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

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 \( \frac{1}{3} \) 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 \( \text{MAX\_ATTESTATIONS} \), 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 (850,000 validators and \( \text{MAX\_ATTESTATIONS} = 128 \)), 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 [43], 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 [11], a Byzantine fault-tolerant (BFT) protocol that tolerates Byzantine failures (i.e., arbitrary failures). Different from conventional BFT protocols [12, 14–16, 22, 23, 46, 47] 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, where stake, in general, refers to the account balance of the validators.

The PoS protocol of Ethereum assumes a partially synchronous network [17], where there exists an unknown upper bound for message processing and transmission. The protocol is a combination of Casper friendly finality gadget (FFG) [10] and a variant of the GHOST fork-choice rule [38] 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 (so the system is safe).

Almost all PoS protocols [11, 21, 25] 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 [34]. This is in sharp contrast to the Nakamoto consensus. While some academic efforts have been made to study PoS in the sleepy model [4, 29], 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\(^\text{\textsuperscript{3}}\). The incentive mechanism has been very successful in practice. According to the report pro-

\(^{\text{\textsuperscript{3}}}\)Note that penalties are different from slashing conditions [10]. Slashing conditions aim to penalize malicious behaviors such as equivocation, e.g., a validator votes for blocks from two conflicting branches. In contrast, penalties discourage inactive participation where validators do not vote.
vided by rated.network, the participation rate of Ethereum has reached 99.6%\(^2\). 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.** We present for the first time an attack on the incentive mechanism of Ethereum PoS, and we call it staircase attack\(^2\). Our work focuses on the rewards and penalties for attestations only, so we use attestation incentives to denote the incentive mechanism we study 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. As a result, 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 \cite{9}: the fraction of the stake controlled by the adversary may continue to increase, posing safety threats. However, \cite{9} 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 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.

**Attack feasibility and insights.** Ethereum has over 850,000 validators as of Oct 2023. 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 850,000 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 the largest mining pool today already controls 32.66\% of the total stake\(^6\), our attack can cause discouragement to honest validators.

---


\(^2\)We call it staircase attack because the branches the adversary constructs in the attack look like a staircase (see Figure. 8).

---

**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 modified version will become effective in the next update of the system. 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 \cite{11}. However, the attacks on the incentive mechanism may make the PoS protocol violate its own assumption (i.e., the fraction of stake controlled by the adversary) and 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, in PoS protocols like Gasper used in Ethereum, 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) is not an easy engineering decision and should be set up carefully.

**Our contributions.** Our paper makes 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, both chains can not be finalized and the system suffers from liveness issues [29,37]. Ethereum fixed the balancing attack [8] using a proposal boosting mechanism [36]. In short, proposal boosting is a "temporary" weight assigned to the block proposed and received in the current slot. As the two branches do not have the same weight, 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 [30]. In all the balancing attacks we are aware of, 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. As a result, the canonical chain jumps from one chain to another and neither branch can not be finalized, causing a liveness issue [28,35]. Ethereum’s upgrade in March 2023 fixed all known bouncing attacks [20]. Our attacks are designed based on the latest specifications.

**Reorg Attack.** In reorg attacks [32,37], the adversary reorganizes the chain to increase the fraction of the blocks from Byzantine validators on the canonical chain to gain more profit [24,44,49], decrease the chain quality [33] and lower the performance of the system. An interesting fact is that the reorg attacks in [32,37] are fixed by proposal boosting mechanism [8]. It was later mentioned informally that the proposal boosting mechanism does not fully address the issue and a revised attack called sandwich attack is proposed [13].

In contrast, to the best of our knowledge, we present the first work that attacks the incentive mechanism by making honest validators suffer from penalties. Additionally, the adversary does not suffer from the slashing conditions.

**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) [7,25,26]. The goal of nothing-at-stake attacks is usually the liveness and the performance of the system. The avalanche attack is an example to PoS GHOST [30]. In the attack, it was shown that by deliberately 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 revives the blocks proposed previously and rewrites the history of the blockchain. Many protocols are designed to prevent the attack [2,3,25,42,45]. The Casper protocol [10] 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) [27]) to select block proposers. The way the randomness is generated usually depends on the blocks in the canonical chain [41]. In practice, the roles of the nodes might be learned in advance. Accordingly, a validator eligible to propose a block can manipulate the value of the randomness (and the validators eligible to propose subsequently) [1,7]. The goal is to improve the adversary’s chances of being selected as a block proposer (to
Accordingly, let $f$ be the number of Byzantine validators, we assume $f < N/3$. We believe this assumption is reasonable as it allows the honest validators to collectively hold more than the fair share. Selfish mining attack was later found to be feasible in some PoS protocol as well [5, 7, 19, 31]. To the best of our knowledge, selfish mining attack has not been identified in Ethereum.

### 3 Review of Ethereum Proof-of-Stake

In this section, we review the Proof-of-Stake (PoS) protocol used by Ethereum 2.0 [11, 35]. Our notations largely follow the specifications by Ethereum foundation [39].

#### 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, and it 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 limited to 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 [11]. 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 it allows the honest validators to collectively hold more than the fair share.

#### 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 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 tree-based view about the blocks proposed by the validators in the form of a block tree $T$. Additionally, a checkpoint is a special type of block and there is one checkpoint block per epoch. 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. We use $ep(b) \leftarrow \lfloor \frac{b}{32} \rfloor$, 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 [11]. HLMD GHOST is a fork-choice rule that recursively chooses the root of 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

---


---

Table 1: Comparison of known attacks against Ethereum PoS. *Slashed* denotes whether at least one Byzantine validator will be slashed. †Synchrony with adversarial delay model assumes that the adversary manipulates the network delay, which is stronger than the partially synchronous model. *The issue has been reported to the community.

<table>
<thead>
<tr>
<th>attack type</th>
<th>scheme</th>
<th>timing assumption</th>
<th>target</th>
<th>slashed*</th>
<th>experimentally confirmed</th>
<th>issue fixed</th>
</tr>
</thead>
</table>
| balancing attacks | Neu, Tas, and Tse. [29]  
Schwarz-Schilling et al. [37]  
Neu, Tas, and Tse. [30] | synchrony with adversarial delay†  
synchrony with adversarial delay† | liveness  
liveness  
liveness | ×  
×  
×  | ×  
×  
×  | ×  
×  
×  |
| bouncing attacks | Nakamura [28]  
Pavloff et al. [35] | partially synchrony  | liveness  | ×  | ×  | ×  |
| reorg attacks | Neuder et al. [32]  
Schwarz-Schilling et al. [37]  
D’Amato et al. [13] | synchrony  | chain quality | ×  | ×  | ×  |
| staircase attack | Our work | synchrony | incentive | ×  | ×  | ×* |

---

gain extra profits or decrease the chain quality).

Selfish mining. Selfish mining is an attack first known in Proof-of-Work (PoW) [18]. In selfish mining, a mining pool (with a number of validators) may collude [48] and gain more revenue than the fair share. Selfish mining attack was later found to be feasible in some PoS protocol as well [5, 7, 19, 31]. To the best of our knowledge, selfish mining attack has not been identified in Ethereum.

---

Ethereum PoS assumes that the network is partially synchronous [17], i.e., there exists an unknown upper bound $\Delta$ for message transmission and processing delay. However, our attack can be launched even if the network is synchronous, i.e., the value of $\Delta$ is known to every validator.

---

# Table 1: Comparison of known attacks against Ethereum PoS

*Slashed* denotes whether at least one Byzantine validator will be slashed. †Synchrony with adversarial delay model assumes that the adversary manipulates the network delay, which is stronger than the partially synchronous model. *The issue has been reported to the community.*
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\(^9\)). 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 in each slot of an epoch and verify the roles of other validators one epoch in advance. 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 highest checkpoint block in \( v_i \)'s canonical chain. In practice, hashes of the checkpoints are included in the attestation. In this paper, we use the checkpoints instead 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 \( T \), 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 the highest checkpoint block 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 as \( V(c) \) is updated throughout the protocol.

**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 attester 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 \( T \), the canonical chain is defined as follows (an example is provided in Figure. 1):

1. Prune any branch \( c \in T \) such that \( J(c) \) is lower than \( LJ \).
2. Recursively calculate the sum of weights of each subbranch. The branch with the largest accumulative weight is considered the heaviest subtree and becomes the canonical chain.

![Figure 1: Example of the fork choice rule HLMD GHOST.](https://example.com/figure1.png)

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 \( T \). In slot \( t \), if \( v_i \) is the proposer, it broadcasts a (PROPOSE, \( t \), \( v_i \), \( H(\text{head}) \), newatts, txs) message to all validators (lines 2-10). Here, \( head \) is the output of HLMD GHOST and \( H(\text{head}) \) is the hash of \( head \), serving as a hash pointer of the parent block. The newatts consists of a set of attestations \( v_i \) has received. Given the canonical chain \( c \), an 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., 4 seconds). Then, \( v_i \) prepares an attestation message (ATTEST, \( t \), \( v_i \), \( H(\text{head}) \), \( LJ \), \( C(c) \)) and sends it to all validators, where \( LJ \) is the source, and the highest checkpoint \( C(c) \) is the target (lines 11-16).

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

---

\(^9\)RANDAO: [https://github.com/randao/randao](https://github.com/randao/randao)
The incentive mechanism consists of rewards that encourage active participation and penalties that discourage inactive behavior [11, 39]. Validators with attestations finalized on-chain will get rewards, and validators who do not have their attestations finalized on-chain pay penalties. We then extend the warm-up attack and show that optimizations resulting from the warm-up attack can make many validators lose incentives. In contrast, Byzantine validators do not suffer from any penalties. The only cost of the incentives Byzantine validators can obtain during the attack are slightly lower than their fair share.

### 3.4 The Attestation Incentive Mechanism

The incentive mechanism consists of rewards that encourage active participation and penalties that discourage inactive behavior [11, 39]. Validators with attestations finalized on-chain will get rewards, and validators who do not have their attestations finalized on-chain pay penalties. Note that penalties differ from the slashing conditions [11]: 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 other validators. Both rewards and penalties are then calculated based on $I_{base}$. The reward is $R \times W_i I_{base}$. 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. $W_i$ is the weight determined by the source, target, and block in the attestation. Additionally, the penalty is $W_i I_{base}$, where $W_i$ is the weight determined by the source and target of the attestation $v_i$ is supposed to send.

### 4 An Attack on the Incentive Mechanism

In this section, we present stadium 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 make $(N - f)/32$ honest validators suffer from penalties. We then extend the warm-up attack and show that if Byzantine validators (owning 29.6% of the total stake) collude, half of the honest validators will be penalized. 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.

![Figure 2: The Ethereum PoS Protocol](image)

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 even if the network is synchronous, $LJ$ might still be changed in the middle of an epoch. Indeed, honest validators may not necessarily have received the blocks and attestations from Byzantine validators. Accord-
Table 1: Legend and Meaning

<table>
<thead>
<tr>
<th>Legend</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>attestation from honest validators with source lower than the checkpoint block of the previous epoch</td>
</tr>
<tr>
<td></td>
<td>attestation from honest validators with source as the checkpoint block of the previous epoch</td>
</tr>
<tr>
<td></td>
<td>attestation from Byzantine validators with source as the checkpoint block of the previous epoch</td>
</tr>
<tr>
<td></td>
<td>discarded attestations</td>
</tr>
<tr>
<td></td>
<td>a block proposed by an honest validator that includes a block proposed by an honest validator that includes</td>
</tr>
<tr>
<td></td>
<td>a block proposed by a Byzantine validator that includes a pruned block</td>
</tr>
<tr>
<td></td>
<td>a block that will be received at time $t$</td>
</tr>
<tr>
<td></td>
<td>a withheld and released block</td>
</tr>
<tr>
<td></td>
<td>attestations included in the block</td>
</tr>
</tbody>
</table>

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

Attaching to the PoS protocol, a validator $v_i$ updates $LJ$ in two cases. First, $v_i$ receives the first block in some epoch $e$ such that $J(c)$ is higher than its $LJ$ (lines 20-22 in Figure. 2). Second, 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 $e$) such that $J(c)$ is higher than its $LJ$ (lines 29-31 in Figure. 2).

We illustrate three examples in Figure. 4 to show three possible cases of a validator when it enters an epoch $e$. Before epoch $e$, $LJ$ is set as $cp$, and attestations by all honest validators in epoch $e$ have target set as block 0.

- **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 by all honest validators are received, the condition $V(c)[0] > 2N/3$ is satisfied so $J(c)$ 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 updated as block 0 (line 21 in Figure. 2).

- **Case 2:** $LJ$ is not updated in an epoch. As shown in Figure. 4b, $v_2$ receives block 0 to block 31, but no blocks have been received in epoch $e + 1$. In this case, the $LJ$ is not updated.

- **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. Different from the case for validator $v_1$, $v_3$ has not received $2N/3$ attestations and $J(c)$ is still $cp$, where $c$ is the chain led by block 33. Later in slot $t$ of epoch $e + 1$, $v_3$ receives blocks 17 to 31, and the 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 30 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 HLMG 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 $v_3$ in Figure. 5 after $v_3$ receives blocks 17-31. In this example, before $v_3$ receives blocks 17-31, branch $c_2$ is 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).

Our attack essentially utilizes this fact to force honest validators to prune a branch led by a block from an honest val-
idator, so the attestations on the pruned branch are discarded, e.g., the attestations included in block 33 in Figure 5. 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 $\frac{(N-f)}{32}$ 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 $\frac{(N-f)}{32}$ 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 also use $cp_{e-1}$ to denote the checkpoint block proposed in the first slot of epoch $e-1$ and assume that all validators receive $cp_{e-1}$ 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 the head of the canonical chain $c$ and withholds $b_i$. According to our assumption, the canonical chain $c$ consists of blocks from all proposers in epoch $e-1$. As none of the validators has received any block in epoch $e$ yet, case 2 mentioned in §4.1 is satisfied: LJ is not updated and is still $cp_{e-2}$ for all honest validators. Thus, attestors of slot $t$ (i.e., attestors that belong to the validator set in slot $t$) will then send attestations using $cp_{e-2}$ as source to all validators.

2. (Figure 6b) At the end of slot $t$, validator $v_i$ sends $b_i$ to all validators. Here, the epoch number of block $b_i$ is greater than $b_i$’s parent block. Additionally, the chain $c$ led by $b_i$’s parent block consists of the attestations from all validators in epoch $e-1$, so $J(c)$ is already set as $cp_{e-1}$. As the epoch number of $J(c) = cp_{e-1}$ is greater than $LJ = cp_{e-2}$, case 1 in §4.1 is satisfied and all honest validators update their LJ to $cp_{e-1}$ (lines 20-22 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 source of $att$ is $cp_{e-2}$. However, the LJ of all honest validators is $cp_{e-1}$. When any proposer creates a new block, $att$, it 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 $\frac{(N-f)}{32}$.

### 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. The goal is that half of the honest validators will be penalized in each epoch after the opportune epoch $e$. Note that in the warm-up attack, only one Byzantine validator launches the attack. In our staircase attack, all Byzantine validators withhold their attestations and blocks, trying to make 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 \( v_t \) of the first slot 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 \( e \), \( v_t \) replays the warm-up attack: \( v_t \) withholds its block \( b_t \) and releases \( b_t \) at the end of slot \( t \), after which all honest validators update their \( LJ \) as \( cp_{e-1} \). In epoch \( e \), the Byzantine validators have two strategies. First, all Byzantine validators in epoch \( e \) withhold their attestations (i.e., \( A_1 \)), 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 \(^{1} \)) 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 \( b_t \) as target is still \( 31(N-f)/32 \). Namely, as we assume \( f = N/3 \), the number of attestations is \( \frac{31}{32} \times \frac{2N}{3} < 2N/3 \), 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. The adversary’s strategy in the first period is to create two forks of the chain, branch \( c_{hon} \) seen by honest validators and branch \( c_{adv} \) withheld by Byzantine validators. In the first period, Byzantine validators prepare attestations with \( b_t \) 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 \( b_t \) 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 \( b_t \) to \( b_j \) consists of the attestations (with source as \( cp_{e-1} \) and target as \( b_t \)) from all attestors (Byzantine and honest) between slot \( t \) and \( t' \). Therefore, we know that if \( \frac{t'}{32} - \frac{t+1}{32} > \frac{f}{3} \), the condition \( V(c_{adv})[b_t] > 2N/3 \) will be satisfied. We show in Lemma 1 that this happens with a probability of 98.84% if the adversary controls \( f = N/3 \) validators.

As \( J(c_{adv}) \) is updated to \( b_t \) after \( c_{adv} \) is released, it is not difficult to see that \( LJ \) will be updated to \( b_t \) 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 \( b_t \)) 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 and attestation in \( A_1 \) will be filtered. Thus, the corresponding attestors will be penalized.

As discussed in step (1), as \( J(c_{hon}) \) will never be updated to \( b_t \) in our attack, the branch \( c_{hon} \) (up to \( b_t \)) will be pruned anyway. Therefore, 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 \( v_j \) 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, i.e., \( \lceil \frac{32}{3} \rceil = 11 \). We now calculate the probability that the

\(^{1}\) 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 $v_j$ 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 $c_{p_{e-1}}$.

(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 $c_{p_{e-1}}$ during this period. The attestations $A_1$ included in $c_{hon}$ have $c_{p_{e-1}}$ 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 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 $c_{p_{e-2}}$ and the (checkpoint) block proposed in the first slot of epoch $e - 1$ is $c_{p_{e-1}}$.

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

$$P_{succ} = 1 - \left( \frac{N - f}{N} \right)^{11} \approx 98.84\%.$$
following the discussion above. Therefore, the size of $A_1$ must
no more than than $N - |A_2 \cup A_3| = N/3$. \hfill \Box

**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 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 re-
wards and attestors that create attestations in $A_1$ will suffer
from penalties. Therefore, 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
$R \times W_r \times I_{base} = \frac{2}{3}W_r \times I_{base}$ and the penalty of each validator is
$W_p \times I_{base}$. Therefore, the incentives received by all honest val-
dicators 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:

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

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

$$I_{hon} = \frac{4}{15}N - \frac{9}{10}f \cdot W_r \times I_{base} \leq 0.$$ 

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

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 $I_{adv} = \frac{1}{3}W_r \times I_{base}$ in each epoch,
about 33% lower than the fair share.

**5 Implementation and Evaluation**

**Implementation.** We implement our attack using Prysm\(^1\),
one of the most widely adopted Ethereum 2.0 beacon chain
implementations written in Golang. 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\(^2\) and will open source them soon.

**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 $f = 3296$ (i.e., $f \approx 8N/27$), $f = 315$, 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

\(^1\)Prysm: https://github.com/prysmaticlabs/prysm
\(^2\)Scripts of our attacks and logs of the experiments: https://anonymous.4open.science/r/Staircase-Attack/
The Workflow of a Byzantine Validator $v_k$

**Global parameters:** slot counter $t$

**Local parameters:** last justified checkpoint $LJ$

---

01. **Upon slot $t$ do**
02. obtain a set $I$ of Byzantine proposers in the current epoch, ordered by the slot numbers the validators are allocated for
03. $\triangleright$ prysm/beacon-chain/rpc/prysm/v1alpha1/validator/proposer.go
04. $\triangleright$ prysm/validator/client/proposer.go
05. as the proposer for slot $t$
06. follow lines 3-9 specified in Figure. 2
07. if $t$ is the first slot in an epoch then
08. send $b$ to all validators at the end of slot $t$
09. elseif $v_k$ is the last Byzantine proposer in $I$ then
10. send $b$ to all Byzantine validators at once and to all honest validators at the beginning of the 17th slot in the next epoch
11. else send $b$ to all validators at once
12. $\triangleright$ prysm/validator/client/attest.go
13. as an attestor for slot $t$
14. follow lines 12-15 specified in Figure. 2
15. 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.

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 $f = 315$ 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.

---

6 Attack Feasibility and Analysis

In this section, we analyze the feasibility of our attacks in the current Ethereum system beyond 1,000 validators, assuming $f = N/3$. Indeed, Ethereum has more than 850,000 validators as of today,

---

Data source (accessed in Oct 2023): [https://www.beaconcha.in/charts/validators](https://www.beaconcha.in/charts/validators)

---
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. Our major modification is that attestations from Byzantine validators do not have to be included in one block. 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 4 slots are withheld. Among them, attestations in 2 slots are included in the block proposed by the last Byzantine proposer (mainly because a block may include attestations from attestors in at most 2 slots, see Appendix B for more information). Therefore, we require the slot of the last Byzantine proposer to be one of the last 4 slots. Under the assumption of $f = 0.33N$, the probability of repeating the attack is:

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

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 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 (850,000 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.
References


A Rewards and Penalties in Ethereum

In this section, we provide more details on attestation rewards and penalties [39]. 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 \(^\text{➀}\), the penalty weight of a validator $W_p$ is set to 40.

\(^\text{➀}\) The value can be found at https://github.com/ethereum/consensus-specs/blob/dev/specs/altair/beacon-chain.md#incentivization-weights
As mentioned in §4, the attestations in $A_2$ are finalized on-chain, so the reward weights of the corresponding attestors $W_i$ 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 $W_i \leq 27W_p/20$.

## B Modified Staircase Attack

In this section, we provide a 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 $\text{judge}$ (lines 41-45). Briefly speaking, given an epoch $e$, the $\text{judge}$ function returns $1$ if the attack can be repeated in the next epoch and $0$ otherwise.

**Lemma 4.** 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.

**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 $c_{hon}^e$. The attestations from honest validators in the last four slots are included in the branch $c_{hon}^{e+1}$. The rest attestations from honest validators are included in the branch $c_{adv}^e$, and $c_{adv}^{e+1}$. We thus consider the following three cases.

1. The branch $c_{hon}^e$ 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 $c_{adv}^{e+1}$ in the last 4 slots is pruned in epoch $e+1$. The attestations from the honest attestations in the last 4 slots are thus discarded. Block $b_j$ can include attestations with a number roughly equal to the number of attestations in most 2 slots. This is because a block can include attestations from 128 committees (i.e., MAXATT = 128) and there are 64 committees (i.e., MAXCOM = 64) in a slot. Therefore, the attestations from honest attestations in the last 2 slots are discarded.
3. The branches $c_{adv}^e$ and $c_{adv}^{e+1}$ are not pruned, and the attestations from the honest validators in $c_{adv}^e$ and $c_{adv}^{e+1}$ are
Thus, the fraction of discarded attestations from honest validators in total attestations is \( R_{\text{hon}} = (13 + 2)/32 \). The expected incentives of an honest validator is then \( I_{\text{hon}} \) is

\[
((1 - R_{\text{hon}})N - f/N + f/N)(1 - R_{\text{hon}})W_r - R_{\text{hon}}W_p) \times I_{\text{base}}.
\]

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

**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 \( W_r I_{\text{base}}, I_{\text{hon}}, \) and \(-W_p I_{\text{base}}\), respectively. Meanwhile, the transition matrix is shown as follows:

\[
P = \begin{pmatrix}
2/3 & P_{\text{suc}}/3 & (1 - P_{\text{suc}})/3 \\
0 & P_{\text{suc}} & 1 - P_{\text{suc}} \\
1 & 0 & 0
\end{pmatrix}.
\]

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

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

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