Holding Back Validators Through De-Anonymization in Ethereum

Anders Malta Jakobsen*, Oliver Holmgaard†

Abstract—This is a placeholder abstract test. The whole template is used in semester projects at Aalborg University (AAU).

1 Introduction

Ethereum is a decentralized network and is one of the most popular blockchains. Currently, Ethereum is the most used blockchain which relies on the consensus mechanism of Proof of Stake (PoS). Ethereum switched from Proof of Work (PoW) to PoS in 2022 with the release of Ethereum 2.0 [1]. A part of this decision was to reduce the requirements for participating in the network. With the increasing number of validators, the network is becoming more active, with a greater number of messages being sent between them.

With all of these messages being sent, the network is becoming increasingly vulnerable to attacks based on information gathering. This makes collection of information, such as IPs of validators, possible and creates a potential threat to the network. This vulnerability led to the paper by Heimbach et al. [2], ”Deanonymizing Ethereum Validators: The P2P Network Has a Privacy Issue” (De-anonymizing Paper), which presents a de-anonymization attack on validators in the Ethereum network. In this paper, we have recreated the de-anonymization attack on Ethereum validators from the De-anonymizing Paper, identifying and linking IPs to active validators.

Even without a validator, the attack enables the gathering of information about validators in the Peer-to-Peer (P2P) network through attestations sent throughout the Ethereum network. Enough information is gathered from logging peers and the messages received to de-anonymize validators. The information gathered from the attack can then be used to perform attacks, such as a Denial-of-Service (DoS) attack on proposers, when combined with the fact that the proposers of each block are known in advance. This could lead to a loss of money for the validators, as they are penalized for being inactive. A potential monetary gain for the attacker is possible if a victim is the proposer in the slot before the attacker.

- *All authors are affiliated with the Dept. of Computer Science, Aalborg University, Aalborg, Denmark*
- *E-mails: *amja23, †oholmg20 @student.aau.dk*

In this paper, we have implemented the de-anonymization attack, documented the results, and compared them against the results from the De-anonymizing Paper. We discuss the incentives and consequences that arise after successfully executing the de-anonymization attack on block proposers. Following this, we describe a possible DoS attack on block proposers made possible after the de-anonymization validators.

This paper was inspired by the work of the De-anonymizing Paper. It has led us to make the following contributions:

- We have successfully executed the de-anonymization attack on validators. The attack was run on the Holesky testnet for ethical reasons.
- We have discovered some similarities with the De-anonymizing Paper, especially the de-anonymization rate of validators through running the attack. We also found differences, such as the amount of validators per peer, due to running on a different network.
- A thorough description of how to implement a DoS attack on block proposers has been developed.

Related Work

Blockchain technology has a history of many attacks, some specific to the Ethereum network. Before choosing the de-anonymization attack in the context of the block proposer DoS attack, we considered other attacks that could be performed on the Ethereum network. section A describes many of these attacks further, but the most closely related attacks are mentioned here.

DoS Attack

Instances of DoS attacks seen on Ethereum range from attacks on the proposers to attacks that seek to slow down the network itself. One of these attacks aims to slow down the network by using underpriced opcodes to create a block that is hard to process in time [3, 4]. However, this has been mitigated in an Ethereum improvement proposal [5]. Another way to slow down the network is to create empty accounts that are hard to process [6, 7]. This attack, however, is outdated and has been mitigated by making it nearly impossible to create empty accounts in the network.

2 Background

To better understand the attack we will be performing in this paper, we need to review some of the concepts used in it. This section dives into the inner workings of the Ethereum network layer and the consensus algorithm. It will also delve into the inspiration for the attack.

2.1 Ethereum and Proof of Stake

Ethereum is a blockchain platform that allows developers to create decentralized applications using smart contracts. Previously operating with a PoW consensus algorithm, Ethereum transitioned to a PoS consensus algorithm in 2022 [1]. This transition was made to reduce the network's energy consumption and increase its scalability. The transition was done in a series of upgrades called the Ethereum 2.0 upgrade.

PoS is a consensus algorithm used to ensure that everyone agrees on the same state of a blockchain. The algorithm includes guidance on whom gets to produce blocks and confirm transactions. In Ethereum, the protocol chooses block proposers from a set of validators based on the amount of cryptocurrency they have staked in the blockchain. All other validators are then given the role of confirming this block, adding it to the canonical chain.

The proposers get rewarded for creating a valid block, and the validators get rewarded for confirming the block. Block proposing happens within epochs of 32 blocks, and the protocol selects a validator as a proposer for each block. The proposer is chosen through a random process weighted by the amount of ether the validator has staked. Ethereum uses the publicly available random number generator (RANDAO) algorithm to simulate this random selection.

The designated proposer must propose a block for the slot, and other validators must verify it within the slot's 12-second time limit. If a fork happens, the validators must choose which fork to follow. Choosing the fork is done by following the Latest Message Driven Greedy Heaviest Observed Subtree (LMD-GHOST) [1] algorithm which chooses the fork with the greatest weight of attestations in its history.

2.2 Subnets

The Ethereum network is split up into smaller networks called subnets. The network's nodes are split into 64 subnets and an additional subnet for attestation aggregation, with each node subscribing to two subnets by default. Being subscribed to a subnet is also referred to as being backbone of a subnet.

These subnets are used to help with the scalability of the network by distributing the load from messages. Most messages are sent between nodes in the same subnet, lessening the load of a node, as it will receive $\frac{2}{64}$ of all messages in the blockchain instead of all messages.

Nodes cannot be a peer with all nodes in a subnet. Hence, the peers a node can reach within the same subnet are called its fanout [2].

Within a subnet, nodes choose a subset of peers in the same subnet to share their messages with. Choosing which nodes belong to this subset is based on a peer score determined by a peer's performance. Nodes send all messages they receive within a subnet to these best-performing peers.

2.3 Attestations

A validator is expected to broadcast a vote each epoch, also called an attestation [8]. This message's overall purpose is to ensure that validators agree on the canonical chain. An attestation is sent by its author to only a single subnet. Afterward, peers in that subnet will naturally broadcast it to other parts of the network.

The blockchain protocol assigns a single validator to a single slot each epoch, where they must publish their attestation. In this attestation, the validator votes for what it sees as the *source* and *target*.

The validator sees the source as the most recent justified block.

The validator sees the target as the first block in the current epoch.

All validators assigned to attest in the same slot are referred to as being on the same committee. Sixteen assigned aggregators per subnet then collect all attestations broadcast in each subnet. These collect votes equivalent to their own, aggregate them, and broadcast them to the network. These are then to be included in the block of the designated slot.

2.4 Validators

A validator is an entity running on a network client, among other things, consisting of an ether balance and a key pair for identifying it [9].

For a validator client to work, it needs to be run in cooperation with a consensus node client. A consensus node client can run an unbounded number of validators. Several validators running in cooperation with a single consensus node mean that they all share the same identity. The shared identity includes data such as IP addresses and the ID of the consensus node.

At each slot, the RANDAO algorithm is used to select a validator responsible for processing transactions, including them in a block, and then propose this block to the blockchain. This procedure is also illustrated by Figure 1. Determining who will be a block proposer is explained in section 2.5. Along with this, the validator is also responsible for attesting blocks proposed by other validators, ensuring the liveness of the chain.

To be able to run a single validator, one must stake 32 ETH, which is $99,776[1]. Having this much money at stake should ensure that a validator acts honestly. Doing so will also earn the validator a reward. However, acting dishonestly will result in the burning or slashing of ether. The rewards and punishments are described in the following sections.

2.4.1 *Validator rewards*

Validators are rewarded for several different actions [10]. Each of these actions is rewarded with different weights, Nevertheless, they all depend on the total amount of staked ether by all validators and the validator's own staked amount. However, when a validator has a balance of 32 ETH, they receive optimal rewards.

A base reward, to be used on the reward weights, is calculated for a single validator as follows, where `BR` is *base*

1. As of 2024-11-18 seen on beaconcha.in

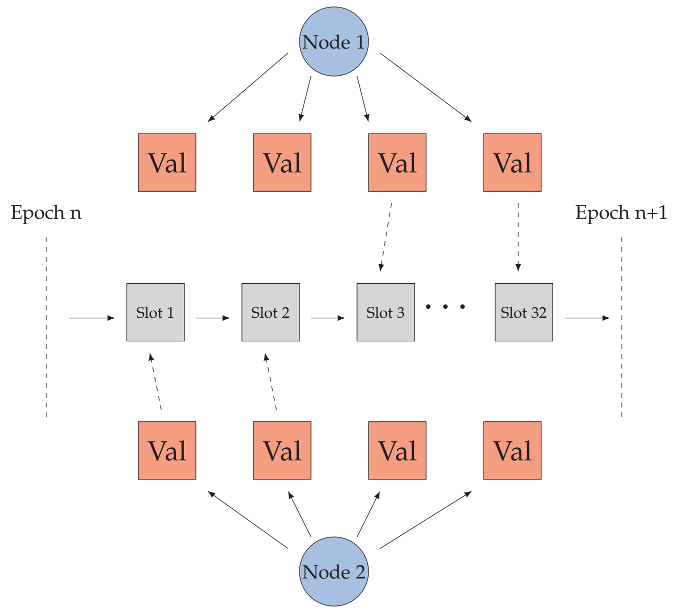

Fig. 1. Illustration of validators selected as proposers through an epoch. Two random nodes are shown, each having several validators. Some of these validators are chosen to propose a block in a designated slot.

reward, EB is *effective balance*, BRF is *base reward factor* set to 64, BRPE is *base rewards per epoch* set to 4, and AB is *active balance*, which is the total staked ether by all validators:

$$BR = EB \cdot \left(\frac{BRF}{BRPE \cdot \sqrt{\sum AB}}\right) \tag{1}$$

The reward a validator receives is then calculated as follows:

$$\frac{\sum weights}{64} \cdot BR \tag{2}$$

What is included in the sum of weights varies depending on the completed task. The weights of the different tasks are the following:

1) Timely source vote: 14
2) Timely target vote: 26
3) Timely head vote: 14
4) Sync reward: 2
5) Proposer weight: 8

The most profitable reward is the proposer reward. This reward is given to the validator whenever it is chosen as a proposer and correctly proposes a block to the blockchain. In this case, instead of being rewarded only once, the proposer is rewarded for each valid attestation included in the block proposed.

$$BR \cdot \frac{8}{64} \cdot \#attestations \tag{3}$$

The maximum number of attestations in a block is 128. Therefore, the proposer can receive a reward up to $BR \cdot \frac{8}{64} \cdot 128$ ETH when proposing a block [10, 11]. Because of that, validators do not want to miss being the proposer. Also, the chance of getting to propose a block is relatively low.

The average amount of staked ether for a validator is over 32 ETH. Furthermore, the validator's probability of being a proposer reaches optimality at 32 staked ether. Therefore, there is an almost equal chance of being a proposer among the over 1 million validators in Ethereum [2].

2.4.2 *Validator punishments*

A validator can also be punished for doing things that negatively affect the chain. Ethereum has two kinds of penalties, burning and slashing, where a validator loses some of their staked ether. Should a validator get penalized and end up below 16 staked ether, they will be removed as a validator [11].

Burning happens every epoch that a validator is offline. If ether gets burned, it is gone forever. To calculate how much ether is retained after burning per offline epoch, n, the following formula is used, with IPQ being the *inactivity penalty quotient* set to 2^{26} [11]:

$$\left(1 - \frac{1}{IPQ}\right)^{\frac{n^2}{2}} \tag{4}$$

This means that a validator being offline in a single epoch still retains 0.99999999255% of their staked ether. A validator with 32 staked ether would, therefore, need to be offline for many epochs before eventually being removed for having under 16 staked ether.

Slashing happens when a validator acts maliciously against the blockchain and cannot be invoked only by being offline. Slashing happens from at least one of three causes [10]:

1) Proposing two different blocks at the same slot
2) Attesting a block that surrounds another block, hereby changing history
3) Double voting - Attesting to two candidates for the same block

If this is detected, $\frac{1}{32}$ of the validator's staked ether is immediately burned, and a 36-day removal period of the validator begins, where the staked ether is gradually burned. After 18 days, an additional penalty is applied. The magnitude of this penalty scales with the total staked ether of the slashed validators in the same period. At worst, a validator can have all their ether burned if many other validators were also slashed during that period.

2.5 RANDAO

As mentioned in section 2.4, the Ethereum protocol chooses block proposers using a random-number-generator called RANDAO[3]. The random selection is decided by the RANDAO algorithm, which is used in every slot. Each chosen proposer will get the RANDAO value, computed at the previous slot, and XOR it with the hash of their private key and epoch number.

This creates what is called the RANDAO reveal. It can be verified with the proposer's public key. Of course, this happens at all 32 slots of an epoch, ensuring the protocol's randomness. At the end of each epoch, the latest reveal constitutes the seed, which is used to determine who the next proposers will be.

2. As of 2024-11-18 seen on beaconcha.in
3. RANDAO - GitHub

Being chosen as a block proposer comes with some duties, such as creating the block. Therefore, they need to know in advance if they will be the proposer of a block in order to perform their duties on time.

The proposer selection among the validators is done two epochs in advance [12]. More specifically, the selection for epoch $n + 2$ happens at the end of epoch n [13]. Hence, if they are proposing a block, a validator knows at least one epoch in advance, at most two epochs.

2.6 ENR

An Ethernet Node Record (ENR) is a record that contains information about a node in the network [14]. Ethereum uses ENRs to package the information sent from node to node. The ENRs are sent during the discovery protocol, where nodes discover each other. The package contains information like the peer's IP address, port, and public key. Because of the nature of the discovery protocol, if one has a node in Ethereum, one can see the ENR of all discovered peers. Moreover, since the ENR contains the IP address and the public key of the peer, the node can see the corresponding IP addresses and public keys of all the peers that have been discovered.

2.7 Inspirational Paper

In the De-anonymizing Paper, the authors show that it is possible to de-anonymize validators on the Ethereum network by observing attestations and subscribing to all subnets [2]. This paper is relevant to our work as it shows that private information about the validators can be obtained on the network. It is also the main inspiration for our attack.

The paper takes advantage of the attestations, including information such as the sender's IP and backbone subnets, to obtain information about the validators. They use a custom version of a Prysm node called RAINBOW that subscribes to all subnets for their setup. They use it to log peers and break peer inputs down into "colors" of the validators. This information consists of all received attestations, their origin and origin subnet; all advertised static subscriptions of peers, and connection status for all peers they interact with.

They also used a network crawler to find peers more quickly in the Ethereum network, enabling broader discovery. However, the crawler and the RAINBOW node are two separate entities.

In the execution of their experiment, they set up four nodes spread out across four different geographical locations: two in Europe, one in Asia, and one in the United States. They let the nodes run for three days, de-anonymized 235,719 validators and contacted 11,219 peers. These peers were also divided into four categories based on a heuristic developed by the authors.

The categories are:

- **De-anonymized:** One or more validators have been discovered on this peer.
- **No validators:** No validators have been discovered on this peer, as no non-backbone attestation was received.
- **64 subnets:** The peer was subscribed to all 64 subnets, so it is impossible to tell backbone attestations from non-backbone attestations.
- **Rest:** Peers that did receive non-backbone attestations, but no validators were identified on the peer.

The heuristic developed for de-anonymizing validators includes four criteria:

1) The proportion of non-backbone attestations for validator v exceed

$$0.9 * \left(\frac{64 - n_{sub}(p)}{64}\right) \tag{5}$$

where nsub(p) is the average number of subnets the peer is subscribed to over the connection's duration.
2) The peer is not subscribed to all 64 subnets.
3) The node receives at least every tenth attestation expected for validator v on peer p.
4) The number of attestations received for validator v from peer p exceeds the mean number of attestations per validator from peer p by two standard deviations.

3 Experimental Protocol

In this section, we will cover the motivation and the workings of the attack that will be performed. We also describe the setup of the experiment that will be conducted and the experiment itself.

3.1 Motivation

3.1.1 Proposer DoS Attack

In this section, we will be describing the attack that acts as motivation for doing our experiment. The attack is a DoS attack aiming to halt the proposers selected for creating blocks in the chain. Ethereum themselves have mentioned it as a potential attack, and with the current implementation of the consensus algorithm, this attack is likely to perform [15, 16].

We are interested in researching the feasibility of this attack and the ones mentioned in section A. Researching feasibility has proven difficult, given that most of our researches involve attacks in the consensus or execution layer. This makes verifying that an attack has occurred challenging. For this reason, we have chosen the *Proposer DoS attack* as it seems exciting and has not been mitigated yet.

The attack is possible because the consensus mechanism uses a publicly known function to choose the upcoming block proposers. Therefore, the adversary can compute this in advance of the upcoming slots in the blockchain so each upcoming proposer is known. After this, the adversary can map the proposer's IP addresses and overload their connection. A successful attack would prevent a proposer from proposing their block in time.

3.1.2 Adversarial Incentives for the Attack

Several reasons can explain why an adversary would be interested in performing this kind of attack. This applies to adversaries running a node with validators and adversaries running a node without validators in the network.

The first reason, not requiring any validators, could be exploiting the inactivity penalization mentioned in section 2.4.2. The adversary could make validators lose money for being inactive. For the adversaries themselves, they do not receive any monetary rewards from this attack. One could be interested in getting people kicked from being a validator for having below 16 staked ether. However, it can quickly be derived from Equation 4 that the DoS attack must be ongoing for at least 9,645 epochs ($\sim$43 days) before a validator goes from 32 ether to under 16 ether.

Having more focus on targeting upcoming proposers, the adversary can limit the rewards that the proposers are promised. Proposers get the greatest reward after proposing a block, see section 2.4.1, so the adversary could limit the rewards given to proposers. In the worst case, this could decrease the total amount of ether, as it is dynamically changing with the chain.

Both reasons only require an Ethereum node, but the adversary does not gain any reward from the attack. Monetary gain could be a possibility if the adversary is in control of a validator. The reason for this is found in the Maximal Extractable Value (MEV) [17].

MEV refers to the maximum value one can extract from a block production opportunity. It is a reward that the proposer can get besides rewards from routine work on the blockchain, as mentioned in section 2.4.1. The idea of MEV is that *searchers* search the network for profitable transaction opportunities. These are, for instance, opportunities where people are willing to pay a high transaction fee to the proposer for inclusion in the block. These transactions are then added to the pool of transactions to be added to the chain. More implementations of these searchers are being made, i.e., Flashbots and MEV-Boost. It is estimated that 90% of Ethereum validators use this kind of software [18].

So, let us assume a scenario in which an adversary has a validator set to propose the block at slot n. It could then be beneficial for the adversary to perform a DoS attack on the proposers of the previous slots. Beginning from slot $n-1$ and down, the more consecutive proposers successfully attacked, the more potential gain the adversary has. The increase in potential gain is because the high-paying transactions, to be included in the previous slots, can now be included in the adversary's slot. Therefore, the adversary potentially gains a greater reward from including those transactions with higher fees now available because of the attack.

3.2 Attack Description

The overall goal of the attack is to be able to withhold validators from performing their eventual proposer duties, as mentioned in [16]. The idea is that the adversary would interrupt validators using a DoS attack. The attack is divided into two parts. First, we need to know validator IP addresses, which are found by de-anonymizing the validators. This part of the attack will be explored below. The second part of the attack is to determine when a specific validator will produce a block. By utilizing the de-anonymized validators, the adversary should be able to perform a DoS attack on the upcoming block proposers.

3.2.1 Getting Validator IPs

When a node is a part of the Ethereum network, it receives messages in the form of attestations from other nodes in the network. These attestations are split into two different categories: backbone and non-backbone attestations. A backbone attestation is typically a routine message sent between peers in the same subnet. Here, the nodes forward messages to each other that they did not construct themselves. To explain non-backbone attestations, we refer directly to the observation in [2].

Observation 1. *An ideal peer will only send us an attestation in a subnet they are not a backbone of if they are the signer of the attestation, and we are in their fanout for the corresponding subnet of the attestation.*

Optimally, all non-backbone attestations that our node receives should be constructed by the node that sent it to us. Therefore, in a perfect scenario, we link the validator associated with the non-backbone attestation to the peer that sent the message.

However, many things are uncertain when sending messages over the internet. For instance, we may never receive a message because of network instability. As stated by [2], nodes can also run non-default configurations, i.e., subscribing to more than two subnets, increasing the number of messages one will receive from that node. We may also wrongly label messages as backbone or non-backbone because of a lack of or delayed information regarding a node's backbone subnets.

Because of this, the authors of the De-anonymizing Paper developed a heuristic to tell if a given validator could be linked with a given node [2]. The heuristic and its application are described in section 2.7.

Assuming that we will be able to de-anonymize validators, we can take advantage of the fact that an Ethereum node always needs to include its ENR when sending messages over the network. As explained in section 2.6, the node IP is always included in the ENR, which this attack takes advantage of.

3.2.2 Finding Proposer Duties for a Proposer DoS attack

Applying the aforementioned heuristic to the logged data will enable us to get a list of de-anonymized validators in which the IP addresses of the nodes are included because of the ENRs. Now that we have linked validators with IP addresses, we need to determine which validators will propose upcoming blocks in the blockchain.

Having de-anonymized validators means that the RANDAO proposer selection is vulnerable, but not limited, to a DoS attack. More specifically, performing the Proposer DoS attack as mentioned in section 3.1 is possible.

In section 2.5, we found that the proposers for an entire epoch are chosen at least one epoch in advance. Gathering the proposer duties can be easily obtained, as Ethereum provides an API endpoint for the blockchain, providing information on the chain [4]. The adversary would only need an Ethereum node to use this API; a validator is not

4. The API calls are found at ethereum.github.io/beacon-APIs

required. Not needing a validator is especially beneficial, as one can perform the attack without staking any money into Ethereum.

The output of a call to the API returns a list of slots. Each component in the list is the chosen validator's public key, ID, and the slot in which they are chosen to propose a block.

Therefore, a list of the de-anonymized validators could be iterated, with the adversary's goal being to match the proposers for the upcoming epoch. The de-anonymization includes the IP addresses of the validators as well as their ID and public key. So, having matched the upcoming proposers with de-anonymized validators, an adversary now ideally knows the IP addresses of several upcoming proposers.

This information allows the adversary to DoS the specific IP address, which could leave the proposer inactive at their designated slot.

3.3 De-anonymization Attack Implementation

Being able to de-anonymize validators revolves around modifying a fork of the Prysm consensus client.

To implement the attack, one must find where the different functionalities are written in the code. Modifying just two classes in the code is enough to make the attack possible: one for handling peers and another for handling attestations.

It is also important to remember that the core functionality of Prysm is never changed. So, no changes have been made to variables already used in the Prysm client.

3.3.1 Peers

The attestation logs include all data also included in the peer logs. Therefore, the peers will be described first. All changes are done to the `beacon-chain` files in Prysm.

A pointer to a `Service` object is typically maintained in the different classes of the Prysm code. This object implements an interface for handling functionality coupled with the given class. The class that handles P2P connections also has such an object, with several specific functions for handling a peer.

Because of said object, information about all earlier or connected peers is available. Set to run every minute, the modified code requests all peers and iterates through them, logging information.

For a peer, by accessing the `service` class in the `p2p` files, we are able to log:

- Node ID
- IP address
- Subscribed subnets
- Connection state
- Connection direction (which node discovered the other node)
- ENR

Also, in each iteration, the collected information is sent to an `SQL` database to process the data later.

3.3.2 Attestations

Everything logged for the peers is also logged for the attestations, as there is always an underlying node sending each attestation.

The class handling the attestations called `validate_beacon_attestation`, also has a `Service` interface implementation. This implementation primarily includes functions specific to the attestations and not peers. Therefore, each `Service` object has a pointer to a configuration of the running chain. Referencing this pointer allows the class to access functions like those in the P2P class.

We log information about an attestation whenever we receive one.

With the proposer DoS attack in mind, we are interested in receiving the public key and ID of the validator, the creator of the attestation. Also, we want the IP address of the peer that sent the attestation to the node.

Getting the public key uses an existing function called `extractPublicKey`, which uses the attestation's data to get the public key and ID of the validator. Getting the IP address is also straightforward, as the peer `Service` object includes a function called `address`, returning the address of the peer who sent the attestation.

In addition to this, we also log the following information available from processing the attestation:

- Subnet which the attestation was sent to
- Slot associated to the attestation
- Source
- Target

After collecting all the information available for an attestation, the data is logged to an `SQL` database.

3.4 Experimental Setup

For the experiment, we modified the implementation of the Prysm consensus client [19]. Prysm is one of the most used clients for connecting with the consensus layer [20]. The Prysm node is modified to subscribe to all 64 network subnets, enabling it to receive as many attestations as possible. Additionally, aiming for experimental consistency with the De-anonymizing Paper, we allow our node to connect to up to 1,000 peers.

The only change is that we log the data as explained in section 3.3. Apart from that, our node performs tasks like any other Ethereum node would.

We aim to de-anonymize validators on as many peers as we come in contact with. However, this requires a lot of computational resources when the node is subscribed to all subnets. Our instance of the modified Prysm node was running in cooperation with a Go-Ethereum execution client. The client was then run on a virtual machine hosted on Strato CLAAUDIA through Aalborg University. The virtual machine had 16 virtual CPUs and 64 GB of RAM available. The physical location of the virtual machine is in Denmark.

While still aiming for consistency concerning the De-anonymizing Paper, we collect data over three days (December 13th, 13:04 UTC to December 16th, 13:12 UTC). Around 300 GB of data was collected into a PostgreSQL database and processed in PostgreSQL.

TABLE 1
Distribution of nodes into the four different categories.

	Nodes	Distribution
Deanonymized validators	462	38.661%
64 subnets	71	5.941%
No validators	422	35.314%
Rest	240	20.084%

The client was connected to the Holesky testnet to start receiving attestations and connecting with other peers. Since the experiment is of an adversarial nature, the client was connected to a testnet instead of the mainnet.

While the client was running, it was able to log information on the received attestations and the peers it discovered. We logged the same peer and attestation data as described in section 3.3 The data that was logged was then stored in a database for further analysis. After this, the peers were sorted into the same four categories mentioned in section 2.7, initially developed in the De-anonymizing Paper [2].

4 Results

4.1 Connected peers

Throughout the experiment, we kept track of the number of peers our node was connected to. The number of connected peers is shown in Figure 2. Our node was connected to at most 299 peers at once, and the average was 148 peers. The number of connected peers fluctuated widely throughout the experiment but continually rose for the first 12 hours. Then, it would heavily drop and rise again. This pattern would repeat, but after the first drop, it drops consistently every 7 hours.

4.2 De-anonymization

During the attack, we discovered a total of 1,195 unique peers. The results of the de-anonymization attack are shown in Table 1. The table showcases the distribution of peers into four categories based on the heuristic described in section 2.7. Here, we see that 38.661% of the peers we have encountered have had at least one validator that has been de-anonymized, meaning that the conditions from section 2.7 are fulfilled. We also see that 5.941% of the peers we discovered are subscribed to all 64 subnets. We did not find validators on 35.314% of the peers we discovered, meaning that we did not receive a single non-backbone attestation from any validator on these peers. Lastly, we discovered that 20.084% of peers did not fit into the other categories. Failing to categorize a peer could be because the peer could be hosting a validator, but we could not, with enough confidence, say that to be the case based on our heuristic.

After the attack, we managed to de-anonymize a total of 492,959 validators corresponding to 28% of the total number of validators in the Holesky network[5]. 158,134 of these validators were non-unique, meaning that they had more than one IP address mapped to them over the duration of the attack. The results are shown in Table 2.

In total, we logged 1,183 unique IP addresses.

5. As of 2024-01-08 seen on holesky.beaconcha.in

TABLE 2
The number of validators located by our modified Prysm node. The first column indicates the total number of validators. The second column indicates validators with more than one IP address mapped to them.

	Validators	Non-Unique Validators
Overall	492,959	158,134

4.3 Validator Distribution

From all the peers we were connected to, we tallied the number of validators on each peer. The cumulative distribution of validators on each peer is shown in Figure 3. Here, we can see that 57% of the peers we connected to have at least one validator. The amount of validators on each peer ranges from 0 to 87,028, with the average being 1,003 validators per peer. We see that the most common number of validators on a peer is 1, with about 7% of the found peers having only one validator. We also saw that 110 and 400 were the most common numbers of validators on a peer once we got above 20 validators on a peer. We found a total of 24 peers that had 10,000 or more validators on them.

5 Discussion

The following section will discuss the results of the experiment. One of the purposes of the paper is to reproduce the attack mentioned in [2]. Therefore, this section compares these results against each other, exploring similarities and differences.

As our experiment handles data that could be coupled to individuals, we also explore ethical considerations regarding our work.

5.1 Ethical considerations

As the paper tackles an attack on the Ethereum network, it is important to consider the ethical implications of the work.

These implications are mainly related to the potential harm that the attack could cause. Because the attack has not been mitigated yet and can cause monetary loss, we have chosen to keep the GitHub repository containing the attack code private.

These considerations are also why we only ran the attack on a testnet, not the mainnet. Running the attack on the testnet has benefits in the form of not having to worry about losing any real ether. There are also more active validators to reach than the mainnet.

Though, it also has some drawbacks. The main drawback is that the testnet does not have the same restrictions for entering a validator as the mainnet, which could affect the attack's results. Since one does not have to put any money into the testnet system to have a validator, the density of irregular validators will be higher. This could affect the results of the attack compared to the original paper, which was run on the mainnet.

Even though the attack is run on a testnet where the money in the system is not a problem, the attack could still have some negative consequences. Since the attack seeks to deanonymize validators by gaining access to the IPs of nodes in the system, it could lead to a loss of privacy. This

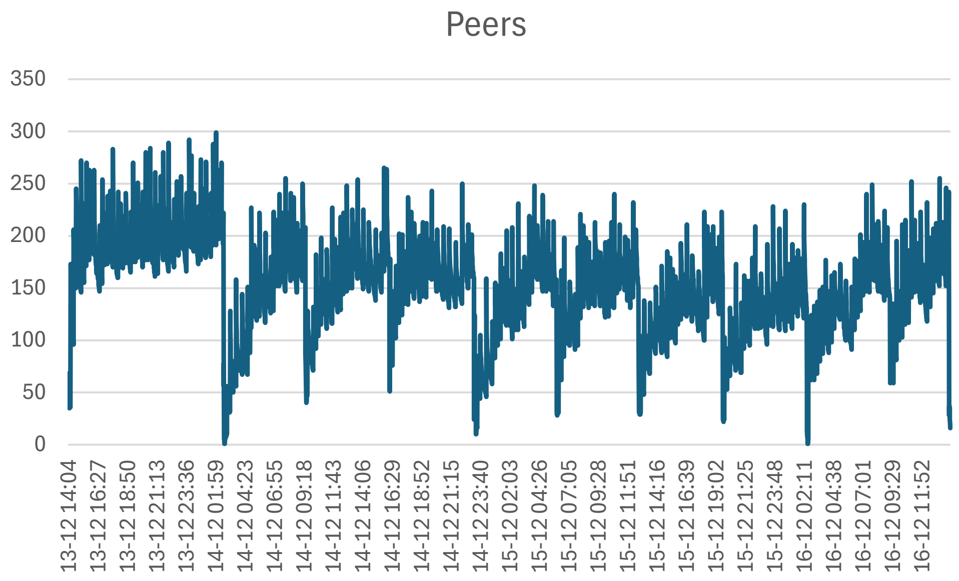

Fig. 2. The number of peers our node was connected to over the time of the experiment. Data points are collected every minute and the average amount of peers connected is 148.

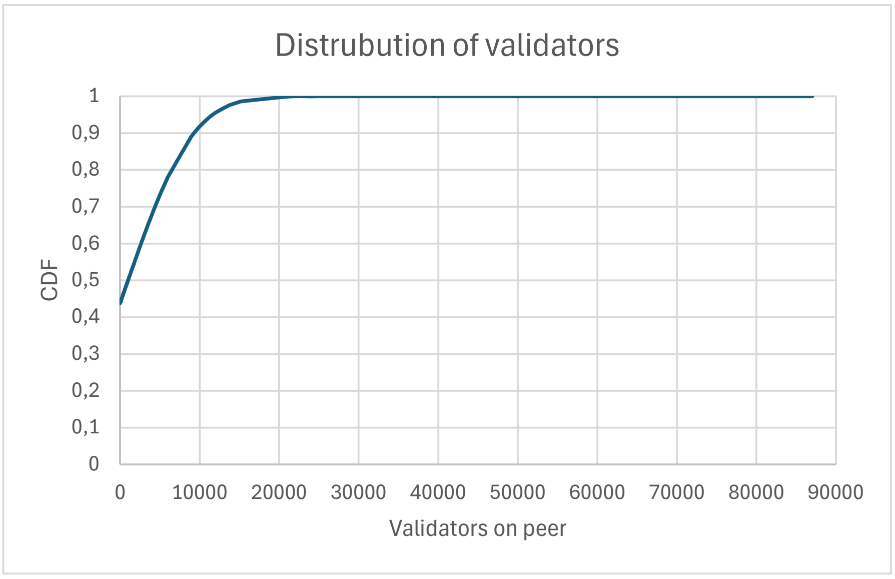

Fig. 3. The cdf showing the distribution of validators on each peer

issue was important to consider when collecting data for the attack.

The data collected was stored in a database and was only used to analyze the attack.

5.2 Comparison of results

5.2.1 Connected peers

As we can see in Figure 2, the number of connected peers fluctuated widely throughout the experiment. It also follows a similar pattern to the one in the original paper regarding sudden drops in the number of connected peers at set intervals. In the original paper, they also experienced sudden drops in the number of connected peers, and they are not sure why this happens. They suspect that it could be due to some artifacts from AWS. However, we also experienced these drops. In our case, we are not running on AWS but on Strato CLAAUDIA, a cloud computing service made available by Aalborg University. So either CLAAUDIA has the same artifacts as AWS, or the drops are due to another reason. We suspect the drops could be due to some garbage collection of peers who have been idle for too long. Each node calculates a peer score for each of its peers to tell whether it should stay connected to that node. Therefore, the drop in peers could be due to how other peers calculated our node's peer score. If something caused our node to get a bad peer score, the number of connected peers to our node could drop. The reason that our node could get a bad peer score could relate to performance issues. In Figure 2, we see that drops happened when around 250 peers were connected. It could be that logging attestations from this many peers hit a bottleneck for our node, slowing it down and hence receiving a bad peer score.

5.2.2 De-anonymization

The distribution of peers into the four different categories differs in some areas. In the De-anonymizing Paper, two of the four nodes show the same de-anonymization rate as we found in our experiment. These are the same two nodes that also share the same peer connection amount as we have. The SO node shows the closest rest rate to our node, the highest of the four nodes. Regarding the no validators category, we see that the De-anonymizing Paper has a higher rate of nodes with no validators on their SO and FR nodes, but similar on VA and ZH. The 64 subnets category is the one that differs the most between the two experiments. The De-anonymizing Paper has a rate of nodes subscribed to all 64 subnets, nearly 9 times smaller than the one we find.

There could be a few reasons for these differences between the results. The main one is the type of network the attack ran on. Since we ran the attack on the Holesky testnet, the behavior of the nodes is different from what it would be if they had been on the mainnet. This is why we see an increase in the number of nodes deviating from the default two subnets and being subscribed to all 64 subnets. The non-default behavior could explain the higher rate of nodes in the rest category since more nodes would be turned on and off and switch IPs more often.

This could also be why the amount of non-unique validators is higher. Because even though we found more validators than the De-anonymizing Paper and de-anonymized a greater portion of the network, we found a higher rate of non-unique validators comparatively. We attribute this to the large number of validators that inhabited some of the nodes we connected to.

5.2.3 Validator distribution

When we compare the results from the De-anonymizing Paper to our results, we see that the distribution of validators on peers is very different. In the De-anonymizing Paper, the amount of validators on each peer tops out at a little over 1,000 validators per peer. Our finding, however, shows that the number of validators on each peer could reach up to 87,028 validators per peer. One explanation could be that the testnet has a higher density of validators than the mainnet because it is easier to become a validator on the testnet as no money is staked. In the De-anonymizing Paper, they also state that 27% of the peers they connected to had only one validator. Opposite to this, we found that only 7% of the peers we connected to had just one validator.

6 Conclusion

After looking into Ethereum and implementing the de-anonymization attack on validators, we have found that the attack is indeed possible. Our results reaffirm the findings of the De-anonymizing Paper and further highlight the Ethereum network's security issues. The issue is that even with zero ether staked, an attacker can gain access to a lot of information about the validators in the network, most importantly their IP addresses. This information can then be used to perform attacks on the network, such as a DoS attack on block proposers, potentially leading to a loss of money for the honest actors within the Ethereum network.

Our research has also shown a difference in the makeup of the Holesky testnet compared to the mainnet. The results of the de-anonymization attack on the Holesky testnet show that there are more irregular validators on the testnet compared to the mainnet. It also shows that the amount of validators on an individual node is many times higher on the testnet than on the mainnet. The difference is likely linked to the low risks of the Holesky testnet. So, even though the two networks are fundamentally the same, the ecosystem within them is different, as expected.

7 Future work

7.1 Potential mitigations

At the moment, there exists an improvement proposal to include a Single Secret Leader Election (SSLE) mechanism, called Whisk, in Ethereum [21].

This method aims to improve the network's security by obfuscating the proposer's identity. It requires the validators to commit to a shared secret, which can be bound to a validator's identity and randomized to match that specific validator's identity.

Every epoch, a random set of validators is chosen to gather commitments from a set of validators using RANDAO. Proposers shuffle the commitments for 8,182 slots ($\sim$24 hours). At each shuffle, the proposer must construct a Zero-Knowledge Proof (ZKP) to prove that it has shuffled honestly while not revealing how the set was shuffled. The proof will then become part of the blockchain for other validators to verify. At the end of the 8,182 slots shuffle, RANDAO is then used to map the shuffled list onto the following 8,182 slots in the same way it has been done since Ethereum used PoS. Validators can now decrypt the commitment that matches their identity and propose a block in the slot that they are assigned to.

This whole process is an example of ZKP where the point is that every validator can prove that it is their turn to propose a block without revealing their identity. If an attacker tried to fetch the upcoming proposers, he would only retrieve one of these commitments. Therefore, it is unfeasible for an adversary to perform the Proposer DoS attack since the adversary would not know which validator to target.

However, it does not prevent the data collection part of the attack, which is used to de-anonymize the validators.

But hindering a DoS attack in itself is a good reason to look at implementing SSLE in Ethereum as a further step to improving the network's security.

7.1.1 More Nodes

The attack we have performed in this paper is based on our ability to gather information about the validators in the network by logging attestations. But since every validator has multiple tasks, those being broadcasting and aggregating attestations and proposing blocks, increasing the number of nodes each validator uses could be a possible solution to the attack. The validators could then use one node for broadcasting and aggregating attestations and another for proposing blocks. This separation would make it harder for the attacker to DoS the proposer when proposing a block since all the attestations the attacker has gathered would be from a different node with a different IP than the one proposing the block.

This mitigation would not stop an attacker from de-anonymizing attesters, but it would make it harder for the attacker to perform a DoS attack on the validators. However, it would increase the system's entry threshold for validators since they would have to run more nodes.

7.1.2 K-anonymity

Another possible solution to the de-anonymization part of the attack could be to implement a K-anonymity system. Using the Prysm Ethereum client, it is possible to choose trusted peers as additional relays for the messages. This would make it harder to de-anonymize validators, as attestations between a K-anonymity group would be sent by an arbitrary peer in that group. Therefore, it would be harder for the adversary to know if the sender of a non-backbone attestation was the creator of it. This means that an attacker must attack K nodes from the same K-anonymity group to halt a proposer in that group.

This solution would make it harder for the attacker to de-anonymize validators and perform the DoS attack. However, this would also require new validators in the system to have a set of trusted peers that they can use as relays for their messages. This would also increase the threshold for entry in the system for validators and the latency within the system due to the increased number of hops the messages would have to go through.

7.2 Building on the experiment

In the De-anonymizing Paper, they ran their experiment on the mainnet while we ran it on a testnet. The difference means that the results of the two experiments could differ due to the dissimilarities in the two networks. It would be interesting to run our experiment on the mainnet to see if the results would more closely align with the original paper.

Another place to look for further research would be to implement the DoS attack as a follow-up to this paper. Since this paper mainly focuses on the data collection part of the attack, it would be interesting to see the consequences of the DoS attack trying to stop a proposer from proposing a block using the data collected. We can see a clear use case for the data collected in the DoS attack, and it is clear that this is a possible attack vector that could be used to obstruct the proposer from proposing a block and gain the rewards that come with it. However, this would need to be tested on a local testnet since, unlike the data collection part of the attack, the DoS attack would harm the network and cause a loss of ether for the validators being attacked if successful.

8 Acknowledgements

We want to express our sincere gratitude to Daniele Dell'Aglio and Michele Albano for their supervision and guidance throughout this project.

We also acknowledge the usage of AI tools such as ChatGPT, GitHub Copilot, and Grammarly. These have been used for clarification and implementation purposes.

References

[1] ethereum.org, "Proof-of-stake (pos)," 2024, Accessed: 23-10-2024.

[2] L. Heimbach, Y. Vonlanthen, J. Villacis, L. Kiffer, and R. Wattenhofer, *Deanonymizing ethereum validators: The p2p network has a privacy issue*, 2024. arXiv: 2409.04366 [cs.CR].

[3] H. Chen, M. Pendleton, L. Njilla, and S. Xu, "A survey on ethereum systems security: Vulnerabilities, attacks, and defenses," *ACM Comput. Surv.*, vol. 53, no. 3, Jun. 2020, ISSN: 0360-0300. DOI: 10.1145/3391195.

[4] A. H. H. Kabla, M. Anbar, S. Manickam, T. A. Al-Amiedy, P. B. Cruspe, A. K. Al-Ani, and S. Karuppayah, "Applicability of intrusion detection system on ethereum attacks: A comprehensive review," *IEEE Access*, vol. 10, pp. 71 632–71 655, 2022. DOI: 10.1109/ACCESS.2022.3188637.

[5] V. Buterin, "Eip-150: Gas cost changes for io-heavy operations," *Ethereum Improvement Proposals*, no. 150, Sep. 2016, Accessed: 18-10-2024.

[6] V. Buterin, "A state clearing faq," Nov. 2016, Accessed: 18-10-2024.

[7] G. Wood, "Eip-161: State trie clearing (invariant-preserving alternative)," *Ethereum Improvement Proposals*, no. 161, Oct. 2016, Accessed: 18-10-2024.

[8] @wackerow, "Attestations," Accessed: 20-12-2024.

[9] Ethereum.org, "Staking," Accessed: 18-11-2024.

[10] @corwintines, @mcmoodoo, @aslikaya, and @nhsz, "Proof-of-stake rewards and penalties," Accessed: 18-11-2024.

[11] Ethereum, "Consensus spec, phase 0," Accessed: 18-11-2024.

[12] @corwintines, @pettinarip, @nhsz, and @nalepae, "Random selection," Accessed: 21-11-2024.

[13] B. Edgington, "Randomness," in *Upgrading Ethereum*, 2023, pp. 134–149.

[14] F. Lange, "Eip-778: Ethereum node records (enr)," *Ethereum Improvement Proposals*, no. 778, Nov. 2017, Accessed: 15-11-2024.

[15] ethereum.org, "Secret leader election," 2024, Accessed: 22-10-2024.

[16] ethereum.org, "Ethereum proof-of-stake attack and defense," 2024, Accessed: 22-10-2024.

[17] @corwintines, "Maximal extractable value," Accessed: 21-12-2024.

[18] A. Wahrstätter, L. Zhou, K. Qin, D. Svetinovic, and A. Gervais, *Time to bribe: Measuring block construction market*, Cryptology ePrint Archive, Paper 2023/760, 2023.

[19] P. Labs, "Prysm," Accessed: 12-01-2024.

[20] E. Alpha, "Client diversity," Accessed: 12-01-2024.

[21] V. Buterin, "Secret non-single leader election," 2024, Accessed: 22-10-2024.

[22] R. Sarenche, E. N. Tas, B. Monnot, C. Schwarz-Schilling, and B. Preneel, "Breaking the balance of power: Commitment attacks on ethereum's reward mechanism," 2024. arXiv: 2407.19479 [cs.CR].

[23] C. Schwarz-Schilling, J. Neu, B. Monnot, A. Asgaonkar, E. N. Tas, and D. Tse, "Three attacks on proof-of-stake ethereum," in *Financial Cryptography and Data Security*, I. Eyal and J. Garay, Eds., Cham: Springer International Publishing, 2022, pp. 560–576, ISBN: 978-3-031-18283-9.

[24] G. Kadianakis, "Whisk: A practical shuffle-based ssle protocol for ethereum," 2024, Accessed: 22-10-2024.

[25] J. Neu, E. N. Tas, and D. Tse, "Two more attacks on proof-of-stake ghost/ethereum," in *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, ser. ConsensusDay '22, Los Angeles, CA, USA: Association for Computing Machinery, 2022, pp. 43–52, ISBN: 9781450398794. DOI: 10.1145/3560829.3563560.

[26] U. Pavloff, Y. Amoussou-Guenou, and S. Tucci-Piergiovanni, "Byzantine attacks exploiting penalties in ethereum pos," in *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2024, pp. 53–65. DOI: 10.1109/DSN58291.2024.00020.

[27] ethereum.org, "What are zero-knowledge proofs?," 2024, Accessed: 15-10-2024.

[28] K. Charbonnet, "A technical introduction to maci 1.0," 2022, Accessed: 15-10-2024.

[29] M. Zhang, R. Li, and S. Duan, *Max attestation matters: Making honest parties lose their incentives in ethereum PoS*, Cryptology ePrint Archive, Paper 2023/1622, 2023.

Appendix A
Attacks on Ethereum

In the following section is an explanation of different attacks that either can or could happen to the ethereum blockchain. There is a description of both the attack and its possible mitigation. If found, there will be a link to a Proof-of-Concept (PoC) of the attack and/or the proposed/implemented mitigation.

A.1 Reorg

One of the reorganization attacks is *Commitment Attacks on LMD GHOST* [22]. It works by using financial incentives to convince validators to vote for a prior block in the chain, by saying it is the block you will be commiting to, to try and exploit there laziness. Possible because around 90% of validators use software called "MEV Boost", which tries to earn you the most money.

Another reorg attack is Short-range reorg, which uses short-range reorgs of the blockchain stipulating consensus to delay the finality of consensus decisions [23]. Such short-range reorgs also allow validators to increase their earnings from participating in the protocol. It does this by withholding a block and then releases it timed with the next honest block to orphan it. This attack requires a large amount of stake to be held by the adversary, with 30% being the aim and everything below only reducing the chance of success.

A reorg attack that has been inspired by the last attack is Low cost long-range reorg attack[23]. This attack works by the adversary avoids competing directly with honest validators of $(k-1)$ committees, as done in the short-range reorg attack. Instead, the adversary uses the technique of balancing attacks to keep honest committee members split roughly in half by ensuring they have different views on what the current head of the chain is. This makes honest nodes work against each other to maintain a tie which the adversary can tip to their liking at any point using only a few votes. A possible mitigation is the recently proposed Transaction Encapsulation approach demonstrates that some cross-layer attacks can be alleviated or prevented by dynamic transaction .

A different reorg attack is called $>= 50\%$ stake attack [16]. The attack works if the adversary has over 50% stake, the adversary would be able to have the majority vote every time in the fork choice algorithm. This would mean that all honest validators need to vote along with the adversary to not get their ETH burned due to the inactivity leak security protocol. Because of said protocol, the adversary would also eventually gain finalization. This attack also makes it possible for the adversary to easily perform a reorg of a block. It is thought that given the large amount of stake needed for the attack to work, this attack would not be feasible.

A.2 DoS

We've found three different kinds of DoS attacks that either were or are possible to perform on Ethereum.

One of the attacks is called *under-priced opcodes* [3, 4]. This attack works because Ethereum has a gas mechanism to reduce abuse of computing resources. Though when a contract has a lot of underpriced opcodes, they will consume many resources. Execution of contracts requires a lot of resources.

To mitigate this, Ethereum has raised the gas cost of opcodes to preserve the number of transactions-per-second [5] [6].

Another attack, which is closely related to the former, is *empty account in the state trie* [3, 4]. This attack was possible because the existence of empty accounts increases the transaction processing time and synchronization. An empty account is an account with zero balance and no code. The attack required the proposer to select only the transactions of the adversary, which could be insured by offering a higher gas price.

The mitigation is a combination of the one explained for *under-priced opcodes* as well as a mitigation for clearing empty accounts [5–7] [7].

The last example of a DoS attack is called *Proposer DoS* [15, 16]. The background to making this attack possible is that the consensus mechanism uses a publicly known function for choosing the upcoming block proposers. The adversary is therefore able to compute this in slight advance of the blockchain, s.t. each proposer is now known. After this, the adversary can map the proposer's IP addresses and overload their connection. A successful attack would leave a proposer unable to propose their block in time.

To prevent this kind of attack, Ethereum plans to use something they call SSLE which ensures that only the selected validator knows that they have been selected [15, 21].

Specifically, a proposal has been made to use an election protocol called Whisk, which is a type of SSLE [24][8][9][10]. It works by each validator submitting a commitment to a secret shared by all validators. The commitments are shuffled s.t. no-one can map commitments to the validators, but each validator knows what commitment belongs to them. This shuffle-phase goes on for a day, 256 epochs, before using the shuffled proposer list the following day. Commitments are chosen at random, and the selected proposer will detect its commitment to know when to propose a block.

The shuffling phase requires validators to occasionally shuffle a subset of candidate proposers. Using a subset is a measure to reduce computation for the validators, as 256 epochs correspond to 8912 proposers. The shuffle requires a validator to construct a ZKP to confirm that the shuffle was performed correctly.

A.3 Balancing Attack

For this type, we found two attacks.

The first attack we call *LMD-specific balancing attack* [25]. This attack exploits the Latest Message Driven (LMD) *proposer boosting* by sending out two competing blocks but giving half the validators one block before the other and the opposite for the other half. This would create a fork in the blockchain.

Although no mitigation is mentioned, it requires $W_p/b + 1$ adversarial slots [11].

Another balancing attack has the adversary exploiting adversarial network delay and strategic voting by a vanishing fraction of adversarial validators to stall the protocol indefinitely [23] [12].

Though this attack does depend on networking assumptions that are highly contrived in practice; those being the attacker having fine-grained control over latencies of individual validators.

A.4 Finality Attack (Bouncing Attack)

For the Ethereum, there exist attacks called finality attacks, also known as bouncing attacks, which have the purpose of denying the blockchain to finalize its block, halting its functionality.

The first attack is a *double finality* attack [16, 26]. It is theoretically possible for an attacker that wants to risk 34% of the total staked ether. Two forks finalize simultaneously, creating a permanent split of the chain.

A way to see a mitigation of this is that it is practically impossible given the current value of 34% of the total staked ether. Also, voting on two different chains, called double voting, is a slash-able offense in the Ethereum chain, so the adversary would get their staked ETH slashed.

6. EIPs/EIPS/eip-150.md at master ethereum/EIPs GitHub
7. EIPs/EIPS/eip-161.md at master ethereum/EIPs GitHub

8. Shuffle_SSLE/rust_code/src at master ethresearch/Shuffle_SSLE GitHub
9. [WIP] Introduce consensus code for Whisk (SSLE) by asn-d6 Pull Request #2800 ethereum/consensus-specs GitHub
10. GitHub - dapplion/lighthouse at whisk
11. where W_p is the proposer boost weight (fx 100 validators/slot: $W_p = 0.7 \cdot 100 = 70$) and b is the fraction of adversaries in the committee in each slot
12. GitHub - tse-group/gasper-gossip-attack

Another finality attack is called *33% finality attack* [16]. Here, all adversarial ($\geq 33\%$) validators can simply go inactive, meaning that a block cannot get $2/3$ attestations which is required to achieve finality. Therefore, the blockchain would not be able to go further, as it would not be able to achieve finality.

The mitigation for this is called *the inactivity leak*. The Ethereum chain penalizes the validators who resist voting or are inactive. Their ETH gets burned until the majority vote has a 2/3 majority. This makes the attack impractical for the attacker, as it is costly to have $\geq 33\%$ of the total staked ETH and their ETH gets burned as well.

A.5 Avalanche Attack

This type only includes a single attack that we call *avalanche attack on proof-of-stake ghost* [25] [13]. The attack uses withheld blocks to make wide subtrees to displace an honest chain. It does this by exploiting the reuse of uncle blocks.

The mitigation for this attack is already a part of the Ethereum blockchain [14]. It is mitigated by LMD, which works together with Greedy Heaviest Observed Subtree (GHOST). The protocol only counts a vote from a validator if the vote is strictly later than the current entry. If two equivocating votes were sent from the same validator at the same time slot, only the earlier message would be counted.

A.6 Bribery

An adversary using bribery attacks on the Ethereum chain could be interested in dictating a choice in some sort of voting mechanism. This is exactly what happens in a described *quadratic funding* attack [27]. The adversary researches votes on the chain and bribes users to vote for what the adversary wants.

In defense of a potential bribery attack, the Ethereum blockchain implements a private voting system called Minimum Anti-Collusion Infrastructure (MACI) [27, 28] [15].

What MACI does is essentially hiding what each person has voted for. It does so by demanding the voters to send their votes encrypted to a central coordinator. This coordinator constructs Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (ZK-SNARK) proofs, which verifies that all messages were processed correctly, and that the final result corresponds to the sum of all valid votes.

As votes are now hidden, the adversary is not able, by oneself, to prove that the bribee voted in way of said bribery. Though the bribee could decrypt their own message and show the vote to the adversary.

MACI has fixed this problem by implementing public key switching. This means that a voter can request a new public key. In addition to this, a vote is only valid if it uses the most recent public key of the voter. Therefore, a bribee can show its first vote obeying the adversary, generate a new public key, and send a new, now honest, vote. The old vote will then become invalid as it uses a deprecated public key.

13. GitHub - tse-group/pos-ghost-attack

14. Proposer LMD Score Boosting by adiasg Pull Request #2730 ethereum/consensus-specs GitHub

15. GitHub - privacy-scaling-explorations/maci: Minimal Anti-Collusion Infrastructure (MACI)

A.7 Staircase Attack

The last type of attack that we cover is called a *staircase attack*. This is a two-part attack, with a warm-up attack and a full attack [29] [16].

First, we will cover the *warm-up attack*. The attack works when the adversary, called a, has to propose the first block of an epoch. From there, the attack works in four parts:

1) The adversary withholds its block at slot t
2) The honest attestors in slot t create attestation with the last block of the previous epoch, called b.
3) Adversary a releases block b_t and honest validators update their checkpoint to block b_t.
4) Attestations with targets different from b_t will be discarded, and honest attestors in slot t will be penalized eventually

This attack is mitigated by what Ethereum calls *honest reorg*, which should prevent intentionally withheld blocks [29].

A block proposed in slot t with fewer than 20% attestations is considered invalid. Though it can be theoretically avoided with network timing, ensuring at least 20% of the attestors get the block.

With the *warm-up attack*, the author goes on to describe a full *staircase attack*. The attack starts with the *warm-up attack*. Then the plan is to manipulate the source of the honest validators. Half the validators should get an outdated *last justified checkpoint* and be penalized. All byzantine validators withhold their blocks and publish them in the middle of the epoch. It should be able to be done as a one-time attack with probability 98.84% if adversary controls N/3 validators [29]. The byzantine validators don't get penalized, but will get a smaller reward than the fair share.

What would make this attack infeasible is that it would require a lot of controlled validators. The author states that for Ethereum to be vulnerable, there would need to be < 16384 validators, which makes the attack as good as impossible given Ethereum has $1.073.406$ daily active validators [29] [17]. Also, Ethereum released a patch fixing this attack after the author pointed it out [18].

16. GitHub - tsinghua-cel/Staircase-Attack: This is the staircase attack implement for the paper "Max Attestation Matters: Making Honest Parties Lose Their Incentives in Ethereum PoS."

17. According to beaconcha.in as of 22-10-2024

18. Confirmation Rule by saltiniroberto Pull Request #3339 ethereum/consensus-specs GitHub