Zero-Knowledge Proof for Attack Prevention in The Ethereum Blockchain

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

The Ethereum network is a decentralized network and is one of the most popular blockchains. At the current time 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 2020 with the release of Ethereum 2.0 [1]. A part of this decision was to reduce the requirements for participation in the network. With the increasing number of validators, the network is becoming increasingly active with the amount of messages being sent between the validators. With all of these messages being sent, the network is becoming increasingly vulnerable to attacks based on information gathering. That is why we in this paper are looking into the possibility of de-anonymizing validators in the Ethereum network by identifying and linkin IPs to active validators.

Without having to be a validator, we have been able to gather information about the validators in the Peer-to-Peer (P2P) network through attestations sent in throughout the Ethereum network. We were able to gather enough information purely from logging messages we received and peers we were connected to, to be able to de-anonymize validators. The information gathered from our attack can then be used to perform attacks such as a Denial-of-Service (DoS) attack on other proposers when combined with the fact that the proposers of each block is known in advance. This could potentially lead to a loss of money for the validators, as they are penalized for being inactive as well as a potential gain for the attacker in the event of victim being the proposer in the slot before the attacker.

This paper was inspired by the work of [2] and we have made our own version of the implementation. This has lead os to make the following contributions:

- We have implemented a de-anonymization attack on validators designed to be running on the Ethereum network. The attack was run on the Holesky testnet for research purposes.
- We have described a possible DoS on block proposers, which is made possible after execution of the de-anonymizing validator attack.
- We have documented our results of the de-anonymization attack and compared those against the results from the original authors of the attack [2]
- We discuss the incentives and consequences that arise after a successful execution of the DoS attack on block proposers.

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

2 Background

To get a better understanding of the attack that we will be performing in this paper, we need to go through some of the concepts that are used in the attack. 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 done to reduce the energy consumption of the network and to increase the scalability of the network. 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. This includes guidance on whom gets to produce blocks and confirm transactions. In Ethereum, it works by choosing 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 then 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 for each block a validator is selected as a proposer by the protocol. The proposer is chosen through a random process weighted by the amount of ether the validator has staked. To simulate this random selection, Etheruem uses the publicly available random number generator (RANDAO) algorithm.

A block needs to be proposed by the designated proposer for the slot and confirmed by other validators within

the 12 second time limit of the slot. If a fork happens, the validators have to choose which fork to follow. This is done by using 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 Attestations

Each epoch, a validator is expected to broadcast what is called an attestation [3]. The overall purpose of this message is to make sure that validators agree on the canonical chain.

A single validator is assigned to a single slot each epoch, where they must publish their attestation. In this attestation, the validator among other things votes for what it sees as the *source* and *target*.

The source is what the validator sees as being the most recent justified block.

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

A committee is then all validators who are set to attest in the same slot. All votes broadcast in each subnet of the committee are collected by 16 assigned aggregators per subnet. These collect the votes that are similar to their own into an aggregation and broadcast this to the network. These are then to be included in the block of the designated slot.

2.3 Subnets

The Ethereum network is split up into smaller networks called subnets. Being subscribed to a subnet is also be referred to as being backbone of a subnet. These subnets are used to help with the scalability of the network. The peers a node can reach within the same subnet are called its fanout [2].

The nodes in the network are split into total of 64 subnets and an additional subnet for attestation aggregates with each node being part of at least two subnets. Within a subnet, nodes choose a subset of peers in the same subnet to share its messages with. Choosing which nodes are a part of this subset is done based on the peers performances. Nodes send all messages they receive within a subnet to these best-performing peers.

2.4 Validators

Our attack is primarily focused on the validators of Ethereum. A validator is an entity running on a validator client, which, among other things, consists of a balance and a key-pair for identifying it [4].

For a validator client to run, it needs to be run in cooperation with a consensus node client. A consensus node client is then able to run an unbounded number of validators. Having several validators running in cooperation with a single consensus node means that they all share identity with that node. This includes data such as IP addresses and the ID of the node.

At each slot, the RANDAO algorithm is used to select a validator to be responsible for processing transactions, including them in a block, and then adding this block to the blockchain. This procedure is also illustrated by Figure 1. How to determine who is going to be a block proposer is explained in section 2.5. Along with this, the validator

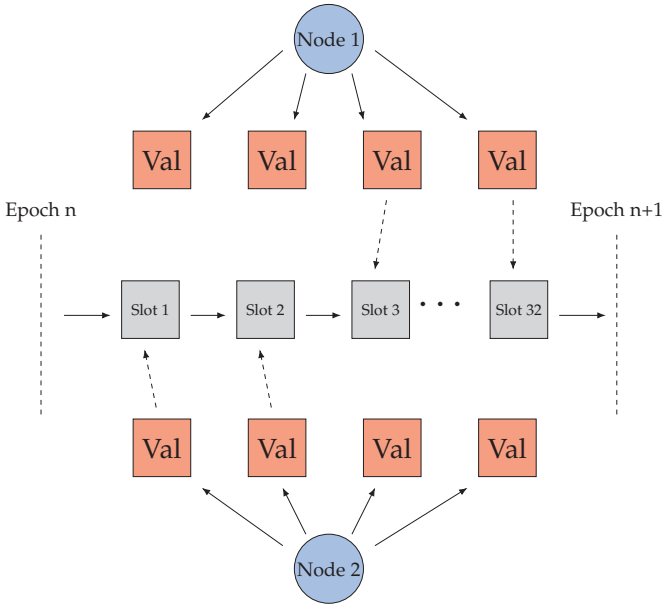

Fig. 1. Illustration of validators selected as proposers through an epoch. Two random nodes are shown, each having several validators. Some of these validators are chosen to propose a block in a designated slot.

is also responsible for attesting blocks proposed by other validators, ensuring the liveness of the chain.

To be able to have a single validator, one must deposit 32 ETH, which is $99,776[1]. Having this much money at stake should be enough to ensure that a validator acts honestly. Doing so will also earn the validator a reward. However, acting dishonest will get your ether burned or slashed. The rewards and punishments are described in the following section.

2.4.1 Validator rewards

Validators are rewarded for several different actions [5]. Each of these actions is rewarded with different weights, but they all depend on the total amount of staked ether by validators and the validator's own staked amount. Though the calculated reward will not get larger above 32 ETH.

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

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

The reward for a validator is then calculated as follows:

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

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

1) Timely source vote: 14
2) Timely target vote: 26
3) Timely head vote: 14

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

4) Sync reward: 2
5) Proposer weight: 8

Though it does not have the largest weight, the most profitable reward is the proposer weight reward. This reward is given to the validator, whenever they are chosen as a proposer and correctly propose a block to the blockchain. In this case, instead of being rewarded this only once, the proposer is getting a reward for each valid attestation included in the block proposed.

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

Because the maximum number of attestations is 128, then in a typical and optimal situation, the proposer will get $BR \cdot \frac{8}{64} \cdot 128$ in reward when proposing a block [5, 6]. Because of that validators does not want to miss being the proposer. especially since the chance if getting to propose a block is relatively low. The reason is first of all that the average amount of staked ether for a validator is over 32 ETH. As the validator's probability of being proposer does not get higher when over 32 staked ether, there is an almost equal chance of being proposer. which means almost equal chance of being proposer, and With Ethereum having over 1 million validators [2].

2.4.2 Validator punishments

A validator can also be punished for doing things that do not contribute to the chain performing as it should. Ethereum has two kinds of penalties, burning and slashing, where a validator loses some of their staked ether. Should a validator lose their ether and end up below 16 staked ether, they will be removed [6].

Burning happens per epoch when 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:

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

with `IPQ` being the *inactivity penalty quotient* set to 2^{26} [6].

This means that a validator being offline in a single epoch still retains $0,99999999255\%$ of their staked ether. A validator would therefore need to be offline for a lot of epochs before eventually being removed for having under 16 staked ether.

Slashing happens when only 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 [5]:

1) Proposing two different blocks at the same slot
2) Attesting a block that surrounds another
3) Double voting - Attesting to 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. Halfway through this, on day 18, an additional penalty is applied. The magnitude of this penalty scales with the total staked ether of the slashed validators in the 36-day period before the slashing event. At worst, a validator can end up having all their ether burned, if enough other validators are also slashed.

2.5 RANDAO

As mentioned in section 2.4, the validators are chosen to propose a block by a random-number-generator called RANDAO[3]. The random selection is decided by the RANDAO algorithm, which is used 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. This, of course, happens at all 32 slots of an epoch, ensuring the randomness of the protocol. At the end of each epoch, the latest reveal constitutes to the seed, which is used to determine who the next proposers are going to be.

Being chosen as a block proposer comes with some duties, such as creating the block. This means that they need to know in advance if they are going to be the proposer of a block.

The proposer selection among the validators is done two epochs in advance [7]. More specifically, the selection for epoch $n + 2$ happens at the end of epoch n [8]. This means that a validator knows at least one epoch, at most two epochs, in advance if they are proposing a block.

2.6 ENR

A Ethernet node record (ENR) is a record that contains information about a node in the network [9]. Ethereum uses ENRs as a way to package the information that is being sent from node to node during the discovery protocol, where nodes discover each other. The package contains information like the node's IP address, port, and public key. Because of the nature of the discovery protocol, if one were to also have a node in Ethereum, one would be able to see the ENR of all discovered nodes. And since the ENR contains the IP address and the public key of the node, you would be able to see the corresponding IP addresses and public keys of all the nodes that have been discovered by the node.

2.7 Inspirational Paper

In the paper "Deanonymizing Ethereum Validators: The P2P Network Has a Privacy Issue" (Deanonymizing Paper) the authors show that it is possible to deanonymize 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 it is possible to get information about the validators on the network. It is also the main inspiration for our attack.

The paper takes advantage of the attestations, including information such as the IP of the sender node, and subnet setup to get information about the validators. For their

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

3. RANDAO - GitHub

setup, they use a custom version of a Prysm node called RAINBOW that subscribes to all subnets, and they use to it log peers and breaking peer inputs down into ”colors” of the validators. This information consists of all attestations, their origin, and their origin subnet, all advertised static subscriptions of our peers and precise connection data for all nodes they interact with.

To enable a broader discovery of the peers, they also used a network crawler to more quickly find the peers in the Ethereum network. Though this crawler and the RAINBOW node are two separate entities.

In their 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 then let the nodes run for three days and managed to deanonymize 235,719 validators and reached out to 11,219 peers. These peers were also divided into four categories based on a heuristic developed by the authors.

The categories are:

- **Deanonymized:** One or more validators have been discovered on this peer.
- **No validators:** No validators have been discovered on this node, as we did not receive a non-backbone attestation.
- **64 subnets:** The peer was subscribed to all 64 subnets, so we are not able to tell backbone attestations from non-backbone attestations.
- **Rest:** Peers where we receive non-backbone attestations, but we were not able to identify any validators.

The heuristics developed for deanonymizing validators include four criteria being:

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 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 we receive for the validator v from the peer p exceeds the mean number of attestations per validator from peer p by two standard deviations.

3 Related Work

The usage of Zero-Knowledge Proofs (ZKPs) in Ethereum is not a new concept. In fact, it currently uses them both on- and off-chain. The following provides a short overview of some of the already existing solutions as well as one still being in development.

3.1 DoS Attack

Some of the instances of DoS attacks that are seen on ethereum ranges from attacks on the proposers to attacks that seek to slow down the network itself. A known attack aims to slow down the network by using underpriced opcodes to create a block that is hard to process [10, 11]. Another way to slow down the network is to create empty accounts that are hard to process [12, 13]. This attack, however, is outdated and has been mitigated by making it near impossible to create empty accounts in the network.

4 Experimental protocol

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

4.1 Motivation

4.1.1 Proposer DoS Attack

In this subsection, we will be describing the attack that we will be using as a basis for our experiment in section 4. The attack is a DoS attack that aims at hitting 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, it seems that this attack is possible to perform [14, 15].

It has been our interest to research the feasibility of this attack and the ones mentioned in section A. This has proven to be a difficult task, given that most of our researched attacks happen in the consensus- or execution layer. Therefore, as a result of the blockchain algorithm, we are not able to clarify the feasibility of the attacks that we have found. For this reason, we have chosen the *Proposer DoS attack* as it seems exciting, has not been mitigated yet, and a potential solutions seems to include a ZKP.

The attack possible is because 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.

4.1.2 Adversarial incentives for the attack

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

The first reason which only requires de-anonymization, could be to exploit the inactivity penalization, which is mentioned in section 2.4.2. The adversary could make validators lose money for being inactive. For the adversaries themselves, they don’t receive any monetary rewards from this attack. One could be interested in getting people kicked from being a validator for being 16 ether. But it can quickly be derived from Equation 4 that it would require the DoS attack to be ongoing for at least 9645 epochs 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 best reward after proposing a block, see section 2.4.1, so the adversary could limit the rewards given out to proposers. In the worst case, this could

lead to a decrease in the total amount of ether, as it is dynamically changing with the chain.

Both of the aforementioned reasons only require a node in Ethereum, but the adversary themselves do not gain any reward from the attack. This 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) [16].

MEV is a reference to the maximum value one can extract from a block production opportunity. It is a reward that the proposer can get besides rewards gotten from working 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 in the chain. There are more implementations of these searchers, fx Flashbots and MEV-Boost. It has been estimated that 90% of all Ethereum validators use this kind of software [17].

So, let us assume a scenario, where an adversary has a validator that is 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 slot $n-1, n-2$, and so on. Beginning from slot $n-1$ and down, the more consecutive proposers that are successfully attacked, the more potential gain there is for the adversary. This is because the high-paying transactions that were to be included in the previous slots now can be included in the adversary's slot. Therefore, the adversary potentially gains a greater reward from including those transactions with higher fees, that are now available because of the attack.

4.2 Attack description

The attack that we are describing is divided in two parts. The overall goal of the attack is to be able to withhold validators from performing their eventual proposer duties as mentioned in [15]. The idea is that the adversary would interrupt validators using a DoS attack. For this, we need information regarding two things; when a specific validator is going to produce a block, and what the IP address of the validator is.

4.2.1 Getting validator IPs

When a node is a part of the Ethereum network, it will receive messages in the form of attestations from other nodes in the network. These attestations can be 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. This means that the nodes are simply forwarding messages to each other that they did not construct themselves. To explain non-backbone attestations, we refer directly to the observation found 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.*

So optimally, all non-backbone attestations that we receive should have been constructed by the node that sent it to us. Therefore, in a perfect scenario, we would couple the validator, associated with the non-backbone attestation, to the peer that sent the message.

Though, there are a lot of things that are uncertain when sending messages over the internet. It may be that we 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 lack of or delayed information regarding a node's backbone subnets.

Because of this, the authors of the Deanonymizing Paper developed heuristics to tell if a given validator could be linked with a given node [2]. The heuristics and their application are described in section 4.4.

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.

4.2.2 Finding proposer duties for a Proposer DoS attack

Applying the aforementioned heuristics 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 should have linked validators with IP addresses, we need to find out which validators are going to 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, it is possible to perform the Proposer DoS attack as mentioned in section 4.1.

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 required. This is especially beneficial, as one can perform the attack without staking any money into Ethereum.

The output of the call to the API returns a list of slots. For 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, where the goal of the adversary would be to match the proposers for the upcoming epoch. Included in the de-anonymization is the IP-addresses of the validator 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 allows for the adversary to DoS the specific IP-address, which could leave the proposer inactive at their designated slot.

4. The API calls are found here.

4.3 DoS attack implementation

As mentioned in section 4.4, the work in this paper revolves around modifying a fork of the Prysm consensus client. An important thing to keep in mind is that the core functionality of Prysm is never changed. So no changes have been made to variables already being used in the Prysm client.

Instead, to be able to implement the attack, it is a case of finding where the different functionalities are written in the code. It is enough to modify just two classes in the code to reach a point where the attack is possible.

As mentioned in section 4.4 two entities are logged, peers and attestations. This translates almost directly to the code in the Prysm client, as each of these can be handled by a class each.

4.3.1 Peers

The logs of the attestations share similarities with the logs of the peers, so the peers will be described first.

A pointer to a `Service` object is typically maintained in the different classes of the Prysm code. This object is an implementation of an interface responsible for handling functionality coupled with the given class. The class that handles P2P connections has got this as well, 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 the information mentioned in section 4.4.

Also in each iteration, the collected information is sent to an `sqlite3` database such the data can be processed later.

4.3.2 Attestations

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

The class handling the attestations also has a `Service` interface implementation. This implementation primarily includes functions specific for 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 as the ones available in the P2P class.

Instead of being logged for once every minute, as in section 4.3.1, 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 of the validator, who is the creator of the attestation. Getting this requires little effort, as there already exists a function which uses the attestation's data to get public keys. We can use this public key in another existing function to get the ID of the validator.

After collection all the information available for an attestation, the data is logged to a database as in section 4.3.1.

4.4 Setup

The primary goal of the experiment is to log as much data from attestations and peers as possible. To do this, we modify the implementation of the Prysm consensus client [18]. Prysm is also one of the most used clients for connecting with the consensus layer [19]. The Prysm node is modified to subscribe to all 64 subnets in the network, which enables it to receive as many attestations as possible. Adding to that, and also aiming for experimental consistency with the Deanonymizing Paper, we allow our node to connect to up to 1,000 peers.

Apart from that, our node performs tasks as any other node in Ethereum would. The only change is that we log the data as explained in section 4.3.

Aiming to log as much data as possible while trying to get as many connections as possible requires a lot of computational resources. 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 as well as 64 GB of RAM available. The physical location of the virtual machine is in Denmark.

While still aiming for consistency in regard to the Deanonymizing Paper, we collect data in the span of 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 afterward also processed in PostgreSQL.

To start receiving attestations and reaching and connecting other peers, the client was connected to the Holesky testnet. 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 from the received attestations as well as on the peers it discovered. Among other things, the client logged peer ENRs, the subnets that the attestations were sent to, the subnet that the peer was in and the public key associated with the attestation. 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, originally developed in the Deanonymizing Paper [2].

5 Results

5.1 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. Over the course of the experiment, the number of connected peers fluctuated a lot but would continually rise for the first 12 hours. Then it would heavily drop and rise again. This pattern would repeat, but after the first drop it would then drop consistently every 7 hours instead.

5.2 Deanonymization

IPs collected = 1183

5.3 Validator Distribution

From all the peers we were connected to, we collected 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 that we connected to have at least one validator. The amount of validators on each

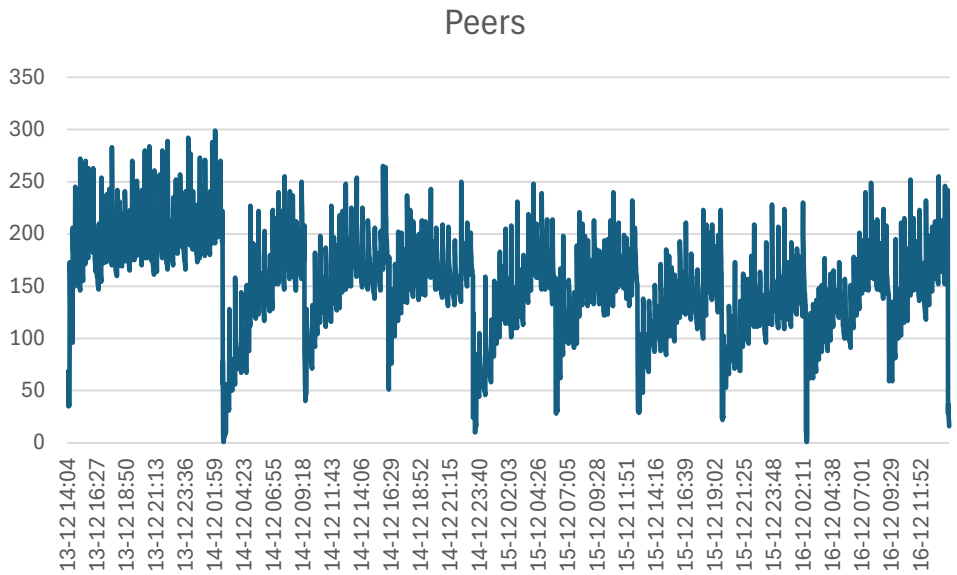

Fig. 2. The number of peers our node was connected to over the time of the experiment. Data points are collected every minute and the average amount of peers connected is 148.

	Nodes	Distribution
Deanonymized validators	513	42.93%
64 subnets	71	5.94%
No validators	422	35.31%
Rest	189	15.82%

TABLE 1
Distribution of nodes into the four different categories

	Validators	Non-Unique Validators
Overall	493,551	158,436

TABLE 2
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 it.

peer ranges from 0 to 87028, with the average being 1003 validators per peer. We see that the most common amount of validators on a peer is 1 with about 7% of the found peers having only one validator. We also saw that having 110 and 400 were the most common amounts of validators on a peer once we get above 20 validators on a peer. We found a total of 24 peers that had 10,000 or more validators on them.

6 Discussion

The following section will discuss the results from 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.

6.1 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 could be caused by the attack. Because of the potential harm, we have decided to not disclose the exact implementation and details of the attack, and limited the information to what is necessary to understand the attack and its implications. This is to prevent any malicious actors from easily replicating the attack and causing harm to the mainnet. This is also why the GitHub repository containing the code for the attack is private.

These considerations are also the reason why we only ran the attack on a testnet and not on the mainnet. Running the attack on the testnet has some benefits in the form of not having to worry about the loss of any real ether. There is also a greater number of active validators to reach compared to 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 in the mainnet, which could affect the results of the attack. Since one does not have to put any money in to the system to become 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.

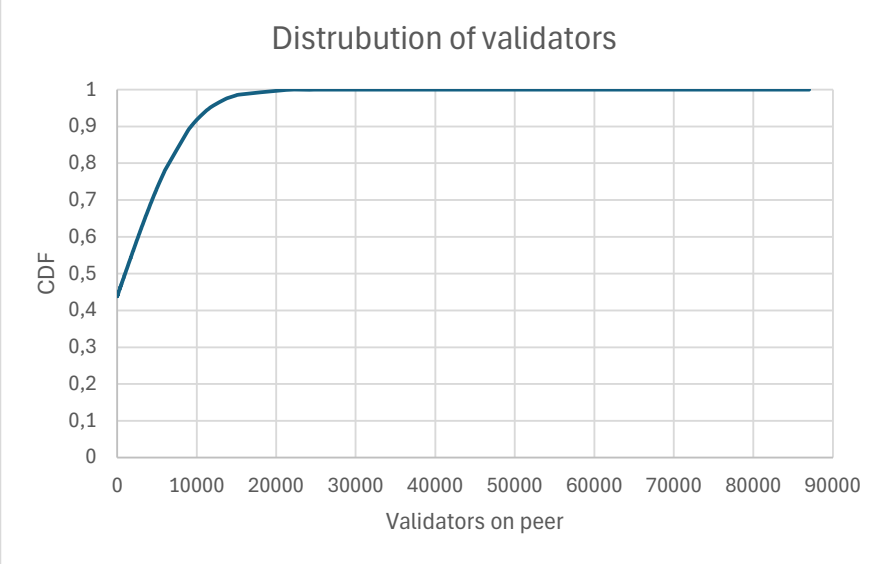

Fig. 3. The cdf showing the distribution of validators on each peer

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

The data that was collected was stored in a database and was only used for the purpose of analysing the attack.

6.2 Comparison of results

6.2.1 Connected peers

As we can see in Figure 2, the number of connected peers fluctuated a lot over the course of the experiment. It also has a similar pattern to the one in the original paper when it comes to the sudden drops in the number of connected peers in 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 too experienced these drops. In our case we are not running on AWS, but on Strato CLAUUDIA, which is a cloud computing service made available by Aalborg University. So either CLAAUDIA has the same artifacts as AWS, or the drops are due to some other reason. We suspect that the drops could be due to garbage collection of peers that have been idle for too long. Each node calculates a peer score for each of their peers to tell whether it should stay connected to that node or not. Therefore, the drop of peers could be a result of how other peers calculated our node's peer score. If something made our node 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. We can see in Figure 2 that drops happen when around 250 peers were connected. It could be that logging attestation from this amount peers hit a bottleneck for our node, making it slow down and hence receive a bad peer score.

6.2.2 Validator distribution

While comparing the results from the original paper to our results, we see that the distribution of validators on peers is a lot different. In the original paper, the amount of validators on each peer tops out at a little over 1000 validators per peer. Our finding however, shows that the amount of validators on each peer could reach up to 87028 validators per peer. One of the explanations for this could be that the testnet has a higher density of validators compared to the mainnet since it is easier to become a validator on the testnet due to money not being a problem. In the original 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.

7 Conclusion

This is a potential conclusion section.

8 Future work

8.1 Potential mitigations

At the moment, there exists an improvement proposal to include a Single Secret Leader Election (SSLE) mechanism, called Whisk, in Ethereum [20].

This method aims to improve the security of the network by obfuscating the identity of the proposer. It does so by making the validators commit to a shared secret which can

be bound to a validator's identity and randomized to match the specific validator's identity.

Every epoch a random set of validators are chosen to gather commitments from a set of validators using RANDAO. Proposers then shuffle the commitments over the duration of 8182 slots. At the end of that period, RANDAO is then used to map the shuffled list onto the 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 is able to prove that it is their turn to propose a block without revealing their identity. If an attacker would try to fetch the upcoming proposers, he would only retrieve one of these commitments. Therefore, it makes it harder for an adversary to perform the Proposer DoS attack since the adversary would not know which validator to target.

Though, 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 security of the network.

8.1.1 More Nodes

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

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

8.1.2 K-anonymity

Another possible solution to the de-anonymization part of the attack could be to implement a k-anonymity system. Using the Prism Ethereum client, it is possible to choose trusted peers as additional relays for the messages. This would make at harder to de-anonymize validators as is would make it difficult to match a validator to a fixed IP address. The group of peers that band together will make up a k-anonymity group. This would then require the attacker to attack multiple nodes at once when trying to stop a proposer from proposing a block.

This solution would make it harder for the attacker to de-anonymize validators and perform the DoS attack. Though, 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 as well as increase the latency within the system due to the increased number of hops the messages would have to go through.

8.2 Further research

Due to time constraints, we were not able to fully reproduce the attack mentioned in [2]. In the original paper, they run their experiment on the mainnet, while we ran our experiment on a testnet. This means that the results of the two experiments could differ due to the differences in the two networks. With more time, 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. This however would need to be tested on a local testnet since unlike the data collection part of the attack, the DoS attack would have a negative impact on the network and cause a loss of ether for the validators that are being attacked if successful.

9 Acknowledgements

We want to express our sincere gratitude to Daniele Dell'Aglio and Michele Albano for their supervision and guidance throughout the duration of 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] @wackerow, "Attestations," Accessed: 20-12-2024.

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

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

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

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

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

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

[10] 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.

[11] 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.

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

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

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

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

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

[17] 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.

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

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

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

[21] 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].

[22] 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.

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

[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* [21]. 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 [22]. 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[22]. 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 [15]. 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* [10, 11]. 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 [23] [5].

Another attack, which is closely related to the former, is *empty account in the state trie* [10, 11]. 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 [12, 13, 23] [6].

The last example of a DoS attack is called *Proposer DoS* [14, 15]. 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 [14, 20].

Specifically, a proposal has been made to use an election protocol called Whisk, which is a type of SSLE [24][7][8][9]. 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 [10].

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 [22] [11].

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 [15, 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.

5. EIPs/EIPS/eip-150.md at master ethereum/EIPs GitHub
6. EIPs/EIPS/eip-161.md at master ethereum/EIPs GitHub
7. Shuffle_SSLE/rust_code/src at master ethresearch/Shuffle_SSLE GitHub
8. [WIP] Introduce consensus code for Whisk (SSLE) by asn-d6 Pull Request #2800 ethereum/consensus-specs GitHub
9. GitHub - dapplion/lighthouse at whisk
10. 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
11. GitHub - tse-group/gasper-gossip-attack

Another finality attack is called *33% finality attack* [15]. 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] [12]. 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 [13]. 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] [14].

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.

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] [15].

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] [16]. Also, Ethereum released a patch fixing this attack after the author pointed it out [17].

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

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

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

15. 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."

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

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