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

TODO

2 Background

In this section, we will go through some of the concepts that will be used in the rest of the paper as well as some surrounding context 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 Proof of Work (PoW) consensus algorithm, Ethereum transitioned to a Proof of Stake (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 that is used to secure blockchain net by helping to create new blocks and confirm transactions. It works by creating validators based on the amount of cryptocurrency they have staked. Then it selects some of these validators as proposers to be the ones to create new blocks, which then are confirmed by the rest of the validators. 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 way the proposer is chosen is through a random process that is weighted by the amount of cryptocurrency the validator has staked, and by using the publicly available random number generator (RANDAO) algorithm to simulate the random selection.

A block needs to be proposed by the designated proposer for the slot and confirmed by other validators within a time limit of the slot, which is 12 seconds. 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.

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

2.2 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 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. The peers a node can reach within the same subnet is called its fanout [2].

2.3 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 [3]. One client can run several validators. 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. How to determine who is going to be a block proposer is explained in subsection 2.4. Along with this, the validator is also responsible for attesting blocks proposed by other validators, ensuring the liveness of the chain.

suggetstion "a picture where all the main actors are visible and connected"

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.

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

2.3.1 Validator rewards

Validators are rewarded for several different actions [4]. 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
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 [4, 5]. 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].

2.3.2 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 [5].

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} [5].

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 [4]:

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.

2.4 RANDAO

As mentioned in subsection 2.3, 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 [6]. More specifically, the selection for epoch $n + 2$ happens at the end of epoch n [7]. This means that a validator knows at least one epoch, at most two epochs, in advance if they are proposing a block.

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

3. RANDAO - GitHub

2.5 ENR

A Ethernet node record (ENR) is a record that contains information about a node in the network [8]. 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.

new titel

make acronym for the paper

2.6 other paper

In the paper "Deanonymizing Ethereum Validators: The P2P Network Has a Privacy Issue" the authors show that it is possible to deanonymize validators on the Ethereum network by observing attestations and subscribing to 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. This paper 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 setup they use a custom version of a Prysm node called RAINBOW that subscribe to all subnets, and they use to log and color the information gathered. 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 we interact with. To help speed up the discovery of the peers they also used a crawler to more quickly find the peers using the discovery protocol and the peer tables. In their execution of their experiment they set up four nodes spread out across four different geographical locations. 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 4 categories based on their heuristic. Those being deanonymized where they located validators on the machine with the heuristic conditions being upheld, No validators where they did not receive a single non-backbone attestation received from the peer, and so they assume that there are no validators on the peer, 64 subnets where they never receive a non-backbone attestation from the peer which makes it impossible to deanonymize the validators and the rest where they got at least one non-backbone attestation but where not able to locate any validators on the peer.

2.7 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 Denial-of-Service (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 [9, 10].

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 Zero-Knowledge Proof (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 pro-

poser's IP addresses and overload their connection. A successful attack would leave a proposer unable to propose their block in time.

Should possibly be explained in more detail

3 Related Work

The usage of 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 [11, 12]. Another way to slow down the network is to create empty accounts that are hard to process [13, 14]. 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 describe the setup of the experiment that will be conducted as well as the experiment itself. The section will also cover the workings of the attack that will ve performed.

4.1 Setup

The setup of the experiment is as follows: First we modified a prysm client to include logging of attestations received by the client and all the peer that the node has discovered. The client was then run on a virtual machine hosted on Strato Claaudia through the Aalborg University. 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 insted of the mainnet. While the client was running, it was able to log attestations received and peers discovered gaining their IPs, the subnets that the attestations were sent to, the subnet that the peer was in, the public key associated with the attestation and more. The data that was logged was then stored in a database for further analysis sorted into categories depending on if the attestation was sent to a subnet that the peer already was subscribed to or not, if the peer is subscribed to every subnet and if the peer never sent an attestation. When it comes to deanonymizing the validators, we used the same heuristic as in the paper by [2]. The heuristic includes four criteria being:

mabye add image of tableheaders here for vizualization

Paper name maybye

- The proportion of non-backbone attestations for validator v exceed
$$0.9 * \left(\frac{64 - n_{sub}(p)}{64} \right) \quad (5)$$
where nsub(p) is the average number of subnets the peer is subscribed over the connection's duration.
- The peer is not subscribed to all 64 subnets.
- TWe receive at least every tenth attestation we expect for the validator v from the peer.
- 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.

is it fine to just copy the criteria from the paper word for word?

4.2 Attack implementation

As mentioned in subsection 4.1, 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 subsection 4.1 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.

How much are we allowed to say? We assume mentioning which classes/functions to modify is unethical

4.2.1 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 Peer-to-Peer (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 subsection 4.1.

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

4.2.2 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 subsubsection 4.2.1, 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 subsubsection 4.2.1.

4.2.3 Data processing

4.2.4 Exploiting RANDAO for a DoS attack

Where should this section be?

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 subsection 2.7.

In subsection 2.4, 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 any financial risks.

The output of the call to the API returns 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.

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 only running a node in the network

The first reason which only requires de-anonymization, could be to exploit the inactivity penalization, which is mentioned in subsubsection 2.3.2. 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 subsubsection 2.3.1, 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 required only a node in Ethereum, but the adversary themselves did not gain any reward from the attack. This could be a possibility if the adversary is in control of a validator .

Note to self: Write about incentive from MEV (include found article)

5 Discussion

This is a potential discussion section.

6 Conclusion

This is a potential conclusion section.

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

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

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

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

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

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

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

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

[11] 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.

[12] 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.

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

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

[15] 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].

[16] 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.

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

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

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

[20] 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.

[21] 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.

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

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

[24] 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* [15]. 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 [16]. 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[16]. 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 [10]. 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* [11, 12]. 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 [17] [5].

5. EIPs/EIPS/eip-150.md at master ethereum/EIPs GitHub

Another attack, which is closely related to the former, is *empty account in the state trie* [11, 12]. 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 [13, 14, 17] [6].

The last example of a DoS attack is called *Proposer DoS* [9, 10]. 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 Single Secret Leader Election (SSLE) which ensures that only the selected validator knows that they have been selected [9, 18].

Specifically, a proposal has been made to use an election protocol called Whisk, which is a type of SSLE [19][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* [20]. 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 [16] [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.

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 asnd6 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

The first attack is a *double finality* attack [10, 21]. 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 slashable offense in the Ethereum chain, so the adversary would get their staked ETH slashed.

Another finality attack is called *33% finality attack* [10]. 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* [20] [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 [22]. 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) [22, 23] [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.

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)

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 [24] [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 [24].

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 [24]. 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 [24] [16]. Also, Ethereum released a patch fixing this attack after the author pointed it out [17].

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