USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Does Finality Gadget Finalize Your Block? A Case Study of Binance Consensus

Rujia Li, *Tsinghua University;* Jingyuan Ding, *Shandong University;* Qin Wang, *CSIRO Data61;* Keting Jia, *Tsinghua University;* Haibin Zhang, *Yangtze Delta Region Institute of Tsinghua University;* Sisi Duan, *Tsinghua University*

https://www.usenix.org/conference/usenixsecurity25/presentation/li-rujia

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

Does Finality Gadget Finalize Your Block? A Case Study of Binance Consensus

Rujia Li
Tsinghua University

Jingyuan Ding
Shandong University

Qin Wang
CSIRO Data61

Keting Jia †
Tsinghua University

Haibin Zhang⋆
Yangtze Delta Region Institute of Tsinghua University

Sisi Duan⋆†
Tsinghua University

Abstract

This paper studies the consensus mechanism of BNB smart chain (BSC) – a top-ranked blockchain platform developed by Binance. Since mid 2023, BSC has integrated a *fast finality* (FF) mechanism into its system. The FF mechanism is borrowed from the friendly finality gadget (FFG) by Ethereum Casper. The idea is to allow validators to vote for blocks and then agree on their order. Such an approach shares some similarities with the consensus mechanism in Byzantine fault-tolerant (BFT) protocols (e.g., PBFT and HotStuff). BSC claims that its FF mechanism can finalize blocks in $O(1)$ time, simultaneously reducing latency and improving stability.

In this paper, we demonstrate that the FF mechanism of BSC is susceptible to attacks. In particular, we provide three different attacks, showing BSC fails to finalize blocks in constant time and may even simply fail to achieve liveness. We validate our results via extensive experimental analysis and provide mitigation solutions.

1 Introduction

The notion of the *friendly finality gadget* (FFG) was first introduced in Casper [1]. FFG was designed as an overlay for a blockchain system to *finalize* blocks. The idea is borrowed from conventional Byzantine fault-tolerant (BFT) protocols [2–4]. Namely, in a system with a known number of N validators, among which at most f are Byzantine, FFG allows validators to *vote* for blocks. After $N-f$ validators send matching votes (also called attestations) for some block b, b is *justified*. After the subsequent block of b is justified, b is *finalized*. Once a block is finalized, its order will never be reversed. In this way, the system achieves *safety* (no double-spending). Besides the fact that FFG can finalize blocks, it can also be used to lower the latency of the underlying blockchain. Thus, many systems adopt FFG or its variants. Most notable examples include Ethereum 2.0 [1] (with a market capitalization of $405.38 billion) and Binance (with a market capitalization of $102.89 billion)[1].

This paper studies BNB Smart Chain (BSC) by Binance. As of writing, BSC supports over 1.499 million users and ranks sixth globally among all cryptocurrencies. Historically, BSC used a variant of the Proof-of-Authority (PoA) protocol called Clique [5, 6] as its core consensus mechanism. With 21 validators, it took about 45 seconds to finalize a block [7]. Due to the lack of finality, in 2023, BSC upgraded its consensus mechanism. It now integrates a *fast finality* (FF) mechanism with its system. FF is a variant of FFG. It still has the *justify-finalize* feature but adds additional constraints to work with its system. By adopting FF, BSC has a clear notion of finality, and the *liveness* property guarantees that blocks continue to be finalized in the system. Meanwhile, the average time it now takes to finalize a block is reduced to only 7.5 seconds. BSC mentions in its documentation that "*the finality of a block can be achieved within two blocks in most cases, this is expected to reduce the chance of chain re-organization and stabilize the block producing further.*" [8].

A case study on FF of BSC. We study the FF mechanism of BSC, both theoretically and experimentally. We present three different attacks on FF and show that FF may completely fail, echoing the conclusion that designing and implementing a consensus mechanism correctly is not easy [9]. Our results are summarized below and also in Table 1. All of our attacks are "risk-free", in the sense that no Byzantine validators will suffer from any penalties or get caught.

Attack-I: CLSO attack. Our first attack exploits an issue for protocols (including BSC) in the *chained* and *leader-speak-only-once* (CLSO) model [10]. In this model, each leader (i.e., a specific validator) proposes one block and is immediately rotated. Under our attack, only one block is expected to be finalized in $O(N)$ time. Our attack extends

⋆ Corresponding authors. Email: bchainzhang@aliyun.com and duansisi@tsinghua.edu.cn

† Sisi Duan and Keting Jia are also with Zhongguancun Laboratory, State Key Laboratory of Cryptography and Digital Economy Security, and Shandong Institute of Blockchains.

[1] Data source for the market capitalization of Ethereum and BNB Smart Chain (by January 2025): https://coinmarketcap.com/currencies.

Table 1: Summary of our attacks on the fast finality (FF) mechanism. An ideal FF mechanism finalizes a block in $O(1)$ time (slots).

Our attacks	Impact of the attacks	Worst case	Experimental results
Attack-I (§5.1): CLSO attack	Slowing down the finality of blocks; honest validators receive attestation rewards lower than their fair share.	When $N = 3f + 1$, only one block can be finalized among $O(N)$ proposed blocks.	Three blocks are finalized every N blocks.
Attack-II (§5.2): Split voting attack	Liveness failure; honest validators receive attestation rewards lower than their fair share.	No blocks can be finalized via FF.	No blocks can be finalized via FF.
Attack-III (§5.3): Synchronization attack	Liveness failure; honest validators receive attestation rewards lower than their fair share.	No blocks can be finalized via FF.	No blocks can be finalized for 390 seconds (130 slots).

the findings for Byzantine fault-tolerant (BFT) protocols [3] in the CLSO model to PoA and is the first such attack for PoA protocols. Moreover, such an issue was previously known for protocols with only a single leader. Surprisingly and somewhat counterintuitively, we show that such an attack strategy works for protocols with multiple leaders (i.e., BSC).

Attack-II: Split voting attack. Our second attack exploits an issue with the first-in-first-vote (FIFV) feature of BSC. BSC requires each validator to vote for the first received block in every *slot* (i.e., the minimum time unit of BSC). We show that by slightly manipulating the order of messages, the FF mechanism fails completely, and the system loses liveness.

Attack-III: Synchronization attack. Finally, we present a simple yet effective approach that uncovers an issue with the design and implementation of the *synchronization* module. The synchronization module is designed to allow fallback validators to catch up with other validators. Under our attack, some honest validators keep synchronizing with other validators and fail to vote. Hence, blocks cannot be finalized during the synchronization period, which significantly hurts liveness. In our experimental analysis, no blocks are finalized for at least 390 seconds (130 slots).

Responsible disclosure. We have contacted the Binance team to disclose the attacks via Github and public channels (i.e., Medium). The attacks have been confirmed and acknowledged by the team.

Our contributions. We make the following contributions:

- We provide a formal treatment of the BNB smart chain and its fast finality mechanism (§3). We present three novel attacks against the fast finality mechanism and show that our attacks can successfully impede liveness (§4-§6).
- By conducting extensive experimentation using the BSC implementation (version 1.4.16), we validate the practicality of our attacks (§7).
- Our case study confirms the challenges in properly designing and implementing the finality gadget on top of an existing consensus mechanism, where a similar observation has been made by previous work [11–14]. Moreover, we discuss some approaches that can mitigate the attacks. (§8).

2 Model

System model and network assumption. Binance Smart Chain (BSC) operates with a set of validators $\mathcal{V} = \{v_1, v_2, \cdots\}$. BSC allows validators to join and leave, so the total number of validators may change over time. However, within one day, the number of validators is fixed. Hence, without loss of generality, we may assume N to be the total number of validators in the system. Among N validators, at most f are Byzantine validators that can arbitrarily deviate from the protocol specifications. Validators that are not Byzantine are honest. BSC assumes that $N > 3f$. Without loss of generality, we assume $N = 3f + 1$ to simplify our description. Such a simplification follows the convention in the literature [2, 3], as we only need to slightly modify some parameters for the $N > 3f$ case.

BSC does not explicitly define its timing assumptions. Since our attacks can be launched in any network condition, we simply assume that the protocol proceeds in a synchronous network, where there exists a known upper bound Δ on message processing and propagation delays.

Validators are connected via a peer-to-peer (P2P) network, also known as a gossip network.

Cryptographic assumption. BSC uses standard hash functions and digital signatures. The hash function is collision-resistant, while digital signatures are unforgeable. We use $\mathsf{Hash}()$ to denote the hash function, and we may ignore digi-

tal signatures in this paper. We say a vote is valid if it includes a valid digital signature from the validator that sends the vote.

Security goals. The consensus mechanism of BSC is specified by the *finalize* event, and each validator finalizes blocks via the event. BSC claims that its consensus mechanism satisfies the following security properties.

- **Safety:** If an honest validator finalizes block b before block b', another honest validator will never finalizes b' before b.
- **Liveness:** The finalized chain eventually grows for all honest validators.

We emphasize that the above two security properties are slightly different from those described in the BSC documentations [8, 15]. BSC follows the notions of Gasper [16] and uses *accountable safety* and *plausible liveness*. In the language of Gasper, the term *accountable* emphasizes the fact that some validators will be caught if they violate the specification of the protocol, e.g., voting for two conflicting chains. As BSC does not have such a feature, we refine their definitions and follow the convention in the literature. Note that the liveness property above is weaker than conventional BFT [2, 3, 17], i.e., a transaction submitted by an honest party will eventually be finalized.

3 The BSC Consensus Protocol

BSC names its consensus mechanism as *Proof-of-Staked-Authority* (PoSA). PoSA combines the Delegated Proof-of-Stake (DPoS) mechanism [18] and the Proof-of-Authority (PoA) [19] mechanism. Briefly speaking, BSC first uses a DPoS-like mechanism to select a set of validators. The selected validators then run a PoA consensus mechanism. Earlier versions of BSC adopt a variant of the Clique protocol [5, 20] proposed by OpenEthereum [21]. Clique only has a *propose* function where $O(N)$ validators propose blocks at a time. The protocol runs in $O(N)$ time, i.e., it takes up to $O(N)$ communication rounds to finalize a block, Specifically, if a block b is extended by $N-f$ blocks, the order of b will not be reversed. After the BEP-126 update, PoSA integrates a fast finality mechanism into its protocol in the hope that the time complexity can be reduced to $O(1)$. Namely, in the optimal case, a block is expected to be finalized after two communication rounds (slots).

In this section, we review the consensus mechanism of BSC. Our study focuses on the improved PoA consensus mechanism (i.e., Clique plus fast finality). As BSC does not have a published paper that formally describes its consensus mechanism, we summarize it based on their documents [8, 15] and codebase. We try our best to use the same notation as the BSC documentation.

3.1 Terminology and Notation

Epochs and slots. PoSA divides time into *epochs* and *slots*. Each epoch consists of $S = 200$ slots. Each slot lasts three seconds. BSC implicitly assumes that each slot has a duration of 2Δ, so Δ is 1.5 seconds. Let t denote the slot number. We have $e \leftarrow \lfloor \frac{t}{S} \rfloor$.

Block, block tree, and canonical chain. We use b to denote a block. A block b includes the slot number, a difficulty value, a hash pointer to the parent block of b, a batch of transactions, and a set of attestations. Each validator maintains a block tree $\mathcal{T}$ to track its received blocks. The first block in the system is called the *genesis* block. Given a block b, the *height* of b is the total number of blocks on the branch led by b in the block tree. We use $h(b)$ to denote the height of b. If b and b' are on the same branch and $h(b) > h(b')$, we say that b extends b'. If $h(b) - h(b') = 1$, b' is the *parent block* of b. Two blocks are *conflicting* if neither b extends b' nor b' extends b. Additionally, we use $slot(b)$ to denote the slot number of b.

Based on $\mathcal{T}$, each validator selects one branch as its *canonical chain*. Each validator only proposes a new block that extends its canonical chain and votes for a block on its canonical chain. The canonical chain is selected via the *fork choice rule*.

Roles of validators. There are three roles in the system [22, 23]: *in-turn validator*, *backup validator*, and *non-turn validator*. In each slot, one validator is in-turn, at most ℓ validators are backup validators, and the rest are non-turn validators, where ℓ is set as $N - (\lfloor N/2 \rfloor + 1)$. Both in-turn and backup validators are eligible to propose blocks. The block by an in-turn validator is designed to have the highest priority among all the proposed blocks in the same slot. Meanwhile, the goal of backup validators is to propose blocks when in-turn validator fails to propose a block. Also, a non-turn validator does not propose a block.

The roles of in-turn and backup validators are selected in a round-robin manner. For example, in slot one, v_1 is the in-turn validator and v_2 to $v_{\ell+1}$ are backup validators; in slot two, v_2 is the in-turn validator and v_3 to $v_{\ell+2}$ are backup validators.

Each block has a *difficulty value*. A block proposed by the in-turn validator has a difficulty value of 2, and a block proposed by each backup validator has a difficulty value of 1.

Vote/attestation and votepool. A vote is also called an *attestation*. An attestation by v_i contains the identifier i, a *source* block sc, a *target* block tg, $h(sc)$, and $h(tg)$. Here, the source block is the last justified block of v_i and the target block is the last block in the canonical chain. The hashes of the blocks are included in the attestation. Each validator stores all the received attestations in its *votepool*.

Justification and finalization. A block b is *justified* if at least $N-f$ attestations for b are received.[2] If another block b' that directly extends b is justified, b and all the blocks on

[2] BSC uses the notion of $2N/3$ (instead of $\lceil \frac{N+f+1}{2} \rceil$) in their documentation. Strictly speaking, doing so is not accurate. BSC currently has 21 validators, it is a coincidence that $2N/3 = \lceil \frac{N+f+1}{2} \rceil$. We use $N-f$ in this paper where no issues will arise.

the branch led by b are *finalized*. The *last justified block* is the highest justified block of the canonical chain, denoted as LJ.

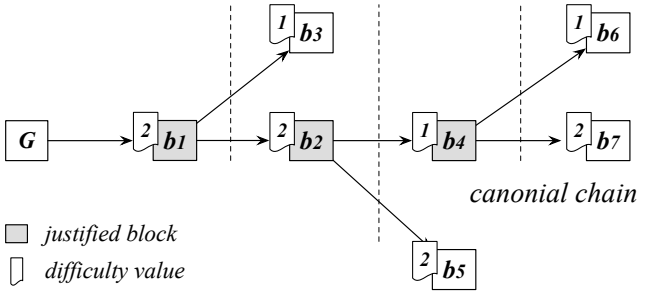


Figure 1: Illustration of the fork choice rule. Given the branch led by b_3 and the branch led by b_2, the height of the last justified block on the branch led by b_2 is higher. The branch led by b_3 is thus pruned. Similarly, the branch led by b_5 is pruned as the height of the last justified block b_5 is lower. For branches led by b_6 and b_7, the branch led by b_7 is the canonical chain, as the chain has a larger sum of difficulty values.

Fork choice rule. As mentioned previously, the fork choice rule takes input the block tree and outputs the head of the canonical chain. Given a block tree $\mathcal{T}$, the canonical chain is defined as follows (an example is shown in Figure 1): (i) Prune any branch $c \in \mathcal{T}$ such that $h(B) < h(LJ)$, where B is the head of branch c; (ii) Given any two branches $\{c_0, c_1\} \in \mathcal{T}$ such that c_0 and c_1 share the same LJ (including the case there LJ is empty), calculate the sum of difficulty values of each branch and prune the branch with a lower sum of difficulty values; (iii) If multiple branches have the same sums of difficulty values, break the tie arbitrarily.

3.2 Workflow

We present the workflow of the BSC consensus mechanism in Figure 3.

Block proposal. The rules for block proposals largely follow the Clique mechanism. We use Figure 2 to illustrate the workflow. In this example, v_1 is the in-turn validator, v_2 and v_3 are backup validators, and v_4 is non-turn validator. At time T_1, i.e., the beginning of slot t, v_1 immediately creates a block with a difficulty value of 2 (line 3-6 of Figure 3) and sends it to all validators (line 12). Meanwhile, v_2 and v_3 sleep for a random period of time. In the current BSC implementation, the sleep time is set as a discrete random delay (*defaultInitialBackOffTime+backOffTime*). *defaultInitialBackOffTime* is set to one second and *backOffTime* is a random delay in the range of $[0, N]$. After waking up, each backup proposer proposes a block if it has not received a block yet. In our example in Figure 2, at T_2, v_2 awakes, creates a block with a difficulty value of 1, and sends the block to all validators. At T_3, v_3 awakes. However, it has already received the block proposed by v_1. In this case, it does not create a block in this slot (line 3-10 of Figure 3).

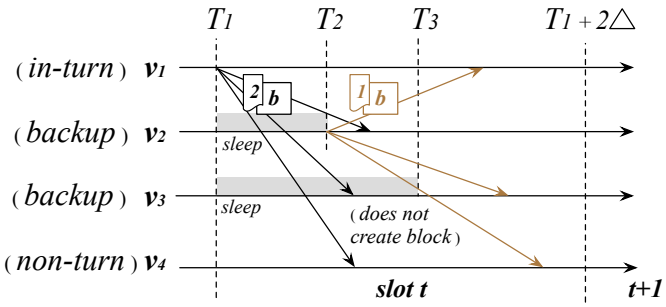


Figure 2: Illustration of block proposal workflow.

Block validation and point-to-point synchronization. Upon receiving block b from an in-turn or backup validator, validator v_i verifies whether b is valid, e.g., whether the transactions are well-formed and valid. Also, v_i compares b with its block tree. If the parent block of b is not included in its block tree, v_i starts a *synchronization* procedure with v_j. We call this period a *point-to-point synchronization period*. In particular, v_i requests for the chain led by b from v_j. After receiving the corresponding blocks, v_i can validate that the chain led by b indeed exists. After the synchronization completes, v_i then adds b (and the chain led by b) to its own block tree.

The fast finality (FF) mechanism. The concept of finality gadget was first proposed in Casper [1]. BSC uses a variant of the finality gadget called fast finality [8]. FF explicitly defines a vote message (also called an attestation) and allows validators to cast their votes on one of the chains. For any proposed block b, after $N - f$ attestations for b are received, b is *justified*. If block b' that extends b is justified, block b is *finalized*. Its correctness largely follows conventional BFT protocols [2, 3].

The voting procedure is triggered after the first block in a slot is received (line 16 of Figure 3). Namely, upon receiving any block b (from v_j) in a slot such that the following conditions hold, a validator v_i immediately casts its vote.

(i) v_i has received the blocks led by b in its block tree (line 17 of Figure 3);
(ii) b is not conflicting with *head*, where *head* is the output of v_i's canonical chain (line 19 of Figure 3);
(iii) T is no later than the beginning of slot $t+1$;

Among these conditions, (i) and (ii) ensure that v_i receives the chain led by the proposed block and then votes for the canonical chain. (iii) requires that each validator only votes for a block within the current slot.

If b is valid, v_i creates an attestation in the form of $\langle v_i, sc, tg, h(sc), h(tg) \rangle$ and sends the attestation to all validators, where sc is its LJ and $tg = b$ (line 21 of Figure 3).

BSC designs several mechanisms for FF. We summarize them as follows.

Mechanism-I (Vote after synchronization): If (i) is not satisfied, (ii)-(iii) are not triggered. Namely, if v_i is in the

The Consensus Protocol

Global parameters: Slot t, Validator set $\mathcal{V}$
Local parameters: Last justified block LJ, Block tree $\mathcal{T}$

01 Upon slot t **do**:
02 **if** v_i is *in-turn validator* **then**
03 Let *head* be the output of fork choice.
04 **if** there are more than $N-f$ attestations for $\mathsf{Hash}(head)$ from its votepool **then** set $atts$ as these attestations
05 Obtain a batch of transactions txs from the mempool
06 Create block $b = \langle t, v_i, \mathsf{Hash}(head), d = 2, atts, txs \rangle$
07 **if** v_i is *backup validator* **then**
08 Repeat lines 03-05
09 Sleep for a random period $\delta \in (0, \Delta)$
10 **if** there does not exists $b' \in \mathcal{T}$ such that $slot(b') = t$ **then**
▷ exploited by attack-I
11 Create block $b = \langle t, v_i, \mathsf{Hash}(head), d = 1, atts, txs \rangle$
12 **if** $b \neq \perp$ **then** send b to all validators

13 Upon receiving a valid block b from v_j such that $slot(b) = t$ and v_j is a valid in-turn validator or backup validator for slot t **do**:
14 (i) Add b to block tree $\mathcal{T}$
15 (ii) Update the last justified block LJ, if applicable
16 **if** b is the first block received in slot t **then**
▷ exploited by attack-II
17 **if** the chain led by b does not belong to $\mathcal{T}$ **then**
▷ exploited by attack-III
18 wait until it finishes point-to-point synchronization
19 Let b' be the parent block of b
20 **if** b' is not the head of the canonical chain **then** *abort*
21 Create an attestation $att = \langle v_i, sc, tg, h(sc), h(tg) \rangle$ and send to all validators

Figure 3: The consensus protocol of BSC. Code for validator v_i.

point-to-point synchronization period, it does not vote.

Mechanism-II (Pack attestations or nothing): Each block b only has two *forms* in terms of including the attestations: b consists of $N-f$ matching attestations for its parent block b'; b does not consist of any attestations. Namely, if a proposer fails to collect $N-f$ matching attestations for its *head*, b does not include any attestations in its proposal.

Mechanism-III (Consecutive honest leaders): The only way to finalize block b (proposed in slot t) is that two consecutive blocks consist of $N-f$ attestations. Namely, block b' proposed in slot $t+1$ consists of $N-f$ matching attestations for b, and block b'' proposed in slot $t+2$ consists of $N-f$ matching attestations for b'. After that, the chain led by b is finalized.

Mechanism-IV (First-in-first-vote): The protocol requires each validator to vote for the first block it receives in each slot (line 13-21 of Figure 3). In particular, $O(N)$ blocks can be proposed in each slot. Among them, only one block has a difficulty value of 2 and the rest blocks have a difficulty value of 1. However, even if a block with a difficulty value of 1 is received before a block with a difficulty value of 2, each validator v_i still triggers the voting event and votes for this block. v_i then stops voting for other blocks in the slot. Meanwhile, if v_i is a backup validator and has not proposed a block yet, it will not propose its own block in the current slot.

Why does FF lower the latency of finalizing a block in the optimistic case? Finality gadget does not necessarily lower the latency of finalizing blocks, even in the optimistic case [24–26]. For instance, in Ethereum PoS, since not all validators vote in every slot, the Casper FFG moves more slowly than the underlying consensus mechanism (i.e., HLMD GHOST) [16]. The latency of finalizing a block is two *epochs* (i.e., 64 slots) in the optimistic case. In contrast, BSC allows all validators to vote in each slot. In the optimistic case, finalizing a block takes 2.5 slots (i.e., 7.5 seconds). Compared to the underlying Clique mechanism (where the order of a block cannot be reversed after $O(N)$ slots), the latency is lowered.

Rewards and slashing. BSC also adopts a rewards and slashing mechanism inspired by Ethereum. In particular, any validator that includes valid attestations in its blocks will receive some rewards. Any validator that generates one of these attestations receives some reward. The rewards are distributed to the validators at the end of each epoch, regardless of whether blocks are finalized or not. Any validator that equivocates (e.g., proposes conflicting blocks or votes for conflicting blocks) will suffer from the *slashing* condition. After being slashed, a validator will suffer from account punishment.

4 Overview of Our Results

We study the fast finality (FF) mechanism of BSC. The fast finality mechanism adopts the voting mechanism from conventional BFT, a well-known technique to finalize blocks quickly. We show that properly designing and implementing FF can be challenging. We present three attacks on the fast finality mechanism and show how the current mechanism fails. We provide an overview of our attacks below.

Attack-I (CLSO attack): Identifying an issue of PoA in the chained and leader-speak-only-once (CLSO) model. Our first attack exploits an issue of protocols in the CLSO model. In this model, every validator can propose one block (and become the *leader* of that slot) and is immediately rotated. BSC also belongs to this model.

Our attack targets the *consecutive honest leaders* mechanism (see *mechanism-III* of §3.2) of BSC. Namely, *three* proposers need to be honest to finalize a block. Considering the case where the adversary can choose the identities of corrupted validators and $N = 3f+1$, there is only *one* occurrence of three consecutive honest proposers. By applying our attack, a block can only be finalized once every $O(N)$ blocks are proposed. Thus, the finality gadget still works, but finality of the blocks is significantly slowed down.

Previously, the CLSO liveness issue has been identified

only for BFT protocols (e.g., Chained HotStuff [10]) where only one leader is selected at a time. In contrast, BSC uses a PoA protocol with multiple leaders (i.e., multiple proposers are selected for each slot). Surprisingly, we present an attack that achieves similar results in BSC.

Attack-II (Split voting attack): Exploiting an issue with the first-in-first-vote (FIFV) mechanism. Attack-II exploits an issue with the first-in-first-vote (FIFV) mechanism (see *mechanism-IV* of §3.2). As described in §3.2, multiple backup validators are allowed to generate blocks in a slot (Line 2 and 7 in Figure 3). Compared to protocols with a single proposer at a time, multiple proposers can handle cases where the in-turn validator does not propose a block.

The FIFV mechanism of BSC requires each validator to vote for the first block received in each slot. Our second attack exploits this fact and manipulates the order of blocks received by honest validators. In this way, honest validators vote for different blocks (all proposed in the same slot). In the end, none of the blocks has more than $N-f$ matching attestations. Such an attack can be launched in *every* slot, so no block can be finalized according to the FF rules.

Attack-III (Synchronization attack): Exploiting an issue with the point-to-point synchronization. In attack-III, we present an extremely simple yet effective attack that significantly compromises liveness. Our attack exploits the design and implementation of the point-to-point synchronization mechanism (see *mechanism-I* of §3.2). Namely, every validator only votes for a block b after it has previously received the blocks led by b. Otherwise, it starts synchronization and skips voting. Our attack-III exploits this design and causes at least one honest validator to remain *stuck* in the point-to-point synchronization period for many epochs.

Later, our experimental results show that in a network with 21 validators, an honest validator can continue its synchronization for up to 130 epochs. In this case, if all Byzantine validators simply do not vote at all, none of the proposed blocks during this long period of time can have more than $N-f$ attestations. Accordingly, the protocol does not finalize any blocks. In fact, none of the honest in-turn validators will include any attestations in their blocks (see *mechanism-II* of §3.2), so all validators lose their rewards for attestations.

5 Our Attacks

5.1 Attack-I: CLSO Attack

As mentioned in §4, our attack-I resembles an issue with protocols in the CLSO model. Attack-I has some constraints on which validators the adversary corrupts, and we assume that f validators are corrupted. In particular, the in-turn validators are ordered in the form of two consecutive honest validators followed by one Byzantine validator. For example, in the case with seven validators $\{v_1, v_2, \cdots, v_7\}$, v_1 and v_4 can be corrupted.

Attack strategies. The attack strategies for the adversary are summarized below.

(1) Whenever an in-turn validator v_i is Byzantine in some slot t, at the beginning of the slot, it creates a block b and sends b to *all* backup validators, but not other non-turn validators. In b, v_i does not include any attestations.
(2) None of the Byzantine validators send any attestations.

After the attack is launched, for any slot t such that the in-turn validator is Byzantine, no validator can collect $N-f$ matching attestations for any blocks. To see why this is the case, we first consider the backup validators. Since all backup validators receive b from v_i, they do not propose any blocks in slot t according to the protocol (line 10 of Figure 3). In practice, this is also related to the actual period each backup validator sleeps. Recall that as mentioned in §3.2, each backup validator sleeps for at least one second before it starts to propose a backup block. To ensure that our attack always succeeds, v_i can send its block *early enough*, e.g., before slot t begins. Thus, no blocks from backup validators are created.

We now consider attestations for b. Since v_i only sends b to backup validators, no validator can receive $N-f$ matching attestations for b. This is because Byzantine validators do not send any attestations (strategy (2)), and only backup validators will vote for b. As there are only $N/2-1$ backup validators and we assume $N > 3f$, obviously $N/2-1 < N-f$. Our statement thus holds.

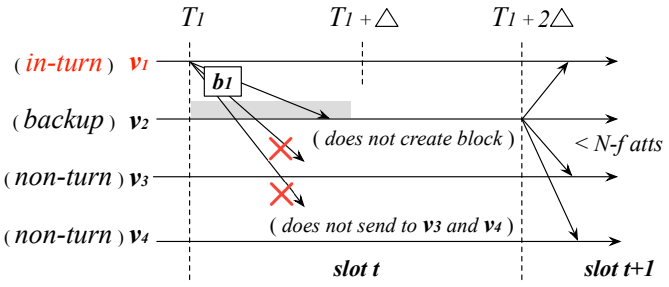


Figure 4: An example of the CLSO Attack. The in-turn validator v_1 is Byzantine. v_1 sends its proposed block b_1 to v_2, but not v_3 and v_4. v_2 does not create its block and votes for b_1. Since v_2 is the only validator that sends an attestation and b_1 is the only proposed block, no validator can receive $N-f$ matching attestations for any block.

We show an example in Figure 4 with four validators, where v_1 is Byzantine. In slot t, v_1 is the in-turn validator, and it sends b_1 immediately to v_2, which is the only backup validator. Since v_3 and v_4 do not receive b, they do not send any attestations. v_1 is Byzantine, so it does not send any attestations. By the end of slot t, the next in-turn validator v_2 and the next backup validator v_3 can collect only one attestation, so none of them will include any attestations in their proposed blocks (see mechanism-II of §3.2).

Why is finality of blocks slowed down? Based on the discussion above, we now discuss why FF fails to achieve improved latency. As discussed in *mechanism-III* (§3.2), BSC requires three consecutive honest leaders to finalize a block. In a system with $N = 3f + 1$ validators, the adversary can corrupt the validators in the form of two consecutive honest in-turn validators followed by one Byzantine in-turn validator. Accordingly, there exists only *one* occurrence of three consecutive honest leaders! To see why, we consider the problem a *positioning* problem where we place validators in a sequence of chain/ring. We already know that the adversary corrupts f Byzantine validators, and we place two honest validators before any Byzantine validator. Thus, we have now already allocated the positions of $3f$ validators. Since there is only one honest validator left, we can place it anywhere so that there is only one occurrence of three consecutive honest leaders.

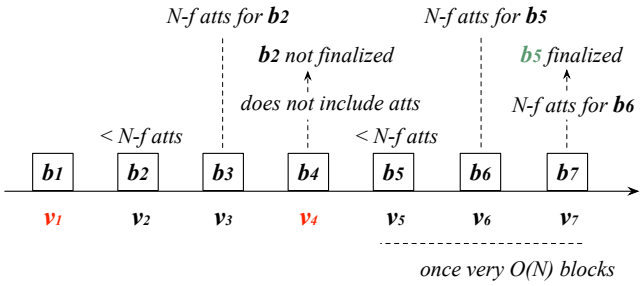


Figure 5: Illustration of why finality of blocks is slowed down in attack-I. Three consecutive honest leaders are required to finalize one block. When $N = 3f + 1$, there is only one occurrence of three consecutive honest leaders. Thus, only one block is finalized once every $O(N)$ blocks are proposed.

We show an example with seven validators in Figure 5. In this example, the adversary corrupts v_1 and v_4. Since v_1 is Byzantine, according to the argument above, no validator can receive $N - f$ attestations for any block in this slot. Validators v_2 and v_3 are honest. v_3 is able to collect $N - f$ matching attestations for b_2 and then include in its block. In the next slot, v_4 is Byzantine. It simply does not include any attestation (strategy (1)). Accordingly, b_2 cannot be finalized (see mechanism-III of §3.2). In this example, the only occurrence of three consecutive honest leaders is v_5, v_6, v_7. The protocol can only ensure that b_5 can be finalized. b_6 and b_7 are not guaranteed to be finalized.

BSC has 21 validators, i.e., $f = 6$ and $N = 3f + 3$. Following the argument above, there are three occurrences of three consecutive honest leaders. Accordingly, three blocks are expected to be finalized out of 21 blocks under attack-I. Later, our experimental results validate our theoretical analysis.

Discussion. As mentioned in §3.2, a similar issue [10] has been identified for Chained HotStuff [3], a representative protocol under the CLSO model. In Chained HotStuff, there is only one leader at a time. In contrast, in BSC, there are multiple *leaders* (backup validators can also propose). One may wonder why this scheme still suffers from a similar liveness issue. In our attack, this is mainly because a backup validator does not propose its block if it has received another block. We believe this approach was meant to lower the network bandwidth consumption. Unfortunately, it introduces new issues.

5.2 Attack-II: Split Voting Attack

We do not put any constraints on the identities of corrupted validators and present attack-II. Attack-II exploits the first-in-first-vote mode to split the votes of honest validators. Once the attack is launched, no validator is able to collect $N - f$ matching attestations for any block in *every* slot. The FF mechanism then completely fails.

Attack strategies. To launch the attack, we manipulate the order of blocks received by honest validators in each slot. The goal is to make some honest validators receive block b first while other validators receive another block b' first. Ideally, the attack strategies are summarized below.

(1) In *every* slot t, the adversary monitors the blocks sent by the in-turn validator and backup validators. For the first two proposed blocks, b_1 and b_2, the adversary manipulates the order of them as follows. It splits all validators into two groups G_1 and G_2. It schedules the order of blocks such that validators in G_1 receive b_1 before b_2, and validators in G_2 receive b_2 before b_1.
(2) For blocks other than b_1 and b_2, schedule them so that all validators still receive the blocks before $T + \Delta$ but after b_1 and b_2, where T is the time slot t begins.

Why does FF fail to work? The attack above splits validators into two groups, making them receive blocks in different orders. Accordingly, no validator can collect $N - f$ matching attestations for any block. We show an example in Figure 6. In this example, there are four validators, all of whom are honest. At time T_1, the in-turn validator v_1 sends b_1 to all validators. The adversary schedules the messages so that v_2 receives b_1 after it finishes sleeping. Thus, v_2 proposes a backup block b_2. The adversary further schedules the messages so that v_1 and v_4 receive b_1 before b_2, and v_2 and v_3 receive b_2 before b_1. According to the FIFV mechanism (each validator casts a vote immediately after receiving the first block proposal), v_1 and v_4 vote for b_1, v_2 and v_3 vote for b_2. In the end, no validator can receive $N - f$ matching attestations. This attack can be repeated in every epoch so that no block can be finalized according to the FF mechanism.

How to provide a more practical attack? One may argue that changing the order of messages is challenging to implement. We now describe an easy approach to instantiating the attack. Note that BSC uses a peer-to-peer gossip network for the underlying network communication. By default, upon receiving a message, each validator forwards the message to

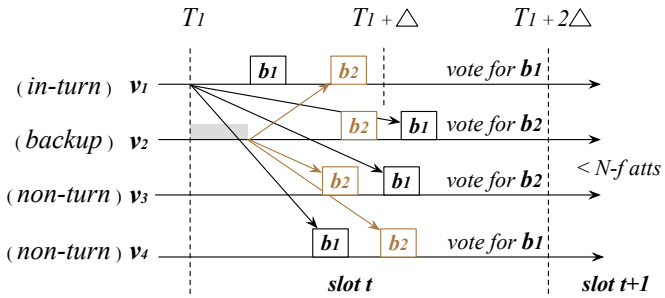


Figure 6: Illustration of attack-II. Validators v_1 and v_4 receive b_1 before b_2, so they vote for b_1. Validators v_2 and v_3 receive b_2 before v_1, so they vote for b_2. No validator is able to collect $N-f$ matching attestations.

a subset of validators (called peers) instead of all validators. Based on this fact, we summarize the attack strategies below.

(1) In every slot t, if the in-turn validator or a backup validator v_i is Byzantine, it creates a block b. When slot t begins, v_i sends b to at least one honest validator v_j but not all validators. Here, v_j should be a non-turn validator. In fact, v_i can send its block slightly ahead of time to ensure that its block arrives earlier than other blocks at v_j. For the rest of the network, all Byzantine validators stop forwarding b to their peers. Such a process will *delay* the arrival of b at other honest validators.

(2) For any other blocks, all Byzantine validators actively forward the blocks to all peers in the gossip network as fast as they can.

(3) None of the Byzantine validators send any attestations.

In our attack strategies mentioned above, we want to ensure that some honest non-turn validator v_j receives b earlier than other blocks. Meanwhile, other in-turn and backup validators will still propose their blocks, which are received earlier than b by honest validators other than v_j.

Discussion. In our attack, as long as Byzantine validators are evenly distributed in the P2P network, we can ensure that honest validators receive blocks in a different order. Thus, the split voting attack can succeed most of the time.

Note that the idea of splitting validators into two groups to slow down finality or even cause liveness failures is not new. Examples include the balancing attacks [13, 27, 28] and the refined liveness attack on Ethereum PoS [11, §4]. For instance, in [11, §4], two in-turn validators (also called block proposers) in two consecutive slots need to be Byzantine to launch an attack. Validators are split into two groups. The adversary schedules messages so that all validators in each group vote for one of the two blocks (by the two Byzantine validators). In this way, fewer than $N-f$ votes for any of the two blocks can be accumulated, and the protocol suffers from liveness failures. Similar to [11], our attack-II also requires the adversary to schedule the order of messages received by honest validators. The main difference is that our attack only requires one in-turn validator to be Byzantine to start the attack. Instead of manipulating the order of two blocks from two slots, we need to manipulate the order of a primary block (by an in-turn validator) and a backup block (by an honest backup validator). Our attack exploits the design of FIFV mechanism, which is unique to BSC.

5.3 Attack-III: Synchronization Attack

We present a synchronization attack, a simple yet effective attack. Attack-III exploits the point-to-point synchronization mechanism where a validator does not vote before it obtains the chain led by a proposed block. Attack-III is not guaranteed to succeed every time it is launched. The actual impact is dependent on the underlying network condition and the peer-to-peer topology. Jumping ahead a little bit, in our experimentation, we vary the network conditions and show that our attack can significantly slow down the finality of the blocks.

Attack strategies. The attack strategies are summarized below.

(1) Whenever an in-turn validator or a backup validator v_i is Byzantine in some slot t, it creates a new block b by setting the parent block field as some random value and then immediately sends b to one honest validator v_j at the beginning of slot t.

(2) None of the Byzantine validators send any attestations.

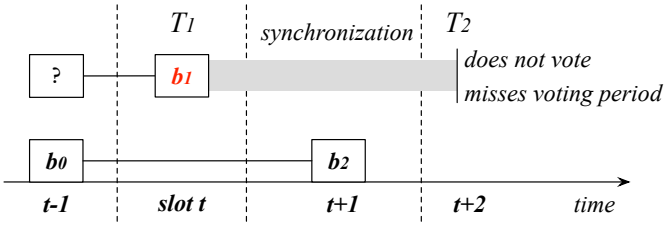


Figure 7: Illustration of attack-III. The figure denotes the view of an honest validator v_j. A Byzantine validator v_i sends a *fake* block b_1 to v_j. According to the protocol, v_j starts point-to-point synchronization with v_i and stops voting until the synchronization period ends. v_i delays the process and waits until the maximum synchronization period ends. In this case, v_j misses the chance to vote for several slots. As Byzantine validators do not vote, no blocks can be finalized during this period.

Why does FF fail to work? The attack strategies mentioned above can be viewed as a denial-of-service (DoS) attack to an honest validator v_j. A Byzantine validator simply creates an invalid block by setting the parent field as a random value and sending it to v_j. After receiving b, v_j triggers the point-to-point synchronization mechanism and begins to synchronize with v_i. As v_i does not have such a parent block, it simply delays until the synchronization period ends. The BSC implementation also requires that before the synchronization period ends, v_j does not vote at all. Consequently, v_j may

miss its voting period for several slots. After the synchronization period ends, another Byzantine validator performs the same strategies again, making v_j fail to vote for a long period of time.

We show an example in Figure 7. In this example, in slot t, the Byzantine validator proposes b_1 that extends a non-existent block. After v_j receives b_1 at T_1, v_j starts to synchronize with the Byzantine validator. After the synchronization period ends at time T_2, validators are already in slot $t+2$. Even if v_j still receives block b_2 in slot $t+1$, it does not vote for b_2. Meanwhile, none of the Byzantine validators send any attestations, so none of the blocks are finalized (we assume $N = 3f+1$ and there are at most $2f$ matching attestations). Such an attack can be repeated so that v_j will not be able to vote at all.

Discussion. Attack-III is closely related to the design and implementation of the synchronization module and the underlying peer-to-peer (P2P) network. In our experimentation, we found that the underlying network topology and the P2P specification both affect the attack. In particular, if multiple Byzantine validators are directly connected with v_j, they can take turns to launch the attack and make v_j unavailable. Additionally, the P2P implementation has several parameters that may affect the attack, which we will discuss in §7.

6 Risk and Reward Analysis

6.1 Risk Analysis

There is no *risk* for launching the attacks. Concretely, in attack-I, the adversary sends its block to a fraction of validators; in attack-II, the adversary schedules the arrival of the messages but does not discard them; in attack-III, the adversary sends invalid blocks earlier than expected. None of these satisfy the slashing conditions, and none of the Byzantine validators in our attacks will get punished. Hence, we consider our attacks risk-free.

6.2 Reward Analysis

We now briefly analyze the rewards received by validators under our attacks. As mentioned in §3.2, there are two types of rewards: (i) every validator that includes attestations (at least $N-f$) in its block receives rewards; (ii) once a block is finalized, the validators that sent the attestations (which are included in the block) receive rewards.

For attack-I, none of Byzantine validators receive any rewards, as long as they do not include attestations in their proposed blocks and do not vote. In contrast, once Byzantine validators include $N-f$ attestations for every three blocks, the corresponding validators receive rewards. We can slightly modify the attack strategies by letting Byzantine validators include $N-f$ attestations in their blocks so that they can also receive the attestation rewards.

For attack-II, none of validators receive $N-f$ attestations in any slot. Thus, no validator receives any rewards.

For attack-III, our goal is to trigger the event where some honest validators will not vote for a long period of time. None of the Byzantine validators will vote. Accordingly, during the period when the attack succeeds and no block can be finalized, no validators can receive any rewards.

In §7, we present experimental results on the rewards received by honest and Byzantine validators under the attacks.

7 Implementation and Evaluation

7.1 Implementation

We implement our attacks in Golang for BSC (version 1.4.16[3], the latest version at the time of writing). We have made our codebase and experimentation logs publicly available[4]. We conduct the experiments on Ubuntu 24.04 LTS, with 8 vCPUs, 32GB RAM, and 500GB NVMe SSD.

Experiment configuration. We establish a local *testnet* with 21 validators, matching the current configuration of the production system. Among the validators, six are Byzantine. Validators are connected via a peer-to-peer (P2P) LAN network.

For attack-I and attack-II, we create a topology such that a Byzantine validator is directly connected to all honest validators, and honest validators are not directly connected. For attack-II, we evaluate two types of P2P connections.

- **Bootnode-based connection:** A BSC bootnode is a predefined network entry point that helps validators discover peers and join the network by providing a list of initial node addresses. Each validator discovers a finite set of peers and connects to some validators based on this discovery.
- **Full connection:** Every pair of nodes is directly connected. If any peer is disconnected, the other peer will immediately try to reconnect.

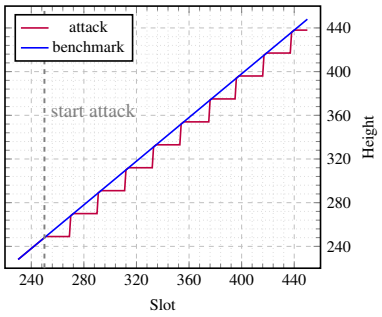
To ensure that a block from a Byzantine validator arrives earlier than other blocks, we modify the codebase such that a Byzantine validator sends its blocks 25 ms, 50 ms, and 75 ms ahead of time.

7.2 Evaluation

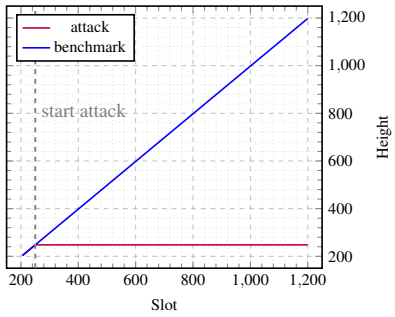
In each experiment, we run more than 2,000 slots on our *testnet* and assess the *fast finality*, i.e., how many blocks are finalized. In each experiment, our attack is launched at the 250-th slot. As a comparison, we also run the network without launching our attack and assess the number of finalized blocks in failure-free cases.

Evaluation on the finalized block. We launch our attack and assess the height of the finalized blocks. In this way, we can assess the number of slots it takes to finalize one block.

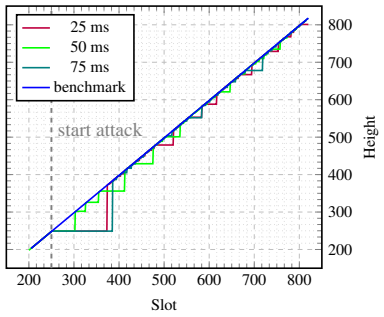
[3] https://github.com/bnb-chain/bsc/releases/tag/v1.4.16
[4] Our codebase: https://doi.org/10.5281/zenodo.15552871



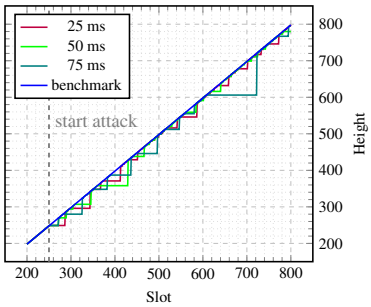
(a) The height of the finalized blocks under attack-I (purple) and the benchmark (blue).



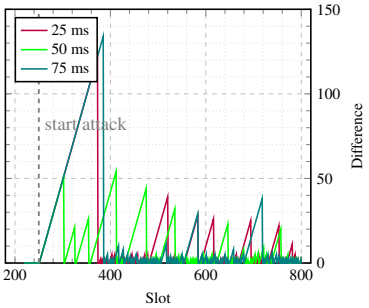
(b) The height of the finalized block under attack-II (purple) and the benchmark (blue).



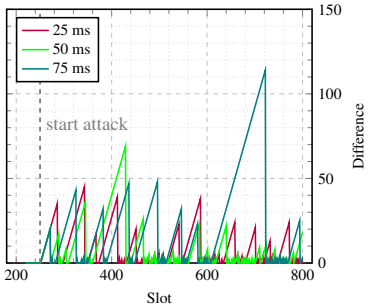
(c) The height of the finalized block under attack-III (**Use bootnode**, advance 25 ms, 50 ms, and 75 ms) and the benchmark (blue).



(d) The height of the finalized block under attack-III (**Full connection**, advance 25 ms, 50 ms and 75 ms) and the benchmark (blue).



(e) The height difference of the finalized block under attack-III and the the benchmark (**Use bootnode**, advance 25 ms, 50 ms and 75 ms).



(f) The height difference of the finalized block under attack-III and the benchmark (**Full connection**, advance 25 ms, 50 ms and 75 ms).

Figure 8: Evaluation the attacks. The figures are best viewed in color.

When no attack is launched (i.e., our benchmark), one block is expected to be finalized in each slot, e.g., as shown in the blue line of Figure 8(a). In each experiment, we launch the attack starting from the 250-th slot.

We first report the results of attack-I in Figure 8(a). Our experimental results match our theoretical analysis in §5.1, i.e., three blocks are finalized among 21 blocks.

Then, we show our attack-II in Figure 8(b). As shown in the figure, after the attack is launched at the 250-th slot, the height of the finalized block stays at height 248. The results also match our theoretical analysis in §5.2.

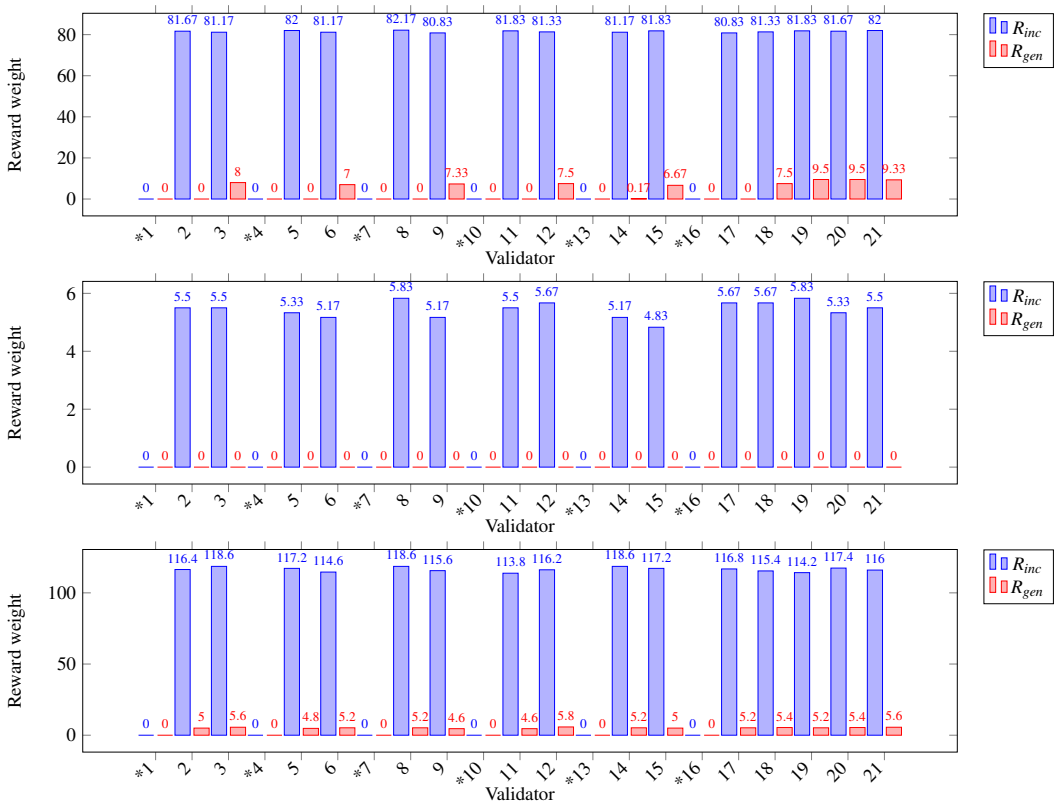
We show the results for attack-III in Figure 8(c)-8(f). We first assess the *bootnode* mode of P2P, where each validator chooses its own peers. We also set up the experiments so that Byzantine validators send their blocks 25 ms, 50 ms, and 75 ms ahead of time (and we call these "advance time"). As shown in Figure 8(c), there exist some periods where no blocks can be finalized. For the three *advance time* we evaluate, the impact when a Byzantine validator sends its blocks 75 ms ahead of time is the highest among the three modes. The result is more visible in Figure 8(e), where we report the height difference of the finalized block under the attack and the benchmark. In the extreme case, no block can be finalized for 130 slots (the leftmost peak). We believe this is because the adversary can ensure that an honest validator is *occupied* for a long time.

We also evaluate the performance using the *full connection* mode, as shown in Figure 8(d) and Figure 8(f). Compared to the experiments in the bootnode mode, the effect in full-connection mode is more *stable*. For advance times of 25 ms, 50 ms, and 75 ms, 10.09, 11.71, and 16.48 blocks cannot be finalized before a new block is finalized, respectively.

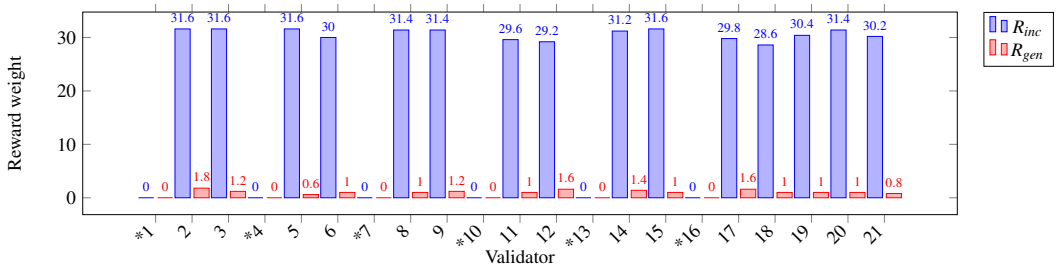
Table 2: Number of finalized blocks (#Blocks) without attacks, number of finalized blocks, and the finalization rate (FR) under attack-I and attack-II.

Attacks	**#Blocks (No Attack)**	**#Blocks (Attack)**	*FR*
attack-I	1999	279	$\approx 14\%$
attack-II	1999	0	0

Finalization rate (FR). To further quantify the analysis, we also measure the *finalization rate* (FR) of the attacks. Based on the consecutive honest leaders mechanism (§3.2), if two blocks consist of $N - f$ attestations, a block is finalized. We then count the number of occurrences of two consecutive blocks with attestations and count the number of



(a) The weight of the reward received by each validator in attack-I, attack-II and attack-III (**Use bootnode**). "*" means malicious validator.



(b) The weight of the reward received by each validator in attack-III (**Full connection**). "*" means malicious validator.

Figure 9: Rewards of the validators under attacks. From top to bottom are attack-I, attack-II and attack-III. The figures are best viewed in color.

Table 3: Number of finalized blocks (#Blocks) and the finalization rate (FR) without attacks and under attack-III (two different P2P modes).

Advance time	No Attack		Use bootnode		Full connection	
	#Blocks	*FR*	#Blocks	*FR*	#Blocks	*FR*
25 ms	1999	100%	513	≈ 25.7%	480	≈ 24.0%
50 ms	1999	100%	493	≈ 24.7%	458	≈ 22.9%
75 ms	1999	100%	515	≈ 25.8%	412	≈ 20.6%

finalized blocks. We use #Blocks to denote the number of finalized blocks. Let Num_b be the number of blocks proposed in each experiment. The finalization rate is calculated as: $FR = \frac{\#\text{Blocks}}{Num_b - 1}$. When no attack is launched, $FR = 1$.

We show the results of attack-I and attack-II in Table 2. In attack-I, only 14% of blocks are finalized. In attack-II, no block is finalized at all. We further show the results for attack-III in Table 3. For the *bootnode* mode, the finalization rate for the three "advance time" is roughly the same. For the *full connection* mode, the finalization rate is lowest when the advance time is 75 ms. This matches our analysis above.

Rewards. We report the rewards received by both honest validators and Byzantine validators under the attack and compare them with the benchmark. We show that the incentives received by honest validators are much lower than their fair share. We use R_{gen} and R_{inc} to denote the rewards for validators who generate attestations and for those who include attestations in their blocks, respectively. As mentioned pre-

Table 4: The average of the reward received by honest validators and Byzantine validators received in an epoch (200 slots) under the attacks and without the attacks. Each reward unit is in BNB.

Attacks	Honest validator		Byzantine Validator		No attack	
	R_{gen}	R_{inc}	R_{gen}	R_{inc}	R_{gen}	R_{inc}
attack-I	81.52	4.83	0	0	200	66.67
attack-II	5.44	0	0	0	200	66.67
III (bootnode)	116.44	5.19	0	0	200	66.67
III (full)	30.64	1.15	0	0	200	66.67

viously, the rewards are distributed to the validators in every epoch regardless of whether blocks are finalized. Each validator that creates a block receives one unit of BNB as a reward. Meanwhile, each validator that has one attestation included in the block receives 1/3 unit of reward.

Accordingly, we report the rewards received by both honest validators and the adversary for each epoch. By default, we consider that the adversary corrupts $f = 6$ validators. As there are 200 slots in each epoch, R_{gen} is 200 units of BNB when there are no attacks. Meanwhile, R_{inc} is about 66.67, as each validator receives 1/3 unit of BNB for each block.

We summarize the average reward received by each honest validator and Byzantine validator in Figure 9 and Table 4. As shown in the figures and the table, honest validators receive much lower rewards than their fair share under attacks, especially the rewards for including attestations in their blocks. This is expected, as few validators can receive $N - f$ attestations under our attacks.

We would like to argue that although Byzantine validators do not receive any rewards, they also do not suffer from slashing conditions. As mentioned in §6.2, by slightly optimizing our attack strategies, Byzantine validators can also receive the rewards for creating attestations. Additionally, our attacks mostly focus on failing the FF mechanism rather than maximizing the rewards of Byzantine validators.

8 How to Make Fast Finality Work?

In this section, we analyze the root causes and discuss the mitigation approaches. We believe BSC can be improved by the following general principles.

Vote "on-time". The first-in-first-vote mechanism does not follow the general principle of handling messages [29–31]. Indeed, even in a synchronous network, there is no guarantee on the fact that a message m sent earlier than m' will be received earlier than m'. Attack-I and attack-II exploit such a feature. Attack-I exploits the issue where an honest validator does not propose a new block after receiving one. Attack-II exploits the fact that any validator votes for the first received block in each slot. Accordingly, there is no way to prevent a faulty validator from sending a message (i.e., block) early enough to only a fraction of honest validators.

Instead of treating the messages according to the order they are received, a better approach is to handle them "on-time". For example, each slot t could proceed as follows, assuming T is the time slot t begins.

- At time T, each proposer proceeds according to the protocol. Upon receiving a block b, each validator adds b to the block tree and does not trigger the voting event. As a backup validator, it proposes a block regardless of whether a block has been received or not.
- At time $T + \Delta + \varepsilon$, validators use the fork choice rule to select the canonical chain, where ε is the maximum duration each validator sleeps. If there is a tie, break the tie deterministically, e.g., if two branches in the block tree have the same sum of difficulties, select the branch led by the block from a validator with a smaller identifier.

The procedures above can already partially solve the problems caused by attack-I and attack-II. First, we enlarge the slot duration from 2Δ to $2\Delta + \varepsilon$. This ensures that all honest validators receive the proposed blocks. Second, all honest validators start to decide which block to vote for at the same time ($T + \Delta + \varepsilon$). By employing a deterministic approach to selecting which block to vote for, we can avoid the issue of honest validators voting for different blocks. For example, even if the in-turn validator does not propose any block, all validators will vote for the block proposed by a backup validator with the lowest identifier. As long as the validator with the lowest identifier is honest, all honest validators receive the block and vote for it. Attack-I and attack-II can be mitigated.

Tricks in the CLSO model. In the study of conventional Byzantine fault-tolerant protocols, several approaches have been proposed to address the liveness issue in the CLSO model. For instance, instead of requiring three consecutive honest leaders, one may choose to design a two-phase protocol such that only two consecutive honest leaders are sufficient to finalize a block [41, 42]. Also, most protocols along this line of research allow validators to finalize blocks without requiring the quorum certificates (each certificate consists of $N - f$ matching attestations) to be consecutive [3, 10]. We believe many of the techniques can be used by BSC to improve the fast finality mechanism.

Randomized leader election. One approach to mitigating Attack-I is to replace the deterministic round-robin in-turn validator selection approach with a randomized leader election (e.g., in-turn validators are selected pseudorandomly [43–45]. Recall that attack-I exploits the fact that three consecutive in-turn validators must be honest to finalize one block. If in-turn validators are selected pseudorandomly, attack-I can be launched only probabilistically.

Making synchronization an asynchronous process. In BSC, the point-to-point synchronization process is designed to be a blocking process, where a validator stops voting before the synchronization period ends. Such a design makes the consen-

Table 5: Comparison of selective known attacks to PoS, PoA, and our work.

Consensus	Attack	Timing assumption	Attacking target					Experimentally confirmed
			Safety violation*	Liveness violation†	Finality gadget*‡	Incentives$^{\diamond}$	Chain quality†	
PoS	Long-range attack [32–34]	synchrony	●	●	●	●	●	✗
	Bouncing attack [14]	psync	-	●	●	●	●	✗
	Balancing attack [11, 13, 27]	synchrony	-	●	●	-	●	✓
	Reorg attack [11, 35, 36]	synchrony	-	-	-	-	●	✗
	Staircase attack [37]	synchrony	-	-	-	●	●	✓
PoA	Time-manipulation [38, 39]	synchrony	-	-	-	●	●	✓
	Fairness attack [40]	synchrony	-	-	-	●	●	✓
	Clone attack [22]	synchrony	●	●	●	●	●	✓
PoSA (ours)	**CLSO attack**	synchrony	-	-	●	●	●	✓
	Split voting attack	synchrony	-	●	●	●	●	✓
	Synchronization attack	synchrony	-	●	●	●	●	✓

* Safety violation denotes the fact that honest validators finalize conflicting blocks. Attacks on PoS in this table are all attacks on Ethereum PoS. Ethereum PoS uses HLMD GHOST as the underlying consensus mechanism and uses Casper as the finality gadget. HLMD GHOST does not have its own finalization rules. Thus, any attacks that cause safety violation can easily fail the finality gadget.

† An attack that causes liveness violation means that no block can be finalized. Such an attack also affects the chain quality.

‡ Attacks on finality gadget focus on slowing down or failing the finality gadget, and such attacks might be related to safety and liveness. Since bouncing attack and balancing attack cause a liveness violation; no blocks can be finalized. We thus consider them attacks on the finality gadget. For PoSA (this work), the underlying consensus mechanism Clique still has its own finalization rules that are independent with the finality gadget. Thus, liveness might not necessarily be violated.

$^{\diamond}$ Attacks on incentives often focus on increasing the incentives received by the adversary and lowering the incentives by honest validators. Meanwhile, these attacks often lower the chain quality of the system.

The ● (red solid dots) denotes the target of the attack. ✓ means the attack is experimentally confirmed on the implementation, while ✗ means the opposite. The symbol "-" means that the attack does not have an impact on the corresponding property or that the impact is not discussed.

sus protocol highly coupled with the underlying implementation of the communication channel. While synchronization might be necessary to prove the existence of the chain, it can be improved in many ways. For instance, synchronization of the blocks can be decoupled from the consensus process as much as possible [46]. However, designing such a decoupling approach correctly might still be challenging. We refrain ourselves from expanding the discussion here and consider it an interesting future work.

9 Related Work

We summarize some of the related work in Table 5.

Attacks against PoS. Based on the properties of the attack target, known attacks are classified into safety attacks, liveness attacks, and incentive attacks. For safety attacks, one example is the long-range attack [32–34]. Long-range attack requires that the adversary can obtain abandoned secret keys and then create forks of the chain. Such an assumption might be overly strong. For liveness attacks, bouncing attacks [14, 24, 25] and balancing attacks [13, 27] target the Ethereum PoS and make the system suffer from liveness issues. Reorg attacks [35, 36] reorganize the chain to lower the chain quality (informally, the chain quality is higher if a larger fraction of blocks are proposed by honest validators). The Staircase attack [37] is a recent attack targeting the incentive mechanism of Ethereum. The idea is to reorganize the chain so that honest validators are penalized even if they follow the protocol.

Besides long-range attacks, most of the known attacks to PoS are reorg attacks, where the blocks proposed by honest validators are discarded (and reorganized). In contrast, our attacks are not reorg attacks, and no blocks are reorganized.

Attacks against PoA. Ekparinya et al. [22] introduce a cloning attack to PoA Clique and Aura. The cloning attack duplicates a pair of public-private keys across two distinct networks, enabling malicious participants to perform double-spending attacks. The slashing condition is a solution to this issue. Zhang et al. [38] propose front-running attacks targeting in-turn validators in the PoA Clique protocol. The attack affects both chain quality and incentives, where the adversary gains 200% rewards compared to its fair share. Wang et al. [40] identify two types of order manipulation attacks that compromise transaction fairness and chain quality. In comparison, the consensus mechanism we study for BSC can

be viewed as an integration of the PoA Clique protocol and the fast finality mechanism. Instead of focusing on the PoA, our attacks focus on the fast finality mechanism. The properties affected by our attacks include liveness, finality, and incentives.

BFT in the CLSO model. Most partially synchronous (there exists an unknown upper bound on message processing and propagation) and synchronous BFT protocols are leader-based [2, 3, 17, 30], where one validator proposes a block and all validators agree on the order. Such protocols usually have a *view change* process that elects a new leader after the current leader fails. If a view change occurs every time a block is proposed, a protocol is turned to one in the CLSO model. Unfortunately, most protocols might be very expensive in rotating leader mode. HotStuff [3] is a protocol that has the same message and communication complexity during normal-case operation and view changes, making it a perfect paradigm under the CLSO model. Its chained version, called Chained HotStuff [10], suffers from the liveness issue. As discussed in §8, several solutions have been proposed to fix the issue, and we believe some techniques can be borrowed to improve BSC as well.

10 Conclusion

We study the fast finality (FF) mechanism of the BNB Smart Chain (BSC). We present three practical attacks that can impede liveness and can also make honest validators fail to receive their rewards. We also provide mitigation solutions.

Acknowledgment

This work was supported in part by the National Key R&D Program of China under 2022YFB2702800, the National Natural Science Foundation of China under 92267203, Beijing Natural Science Foundation under M23015, China Postdoctoral Science Foundation under 2023M741949, and Tsinghua Shuimu Scholar. This is also supported by Zhongguancun Laboratory. Haibin was additionally supported in part by the National Natural Science Foundation of China under 62272043, Major Program of Shandong Provincial Natural Science Foundation for the Fundamental Research under ZR2022ZD03, Yangtze Delta Region Institute of Tsinghua University, Zhejiang (N0.LZZLX24F007). Qin Wang did not receive any specific funding for this work.

11 Ethics Considerations and Compliance with the Open Science Policy

Our experiments are conducted on a local testing platform using open-source libraries and public datasets, without connecting to any external or live systems. These experiments do not involve any issues related to animals, human beings, the environment, healthcare, or military factors. We have addressed numerous ethical considerations in our experimental design, strictly adhering to the ethical principles outlined in the Menlo Report [47].

11.1 Research Ethics Considerations

We are committed to complying with all relevant research ethics considerations. In particular, we are committed to the following principles:

- **Respect for Persons:** Our research does not involve human subjects or personal data. We respect the work of other researchers and properly cite all relevant prior work.
- **Beneficence:** Our findings highlight a critical vulnerability in fast finality mechanism of BNB Smart Chain.
- **Justice:** We strive to ensure our proposed modifications do not disproportionately impact or disadvantage any particular group of BSC users or validators. We have already disclosed our findings to BSC developers.
- **Respect for Law and Public Interest:** No actions have been taken to exploit the identified vulnerabilities; instead, our research was conducted with the goal of improving the BSC system.

11.2 Compliance with the Open Science Policy

We are committed to the principles of open science, ensuring transparency, reproducibility, and accessibility throughout the research process. We adhere to the following practices:

- **Data Availability:** All experimental data used in our analysis will be made publicly available in a repository upon publication.
- **Code Availability:** To support open review, our implementation is open-sourced and shared via Zenodo[5].
- **Reproducibility:** We document our methodology to promote the reproducibility of our results.

References

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

[2] Miguel Castro, Barbara Liskov, et al. Practical Byzantine fault tolerance. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, volume 99, pages 173–186, 1999.

[3] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 347–356, 2019.

[5] https://doi.org/10.5281/zenodo.15552871

[4] João Sousa, Eduardo Alchieri, and Alysson Bessani. State machine replication for the masses with bft-smart. In *DSN*, pages 355–362, 2014.

[5] Clique in Go-Ethereum. https://github.com/ethereum/go-ethereum/blob/master/consensus/clique/clique.go, 2021.

[6] Péter Szilágyi. Eip-225: Clique proof of authority consensus protocol, 2017. Ethereum Improvement Proposals, accessed: Dec.,2021.

[7] BNB Chain Documentation Team. BNB smart chain: Introduction and security, 2025. Accessed: 2025-01-23.

[8] BSC Team. BEP-126: Introduce fast finality mechanism. https://forum.bnbchain.org/t/bep-126-draft-introduce-fast-finality-mechanism/123, 2024.

[9] Christian Cachin and Marko Vukolić. Blockchain consensus protocols in the wild. In *International Symposium on Distributed Computing (DISC)*, 2017.

[10] Neil Giridharan, Florian Suri-Payer, Matthew Ding, Heidi Howard, Ittai Abraham, and Natacha Crooks. BeeGees: stayin'alive in chained BFT. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 233–243, 2023.

[11] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. Three attacks on proof-of-stake Ethereum. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 560–576. Springer, 2022.

[12] Francesco D'Amato, Joachim Neu, Ertem Nusret Tas, and David Tse. Goldfish: No more attacks on Ethereum?! *International Conference on Financial Cryptography and Data Security (FC)*, 2024.

[13] Joachim Neu, Ertem Nusret Tas, and David Tse. Ebb-and-flow protocols: A resolution of the availability-finality dilemma. In *IEEE Symposium on Security and Privacy (SP)*, pages 446–465. IEEE, 2021.

[14] Ulysse Pavloff, Yackolley Amoussou-Guenou, and Sara Tucci-Piergiovanni. Ethereum proof-of-stake under scrutiny. In *ACM/SIGAPP Symposium on Applied Computing (SAC)*, pages 212–221, 2023.

[15] Binance Chain Developers. Binance Chain Whitepaper. https://github.com/bnb-chain/whitepaper/blob/master/WHITEPAPER.md, 2022. Accessed: 2025-01-13.

[16] 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.

[17] Sisi Duan, Haibin Zhang, Xiao Sui, Baohan Huang, Changchun Mu, Gang Di, and Xiaoyun Wang. Dashing and star: Byzantine fault tolerance from weak certificates. In *IEEE European Conference on Computer Systems (EuroSys)*, 2024.

[18] Chao Li, Balaji Palanisamy, Runhua Xu, Li Duan, Jiqiang Liu, and Wei Wang. How hard is takeover in DPoS blockchains? understanding the security of coin-based voting governance. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 150–164, 2023.

[19] Gavin Wood. PoA private chains. https://github.com/ethereum/guide/blob/master/poa.md, November 2015. Retrived: Dec., 2021.

[20] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, Vladimiro Sassone, et al. PBFT vs proof-of-authority: Applying the CAP theorem to permissioned blockchain. In *CEUR Workshop Proceedings*, volume 2058. CEUR-WS, 2018.

[21] OpenEthereum. OpenEthereum. *accessed: Sep., 2024* https://github.com/openethereum/openethereum.

[22] Ekparinya Parinya, Gramoli Vincent, and Jourjon Guillaume. The attack of the clones against proof-of-authority. In *Annual Network and Distributed System Security (NDSS) Symposium*, 2020.

[23] Xinrui Zhang, Rujia Li, Qin Wang, Qi Wang, and Sisi Duan. Time-manipulation attack: Breaking fairness against proof of authority Aura. In *Proceedings of the ACM Web Conference (WWW)*, pages 2076–2086, 2023.

[24] Alistair. Beacon chain casper mini-spec. https://ethresear.ch/t/beacon-chain-casper-mini-spec/2760/19, 2018. Accessed: 2025-05-18.

[25] nrryuya. Analysis of bouncing attack on ffg. https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113, 2019. Accessed: 2025-05-18.

[26] Suryanarayana Sankagiri, Xuechao Wang, Sreeram Kannan, and Pramod Viswanath. Blockchain CAP theorem allows user-dependent adaptivity and finality. In *International Conference on Financial Cryptography and Data Security (FC)*, pages 84–103. Springer, 2021.

[27] Joachim Neu, Ertem Nusret Tas, and David Tse. Two more attacks on proof-of-stake ghost/ethereum. In *ACM*

Workshop on Developments in Consensus (ConsensusDay@CCS), pages 43–52, 2022.

[28] Joachim Neu, Ertem Nusret Tas, and David Tse. A balancing attack on gasper, the current candidate for eth2's beacon chain. https://ethresear.ch/t/a-balancing-attack-on-gasper-the-current-candidate-for-eth2s-beacon-chain/8079, October 2020. Ethereum Research.

[29] Haochen Wang, Qidi You, and Sisi Duan. Synchronous Byzantine agreement with $o(n)$ messages and $o(1)$ expected time. *IEEE Transactions on Information Forensics and Security (TIFS)*, 2024.

[30] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *IEEE Symposium on Security and Privacy (SP)*, pages 106–118. IEEE, 2020.

[31] Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. Optimal good-case latency for rotating leader synchronous BFT. In *International Conference on Principles of Distributed Systems (OPODIS)*, 2022.

[32] Vitalik Buterin. Proof of stake: How I learned to love weak subjectivity, November 2014. Accessed: 2025-01-16.

[33] Evangelos Deirmentzoglou, Georgios Papakyriakopoulos, and Constantinos Patsakis. A survey on long-range attacks for proof of stake protocols. *IEEE Access*, 7:28712–28725, 2019.

[34] Sarah Azouvi, George Danezis, and Valeria Nikolaenko. Winkle: Foiling long-range attacks in proof-of-stake systems. In *ACM Conference on Advances in Financial Technologies (AFT)*, pages 189–201, 2020.

[35] 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.

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

[37] Mingfei Zhang, Rujia Li, and Sisi Duan. Max attestation matters: Making honest parties lose their incentives in Ethereum PoS. In *USENIX Security Symposium (USENIX Sec)*, pages 6255–6272, 2024.

[38] Xinrui Zhang, Qin Wang, Rujia Li, and Qi Wang. Frontrunning block attack in PoA clique: A case study. In *IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–3, 2022.

[39] Jianting Zhang, Wuhui Chen, Sifu Luo, Tiantian Gong, Zicong Hong, and Aniket Kate. Front-running attack in sharded blockchains and fair cross-shard consensus. In *The Network and Distributed System Security (NDSS) Symposium*, 2024.

[40] Qin Wang, Rujia Li, Qi Wang, Shiping Chen, and Yang Xiang. Exploring unfairness on proof of authority: Order manipulation attacks and remedies. In *ACM on Asia Conference on Computer and Communications Security (AsiaCCS)*, pages 123–137, 2022.

[41] Xiao Sui, Sisi Duan, and Haibin Zhang. Marlin: Two-phase BFT with linearity. *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022.

[42] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *International Conference on Financial Cryptography and Data Security (FC)*, page 296–315, 2022.

[43] Edgington Ben. The eth2 book: Upgrading ethereum, a technical handbook on ethereum's move to proof of stake and beyond. *Retrived May 2025,* https://eth2book.info/capella/, 2023.

[44] Kaya Alpturer and S Matthew Weinberg. Optimal RANDAO manipulation in Ethereum. In *6th Conference on Advances in Financial Technologies (AFT)*, 2024.

[45] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*, pages 51–68, 2017.

[46] Consensys. Teku: Ethereum 2.0 client by consensys. https://github.com/Consensys/teku, 2024. Accessed: 2025-05-19.

[47] Michael Bailey, David Dittrich, Erin Kenneally, and Doug Maughan. The menlo report. *IEEE Security and Privacy (SP)*, 10:71–75, 2012.

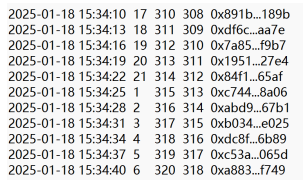
A Screenshots of Our Experiments

Figure 10 presents a sequence of block records generated by separately running normal and malicious BSC validators. From left to right, the columns show the block timestamps, validator indexes, current block numbers (in decimal), last finalized block numbers, and current block hashes (in hexadecimal). In the normal case (Figure 10(a)), the finalized

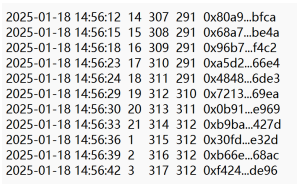
Table 6: Symbols and definitions.

Symbol	Definition
$\mathcal{V}$	The set of validators in the system
N	The total number of validators in the system
f	The maximum number of Byzantine validators
t	Slot number
Δ	Msg processing/propagation upper bound
v_i	A validator in $\mathcal{V}$
ℓ	The number of backup validators.
b	A block
$\mathcal{T}$	The block tree maintained by a validator
$h(b)$	The height of block b in the block tree
$slot(b)$	The slot number of block b
LJ	The last justified block in the canonical chain.
$\mathsf{Hash}(x)$	Hash of x
sc	The source block of an attestation
tg	The target block of an attestation
att	An attestation/vote created by a validator
d	The difficulty value of a block
e	The epoch number
head	The leaf block of the canonical chain
votepool	The set of all attestations stored by a validator

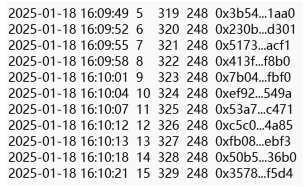
block number consistently lags behind the current block number by two. Under attack scenarios (Figures 10(b)–10(d)), the finalized block number ceases to update, with the extent of disruption varying by attack.



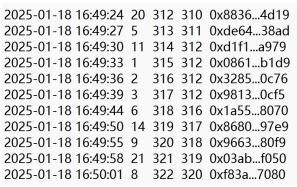
(a) Case 1:Normal



(b) Case 2:attack-I



(c) Case 2:attack-II



(d) Case 2:attack-III

Figure 10: Screenshots of running BSC private testnet