the victim node, as evil nodes will deliberately withhold information about the victim’s existence.

For more information about such Sybil attacks, see this recent research paper on DHT Eclipse attacks. Furthermore, Dankrad’s DAS networking protocol proposal describes how the S/Kademlia DHT protocol suffers from such attacks and shows the need for a proof of validator

protocol.

Proof of Validator

The above attack motivates the need for a proof of validator

protocol: If only validators can join the DHT, then an attacker who wants to launch a Sybil attack must also stake a large amount of ETH.

Using our proof of validator

protocol we ensure that only beacon chain validators can join the DHT and that each validator gets a unique DHT identity.

Furthermore, for validator DoS resilience, we also aim to hide the identity of the validators on the networking layer. That is, we don’t want attackers to be able to tell which DHT node corresponds to which validator.

To fulfill these objectives, the proof of validator protocol must meet the following requirements:

- Uniqueness

: Each beacon chain validator must be able to derive a single, unique keypair. This property not only restricts the number of nodes a Sybil attacker can generate, but also enables network participants to locally punish misbehaving nodes by blocklisting their derived keypair

- Privacy

: Adversaries must be unable to learn which validator corresponds to a particular derived public key

- Verification Time

: The protocol’s verification process must be efficient, taking less than 200ms per node, enabling each node to learn at least five new nodes per second

Such a proof of validator

protocol would be used by Bob during connection establishment in the DHT layer, so that Alice knows she is speaking to a validator.

Proof of Validator protocol

Our proof of validator

protocol is effectively a simple anonymous credential scheme. Its objective is to enable Alice to generate a unique derived key, denoted as D

, if and only if she is a validator. Subsequently, Alice uses this derived key D

within the networking layer.

In designing this protocol, our objective was to create a solution that was both straightforward to implement and analyze, ensuring it meets the outlined requirements in an efficient way.

Protocol overview

The protocol employs a membership proof subprotocol

, wherein Alice proves she is a validator by demonstrating knowledge of a secret hash preimage using ZK proofs. Alice then constructs a unique keypair derived from that secret hash preimage.

The membership proof subprotocol

can be instantiated through different methods. In this post, we show a protocol using Merkle trees

and a second protocol using lookups

.

While both approaches demonstrate acceptable efficiency, they feature distinct tradeoffs. Merkle trees rely on SNARK-friendly hash functions like Poseidon (which may be considered experimental). On the other hand, efficient lookup protocols rely on a powers-of-tau trusted setup of size equal to the size of the validator set (currently 700k validators but growing).

Now let’s dive into the protocols:

Approach #1:

Merkle Trees

Merkle trees have seen widespread use for membership proofs (e.g. see Semaphore). Here is the tradeoff space when designing a membership proof using Merkle trees:

- Positive:

No need for trusted setup

- Positive:

Simple to understand

- Negative:

Relies on SNARK-friendly hash functions like Poseidon

- Negative:

Slower proof creation

Below we describe the proof of validator protocol based on Merkle trees:

Proof-of-validator protocol using Merkle trees

Every validator i

registers a value p_i

on the blockchain, such that p_i = Hash(s_i)

. Hence, the blockchain contains a list {p_i}

such that:

p_1 = Hash(s_1), \ldots, p_n = Hash(s_n)

where the p_i

are public and the s_i

are secret. The blockchain creates and maintains a public Merkle root R

for the list of public p_i

values.

Suppose Alice is a validator. Here is how she can compute and reveal her derived key D

given her secret value s_i

:

1. Set derived key D

to equal D = s_i G

1. Prove in zero-knowledge with a general purpose zkSNARK that there is a valid Merkle path

from p_i

to the merkle root R

, plus the following statement:

p_i = Hash(s_i) \ D = s_iG \

At the end of the protocol, Alice can use D

in the DHT to sign messages and derive her unique DHT node identity.

Now let’s look at a slightly more complicated, but much more efficient, solution using lookups.

Approach #2:

Lookups

Here is the tradeoff space of using lookup protocols like Caulk:

- Positive:

Extremely efficient proof creation (using a preprocessing phase)

- Positive:

Protocol can be adapted to use a regular hash function instead of Poseidon

- Negative:

Requires a trusted setup of big size (ideally equal to the size of validators)

Below we describe a concrete proof of validator protocol:

Proof of validator protocol using lookups

Exactly like in the Merkle approach, every validator i

registers a new value p_i

on the blockchain such that:

$p_1 = Hash(s_1), \ldots, p_n = Hash(s_n)$

where the p_i are public and the s_i are secret. The blockchain creates and maintains a KZG commitment R to the vector of all p_i values.

Suppose Alice is a validator. Here is how she can compute and reveal her derived key D given her secret value s_i:

1. Set derived key D to equal $D = s_i G$

1. Reveal commitment $C = p_i G + s_i H$ where H is a second group generator.

You can view $(D, C) = (s_i G, p_i G + s_i H)$ as an El-Gamal encryption of p_i under randomness s_i. Assuming $Hash$ is a random oracle, s_i is not in any way revealed so this is a valid encryption of s_i.

1. Prove using a Caulk+ proof that C is a commitment to a value in the set $\{p_i\}$ represented by commitment R.

1. Prove with a Sigma protocol that s_i is consistent between D and C.

1. Prove with a general purpose zkSNARK that $p_i = Hash(s_i)$ and that $C = p_iG + s_iH$.

At the end of the protocol, the validator uses D as her derived key on the networking layer.

Efficiency

We benchmarked the runtime of our membership proof protocol (link to the benchmark code) in terms of proof creation and verification. Note that while the membership proof is just one part of our proof of validator protocol, we expect it to dominate the overall running time.

Below we provide benchmark results for a merkle tree membership proof

using the Halo2

proof system with IPA

as the polynomial commitment scheme. IPA is a slower scheme than KZG but it doesn't require a trusted setup maximizing the advantages of the merkle tree approach.

Prover time

Verifier time

Proof size

4 milion validators (Depth = 22)

325ms

13.8ms

2944 bytes

16 milion validators (Depth = 24)

340ms

14ms

2944 bytes

67 milion validators (Depth = 26)

547ms

21ms

3008 bytes

We observe that both the prover and verifier times align well with our efficiency requirements. For this reason, we decided against benchmarking the Caulk-based approach, as its performance is expected to be significantly better in all categories (especially prover time and proof size).

Benchmarks were collected on a laptop running on an Intel i7-8550U (five years old CPU).

Discussion

Rotating identities

The uniqueness

property of the proof of validator

protocol ensures that each network participant possesses a distinct derived keypair. However, for certain networking protocols, it might be advantageous to allow validators to have rotating identities, where their derived keys change periodically, perhaps daily.

To implement this, we can adapt the protocol to generate rotating derived keys based on a variable string, such as a daily changing value. For instance, let $D = r_iG$

: to prove the validity of this rotated key, we can utilize a SNARK proving that $r_i = Hash(s_i || \mathsf{daily string})$

. Additionally, the SNARK must prove that $p_i = Hash(s_i)$

and conduct a membership proof on p_i

.

In such a scenario, if Eve misbehaves on a particular day, Alice can blocklist her for that day. However, on the next day, Eve can generate a new derived key, which is not blocklisted. If we wanted to be able to permanently blocklist validators based on their rotating identity we would need a more advanced anonymous credentials scheme like SNARKBlock.

Why not use the identity BLS12-381 public key?

An alternative (perhaps simpler) approach would be to build a commitment out of all validator identity BLS12-381 keys and do a membership proof on that commitment.

However, this approach would require validators to insert their identity private key into the ZK proof system to create a valid membership proof and compute the unique derived key.

We decided to not take this approach because it's not good practice to insert sensitive identity keys into complicated cryptographic protocol, and it would also make it harder for validators to keep their main identity key offline.

Future research directions

- Can we avoid SNARK circuits entirely and perform the membership proof and key derivation in a purely algebraic way?
- Related: Can we have an efficient proof of membership

protocol without a trusted setup and without relying on SNARK-friendly hash functions?

Acknowledgements

Thanks to Enrico Bottazzi, Cedoor, Vivian Plasencia and Wanseob for the help in navigating the web of membership proof codebases.