Max Attestation Matters: Making Honest Parties Lose Their Incentives in Ethereum PoS

Mingfei Zhang
Shandong University
mingfei.zh@outlook.com

Rujia Li ✉
Tsinghua University
rujia@tsinghua.edu.cn

Sisi Duan ✉
Tsinghua University
duansisi@tsinghua.edu.cn [1]

Abstract

We present staircase attack, the first attack on the incentive mechanism of the Proof-of-Stake (PoS) protocol used in the Ethereum 2.0 beacon chain. Our attack targets the penalty of the incentive mechanism that penalizes inactive participation. Our attack can make honest validators suffer from penalties, even if they strictly follow the specifications of the protocol. We show both theoretically and experimentally that if the adversary controls 29.6% stake in a moderate-size system, the attack can be launched continuously, so eventually *all* honest validators will lose their incentives. In contrast, the adversarial validators can still receive incentives, and the stake owned by the adversary can eventually exceed the $1/3$ threshold (system assumption), posing a threat to the security properties of the system.

In practice, the attack feasibility is directly related to two parameters: the number of validators and the parameter MAX_ATTESTATIONS, the maximum number of attestations (i.e., votes) that can be included in each block. We further modify our attack such that, with the current system setup (900,000 validators and MAX_ATTESTATIONS =128), our attack can be launched continuously with a probability of 80.25%. As a result, the incentives any honest validator receives are only 28.9% of its fair share.

1 Introduction

Ethereum [50], one of the most popular blockchain systems, upgraded to 2.0 in Sep 2022. The system now uses a Proof-of-Stake (PoS) protocol called Gasper as its core consensus scheme [12], a Byzantine fault-tolerant (BFT) protocol that tolerates Byzantine failures (i.e., arbitrary failures). Different from conventional BFT protocols [13, 16–18, 24, 25, 47, 53, 54] that assume the adversary does not control over one-third of nodes (also called validators) or Nakamoto consensus (the consensus protocol of Bitcoin and Ethereum 1.0) that assumes the adversary does not control over 50% computational power, PoS assumes that the adversary does not control more than one-third of the total stake. Here, the stake in general refers to the account balance of the validators.

The PoS protocol of Ethereum assumes a partially synchronous network [19], where there exists an unknown upper bound for message processing and transmission. The protocol is a combination of Casper friendly finality gadget (FFG) [11] and a variant of the GHOST fork-choice rule [44] called Hybrid Latest Message Driven Greedy Heaviest-Observed Sub-Tree (HLMD GHOST). The protocol is epoch-based, and there are many slots in each epoch, divided by physical clocks. In each epoch, HLMD GHOST selects the *canonical chain* based on the received block proposals. Informally speaking, an honest block proposer will extend the canonical chain when creating a new block, and an honest validator only votes for blocks on the canonical chain (so the system is somewhat *live*). Additionally, Casper is a gadget that essentially counts the number of votes (also called attestations) so eventually some blocks are finalized and honest validators will finalize the same chain (and the system is *safe*).

Almost all PoS protocols [12, 23, 27] make an implicit assumption similar to conventional BFT: all honest validators are always online, and the system cannot support an unknown number of validators that may go to sleep [36]. This is in sharp contrast to the Nakamoto consensus. While some academic efforts have been made to study PoS in the sleepy model [5, 31], Ethereum utilizes the *incentive* mechanism to encourage validators to stay online. The incentive mechanism for attestations consists of *rewards* and *penalties*. In particular, validators whose attestations are finalized on-chain will receive rewards, and validators whose attestations are not finalized on-chain for a sufficiently long period of time will suffer from penalties (see §3.4). The incentive mechanism has been very successful in practice. According to the report provided by *rated.network*, the participation rate of Ethereum has

[1] Sisi is also with Zhongguancun Laboratory, Shandong Institute of Blockchains, and Beijing National Research Center for Information Science and Technology

reached 99.6%[1]. Indeed, validators are incentivized to make the system both safe and live so they can continue gaining rewards from the blockchain.

An attack on the attestation incentive mechanism. For the first time, we present an attack on the incentive mechanism of Ethereum PoS, and we call it *staircase attack* (mainly because the branches the adversary constructs in the attack look like a staircase). Our work focuses on the rewards and penalties for attestations only, so we call them *attestation incentives* in this paper. Our attack can be launched even if the network is synchronous, i.e., there exists a known upper bound for message transmission and processing. The goal of our attack is to force honest validators to be penalized, even if they strictly follow the specifications of the protocol. We begin with a warm-up attack where a single Byzantine validator, upon some *opportune epoch*, is able to make some honest validators suffer from penalties without any cost. We then extend the attack to a scenario where Byzantine validators controlling 29.6% of the total stake may collude. After an opportune epoch, the Byzantine validators can make half of the honest validators suffer from penalties in *every* epoch. Eventually *all* honest validators will lose their incentives. Meanwhile, none of the Byzantine validators suffer from any penalties at all. The consequence of our attack is thus the same as *discouragement attack* [10]: the fraction of the stake controlled by the adversary may continue to increase, posing safety threats. However, [10] only briefly mentions the concept and does not provide the attack strategies. Therefore, we consider our attack the first concrete instantiation of a discouragement attack.

Our attack utilizes a parameter used in both HLMD GHOST and Casper called the *last justified checkpoint LJ*. In Ethereum PoS, Casper updates the *LJ* parameter, and HLMD GHOST determines the canonical chain based on the *LJ* parameter. The design of Ethereum PoS identifies the *LJ* parameter as a *frozen* parameter that is not supposed to change within an epoch. However, as Byzantine validators may withhold their blocks (and attestations) and release them at any time, an honest validator might update *LJ* in the middle of an epoch. We show that by deliberately packing the attestations from *all* Byzantine validators into one block and withholding such a block, the honest validators always change their *LJ* in the middle of an epoch. As a result, the canonical chain (output by HLMD GHOST) may switch from an old branch to a new one. Thus, attestations from honest validators included in the old branch will be discarded and the corresponding honest validators will be penalized.

Evaluation of the attack. We implement our attack using an Ethereum implementation *Prysm (Capella version)* and conduct experiments using 1,000 validators. Our experimental results match our theoretical analysis: if the adversary controls 29.6% validators, *all* honest validators lose their incentives. As the fraction of stake controlled by the adversary grows, honest validators may suffer from stake loss. For instance, if the adversary controls 33.3% stake, all honest validators are expected to suffer from a 20% stake loss compared to their fair share.

[1] Data source (accessed in Feb 2024): https://www.rated.network/

Attack feasibility and insights. Ethereum has over 900,000 validators as of Feb 2024. When demonstrating the feasibility of our attack in such a large-scale system, we find a somewhat surprising result. Specifically, the feasibility of our attack is related to two parameters: the number of validators and the number of attestations each block can carry, i.e., the MAX_ATTESTATIONS parameter. Based on the system setup of Ethereum, if we fix the MAX_ATTESTATIONS parameter, our attack can be launched for a system with fewer than 16,384 validators. Alternatively, for a system with 900,000 validators, the attack can be launched if the MAX_ATTESTATIONS parameter is increased from 128 to 2,048.

We further modify our attack to *accommodate* the system parameters of today's Ethereum system. With the modification, our attack can be continued in each epoch with a probability of 80.25%, given that the adversary controls 33% of the total stake. As a result, the attestation incentives of an honest validator become only 28.9% of the fair share. As reported by *rated.network*, the largest mining pool today already controls 32.66% of the total stake. Therefore, our attack can cause discouragement to honest validators.

It is also worth mentioning that our warm-up attack is additionally affected by the *honest reorg* mechanism [46]. To successfully launch the warm-up attack, the adversary needs to carefully release a block *late* enough and the timing is highly related to the network condition.

Responsible disclosure. For ethical reasons, we disclosed our findings to the Ethereum Foundation in April 2023, and the development team has taken measures since then to mitigate the attack. With the mitigation, the probability of continuing the attack in each epoch is significantly reduced. The mitigation is already effective after the Deneb upgrade in March 2024. The communication and the mitigation can be found at https://github.com/ethereum/consensus-specs/pull/3339#issuecomment-1637117341.

Our attack and analysis show some interesting insights that may lead to future research directions. First, the incentive mechanism is usually considered an economic factor in a system and has never been considered in the design of Byzantine-fault tolerant consensus protocols [12]. However, the attacks on the incentive mechanism may make the PoS protocol violate the security goals. Indeed, the stake of the adversary may exceed one-third of the total stake so eventually the safety and liveness of the system might be violated. Therefore, it is interesting to learn whether the security properties regarding the incentive mechanism can be considered in the design of the consensus protocols. Second, the number of attestations that can be included in each block proposal is closely related to both the incentives of validators and the security goals of the system. Such a counter-intuitive finding shows that the value of the MAX_ATTESTATIONS parameter (and its closely related parameters) should be set up carefully.

Our contributions. We make the following contributions.

- We propose staircase attack on the incentive mechanism of Ethereum's PoS protocol. We begin with an attack without considering the number of validators and the value of the MAX_ATTESTATIONS parameter. We show that when the adversary controls at least 29.6% of the total stake, the attack can be launched continuously and eventually make *all* honest validators lose their incentives.
- We implement the attack using 1,000 validators. By conducting each experiment for one day (225 epochs), we show that an adversary controlling 29.6% stake makes *all* honest validators lose their incentives; if an adversary controls 33% stake, all honest validators are expected to suffer from a 20% of stake loss compared to their fair share.
- Our feasibility analysis shows that the number of validators and the MAX_ATTESTATIONS parameter have a direct impact on whether the attack can be continuously launched. Based on today's system setup of Ethereum, our attack can be launched under the following two conditions: the system has no more than 16,384 validators; the MAX_ATTESTATIONS parameter is increased from 128 to 2,048. We further modify our attack such that with a probability of 80.25%, the attack can be continued in the next epoch and the expected attestation incentives of honest validators become only 28.9% of their fair share.

2 Related Work

Identified attacks against Ethereum PoS only. Many efforts have been made to analyze Ethereum PoS since Ethereum announced its plans to upgrade to 2.0, as summarized in Table 1.

Balancing attack. A balancing attack aims to split honest validators into two parts, forcing them to vote for two conflicting branches with the same weight. Consequently, neither chain can be finalized and the system suffers from liveness issues [31, 43]. Ethereum fixed the balancing attack [9] using a *proposal boosting* mechanism [40]. In short, proposal boosting is a "temporary" weight assigned to the block in the current slot. As the two branches do not have the same weight anymore, the situation created in the balancing attack will not last forever. Later, it was shown that the balancing attack can be revised to bypass the proposal boosting mechanism [32]. In the balancing attack, the cost of the adversary is that at least one Byzantine validator will suffer from the slashing condition. According to the specification of Ethereum, slashed validators will suffer from stake loss and eventually be removed from the system.

Bouncing Attack. In a bouncing attack, checkpoints from two branches are justified one after the other. Therefore, the canonical chain jumps from one chain to another and neither branch can be finalized, causing a liveness issue [30, 37]. Ethereum's upgrade in March 2023 fixed all known bouncing attacks [22]. Our attacks are designed based on the latest specifications.

Reorg Attack. In reorg attacks [34, 42, 43], the adversary reorganizes the chain to increase the fraction of the blocks from Byzantine validators on the canonical chain. The goal is to gain more profit [26,51,56], decrease the *chain quality* [35], or lower the performance of the system. There are two types of reorg attacks: short reorg attacks (SRA) and long reorg attacks (LRA) [42]. In SRA, the attacker makes the adversarial branch heavier than the honest branch so the honest branch becomes orphaned [34, 43]. An interesting fact is that the SRA in [34, 43] are fixed by the proposal boosting mechanism [9]. It was later mentioned informally that the proposal boosting mechanism does not fully address the issue and a revised SRA called *sandwich attack* is proposed [15].

Meanwhile, in LRA, the attacker forces the honest branch to be pruned and the number of blocks affected is usually larger than that in SRA [2, 38, 41]. Two types of LRA have been found: unrealized justification attacks [2] and justification withholding attacks [38, 41]. In unrealized justification attacks, the validator in the first slot of an epoch needs to be Byzantine. The Byzantine validator creates a fork by proposing a block that extends an older block so the canonical chain is re-organized. In justification withholding attacks, the validators in the last few slots of an epoch need to be Byzantine and the Byzantine validators withhold their blocks. In the middle of the subsequent epoch, the withheld blocks are released so the *LJ* of honest validators is updated in the middle of an epoch and the honest branch is pruned. A patch is provided recently such that branches with *enough* attestations are not pruned to prevent all LRA known so far [2, 39].

In comparison, our staircase attack can be viewed as a long reorg attack, specifically a justification withholding attack. Different from the justification withholding attack that requires validators in consecutive slots to be Byzantine, our attack only requires the last Byzantine validator in an epoch to withhold the attestations and the block. Therefore, our attack is much easier to launch. Besides, the patch for the LRA does not affect our attack.

Identified attacks against PoS in general. We summarize some identified attacks against PoS in general, some of which are applicable to Ethereum PoS as well.

Nothing-at-stake attack. A nothing-at-stake attack refers to the attack where an adversary is willing to contribute to multiple forks at no cost (as the validators in PoS do not have to compete to propose or vote) [8, 27, 28]. The goal of nothing-at-stake attacks is usually the liveness and the performance of the system. The *avalanche attack* is one example of PoS GHOST [32]. In the attack, it was shown that by deliberately

attack type	scheme	timing assumption	target	slashed*	experimentally confirmed	issue fixed
balancing attacks	Neu, Tas, and Tse. [31]	synchrony	liveness	✗	✗	✓
	Schwarz-Schilling et al. [43]			✗	✗	✓
	Neu, Tas, and Tse. [32]			✓	✗	✗
bouncing attacks	Nakamura [30]	partially synchrony	liveness	✗	✗	✓
	Pavloff et al. [37]				✗	✓
short reorg attacks	Neuder et al. [34]	synchrony	chain quality	✗	✗	✓
	Schwarz-Schilling et al. [43]				✗	✓
	D'Amato et al. [15]				✗	✓
long reorg attacks	Asgaonkar [2]	synchrony	chain quality	✗	✗	✓
	Ryan [41]				✗	✓
	Potuz [38]				✗	✓
staircase attack	Our work	synchrony	incentive	✗	✓	✗⋆

Table 1: Comparison of known attacks against Ethereum PoS. **Slashed* denotes whether at least one Byzantine validator will be slashed. According to the specification of Ethereum, slashed validators will suffer from stake loss and eventually be removed from the system. ⋆The issue has been reported to the community.

working on multiple forks at the same time, the canonical chain consists of blocks only from the adversary, so the system is not live anymore. Note that Ethereum does not suffer from nothing-at-stake attack, as a validator that equivocates (e.g., it votes for two conflicting blocks) will be slashed.

Long-range attack. In a long-range attack, the adversary first acquires the secret keys of some validators after they withdraw their stake (and leave the system). The adversary then revises the blocks proposed previously and rewrites the history of the blockchain. Many protocols are designed to prevent the attack [3, 4, 27, 49, 52]. The Casper protocol [11] used in Ethereum is a solution to the long-range attack.

Grinding attack. Most PoS protocols use pseudorandom functions (e.g., verifiable random functions (VRF) [29]) to select block proposers. The way the randomness is generated usually depends on the blocks in the canonical chain [48]. In practice, the roles of the nodes might be learned in advance [6]. Accordingly, a validator eligible to propose a block can manipulate the value of the randomness (and the validators eligible to propose subsequently) [1, 8]. The goal is to improve the adversary's chances of being selected as a block proposer (to gain extra profits or decrease the chain quality).

Selfish mining. Selfish mining is an attack first known in Proof-of-Work (PoW) [20]. In selfish mining, a mining pool (with a number of validators) may collude [55] and gain more revenue than the fair share. Selfish mining attack was later found to be feasible in some PoS protocols as well [6, 8, 21, 33]. To the best of our knowledge, selfish mining attack has not been identified in Ethereum 2.0 yet.

3 Review of Ethereum Proof-of-Stake

In this section, we review the Proof-of-Stake (PoS) protocol used by Ethereum 2.0 [12, 37]. Our notations largely follow the specifications by Ethereum foundation [45].

3.1 System Model

Nodes that participate in the PoS protocol are called *validators*. To become a validator, one must deposit at least 32 ETH as an initial stake in its account. Each validator's voting power is weighted by its stake. There are N validators $\{v_1, v_2, \cdots, v_N\}$, where N may change over time as validators join and leave the system. Each validator holds a private/public key pair. Validators may be Byzantine and arbitrarily deviate from the protocol. Non-faulty validators are called honest validators. Ethereum PoS assumes that the stake controlled by Byzantine validators is less than one-third of the total stake.

To facilitate the description of our attack and without loss of generality, we assume that N does not change and each validator is assumed to hold at least 32 ETH. In this way, we normalize each validator's balance to *1 unit* [12]. This enables us to tally the number of attestations instead of taking into consideration the fractions of the validators' balance. Accordingly, let f be the number of Byzantine validators, we assume $f < N/3$. We believe this assumption is reasonable as according to the data from *beaconcha.in*, the average actual balance of the validators is 32.08 ETH[2].

Ethereum PoS assumes that the network is partially synchronous [19], i.e., there exists an unknown upper bound Δ for message propagation and processing delay. However, our

[2]Data source (accessed in Feb 2024): https://www.beaconcha.in/charts/average_balance

attack can be launched even if the network is synchronous, i.e., the value of Δ is known to every validator.

3.2 Terminology and Notation

Ethereum PoS proceeds in epochs, and each epoch consists of 32 slots. Given a slot number t, every validator is able to obtain the epoch number $e \leftarrow \lfloor \frac{t}{32} \rfloor$. Each slot lasts for 12 seconds in the current production system. Within a slot, only a single block can be proposed. A *block b* consists of the slot number, a hash pointer to the parent block, a batch of transactions, and a set of attestations (i.e., votes, to be described shortly). Given a block b, the *branch* led by b is the path from b to the *genesis block* (the first block of the blockchain history). Each validator maintains a *block tree* $\mathcal{T}$ about the blocks proposed by the validators. Additionally, a *checkpoint* is a special block b. By default, the checkpoint is the first block proposed in each epoch. If such a block does not exist, the most recent preceding block becomes the checkpoint. A checkpoint is denoted as a pair (b,e), where b is the block and e is an epoch number. For simplicity, we use the block instead. We use $\mathsf{ep}(b) \leftarrow \lfloor \frac{t}{32} \rfloor$ to denote the epoch number of block b proposed in slot t.

Ethereum PoS consists of Hybrid Latest Message Driven Greedy Heaviest-Observed Sub-Tree (HLMD GHOST) and Casper FFG [12]. HLMD GHOST is a fork-choice rule that recursively chooses the root of the *heaviest subtree* and outputs the leaf block as *head*. The chain led by head is also called the *canonical chain*. Each honest validator will only vote for the head of the canonical chain it is aware of or create a new block by extending the canonical chain. Additionally, Casper FFG helps finalize the checkpoints, and once a checkpoint is finalized, all the blocks on the chain led by the checkpoint are finalized. As attestations are contained in blocks, we can also say that the attestations are finalized on-chain.

Attestor and proposer. In each epoch, N validators are randomly and evenly divided into 32 validator sets (determined by the RANDAO protocol[3]). Each validator set is allocated for one slot of the epoch and validators (also called *attestors*) in this set are allowed to vote. Additionally, in each slot, one validator is randomly selected as the *proposer*, also according to RANDAO. The proposer is the only validator that is allowed to propose a block in the slot. Our work simply assumes RANDAO is a pseudorandom function that randomly determines the roles of the validators, i.e., the expected number of honest attestors in each slot is $(N-f)/32$. Every validator is able to determine its role one epoch in advance and also verify the roles of other validators. A message from validator v_i is considered invalid if v_i's role is not verified.

Attestation. In Ethereum, a vote is also called an *attestation*. An attestation (*att*) by validator v_i consists of a slot number, two checkpoints (*source* and *target*), and the hash of a block b. We say *att* is an attestation for block b. By default, *source* is v_i's *last justified checkpoint* (i.e., *LJ*, to be described shortly). The *target* field is the checkpoint block of the current epoch in v_i's canonical chain. Both *source* and *target* are needed to identify the chain each attestor votes for. In an attestation, the *source* or the *target* field is denoted as the hash of a checkpoint block b and the epoch number. In this paper, we omit the epoch number for ease of understanding. Notably, the block b is the output of the HLMD GHOST.

Justification and finalization. The justification and finalization rules are defined in Casper for checkpoints only. Specifically, if attestations from more than two-thirds validators with the same *source* and *target* are received, the *target* checkpoint is *justified*. Additionally, if the descendant checkpoint of a checkpoint cp is justified, cp becomes *finalized*. If cp is finalized, all the blocks on the chain led by cp are finalized and the order of the finalized blocks will never be reversed.

Given a branch c in v_i's block tree $\mathcal{T}$, we use $V(c)$, $J(c)$, and $C(c)$ to denote the number of attestations included in chain c, the last justified checkpoint based on $V(c)$, and checkpoint block of the current epoch in c, respectively. Given a block b and a branch c led by b's parent block, if the slot numbers of b and b's parent block indicate that b is from a new epoch, *LJ* is updated to $J(c)$. Ideally, *LJ* is updated at the beginning of an epoch and is not changed during the epoch. In contrast, $J(c)$ is updated whenever $V(c)$ is updated.

Fork choice rule. The fork choice rule HLMD GHOST defines the canonical chain and HLMD GHOST outputs the head of the canonical chain. In particular, the block each attestor votes for is the head of its canonical chain. Meanwhile, a block proposer will extend the head of the canonical chain when creating a new block b, i.e., by setting the hash pointer of b as the hash of the head. Given a block tree $\mathcal{T}$, the canonical chain is defined as follows (an example is provided in Figure. 1):

1. Prune any branch $c \in \mathcal{T}$ such that $J(c)$ is lower than *LJ*. Then calculate the *weight* of each block in the chain. Here, weight is determined by the attestations. In particular, if there exists an attestation for block b, the weight of b is incremented by one.
2. Recursively calculate the sum of weights of each sub-branch. The branch with the largest accumulative weight is considered *the heaviest subtree* and becomes the canonical chain.

3.3 Workflow of Ethereum PoS

We summarize the workflow of Ethereum PoS in Figure. 2. Each validator maintains three local parameters: the last justified checkpoint *LJ*, a set of received attestations *Atts*, and the block tree $\mathcal{T}$.

In slot t, if v_i is the proposer, it broadcasts a

[3]RANDAO: https://github.com/randao/randao

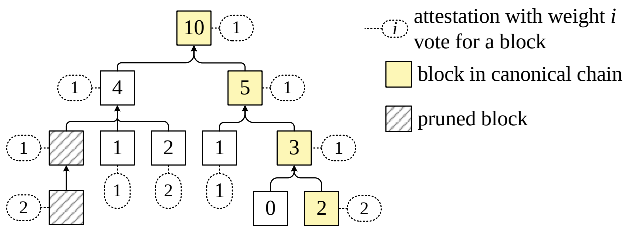


Figure 1: Example of the fork choice rule HLMD GHOST. Attestations are represented in dashed rounded rectangles and blocks are represented in solid rectangles. The number shown in each attestation denotes the weight of the attestation and the number shown in each block denotes the weight of the subtree. The leftmost branch c is pruned (because $J(c)$ is lower than *LJ*), and its weight is not calculated. The rightmost branch is the canonical chain.

$(\text{PROPOSE}, t, v_i, H(head), newatts, txs)$ message to all validators (lines 1-10). Here, *head* is the output of HLMD GHOST and $H(head)$ is the hash of *head*, serving as a hash pointer of the parent block. The *newatts* field consists of a set of attestations v_i has received. Given the canonical chain c, any attestation that satisfies the following two requirements will be included in *newatts*: 1) the attestation has not been included in c so far; 2) the *source* is the same as *LJ* and the *target* is the same as $C(c)$. If v_i is an attestor, it waits until 1/3 time of slot t has elapsed (i.e., four seconds) and updates its *LJ*. Then, v_i prepares an attestation message $(\text{ATTEST}, t, v_i, H(head), LJ, C(c))$ and sends it to all validators, where *LJ* is the *source*, and the checkpoint of current epoch $C(c)$ is the *target* (lines 11-19).

Upon receiving a block b from the proposer v_j of slot t', v_i checks: 1) whether the epoch number of b is higher than the epoch number of b's parent block; 2) whether the epoch number of $J(c)$ is higher than *LJ*, where c is the branch led by b (line 24). If so, v_i updates *LJ* to $J(c)$. Validator v_i then processes the attestations included in b and updates its block tree $\mathcal{T}$, i.e., for each branch $c \in \mathcal{T}$, update $J(c)$, $V(c)$, and $C(c)$ (lines 27-31). Last, if b is a block from the prior epoch and $\mathsf{ep}(J(c)) > \mathsf{ep}(LJ)$, v_i also updates its *LJ* (lines 33&34).

It is worth mentioning that a validator v_i updates *LJ* in three cases. First, v_i receives the first block in some epoch e such that $J(c)$ is higher than its *LJ* (lines 23-25 in Figure. 2). Second, before v_i prepares an attestation, it updates its *LJ* according to the messages it receives (lines 14-16 in Figure. 2). Last, although v_i already enters epoch e, it has received a sufficiently large number of attestations for an epoch lower than e (e.g., included in the blocks withheld in previous epoch(s) and released in epoch e[4]) such that $J(c)$ is higher than its *LJ* (lines 32-34 in Figure. 2). Before epoch e begins, the epoch number of *LJ* of any honest validator is at most $e-2$.

[4]Note that according to the design of the system, a block tagged with epoch e' where $e' < e$ can still be accepted by validator v_i even if v_i already enters epoch e.

The Ethereum Protocol for Validator v_i

global parameters: slot t
local parameters: block tree $\mathcal{T}$, attestation set *Atts*, last justified checkpoint *LJ*.

---

01 **upon** slot t **do**
02 **as** the proposer for slot t
03 let *head* be the output of HLMD GHOST
04 let c be canonical chain
05 **for** each attestation *att* in *Atts*
06 **if** *att* is not included in c **and** *source* of *att* is *LJ* **and** *target* of *att* is $C(c)$ **then**
07 $newatts \leftarrow newatts \cup \{att\}$
08 obtain a batch of transactions *txs* from mempool
09 create block $b = (\text{PROPOSE}, t, v_i, H(head), newatts, txs)$
10 send b to all validators
11 **as** the attestor for slot t
12 wait until 1/3 time of slot t has elapsed
13 let *head* be the output of HLMD GHOST
14 ▷ Update *LJ*
15 **if** $\mathsf{ep}(t) > \mathsf{ep}(head)$ **and** $\mathsf{ep}(J(c)) > \mathsf{ep}(LJ)$ **then**
16 $LJ \leftarrow J(c)$
17 let c be canonical chain
18 create attestation $att = (\text{ATTEST}, t, v_i, H(head), LJ, C(c))$
19 send *att* to all validators

---

20 **upon** receiving $b = (\text{PROPOSE}, t', v_j, H(head), newatts, txs)$ from the proposer v_j of slot t' **do**
21 $\mathcal{T} \leftarrow \mathcal{T} \cup \{b\}$
22 let c be the branch led by b
23 ▷ Update *LJ*
24 **if** $\mathsf{ep}(b) > \mathsf{ep}(head)$ **and** $\mathsf{ep}(J(c)) > \mathsf{ep}(LJ)$ **then**
25 $LJ \leftarrow J(c)$
26 ▷ Update $J(c)$
27 **for** each $att \in newatts$
28 let *tar* be the *target* in *att*
29 $V(c)[tar] \leftarrow V(c)[tar] + 1$
30 **if** $V(c)[tar] \geq 2N/3$ **and** $\mathsf{ep}(tar) > \mathsf{ep}(J(c))$ **then**
31 $J(c) \leftarrow tar$
32 ▷ Update *LJ* (exploited by our staircase attack)
33 **if** $\mathsf{ep}(b) < \lfloor \frac{t}{32} \rfloor$ **and** $\mathsf{ep}(J(c)) > \mathsf{ep}(LJ)$ **then**
34 $LJ \leftarrow J(c)$
35 **upon** receiving attestation *att* (with message type (ATTEST)) from validator v_j of slot t' **do**
36 $Atts \leftarrow Atts \cup \{att\}$

Figure 2: The Ethereum PoS Protocol. Codes are shown for validator v_i. $H()$ denotes the hash function.

3.4 The Attestation Incentive Mechanism

The incentive mechanism consists of *rewards* that encourage active participation and *penalties* that discourage inactive behavior [12,45]. Validators with attestations finalized on-chain will receive rewards, and validators who do not have their attestation finalized on-chain (after a sufficiently large number of slots) will get penalties. Note that penalties differ from the *slashing conditions* [12]: slashing conditions penalize behaviors such as equivocation, while penalties only act on inactive behavior. The slashing condition is also part of the incentive mechanism of the system. We thus use attestation incentives to denote rewards and penalties in this paper.

Generally speaking, the rewards and penalties of each validator depend on its own stake, the total amount of stake, and the corresponding attestation. For each validator v_i, a *base reward* I_{base} is first decided, which is an equation of v_i's stake and stake of all validators. Both rewards and penalties are then calculated based on I_{base}. The reward is $R \times W_r I_{base}$. Here, R is the *rewards scale with participation*, which is related to v_i's stake and the total stake of attestors who have their attestations finalized on-chain. W_r is the weight determined by the *source*, *target*, and block in the attestation. Additionally, the penalty is $W_p I_{base}$, where W_p is the weight determined by the *source* and *target* of the attestation v_i is supposed to send. If a validator submits an attestation in an epoch e and the attestation is finalized on-chain, the validator receives a $R \times W_r I_{base}$ reward. On the contrary, if the attestation is not finalized on-chain, the validator gets a penalty with amount $W_p I_{base}$ and no reward for epoch e.

4 An Attack on the Incentive Mechanism

In this section, we present staircase attack, an attack on the incentive mechanism of the Ethereum PoS protocol. The goal of this attack is to make honest validators suffer from penalties, even if the network is synchronous and honest validators strictly follow the specification of the PoS protocol.

We begin with a warm-up attack where a single Byzantine validator can manipulate the *target* of attestations from honest validators and make $(N-f)/32$ honest validators suffer from penalties. We then show that if Byzantine validators (owning 29.6% of the total stake) collude, the colluding validators can manipulate both the *source* and *target* of attestations from half of the honest validators. After the attack is started, it can be launched in *every* epoch. Eventually, *all* honest validators will lose their incentives. In contrast, Byzantine validators do not suffer from any penalties. The only cost is that the incentives Byzantine validators can obtain during the attack are slightly lower than their fair share.

We show in Figure. 3 the meaning of legends used in the figures of this paper.

4.1 Overview of the Attack Methodology

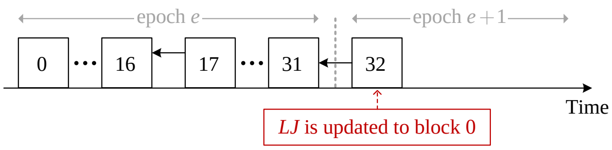
Our attack exploits the fact that *LJ* and the latest checkpoint might be changed in the middle of an epoch, as mentioned in §3.3. For ease of understanding, we illustrate three examples in Figure. 4 to show three possible cases. Before epoch e, *LJ* is set as *cp*, and attestations by all honest validators in epoch e have *target* set as block 0.

Legend	Meaning
	attestations from honest validators with *source* lower than the checkpoint block of the previous epoch
	attestations from honest validators with *source* as the checkpoint block of the previous epoch
	attestations from Byzantine validators with *source* as the checkpoint block of the previous epoch
	discarded attestations
	a block proposed by an honest validator that includes
	a block proposed by an honest validator that includes
	a block proposed by a Byzantine validator that includes
	a pruned block
t	a block that will be received at time t
	a withheld and released block
	attestations included in the block

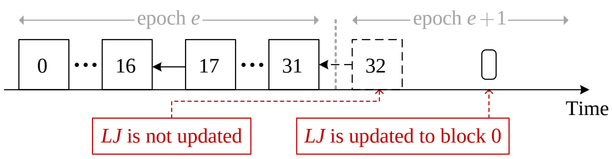
Figure 3: The meaning of legends used in the figures of this paper. Figures in this paper are best viewed in color.

- **Case 1: *LJ* is updated upon receiving the first block of an epoch.** As shown in Figure. 4a, in epoch e, v_1 receives block 0 to block 31. The attestations of all honest validators are included in the blocks. As attestations from all honest validators are received, the condition $V(c)[0] > 2N/3$ is satisfied and $J(c)$ is updated to block 0. In epoch $e+1$, v_1 receives block 32 in the first slot. As the epoch number of $J(c)$ is greater than *LJ*, *LJ* is set as block 0 (line 24 in Figure. 2). Honest attestors all use block 0 as *source* and block 32 as *target* in their attestations i n epoch e.
- **Case 2: *LJ* is updated before preparing an attestation.** As shown in Figure. 4b, v_2 receives block 0 to block 31, but no blocks have been received in epoch $e+1$. *LJ* is not updated to block 0 until v_2 becomes an attestor (line 14 in Figure. 2). Before v_2 prepares its attestation, it updates its *source* as block 0 and *target* as block 31.
- **Case 3: *LJ* is updated in the middle of an epoch.** As shown in Figure. 4c, validator v_3 has received block 0 to block 16 in epoch e. In epoch $e+1$, v_3 receives block 33, the parent of which is block 16. As blocks 17-31 are withheld or delayed, v_3 has not received $2N/3$ attestations and $J(c)$ is still *cp*, where c is the chain led by block 33. The attestation from v_3 has *cp* as *source* and block 16 as *target*. Later, in slot t of epoch $e+1$, v_3 receives blocks 17 to 31, blocks created in epoch e but received by v_3 in slot t of epoch $e+1$. Now, the blocks on the branch led by block 31 include enough attestations in epoch e. Therefore, $J(c)$ and *LJ* are updated to block 0 (line 33 in Figure. 2).

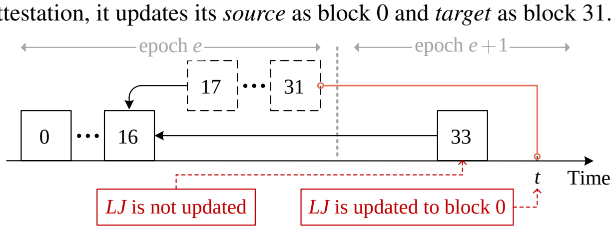
It is worth mentioning that after *LJ* is updated, any branch c such that $J(c)$ is lower than *LJ* will be pruned in the block tree, and HLMD GHOST will not output a block from a pruned branch. Take the case for validator v_3 as an example, we show the block tree of v_3 after v_3 receives blocks 17-31 in Figure. 5. In this example, before v_3 receives blocks 17-31, branch c_2 is



(a) v_1 receives block 32, the first block in epoch $e+1$. v_1 updates its LJ to block 0.



(b) v_2 has not received any blocks in epoch $e+1$ yet and does not update its LJ until it prepares an attestation. Before v_2 prepares its attestation, it updates its *source* as block 0 and *target* as block 31.



(c) v_3 receives block 33 and the chain led by block 33 does not include $2N/3$ attestations. v_3 does not update its LJ. When v_3 later receives blocks 17-31, the chain led by block 31 includes $2N/3$ attestations with *target* as block 0. v_3 updates LJ to block 0.

Figure 4: Examples of updating LJ.

the canonical chain. After LJ is updated to block 0, branch c_1 led by block 31 becomes the canonical chain. Branch c_2 led by block 33 will then be pruned as $J(c_2) = cp$, lower than LJ (block 0).

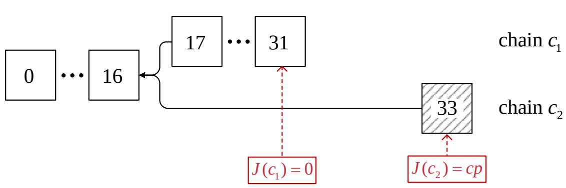


Figure 5: Validator v_3's block tree.

Our attack utilizes this fact to *force* honest validators to prune a branch led by a block from an honest validator, so the attestations on the pruned branch are discarded, e.g., the attestations included in block 33 in Figure. 5 with cp as *source* and block 16 as *target* are discarded. Moreover, these attestations will never be finalized on-chain, as their *source* is lower than LJ (lines 5-7 in Figure. 2) and the corresponding validators will be penalized. In the following, we show that one Byzantine validator is able to make the attestations from $(N-f)/32$ honest validator be discarded. We then show that if all Byzantine validators collude, they can utilize the re-organization of the chain to make *half* honest validators suffer from penalties in every epoch.

4.2 The Warm-up Attack

We now present a warm-up attack, where a single adversarial validator v_i can launch the attack, and $(N-f)/32$ honest validators are expected to be penalized. To kick-start the attack, v_i waits for an *opportune epoch*. An epoch e is deemed opportune if v_i is eligible to propose in the first slot of epoch e (let the slot number be t).

Before epoch e, honest validators maintain a consistent view of the canonical chain, and LJ of all honest validators is the same checkpoint. According to the discussion in §3, we know that before epoch e begins, the epoch number of LJ is $e-2$, and we use cp_{e-2} to denote the LJ of all validators. We use cp_{e-1} to denote the checkpoint block proposed in the first slot of epoch $e-1$ and b to denote the last block proposed in epoch $e-1$. We assume that all validators receive cp_{e-1} and b in epoch $e-1$. Given such an opportune epoch e, the attack strategies of v_i are summarized below.

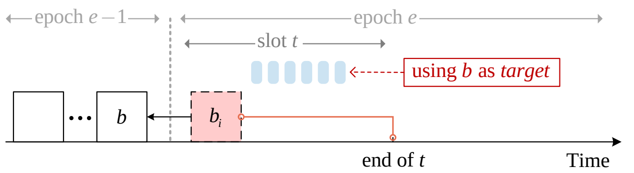
(1) (Figure. 6a) Validator v_i first creates a block b_i that extends block b and withholds b_i. As none of the validators has received the first block in epoch e yet, case 2 mentioned in §4.1 is satisfied: LJ is updated to cp_{e-1} when attestors in slot t prepare their attestations. Thus, attestors in slot t then send attestations using cp_{e-1} as *source* and b as *target*.

(2) (Figure. 6b) At the end of slot t, validator v_i sends b_i to all validators. Here, b_i becomes the head of the canonical chain, and all honest validators set their checkpoint block as b_i. Additionally, the chain c led by b consists of the attestations from all validators in epoch $e-1$, so $J(c)$ is already set as cp_{e-1}. As the first block of epoch e is received, case 1 in §4.1 is satisfied and all honest validators update their LJ to cp_{e-1} (lines 23-25 in Figure. 2).

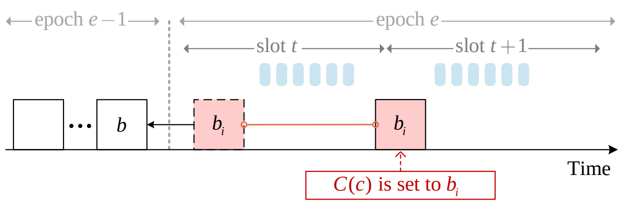
(3) (Figure. 6c) After slot t, any attestations created by honest attestors in slot t will be discarded by all honest validators. To see why, given any attestation *att* mentioned in step (1), the *target* field of *att* is b. However, the checkpoint of epoch e for all honest validators is b_i. When any proposer creates a new block, *att* will be filtered (lines 6&7 in Figure. 2). Therefore, all honest attestors in slot t will be penalized. According to our discussion in §3.2, the expected number of honest attestors penalized is thus $(N-f)/32$.

4.3 The Staircase Attack

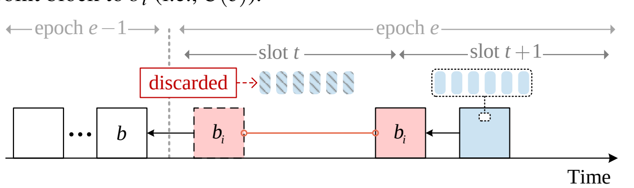
We are now ready to present our staircase attack, in which Byzantine validators collude to launch the attack. Here, we assume the adversary controls a set of Byzantine validators to launch the attack and later discuss the number of Byzantine validators the adversary needs to control. Recall that our warm-up attack manipulates the *target* of attestations from



(a) Step 1: v_i withholds block b_i. Honest attestors in slot t create attestation with b as *target*.



(b) Step 2: v_i releases b_i and all honest validators update their checkpoint block to b_i (i.e., $C(c)$).



(c) Step 3: Attestations with *target* different from b_i will be discarded and honest attestors in slot t will eventually be penalized.

Figure 6: Warm-up attack where v_i is the Byzantine validator. Before epoch e begins, all validators receive the last block b in epoch $e-1$.

honest validators. Our full staircase attack relies on the warm-up attack to kick-start the attack. After that, the staircase attack manipulates the *source* of honest validators instead. The idea is to let half of the honest validators use an outdated *LJ* as *source* and eventually be penalized. To do so, all Byzantine validators withhold their attestations and blocks, making *LJ non-frozen* and updated in the middle of an epoch. After that, the attestations released before *LJ* is updated are not included in the canonical chain, resulting in penalties for the corresponding honest validators.

Notations. We define several notations to assist the explanation of our attack. We divide each epoch e into two periods: the first period consists of slots before *LJ* is updated; the second period consists of the rest of the slots in epoch e. Honest attestors in the first period create attestations with some checkpoint cp as the *source*, while honest attestors in the second period create attestations with cp' as the *source*, where cp' is higher than cp. We partition attestations from all validators into three sets: A_1 denotes the attestations from honest attestors in the first period; A_2 denotes the attestations from honest attestors in the second period; A_3 denotes the attestations from all Byzantine attestors in epoch e. Our goal is to make the attestations in A_2 and A_3 share the same *source* and *target*, and the number of attestations in $A_2 \cup A_3$ is greater than $2N/3$, so the canonical chain can be *manipulated* by the adversary and attestations in A_1 will be discarded. As illustrated in Figure 3, we use green, blue, and red rounded rectangles to represent the attestations in A_1, A_2, and A_3, respectively.

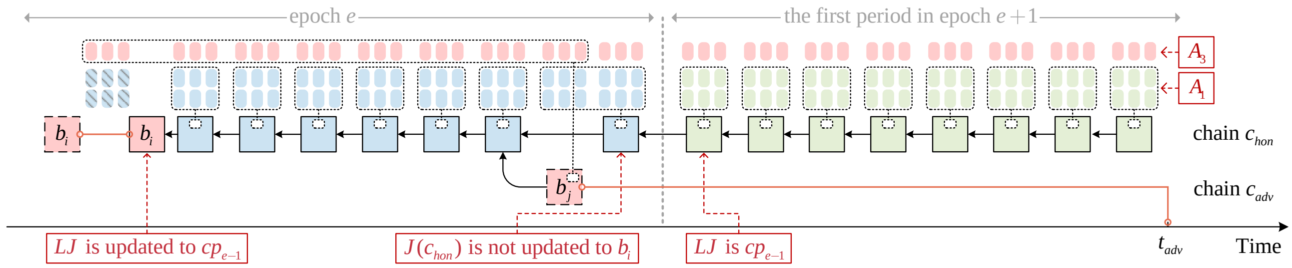
We use c_{adv} to denote the branch withheld by Byzantine validators. In our attack, after c_{adv} is released, *LJ* is updated by all honest validators. Similarly, c_{hon} is the branch seen by honest validators before c_{adv} is released.

The (one-time) attack. To kick-start a staircase attack, Byzantine validators also need to wait for an opportune epoch, the condition of which is exactly the same as our warm-up attack: the proposer v_i of the first slot t in epoch e is Byzantine. Similar to our warm-up attack, we use cp_{e-2} to denote the *LJ* of all validators and cp_{e-1} to denote the checkpoint block proposed in the first slot of epoch $e-1$. The strategies of our staircase attack are summarized below.

(1) (Figure. 7a) In slot t of epoch e, v_i replays the warm-up attack: v_i withholds its block b_i and releases b_i at the end of slot t, after which all honest validators set their checkpoint of epoch e as b_i. In epoch e, the Byzantine validators have two strategies. First, *all* Byzantine validators in epoch e withhold their attestations with cp_{e-1} as *source* and b_i as *target* (i.e., A_3), regardless of which slot each validator is the designated attestor. Second, the *last* Byzantine proposer v_j in slot t_j (all proposers in the rest of epoch e are honest[5]) in epoch e includes the attestations from the Byzantine validators in its block b_j and withholds b_j. Note that not *all* the attestations from Byzantine validators are included in b_j (as according to the protocol, a block in some slot cannot include attestations with a higher slot number). In this case, the attestations not included in b_j can simply be included in blocks proposed in epoch $e+1$ and the corresponding validators will still receive their rewards.
At the end of epoch e, the chain c seen by all honest validators consists of the attestations from all honest attestors from slot $t+1$ to $t+31$ (the last slot in epoch e) where *source* of these attestations is cp_{e-1}. The expected number of attestations on chain c is then $31(N-f)/32$, equal to the number of honest attestors from slot $t+1$ to $t+31$. Note that even if the attestations from honest attestors in slot $t+31$ are received by all honest validators and the proposer of the first block in epoch $e+1$ includes these attestations, the maximum number of attestations with b_i as *target* is still $31(N-f)/32$. Namely, as we assume $f = N/3$, the number of attestations is $\frac{31}{32} \times \frac{2N}{3} < \frac{2N}{3}$, the requirement for validators to update their $J(c)$. Therefore, at the end of epoch e, *LJ* and $J(c)$ of *all* honest validators is cp_{e-1}.

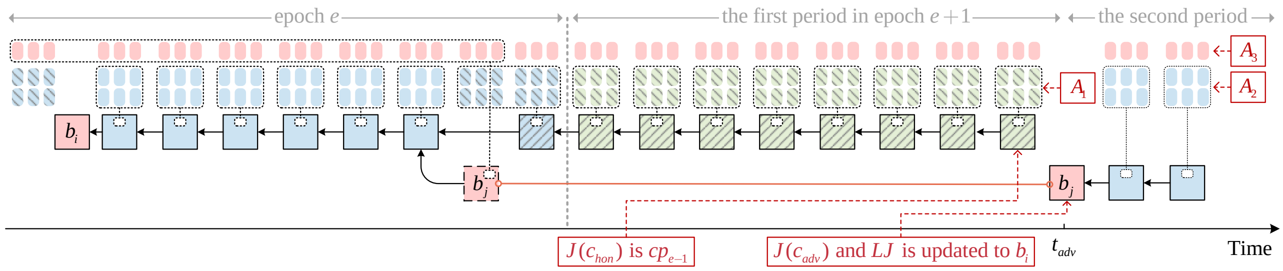
(2) (Figure. 7b) In epoch $e+1$, we divide the epoch into two periods. Let t' be the first slot in epoch $e+1$, i.e., $t' = t+32$. For now, we assume the first period ends at the end of slot $t_{adv}-1$ and discuss the value of t_{adv} later.

[5] According to the RANDAO protocol, the roles of each validator in epoch e can be predicted before epoch e begins.



(a) Step 1: In epoch e, the first Byzantine proposer v_i replays the warm-up attack. All Byzantine validators withhold their attestations. The last Byzantine proposer v_j includes the attestations from the Byzantine validators in its block b_j and withholds b_j. *LJ* and $J(c)$ of all honest validators are cp_{e-1}.

(b) Step 2: In the first period of epoch $e+1$ (before slot t_{adv} begins), branch c_{hon} is seen by honest validators and branch c_{adv} is withheld by Byzantine validators. *LJ* is not updated and is still cp_{e-1} during this period. The attestations A_1 included in c_{hon} have cp_{e-1} as the *source*.



(c) Step 3: At the beginning of slot t_{adv}, Byzantine validators release c_{adv}. As c_{adv} includes a sufficiently large fraction of attestations with *target* as b_i, *LJ* is updated to b_i. In the second period, c_{hon} is pruned and c_{adv} becomes the canonical chain. Attestations A_1 included in c_{hon} are discarded and the corresponding validators will be penalized.

Figure 7: One-time staircase attack where all Byzantine validators collude. We assume that before epoch e begins, the *LJ* of all validators is cp_{e-2} and the (checkpoint) block proposed in the first slot of epoch $e-1$ is cp_{e-1}.

There are two forks of the chain, branch c_{hon} seen by honest validators and branch c_{adv} withheld by Byzantine validators. The adversary's strategy in the first period is to prepare attestations with b_i as *source* and b_j as *target* and withhold their attestations. The branch c_{hon} thus consists of blocks from honest proposers for slots $[t', t_{adv}-1]$. The attestations included in c_{hon} will all have *source* as cp_{e-1}, and we let the set of attestations be A_1.

(3) (Figure. 7c) At the beginning of slot t_{adv} (i.e., the second period), the adversary releases the withheld chain c_{adv} and our goal here is for all honest validators to update their *LJ* to b_i and make c_{adv} become the canonical chain. Here, we need to dive into the attestations in c_{adv}. According to the discussion in step (1), the branch from b_i to b_j consists of the attestations (with *source* as cp_{e-1} and *target* as b_i) from *all* attestors (Byzantine and honest) between slot t and t_j (the slot of b_j). If the number of the attestations is greater than $2N/3$ (i.e., $V(c_{adv})[b_i] > 2N/3$), all honest validators will set their $J(c_{adv})$ as b_i. Therefore, we know that if $\frac{t_j-t-1}{32} > \frac{2}{3}$, the condition $V(c_{adv})[b_i] > 2N/3$ will be satisfied. We show in Lemma 1 that if the adversary controls $f = N/3$ validators, this happens with a probability of 98.84%.

As $J(c_{adv})$ is updated to b_i after c_{adv} is released, it is not difficult to see that *LJ* will be updated to b_i in the middle of an epoch! Moreover, as $J(c_{hon})$ is still cp_{e-1} in the second period of epoch $e+1$, the branch c_{hon} (up to b_i) will be pruned and c_{adv} becomes the canonical chain. Any attestations in A_1 will be discarded since the *source* field in A_1 is different from *LJ* of validators in the second period. Thus, the corresponding attestors will be penalized.

Value of t_{adv} in one-time attack. As discussed in step (1), $J(c_{hon})$ will never be updated to b_i in our attack. Therefore, the branch c_{hon} (up to b_i) will be pruned. To maximize the number of honest attestors that will be penalized (i.e., $|A_1|$), we can set up t_{adv} as large as possible. In fact, t_{adv} can be the last slot in epoch $e+1$, so almost *all* honest validators will be penalized.

Lemma 1. *Assuming that slot t is the first slot of epoch e and $f = N/3$. Given the last slot t_j of epoch e in which the proposer v_i is Byzantine (all proposers in the rest of epoch e are honest), the probability that $\frac{t_j-t-1}{32} \geq 2/3$ is at least* 98.84%.

Proof. According to the definition, $\frac{t_j-t-1}{32} < 2/3$ happens only if any proposer in slot t_j+1 to slot $t+31$ is honest,

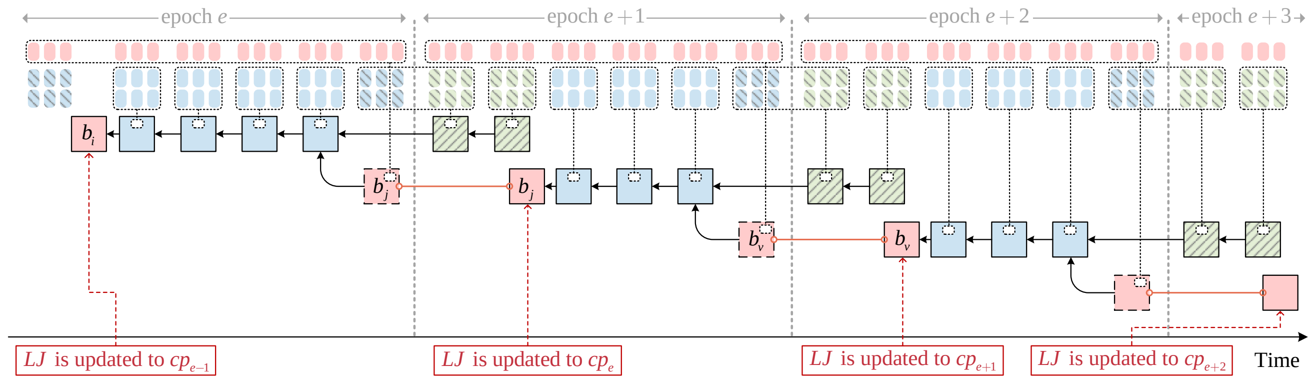


Figure 8: Staircase attack where the attack is repeated in every epoch. The figure is best viewed in color. The green, blue, and red rounded rectangles represent attestations in A_1, A_2, and A_3 respectively.

i.e., $\lfloor \frac{32}{3} \rfloor = 11$. We now calculate the probability that the proposers in the last 11 slots in epoch e are all honest. As the proposer of each slot is selected pseudorandomly according to RANDAO, the probability of the above situation is:

$$(\frac{N-f}{N})^{11} \approx 1.16\%.$$

The probability that there is at least one Byzantine proposer in one of the last 11 slots is thus:

$$P_{succ} = 1 - (\frac{N-f}{N})^{11} \approx 98.84\%.$$

□

Repeating the attack for *every* epoch. Our attack above can in fact be continued in *every* epoch after $e+1$. This is achieved by simply having the last Byzantine validator in *each* epoch e' withhold its block just as v_j does and then repeat steps (2) and (3) in epoch $e'+1$. For instance, in epoch $e+1$ of the (one-time) attack, the last Byzantine validator v_k in slot t_k starts to withhold its block b_k and all Byzantine attestors in the rest of epoch $e+1$ withhold their attestations. In epoch $e+2$, branch c_{hon}^{e+2} that extends the canonical chain is seen by all honest validators and Byzantine validators withhold a branch c_{adv}^{e+2} led by b_k. After the end of the first period of $e+2$, the withheld branch is released, in the hope that $J(c_{adv}^{e+2})$ and LJ is updated to the first block in epoch $e+1$ (denoted as cp_{e+1}) and $J(c_{hon}^{e+2})$ is still b_i (the first block in epoch e). After LJ of honest validators is updated to cp_{e+1}, c_{hon}^{e+2} will be discarded.

Value of t_{adv} in staircase attack. Notably, same as our one-time attack, only the attestations in A_2 and A_3 can be included in c_{adv}^{e+2}. Therefore, we need to ensure that $J(c_{adv}^{e+2})$ can be updated to cp_{e+1} (and also every epoch after $e+2$). Now it becomes clear why we additionally need to define A_2 and A_3 in addition to A_1. We already know that the set of attestations from Byzantine validators is A_3 and $|A_3| = f < N/3$. To ensure that the branch c_{adv}^{e+2} consists of $2N/3$ attestations, set A_2 must consist of more than $2N/3 - f$ attestations. To satisfy this condition, we can set A_1 to A_3 with roughly the same size and set t_{adv} as the middle of an epoch e', i.e., $t_{adv} = 16 + t'$, where t' is the first slot in e'.

Lemma 2. *In our staircase attack, honest attestors that create attestations in A_1 will be penalized and $|A_1| \leq N/3$, regardless of the fraction of f in N.*

Proof. Let $|A_3|$ be f. $|A_2|$ must be greater than $2N/3 - f$ for the branch by the adversary to become the canonical chain, following the discussion above. Therefore, the size of A_1 must no more than than $N - |A_2 \cup A_3| = N/3$. □

Theorem 1. *According to the configuration in Ethereum where $W_r \leq 27W_p/20$ (see Appendix A for more details), the expected incentive of any honest validators becomes lower than 0 when $f \geq 8N/27 \approx 29.6\%N$.*

Proof. The attestation incentive each validator receives is the difference between the rewards and the penalties. According to our attack, attestors that create attestations in A_2 will receive rewards and attestors that create attestations in A_1 will suffer from penalties. To lower the expected incentives received by honest validators, we can simply let $|A_1| = N/3$ according to Lemma 2. According to our discussion §3.4, to determine the concrete amount, we also need to calculate R, the rewards scale with participation. As attestations in A_2 and A_3 are included in the canonical chain, $R = |A_2 \cup A_3|/N \approx 2/3$. Therefore, the reward received by each honest validator is $R \times W_r I_{base} = \frac{2}{3} W_r I_{base}$ and the penalty of each validator is $W_p I_{base}$. Accordingly, the incentives received by all honest validators are $(\frac{2}{3}|A_2|W_r - |A_1|W_p)I_{base}$. Our goal is to learn the expected incentives received by any honest validator. As the number of honest validators is $|A_1 \cup A_2|$, the expected incentives of each validator becomes:

$$I_{hon} = (\frac{2}{3}|A_2|W_r - |A_1|W_p)\frac{I_{base}}{|A_1 \cup A_2|}.$$

As $|A_1| = N/3$ and $|A_1 \cup A_2| = N - f$, we have:

$$I_{hon} = (\frac{4}{15}N - \frac{9}{10}f)\frac{W_p I_{base}}{N - f} \leq 0.$$

We can simply set $\frac{4}{15}N - \frac{9}{10}f \leq 0$ to satisfy the equation above, so $f \geq 8N/27$. □

On the contrary, Byzantine validators do not suffer from penalties. However, the rewards they receive will be decreased. Following the discussion in Theorem 1, the reward each Byzantine validator receives is $I_{adv} = \frac{2}{3}W_r I_{base}$ in each epoch, about 33% lower than the fair share.

5 Implementation and Evaluation

Implementation. We implement our attack using *Prysm*[6], one of the most widely adopted Ethereum 2.0 beacon chain implementations written in Golang. The Prysm version is in Capella[7], the latest version at the time of writing. We modify the codes in the Prysm as specified in Figure. 9, where changes we make on top of the PoS protocol are highlighted in red and the files we modify are included as comments. Our implementation has exactly the same effect as the attack mentioned in §4.3 while the actual implementation is slightly different. We have made the scripts for our attacks and the logs of our experiments available[8] and will open source them.

Experiment configuration. We establish a local *testnet* with 1,000 validators connected through a P2P LAN network and vary the fraction of Byzantine validators to understand the impact of our attack. We choose the LAN network as the communication delay is negligible, demonstrating that our attack can be launched even in a fully synchronous network.

We vary the number of Byzantine validators using $f = 296$ (i.e., $f \approx 8N/27$), $f = 310$, $f = 320$, and $f = 333$ and assess the incentives received by the validators. In each experiment, we fix the identities of f Byzantine validators, run our testnet for one day (225 epochs), and collect the attestation incentive of each honest validator from their logs. As a comparison, we also run the testnet for one day without launching the attack and collect the incentives of all validators, the value of which is also known as the *fair share*.

Evaluation results. We evaluate the *incentive loss rate* of honest validators. In each experiment, we report the loss rate of an honest validator in Figure. 10. The incentive loss rate is calculated as follows. The incentive loss of an honest validator is the difference between its fair share and the incentives it receives when the attack is launched. The percentage of incentive loss in the fair share is called the incentive loss rate. The incentive loss rate can be interpreted as follows. If the attack is not launched, the incentive loss rate is supposed to be 0%. If the incentive loss rate is close to 100%, *all* honest validators will lose their incentives.

Our results show a notable trend: as the attack is being launched, the incentive loss rate of each honest validator increases significantly and then stabilizes. For $f = 296$, the loss rate eventually stabilizes at 100%, matching our results in Theorem 1. For $f = 310$, $f = 320$, and $f = 333$, the incentive loss rate exceeds 100% and all the honest validators are expected to suffer from stake loss. For instance, when $f = 333$, the incentive loss rate is close to 120%, i.e., honest validators suffer from a 20% stake loss compared to their fair share.

```
The Workflow of a Byzantine Validator v_k
global parameters: slot counter t
local parameters: last justified checkpoint LJ.
-----------------------------------------------------------
upon slot t do
  obtain a set I of Byzantine proposers in the current epoch,
  ordered by the slot numbers the validators are allocated for
  ▷ prysm/beacon-chain/rpc/prysm/v1alpha1/validator/proposer.go
  ▷ prysm/validator/client/propose.go
  as the proposer for slot t
    follow lines 3-9 specified in Figure. 2
    if t is the first slot in an epoch then
      send b to all validators at the end of slot t
    elseif v_k is the last Byzantine proposer in I then
      send b to all Byzantine validators at once and to all honest
      validators at the beginning of the 17th slot in the next epoch
    else send b to all validators at once
  ▷ prysm/validator/client/attest.go
  as an attestor for slot t
    follow lines 12-18 specified in Figure. 2
    send att to the last Byzantine proposer in I
```

Figure 9: The workflow of a Byzantine validator v_k. The main changes made on top of Figure. 2 are highlighted in red.

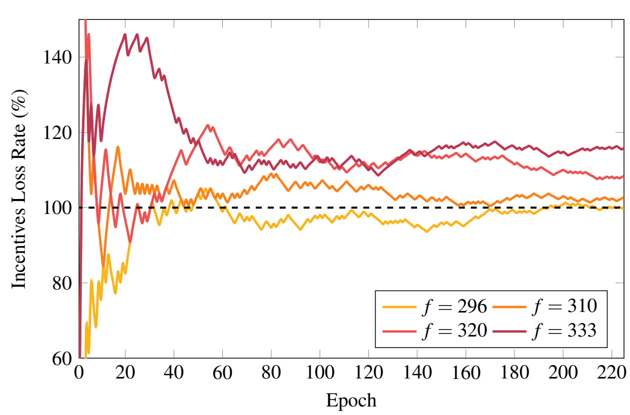


Figure 10: The incentives loss rate of an honest validator for experiments with 1,000 validators and f Byzantine validators. Each experiment is launched for one day (225 epochs).

[6] Prysm: https://github.com/prysmaticlabs/prysm

[7] Capella: https://github.com/ethereum/consensus-specs/tree/dev/specs/capella

[8] Scripts of our attacks and logs of the experiments: https://github.com/tsinghua-cel/Staircase-Attack

6 Attack Feasibility and Analysis

In this section, we analyze the feasibility of our attacks in the current Ethereum system with realistic settings. So far, we have presented three attacks on the incentive mechanism of Ethereum: a warm-up attack where a single Byzantine validator can launch the attack and make some honest validators suffer from penalties; a one-time attack where Byzantine validators collude to make all honest validators suffer from penalties; and the staircase attack where Byzantine validators continuously launch the attack in every epoch and all honest validators lose their incentives.

Feasibility of warm-up attack. Recall that in our warm-up attack, the Byzantine validator v_i only needs to delay its block b_i. In practice, the feasibility is related to *honest reorg* currently implemented in Ethereum. *Honest reorg* is a mechanism that prevents intentionally withheld blocks. In particular, a block proposed in slot t with fewer than 20% attestations from attestors in slot t is considered invalid. If v_i releases its block at the end of slot t, the block will be reorganized and our warm-up attack may fail. To prevent b_i from being considered invalid due to honest reorg, v_i can carefully release a block *late* enough to ensure that at least 20% attestors (but not all) receive b_i. In this way, the warm-up attack can still succeed and around 80% honest attestors in the slot will be penalized (i.e., approximately $(N-f)/40$). Note that in practice, the concrete release time is highly related to the actual network condition in the Ethereum network. We leave the analysis as future work.

Feasibility of one-time attack and staircase attack. We have validated the feasibility of our attacks in the current Ethereum system beyond 1,000 validators. Indeed, Ethereum has more than 900,000 validators as of today[9], we thus analyze whether our attacks can be launched in such a large-scale system. Our analysis reveals a somewhat surprising result: the feasibility of the attacks is directly related to the number of validators and the maximum number of attestations in each block (i.e., MAX_ATTESTATIONS in Ethereum (MaxAtt for short)).

Ethereum currently employs a committee-based scheme to aggregate the attestations using the BLS signature [7]. Recall that every slot has $N/32$ validators randomly selected by RANDAO. The validators in each slot are further divided into several committees [45]. In every slot, matching attestations (same *target*, *source*, and *head*) in the same committee are aggregated into one aggregated attestation. We use N_c to denote the number of committees in each epoch and N_a to represent the number of aggregated signatures in every block.

In the most recent configuration of Ethereum, N_a is limited by MaxAtt [45]. The value of N_c is determined as follows (also as shown in Figure. 11), where MaxCom is the maximum number of committees in a slot (the actual parameter name is MAX_COMMITTEES_PER_SLOT [45]):

$$N_c = \begin{cases} N/128, & \text{if } N \leq 4096 \times \text{MaxCom}; \\ 32 \times \text{MaxCom}, & \text{otherwise.} \end{cases} \quad (1)$$

Currently, the parameter MaxAtt is 128 and the parameter MaxCom is 64. Thus we have $N_a = \text{MaxAtt} = 128$, $N_c = 32 \times \text{MaxCom} = 2048$, and $N_c = 16N_a$.

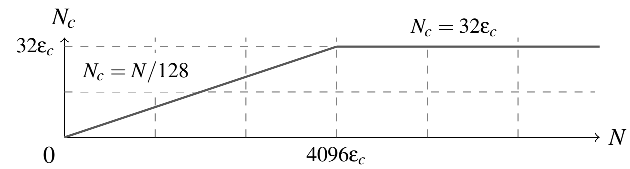


Figure 11: The value of N_c as N grows. We use ε_c to denote MaxCom.

Lemma 3. *To launch staircase attack, $N_a \geq N_c$.*

Proof. In our staircase attack, the block proposed by the last Byzantine validator in an epoch includes attestations from all Byzantine validators. Therefore, N_a is no less than the number of aggregated attestations from Byzantine validators.

According to the RANDAO protocol, Byzantine attestors are randomly distributed among N_c committees. Since attestations across committees cannot be aggregated, the number of aggregated attestations from Byzantine validators is no less than N_c, i.e., $N_a \geq N_c$. □

Based on Lemma 3, we now analyze the impacts of system parameters and the number of validators to our attack. We also provide a modified attack to make our attack feasible in today's Ethereum system.

Impacts of the maximum number of attestations to our attack. To satisfy the condition $N_a \geq N_c$ specified in Lemma 3, we need either a larger N_a or a smaller N_c. We have the following observations:

- If the MaxAtt parameter increases from 128 to 2,048, a block can include aggregated attestations from *all* Byzantine validators. This is mainly because $N_c = 16N_a$ in the current system, and a 16-fold increase in MaxAtt would make $N_a = N_c$. Alternatively, we can also set MaxCom to MaxCom/16 to achieve the same result (as N_c depends on MaxCom), i.e., by decreasing MaxCom from 64 to 4.
- There are many alternative ways for $N_a = N_c$ to be satisfied. For instance, if the parameter MaxAtt increases from 128 to 512 and the parameter MaxCom decreases from 64 to 16, the condition $N_a \geq N_c$ is also satisfied.

Such a counter-intuitive finding shows that the values of system parameters such as MaxAtt and MaxCom are not an easy engineering decision and should be set up carefully.

Impacts of the number of the number of validators to our attack. If we fix the MaxAtt and MaxCom parameters,

[9]Data source (accessed in Feb 2024): https://www.beaconcha.in/charts/validators

our attack can be launched with fewer than 16,384 validators. This is because when we fix the value of parameters, we need to set $N_c \leq N_a = 128$ according to Lemma 3. According to equation (1), setting N_c as $32 \times \mathsf{MaxCom}$ will never make $N_c \leq 128$. As a result, we can set $N_c = N/128$ for $N_c \leq N_a = 128$ to hold. Accordingly,

$$N \leq 128 \times 128 = 16384.$$

Namely, if we do not change the system parameters, Ethereum becomes vulnerable to our attack when the number of validators is below 16,384.

A modified attack on the current system. We can modify our attack to be feasible in the current Ethereum system. The main reason why our attack might fail in the current system is that a block cannot include attestations from all Byzantine validators. Indeed, based on the current system parameters, a block can include attestations from 128 committees (i.e., MaxAtt =128), and there are 64 committees (i.e., MaxCom =64) in each slot. Since the Byzantine attestors are distributed in every committee and attestations across committees cannot be aggregated, a block can include attestations of Byzantine attestors from at most two slots. Our modified attack thus requires more Byzantine validators to release the withheld attestations. In particular, the attestations from Byzantine validators in the first 28 slots are directly sent to all validators in the system. Only the attestations from Byzantine validators in the last four slots are withheld. Among them, attestations in two slots are included in the block proposed by the last Byzantine proposer. Therefore, we require the slot of the last Byzantine proposer to be one of the last four slots (see Appendix B for more information). Under the assumption of $f = 0.33N$, the probability of repeating the attack is:

$$P_{succ} = 1 - \left(\frac{N-f}{N}\right)^4 \approx 80.25\%.$$

We provide the details of the modified attack in Appendix B. If the attack cannot be repeated in the next epoch, the adversary can wait for another opportune epoch to restart the attack. Recall that our attack can be started if the proposer of the first slot of an epoch is Byzantine. The probability of starting an attack is thus 33%.

Combining the discussion above, we show the attestation incentives of honest validators in Theorem 2 in Appendix B. Specifically, if the adversary controls 33% of the total stake, the attestation incentives of an honest validator become only 28.9% of the fair share.

7 Discussion

Mitigation. We show two possible solutions as mitigation of the attack. One solution is to modify the tree pruning rules in HLMD GHOST. In the current fork choice rule (see §3.2), the branch that does not include the latest justification information will be pruned. Our attack takes advantage of this to force honest validators to be penalized. Relaxing the pruning rules and still keeping the honest branch in the HLMD GHOST tree can mitigate our attack. Alternatively, adjusting the concrete amount of rewards and penalties can also mitigate our attack. Currently, the amount of penalties and rewards are almost identical. If the penalties in the attestation incentive mechanism are lowered, our attack can be mitigated accordingly.

Note that Ethereum is taking the first approach as mentioned in our responsible disclosure. In particular, chain c will not be pruned if the difference between $\mathtt{ep}(J(c))$ and the current epoch is no more than two epochs. This solution can mitigate our attack but cannot fully prevent our staircase attack from happening. In particular, the Byzantine validators can delay the attestations for a longer period of time and the honest chain will still be pruned.

Situations observed in practice. Situations similar to our warm-up attack are in fact quite common in practice. According to beaconcha.in, at least one validator in each epoch is penalized because of *incorrect* attestations. Such events are often caused by natural network delay such that honest attestors do not receive the first block in an epoch before sending their attestations.

Extension of our attack. As mentioned in §2, our staircase attack is one kind of reorg attack. Although we focus on attestation incentives in this paper, our attack can be extended to cause other effects, given its nature as a reorg attack. For example, the adversary can manipulate the order of the transactions and obtain more MEV (Maximal Extractable Value) [14]. Besides, strategically choosing the parent block for the withheld blocks allows the adversary to reorg more blocks from the honest validators. This may lower the chain quality and pose a liveness threat to the system.

Future directions. Our counter-intuitive finding about MAX_ATTESTATIONS parameter setting may lead to future research directions. Indeed, having a larger MAX_ATTESTATIONS will enhance the performance of the system as more attestations are aggregated. In fact, the parameter is even related to the liveness of the system. However, our attack shows that a larger MAX_ATTESTATIONS might make honest validators suffer from higher incentive loss. Therefore, it is interesting to learn whether there exists some sweet spot for the MAX_ATTESTATIONS parameter and how the MAX_ATTESTATIONS, N, and the incentives are correlated. Furthermore, it would be useful to consider security properties related to incentives when designing consensus protocols.

8 Conclusion

We present staircase attack, the first attack on the incentive mechanism of the Ethereum Proof-of-Stake (PoS) protocol. Without considering the constraints of system parameters such as the number of validators, we show that an adversary that controls 29.6% stake can launch the attack and eventually *all* honest validators lose their attestation incentives. As the fraction of stake controlled by the adversary increases, honest validators may even suffer from stake loss. Moreover, considering the values of system parameters, the feasibility of our attack is closely related to two parameters: the number of validators and MAX_ATTESTATIONS, the maximum number of attestations included in each block. With the current Ethereum setup (900,000 validators and MAX_ATTESTATIONS =128), we show that an adversary that controls 33% stake can make honest validators suffer from no incentives with a probability of 80.25%. Our attack shows that in addition to the safety and liveness properties considered in conventional consensus protocols, properties regarding the incentives might also be worth investigating in today's blockchain systems.

Acknowledgment

The authors would like to thank our shepherd and the Usenix Security reviewers for their helpful comments that greatly improved our paper. This work was supported in part by the National Key R&D Program of China under 2022YFB2701700, Beijing Natural Science Foundation under M23015, and the National Natural Science Foundation of China under 92267203, China Postdoctoral Science Foundation under 2023M741949 and Tsinghua Shuimu Scholar.

References

[1] Musab A Alturki and Grigore Roşu. Statistical model checking of randao's resilience to pre-computed reveal strategies. In *Formal Methods. FM 2019 International Workshops*, pages 337–349, 2019.

[2] Aditya Asgaonkar. Unrealized justification reorgs. https://notes.ethereum.org/@adiasg/unrealized-justification. (accessed in Feb 2024).

[3] Sarah Azouvi, George Danezis, and Valeria Nikolaenko. Winkle: Foiling long-range attacks in proof-of-stake systems. In *AFT*, pages 189–201, 2020.

[4] Sarah Azouvi and Marko Vukolić. Pikachu: Securing pos blockchains from long-range attacks by checkpointing into bitcoin pow using taproot. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, pages 53–65, 2022.

[5] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. *IACR Cryptol. ePrint Arch.*, 2016(919), 2016.

[6] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. In *AFT*, pages 12–24, 2020.

[7] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT*, pages 514–532, 2001.

[8] Jonah Brown-Cohen, Arvind Narayanan, Alexandros Psomas, and S Matthew Weinberg. Formal barriers to longest-chain proof-of-stake protocols. In *EC*, pages 459–473, 2019.

[9] Vitalik Buterin. Proposal for mitigation against balancing attacks to lmd ghost. https://notes.ethereum.org/@vbuterin/lmd_ghost_mitigation. (accessed in Feb 2024).

[10] Vitalik Buterin. Discouragement attacks. 2018.

[11] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[12] Vitalik Buterin, Diego Hernandez, Thor Kamphefner, Khiem Pham, Zhi Qiao, Danny Ryan, Juhyeok Sin, Ying Wang, and Yan X Zhang. Combining ghost and casper. *arXiv preprint arXiv:2003.03052*, 2020.

[13] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *TOCS*, 20(4):398–461, 2002.

[14] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *SP*, 2020.

[15] Francesco D'Amato and Caspar Schwarz-Schilling. Proposer boost considerations. https://notes.ethereum.org/@casparschwa/H1T0k7b85. (accessed in Feb 2024).

[16] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *EuroSys*, pages 34–50, 2022.

[17] Sisi Duan, Xin Wang, and Haibin Zhang. Practical signature-free asynchronous common subset in constant time. In *CCS*, 2023.

[18] Sisi Duan and Haibin Zhang. Foundations of dynamic bft. In *SP*, pages 1317–1334, 2022.

[19] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *JACM*, 35(2):288–323, 1988.

[20] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. *Communications of the ACM*, 61(7):95–102, 2018.

[21] Matheus VX Ferreira and S Matthew Weinberg. Proof-of-stake mining games with perfect randomness. In *EC*, pages 433–453, 2021.

[22] Fork choice upgrade. https://github.com/ethereum/consensus-specs/pull/3290. (accessed in Feb 2024).

[23] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68, 2017.

[24] Neil Giridharan, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Bullshark: DAG BFT protocols made practical. In *CCS*, 2022.

[25] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: a scalable and decentralized trust infrastructure. In *DSN*, pages 568–580, 2019.

[26] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *CRYPTO*, pages 451–480, 2020.

[27] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *CRYPTO*, pages 357–388, 2017.

[28] Wenting Li, Sébastien Andreina, Jens-Matthias Bohli, and Ghassan Karame. Securing proof-of-stake blockchain protocols. In *ESORICS*, pages 297–315, 2017.

[29] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *FOCS*, pages 120–130, 1999.

[30] Ryuya Nakamura. Analysis of bouncing attack on ffg. https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113. (accessed in Feb 2024).

[31] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *SP*, pages 446–465, 2021.

[32] Joachim Neu, Ertem Nusret Tas, and David Tse. Two more attacks on proof-of-stake ghost/ethereum. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus*, pages 43–52, 2022.

[33] Michael Neuder, Daniel J Moroz, Rithvik Rao, and David C Parkes. Selfish behavior in the tezos proof-of-stake protocol. *arXiv preprint arXiv:1912.02954*, 2020.

[34] Michael Neuder, Daniel J Moroz, Rithvik Rao, and David C Parkes. Low-cost attacks on ethereum 2.0 by sub-1/3 stakeholders. *arXiv preprint arXiv:2102.02247*, 2021.

[35] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *PODC*, pages 315–324, 2017.

[36] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *ASIACRYPT*, pages 380–409, 2017.

[37] Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. Ethereum proof-of-stake under scrutiny. In *SAC*, pages 212–221, 2023.

[38] Potuz. Justification widtholding attacks. https://hackmd.io/o9tGPQL2Q4iH3Mg7Mma9wQ. (accessed in Feb 2024).

[39] Potuz. Witholding attack mitigation. https://hackmd.io/a8vbgF6YR0-j6T9LpcYB3g. (accessed in Feb 2024).

[40] Proposer lmd score boosting #2730. https://github.com/ethereum/consensus-specs/pull/2730. (accessed in Feb 2024).

[41] Danny Ryan. Epoch reorg. https://notes.ethereum.org/VH_B3kEVQFav4roEgYuCjA. (accessed in Feb 2024).

[42] Danny Ryan. Fork choice bugfix disclosure. https://notes.ethereum.org/@djrtwo/2023-fork-choice-reorg-disclosure. (accessed in Feb 2024).

[43] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake ethereum. In *FC*, pages 560–576, 2022.

[44] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. In *FC*, pages 507–527, 2015.

[45] Ethereum proof-of-stake consensus specifications. https://github.com/ethereum/consensus-specs. (accessed in Feb 2024).

[46] Michael Sproul. Allow honest validators to reorg late blocks. https://github.com/ethereum/consensus-specs/pull/3034. (accessed in Feb 2024).

[47] Xiao Sui, Sisi Duan, and Haibin Zhang. Marlin: Two-phase bft with linearity. In *DSN*, pages 54–66, 2022.

[48] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi,

Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *SP*, pages 444–460, 2017.

[49] Ertem Nusret Tas, David Tse, Fangyu Gai, Sreeram Kannan, Mohammad Ali Maddah-Ali, and Fisher Yu. Bitcoin-enhanced proof-of-stake security: Possibilities and impossibilities. In *SP*, pages 126–145, 2023.

[50] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 2014.

[51] Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. Sok: Mev countermeasures: Theory and practice. *arXiv preprint arXiv:2212.05111*, 2022.

[52] Bennet Yee. Keep your transactions on short leashes. *arXiv preprint arXiv:2206.11974*, 2022.

[53] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *PODC*, 2019.

[54] Haibin Zhang and Sisi Duan. Pace: Fully parallelizable bft from reproposable byzantine agreement. In *CCS*, 2022.

[55] Haoqian Zhang, Mahsa Bastankhah, Louis-Henri Merino, Vero Estrada-Galiñanes, and Bryan Ford. Breaking blockchain rationality with out-of-band collusion. *arXiv preprint arXiv:2305.00554*, 2023.

[56] Haoqian Zhang, Louis-Henri Merino, Vero Estrada-Galinanes, and Bryan Ford. Flash freezing flash boys: Countering blockchain front-running. In *International Conference on Distributed Computing Systems Workshops*, pages 90–95, 2022.

A Rewards and Penalties in Ethereum

In this section, we provide more details on attestation rewards and penalties [45]. The reward weight of a validator W_r is set to 54 if the corresponding attestation is finalized on-chain and the *head* field in the attestation is correct (i.e., the block matches the head of the canonical chain). If the attestation is finalized on-chain but the *head* in the attestation is incorrect, the reward weight W_r is set to 40. Finally, if the corresponding attestation is not finalized on-chain[10], the penalty weight of a validator W_p is set to 40.

As mentioned in §4, the attestations in A_2 are finalized on-chain, so the reward weights of the corresponding attestors W_r are at most 54. Meanwhile, the attestations in A_1 are discarded, so the penalty weight of the corresponding attestors W_p is 40. Thus, we have $W_r \leq 27W_p/20$.

[10]The value can be found at https://github.com/ethereum/consensus-specs/blob/dev/specs/altair/beacon-chain.md#incentivization-weights

B Modified Staircase Attack

In this section, we show the modified staircase attack. The modified attack can be launched based on the latest system configuration of Ethereum. We show the workflow of the modified attack in Figure. 12. Compared to the attack presented in §4.3, we make three major changes. First, as shown in Figure. 13, there are now three statuses for each Byzantine validator: idle, repeat, and stop. The status is *idle* if the attack is not being launched. The status is *repeat* if the attack is being launched and can be repeated to the next epoch. The status is *stop* if the attack cannot be repeated and will be stopped at the end of the epoch. Here, the last epoch of the attack becomes similar to that in our one-time staircase attack. Second, the attestations from Byzantine validators in the first 28 slots are not withheld. Instead, they are directly sent to all validators in the system. The attestations from the Byzantine attestors in the last four slots are sent to the last Byzantine proposer v_j. Third, to continuously launch the attack, slot t_{adv} (when the last Byzantine proposer v_j releases the block b_j) is not set to the middle of an epoch. Instead, it is set to the beginning of the 14-th slot of an epoch.

We also introduce a function called *judge* (lines 41-45). Briefly speaking, given an epoch e, the *judge* function returns 1 if the attack can be repeated in the next epoch and 0 otherwise.

Lemma 4. *In the modified attack, the number of attestations included in the canonical chain can not exceed* $2N/3$ *at the end of an epoch.*

Proof. After the attack is launched, none of the honest validators update their *LJ* before the last Byzantine validator releases the withheld block. If the status is *repeat*, the attestations from the honest validators in the first 13 slots are discarded and attestations from the honest validators in the last slot are not included. On the contrary, Byzantine validators use the new *LJ* as *source* of their attestation. Therefore, the number of attestations included in the canonical chain is smaller than $18(N-f)/32+28(N-f)/32 = 2N/3$. For the *stop* status, the proof is conducted in the same manner. After the warm-up attack is launched, none of the honest validators update their *LJ* before the 28-th slot. On the contrary, Byzantine validators use the new *LJ* as *source* of their attestation. Thus, the number of attestations is lower than $2N/3$ and the checkpoint is not set correctly. □

Lemma 5. *If the adversary controls 33% of the total stake and the attack is being launched, all honest validators lose their incentives or even suffer from stake loss.*

The Workflow of a Byzantine Validator v_k

global parameters: slot counter t, attack state $status_{cr}$, $status_{nxt}$
local parameters: last justified checkpoint LJ.

```
01 init status_cr ← idle, status_nxt ← idle
02 upon slot t do
03   let current epoch number be e ← ⌊t/32⌋
04   ▷ prysm/validator/client/runner.go
05    if t is the first slot in epoch e then
06      if status_cr = repeat then
07        if judge(e) = 1 and judge(e+1) = 1 then
08          status_nxt ← repeat
09        if judge(e) = 1 and judge(e+1) = 0 then
10          status_nxt ← stop
11      if status_cr = stop then
12        status_nxt ← idle
13   ▷ prysm/beacon-chain/rpc/prysm/v1alpha1/validator/proposer.go
14   ▷ prysm/validator/client/propose.go
15   as the proposer for slot t
16     follow lines 3-9 specified in Figure. 2
17     if status_cr = idle then
18       if t is the first slot in epoch e then
19         send b to 20% honest validators at 1/3 of slot t
20         if judge(e) = 1 and judge(e+1) = 1 then
21           status_nxt ← repeat
22         if judge(e) = 1 and judge(e+1) = 0 then
23           status_nxt ← stop
24       else status_nxt ← idle
25     elseif v_k is the last Byzantine proposer in I then
26       send b to all Byzantine validators at once
27       set t_adv as follows
28       if status_cr = repeat then
29         t_adv ← the 14-th slot in epoch e+1
30       if status_cr = stop then
31         t_adv ← the last slot in epoch e+1
32       send b to all honest validators at the beginning of slot t_adv
33     else send b to all validators
34   ▷ prysm/validator/client/attest.go
35   as an attestor for slot t
36     follow lines 12-18 specified in Figure. 2
37     if status_cr = repeat or status_cr = stop then
37       if t mod 32 < 28 then
37         send att to all validators
37       else
38         send att to the last Byzantine proposer in I
39   if t is the last slot in epoch e then
40     status_cr ← status_nxt
41 function judge(epoch e)
42   obtain a set I of Byzantine proposers in epoch e
43   if there is a Byzantine proposer in the last four slots then
44     return 1
45   return 0
```

Figure 12: The workflow of a Byzantine validator v_k.

Proof. We consider the attestations from honest validators in epoch e while the attack is being launched. The attestations from honest validators in the first period of epoch e are included in the branch c^{e}_{hon}. The attestations from honest validators in the last four slots are included in the branch c^{e+1}_{hon}. The rest attestations from honest validators are included in the branch c^{e}_{adv} and c^{e+1}_{adv}. We thus consider the following three cases.

(1) The branch c^{e}_{hon} in the first 13 slots is pruned in epoch e. Attestations from the honest attestations in the first 13 slots are thus discarded.

(2) The branch c^{e+1}_{hon} in the last four slots is pruned in epoch $e+1$. Block b_j can include attestations with a number roughly equal to the number of attestations in most two slots. Therefore, the attestations from honest attestations in the last two slots are discarded.

(3) The branches c^{e}_{adv} and c^{e+1}_{adv} are not pruned, and the attestations from the honest validators in c^{e}_{adv} and c^{e+1}_{adv} are finalized on-chain.

Thus, the fraction of discarded attestations from honest validators in total attestations is $R_{hon} = (13+2)/32$. The expected incentives of an honest validator is then I_{hon} is

$$(((1-R_{hon})\frac{N-f}{N}+\frac{f}{N})(1-R_{hon})W_r - R_{hon}W_p)\times I_{base}.$$

Let $f = 0.33N$ and $W_r = 27W_p/20$, we have the incentive $I_{hon} \approx -0.0382 \times W_r I_{base} < 0$, where $W_r I_{base}$ is the fair share of an honest validator. □

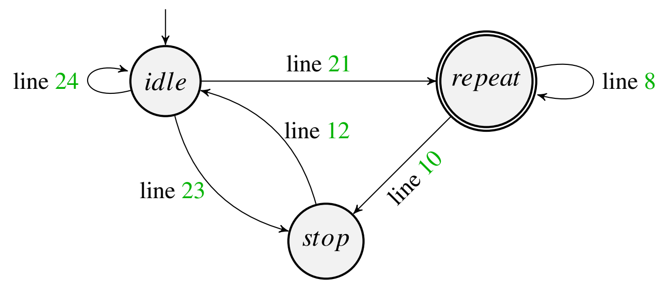


Figure 13: The status of the attack. Line numbers refer to those in Figure. 12.

Theorem 2. *If the adversary controls 33% of the total stake, our modified attack makes the incentives of an honest validator decrease to 28.9% of its fair share.*

Proof. The incentive of an honest validator in each epoch can be modeled as a discrete-time Markov chain. In particular, there are three states of the attack: the *idle* status (staircase attack is not launched), the *repeat* status (staircase attack is launched and being repeated in every epoch), and the *stop* state (staircase attack can not be repeated and conduct a one-time attack to penalize all honest validators). In each state, the incentives of honest validators are $W_r I_{base}$, I_{hon}, and $-W_p I_{base}$, respectively. Meanwhile, the transition matrix is shown as follows:

$$\mathbf{P} = \begin{pmatrix} 2/3 & P_{succ}/3 & (1-P_{succ})/3 \\ 0 & P_{succ} & 1-P_{succ} \\ 1 & 0 & 0 \end{pmatrix}.$$

By applying the matrix to a n-epoch transition, we have matrix $\mathbf{P^{(n)}} = \mathbf{P^n}$. The $\mathbf{P^{(n)}}$ becomes stable after $n = 27$, we have

$$\mathbf{P^{(n)}} \approx \begin{pmatrix} 0.372 & 0.504 & 0.124 \\ 0.372 & 0.504 & 0.124 \\ 0.372 & 0.504 & 0.124 \end{pmatrix},$$

for $n \geq 27$. Thus, the incentive of an honest validator is equal to any entry in the matrix $\mathbf{P^{(n)}}$ multiplied by $(W_r I_{base}, I_{hon}, -W_p I_{base})^{\mathrm{T}}$, which is approximately $0.289 \times W_r I_{base}$.

$\square$