Don’t Trust, Verify: The Case of Slashing from a Popular Ethereum Explorer

Zhiguo He*
University of Chicago and NBER
Chicago, Illinois, USA
zhiguo.he@chicagobooth.edu

Jiasun Li
George Mason University
Fairfax, Virginia, USA
jli29@gmu.edu

Zhengxun Wu
Independent
New York City, New York, USA
wuzhengxun@outlook.com

ABSTRACT

Blockchain explorers are important tools for quick look-ups of on-chain activities. However, as centralized data providers, their reliability remains under-studied. As a case study, we investigate Beaconcha.in, a leading explorer serving Ethereum’s proof-of-stake (PoS) update. According to the explorer, we find that more than 75% of slashable Byzantine actions were not slashed. Since Ethereum relies on the “stake-and-slash" mechanism to align incentives, this finding would at its face value cause concern over Ethereum’s security. However, further investigation reveals that all the apparent unslashed incidents were erroneously recorded due to the explorer’s mishandling of consensus edge cases. Besides the usual message of using caution with centralized information providers, our findings also call for attention to improving the monitoring of blockchain systems that support high-value applications.

CCS CONCEPTS

• **Computer systems organization → Reliability**; • **Information systems → Search interfaces**.

KEYWORDS

blockchain, explorer, distributed consensus

ACM Reference Format:
Zhiguo He, Jiasun Li, and Zhengxun Wu. 2023. Don’t Trust, Verify: The Case of Slashing from a Popular Ethereum Explorer. In *Companion Proceedings of the ACM Web Conference 2023 (WWW ’23 Companion), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3543873.3587555

1 INTRODUCTION

Blockchain explorers are important tools for users to quickly query on-chain activities. They promote inclusiveness by allowing anyone to monitor a blockchain’s performance without having to incur the overhead of running a node. Despite blockchain explorers’ popularity, few studies have looked into how reliable they are. Given the crucial roles that explorers play in a blockchain’s ecosystem (and especially for proof-of-stake chains that are subject to “weak-subjectivity" [7]), it is important to ensure explorers’ accuracy, expose patterns regarding how mistakes may potentially occur, and reemphasize the “don’t trust, verify" motto with concrete examples.

In this paper, we take a first look at this issue by analyzing beaconcha.in, a leading explorer for Ethereum’s consensus layer (often known as the Beacon chain) within its proof-of-stake (PoS) update. After briefly explaining how Ethereum’s new consensus layer works in Section 2, we document several empirical facts from the explorer’s data in Section 3. Although beaconcha.in is the more comprehensive and often more reliable explorer among its peers,[1] we still find errors with significant implications: for example, according to the explorer, more than 75% of “slashable" consensus rule violations were left unslashed — which by appearance suggests a major failure of Ethereum’s consensus design, which heavily relies on a “stake and slash" mechanism to ensure compliance. However, further investigation shows that all these findings were due to the explorer’s mistakes in encoding validators, which reflects the negligence of consensus edge cases. We provide detailed explanations of what went wrong to help avoid future incidents.

While our concrete findings come from Ethereum, we believe the learned lessons carry broader implications for other blockchains and decentralized applications (dApps) running on them. As we will see, the explorer errors we find reflect a fundamental trade-off between computation and data availability that any blockchain would face when upholding decentralization: On the one hand, it is crucial to make on-chain data light/pruned to lower validator overheads; On the other hand, such efforts increase the computational costs for other stakeholders to monitor/verify on-chain activities later. Striking the right balance between the two competing forces is thus crucial for the security of the decentralized applications running on these chains. For many dApps, the heavy cost in the latter channel creates its own centralization forces, in that many high-profile dApps rely on centralized information providers.[2] Hence, the explorer mistake we identify again calls for more attention to the reliability of information providers and hopefully inspire community efforts toward a more robust ecosystem.

2 CONSENSUS ON ETHEREUM POS: A BRIEF OVERVIEW

This section gives a brief introduction to Ethereum’s recent upgrade to a proof-of-stake (PoS) system, which is often known as Ethereum 2.0 to separate from the proof-of-work powered “Ethereum 1.0."[3] The upgrade has been planned to take place over multiple phases, with phase 0 creating a new PoS-based blockchain known as the Beacon chain. The Beacon chain went online on Dec 1, 2020, and merged with Ethereum 1.0 on Sept 6, 2022. Our empirical analysis focuses on the Beacon chain. At a high level, the PoS mechanism in the Beacon chain works as follows (see Figure 1 for an illustration):

As a permissionless blockchain, anyone can stake 32 ETH and become an Ethereum 2.0 validator to participate in the consensus

*Authors are listed alphabetically.

[1]Regarding comprehensiveness, beaconcha.in is the only explorer we are aware of that displays detailed attestation information; regarding accuracy, there are incidents where beaconcha.in disagrees with another major explorer beaconscan, with the former being correct according to running a local consensus client node.
[2]See the Nov 2020 Infura outage for a vivid illustration. More recently, many other potentially centralized critical infrastructures for dApps have also emerged, including e.g., 0x’s Request for Quote (RFQ), Walletconnect, Defilemma, etc.
[3]For exact implementation details, see Ethereum’s consensus specs [link].

formation process, which proceeds in time units known as *epochs*. Before an epoch starts, the set of active validators are determined and pseudo-randomly assigned (using the chain's state) to their respective roles: Some validators are chosen to propose new blocks while all validators (including the proposers) are assigned to make attestations (votes). The assignment makes sure that within a given epoch, each validator has the right to attest once and only once, and each chosen proposer has the additional right to propose one and only one new block. Specifically, an epoch is divided into 32 *slots* each lasting for 12 seconds. In a given epoch, 32 out of all active validators are chosen as proposers, with each slot having one proposer. All validators (including the proposers) are subdivided into 32 groups so that each group of validators is assigned to attest for one of the 32 slots.[4] Every proposed block contains certain history of the Beacon chain, including the hash of a previous block, past attestations, execution layer transactions (after the merge), or occasionally slashing violation evidence (to be detailed later). Finally, each epoch also defines a *checkpoint* block, which is typically the block proposed in the first slot of the epoch.[5]

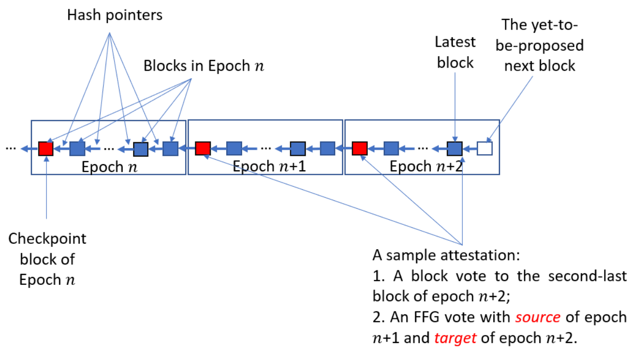
Hash pointers
Blocks in Epoch n
Latest block
The yet-to-be-proposed next block
Epoch n
Epoch n+1
Epoch n+2
Checkpoint block of Epoch n
A sample attestation:
1. A block vote to the second-last block of epoch n+2;
2. An FFG vote with source of epoch n+1 and target of epoch n+2.

Figure 1: An Illustration of the Beacon Chain Structure

This figure illustrates an ideal structure of the Beacon chain. Each epoch contains a sequence of blocks proposed by pre-scheduled block proposers. Blocks are connected with each other via hash pointers. The first block of each epoch is denoted as the checkpoint of the epoch. Within a given epoch, each validator makes one and only one attestation. In the figure, a validator makes an attestation that votes for the second-last block of epoch $n + 3$ and FFG votes for source $n + 2$ and target $n + 3$.

An attestation contains two types of votes. First, it indicates which newly proposed block it votes for.[6] Attesting to a block indicates an endorsement of the block as the latest block. Such votes thus follow the "longest chain" rule in Nakamoto consensus (as adopted by Bitcoin and Ethereum 1.0). Second, in addition to voting for a new block proposal, each attestation also additionally includes an *FFG vote* for checkpoints.[7] An FFG vote specifies both a *target* checkpoint and a *source* checkpoint, with the latter necessarily proceeding the former. While FFG votes do not have apparent analogies in Nakamoto consensus, they can be understood at a high level as a specific type of multi-round voting messages to help finalize blocks in the spirit of Byzantine Fault Tolerance (BFT) protocols.[8] All proposals and attestations are broadcast to peers as messages.[9] Compliant proposals and attestations bring their respective rewards.[10]

2.1 Slashing conditions

To ensure the blockchain's integrity, all validators are expected to comply with certain rules when proposing new blocks or making attestations. Roughly speaking, these rules require validators to never contradict themselves. Violators may be caught and "slashed," that is, be deprived of the privilege to validate (and thus collect rewards) anymore and stakes deducted according to a predefined rule. These violations can be categorized into "double proposal", "double vote", and "surround vote", as further explained below:

(1) Double proposal: a proposer proposes two conflicting blocks. A double proposal resembles "equivocating" different messages in a BFT context, which is prevented (with an overwhelming probability) in Nakamoto consensus by the "proof-of-work" requirement that makes it costly to create different proposals. Figure 2 gives an illustrative example;

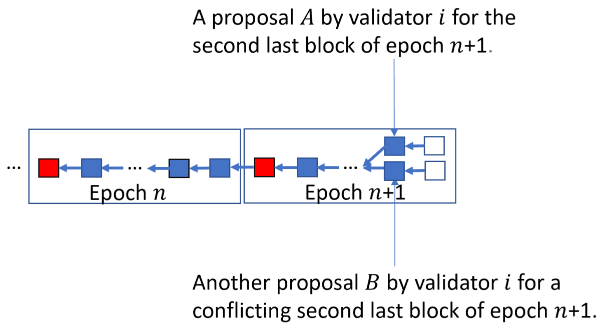
A proposal A by validator i for the second last block of epoch n+1.
Epoch n
Epoch n+1
Another proposal B by validator i for a conflicting second last block of epoch n+1.

Figure 2: Illustrations of double proposals

This figure gives an example of double proposals: Validator i makes two proposals A and B with two conflicting blocks at the time.

(2) Double vote: a validator sends two different attestations with the same target epoch number. Figure 3 gives an illustrative example;

(3) Surround vote: a validator casts two FFG votes A and B so that Source(A)< Source(B)<Target(B)<Target(A), where $\forall i \in \{A, B\}$, Source(i) and Target(i) denote the epoch numbers of FFG vote i' source and target, respectively. Figure 4 gives an illustrative example.

[4]Each group is further divided into committees — a legacy of the deprecated "sharding" plan. There are discussions to remove this further division into committees.

[5]If the block in the first slot is missing, then the checkpoint is defined as the latest preceding block (which may belong to a previous epoch).

[6]Ideally, all validators assigned to attest in a particular slot vote for the block proposed in the same slot. However, due to network latency, some validators may have not received the current-slot block before the slot expires, and these validators may instead vote for blocks proposed in earlier slots.

[7]FFG stands for Casper: the Friendly Finality Gadget, which is a protocol designed on top of a running blockchain for finalizing blocks in a Byzantine Fault Tolerance (BFT) fashion. See [8].

[8]Specifically, once a checkpoint has gathered FFG votes from more than (weighted by stakes) $\frac{2}{3}$ of all validators (reaching a supermajority), the checkpoint becomes justified. Once the immediately succeeding checkpoint of a previously justified checkpoint becomes justified, the previously justified checkpoint becomes finalized.

[9]In practice, to reduce bandwidth/storage usage, validators are further grouped into several committees so that many communications only happen within committees. Ethereum 2.0 adopts the BLS threshold signature ([5]) so that within-committee communications are aggregated for cross-committee communications.

[10]See detailed explanations on reward schedules here.

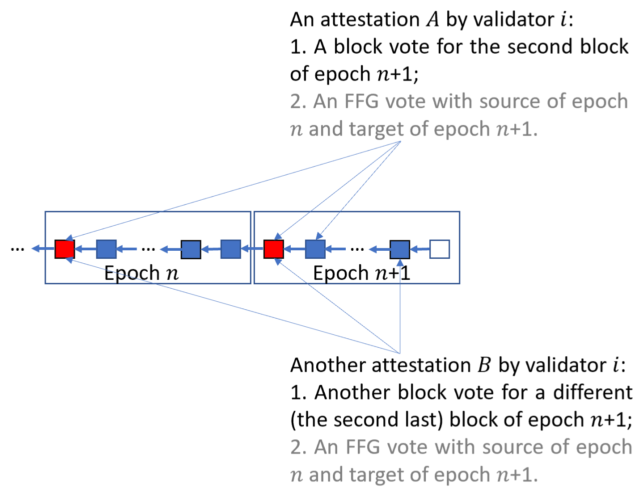
An attestation A by validator i:
1. A block vote for the second block of epoch n+1;
2. An FFG vote with source of epoch n and target of epoch n+1.
Epoch n
Epoch n+1
Another attestation B by validator i:
1. Another block vote for a different (the second last) block of epoch n+1;
2. An FFG vote with source of epoch n and target of epoch n+1.

Figure 3: Illustrations of double votes

This figure gives an example of double votes: Validator i casts two votes with the same target number but different attestation contents, in that attestation A votes for the second block of epoch $n+1$, while attestation B votes for a different (the second last) block of epoch $n+1$. Attestations A and B thus constitute double votes.

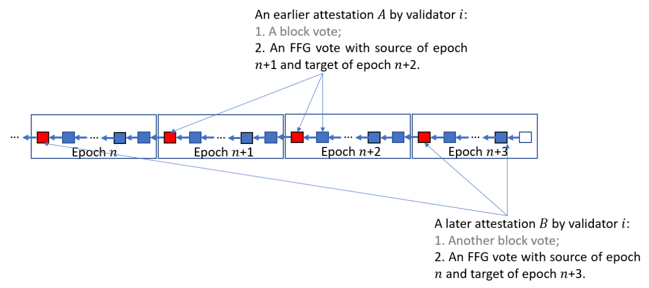
An earlier attestation A by validator i:
1. A block vote;
2. An FFG vote with source of epoch n+1 and target of epoch n+2.
Epoch n
Epoch n+1
Epoch n+2
Epoch n+3
A later attestation B by validator i:
1. Another block vote;
2. An FFG vote with source of epoch n and target of epoch n+3.

Figure 4: Illustrations of surround votes

This figure gives an example of an FFG vote "surrounding" a previous FFG vote: validator i's attestation A during epoch $n+2$ specifies a source of epoch $n+1$ and a target of epoch $n+2$, while later during epoch $n+3$ the same validator sends a new attestation B which specifies a source of epoch n and target of epoch $n+3$.

A rough intuition for slashing double or surround votes. In BFT protocols, "honest" behaviors, that is, to not deviate from the protocol's specified forwarding and voting strategies, ensure any record that has reached consensus to never be overturned under certain security conditions, say more than two-thirds of nodes are honest (see e.g. [9]; in contrast, Bitcoin does not have such a feature as Bitcoin blocks are never 100% finalized). In the context of Beacon chain, [8] show that for two conflicting checkpoints to ever get finalized, it necessarily requires more than one-third of validators to have cast two conflicting FFG votes that constitute a pair of either double votes or surround votes. Therefore, if fewer than one-third of validators commit such violations, then the Beacon chain will be "safe" in the sense that no conflicting checkpoints will ever be finalized. The threat of slashing aims to deter any validator from committing these violations, and thus ensure the "$< \frac{1}{3}$" condition.

We further provide an intuitive explanation of why the absence of surround or double votes is sufficient for the safety of finalized checkpoints. Indeed, we show that if two conflicting checkpoints ever both get finalized, then more than $\frac{1}{3}$ validators must have cast either surround or double votes.

First, recall from Footnote 8, a checkpoint becomes finalized when its immediate next checkpoint becomes justified, that is, having received more than $\frac{2}{3}$ of FFG votes from all validators as a target. Also recall from Footnote 4 that each epoch is divided into 32 slots. We then prove the argument by contradiction: Suppose two conflicting checkpoints A and B both get finalized (with $e(A)$ and $e(B)$ denoting the epoch number of checkpoints A and B). Discuss two scenarios: (1) If A and B are for the same epoch, that is, $e(A) = e(B)$, then more than $\frac{2}{3}$ validators have included A as target in their FFG votes, and (not necessarily the same set of) more than $\frac{2}{3}$ validators have included B as target in their FFG votes. By the pigeon hole principle, at least $\frac{2}{3} + \frac{2}{3} - 1 = \frac{1}{3}$ validators have included both A and B as targets in their FFG votes. These validators then have committed double votes. (2) If A and B are for different epochs, that is, $e(A) \neq e(B)$. Without loss of generality, assume that A has a smaller epoch number than B, that is, $e(A) < e(B)$. Since A and B conflict, B also conflicts with A's immediately next checkpoint A', which is justified by definition. Then $e(B) > e(A') = e(A) + 1$. Denote C as a justified checkpoint that has the smallest epoch number among the set of all justified checkpoints that conflict with A and have epoch number larger than $e(A)$. Notice that C is well-defined because the set is not empty (for example, B belongs to the set). Then all FFG votes that justify C must have C as target and a source checkpoint D with epoch number smaller than $e(A)$. Therefore, more than $\frac{2}{3}$ validators have included C as target and D as source in their FFG votes, while the finalization of A upon A''s justification means that (not necessarily the same set of) more than $\frac{2}{3}$ validators have included A' as target and A as source in their FFG votes. By the pigeon hole principle, at least $\frac{2}{3} + \frac{2}{3} - 1 = \frac{1}{3}$ validators have included both source-A/target-A' and source-D/target-C in their FFG votes. These validators then have committed surround votes.

2.2 Slashing detection in practice

When any of the above violations are committed, evidence of such violations may be gathered by some validator (known as the whistleblower) and then included by a proposer in a new block to trigger slashing of the offending validator. However, if no whistleblower detects such a violation (due to either costly detection or inadequate incentives),[11] or if the proposed block that includes a whistleblowing message fails to reach consensus (e.g., orphaned), then some slashable violations may be left unslashed. This theoretical possibility originally motivated us to look for potentially unslashed but slashable violations.

3 ERRONEOUS EXPLORER RECORDS

To investigate the accuracy of explorer's records on slashing events, we collect data from beaconcha.in, one of the most popular blockchain explorers of the Beacon chain. The explorer displays Beacon chain records in reader-friendly web pages, as well as exposes APIs to access their back-end data. Our data include all proposal/attestation

[11] In practice, both channels may be at work. On the resource cost in detecting violations, see e.g. the documentation of Prysm, one of the most popular Ethereum consensus client software, which states that "*Slasher ... uses significantly more disk space when running on mainnet.*" The same document also highlights the lack of incentives to whistleblowers: "*Running a slasher is not meant to be profitable.*"

records in the first 1.75 million blocks (from the genesis block on Dec 1, 2020 to August 1, 2021).

3.1 (Correctly) recorded slashing incidents

Slashing happens from time to time. Figure 5 chronicles by black solid bars all conflicting proposals/attestations involved in all slashing incidents in our sample (from the explorer). We are able to verify the signatures of these proposals/attestations, supporting this part of the explorer's records.[12]

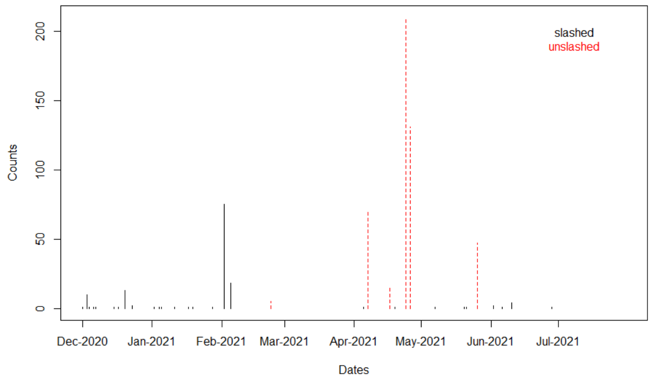

Figure 5: Slashed and seemingly slashable incidents over time

This figure plots for every day within our sample the count of slashing incidents (black solid bars) and seemingly slashable (yet unslashed) violations (red dashed bars). The sample includes the first 1.75 million Beacon chain blocks from genesis (December 1, 2020 to August 1, 2021).

Overall, as expected in the Ethereum community, slashing events tend to be rare: Out of the first 1.75 million blocks, there are just 156 recorded slashing incidents, including 15 proposer violations and 144 attester violations. Furthermore, slashing incidents tend to cluster. For example, out of all slashing incidents, 75 of them happened on the same day. There could be prolonged periods during which no slashes take place until suddenly "many things go wrong."

3.2 Unslashed (seemingly) slashable violations

While it may not be surprising to see detected slashing violations (ultimately this is what the slashing mechanism was supposed to do), our data from the explorer (which we later prove to be erroneous) also give a surprising finding in that many slashable violations seem to have dodged slashing. Figure 5 chronicles by red dashed bars all the proposals/attestations involved in such seemingly unslashed incidents.[13]

The number of seemingly unslashed violations is large compared to actual slashes: 478 unslashed violations, including 404 double votes and 74 surround votes (but no double proposals). In comparison, recall that the actual number of slashed attestation violations is 144. Hence, $\frac{478}{478+144}$, or more than 75% attestation violations seem to have dodged slashing according to the explorer.

Before proceeding, we make a digression to highlight a separate methodological contribution regarding how to efficiently identify surround votes, as the community has pointed out that it is a computationally non-trivial task to look for slashable offenses, and especially surround votes.[14] The trick is to recognize that surround votes necessarily involves attestations whose target and source slot numbers differ by at least 3. By first singling out these attestations, we significantly reduce the workload and speed up surround vote detection.

[12]Appendix A further includes a more detailed list of these incidents.
[13]Appendix B further includes a more detailed list of these incidents.

3.3 (Incorrect) explorer records

The findings from the previous section, at face value, would seriously question the effectiveness of the "stake-and-slash" mechanism in detecting and deterring misbehaving violations. However, two observations point to potential mistakes in beaconcha.in. First, as suggestive evidence, in each seemingly (according to the explorer) unslashed violation, at least one of the involved conflicting attestations does not appear in first-hand data when we directly sync an Ethereum consensus node; Second, as definitive evidence, none of these "phantom" attestations has valid signatures, indisputably proving that they are incorrect.

In addition, these "phantom" attestations (345 in total) all have the following features: (1) each of them involves a small number of validators (no more than six, with 1-validator and 2-validator attestations being the most common; See Figure 6); (2) blocks containing them are all in slots that feature orphaned blocks. These observations will turn out to be useful for our follow-up analysis in the next section.

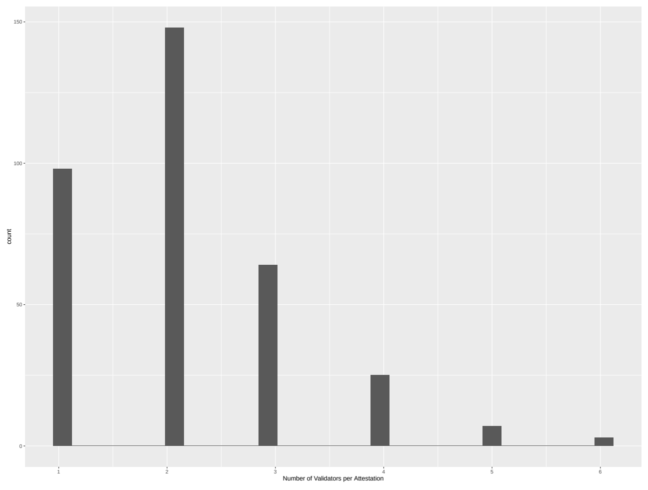

Figure 6: Validator counts per "phantom" attestation

This figure breaks down all "phantom" attestations by the number of validator(s) in each of them (ranging from 1 to 6), and then plots a histogram for the number of attestations within each category.

3.4 Uncovering the cause of the explorer's mistake

We believe it is important to understand where the "phantom" came from to help fix any existing explorer mistakes and also expose patterns for mitigating future occurrences. We thus further investigate

[14]See e.g. Protolambda's discussion on resource consumption. Various client implementations also cover similar issues, see e.g. here and here.

the root cause of the explorer's mistake and uncover the explorer's mishandling of a subtle consensus edge case. For ease of exposition, the discussion below follows our original uncovering process.

(1) Although none of the phantom attestations has correct signatures, for all 1-validator phantom attestations it is feasible to show that all signatures are valid once we correct attesting validator indices by brute force — iterating over all (~400K) validators to look for one that renders a correct signature.
(2) The last finding suggests that all 1-validator phantom attestations have indeed been created, though by validators other than those indicated by beaconcha.in. In other words, the explorer messed up validator labeling for these attestations. For example, according to the explorer, the second attestation in slot 608067 (containing only one validator 4219) is part of an unslashed double vote. This attestation does not have the right signature if we verify it with validator 4219's public key; however, it will have the right signature if we replace validator 4219's public key with validator 56119's.
(3) Why did the explorer mislabel validator 56119 as 4219? We notice that this attestation's target number points to the checkpoint of epoch 19002, which is typically its first slot, or slot 32*19002 = 608064. However, this attestation's target root actually points to block 608004, suggesting that validator 56119 likely experienced a network delay and missed subsequent blocks after slot 608004. According to Ethereum's "lookahead" rule, a node should use information (specifically, a field known as "randao_mix") in the last available chain state in epoch $19002 - 2 = 19000$ (which should correspond to slot $19000 \times 32 + 31 = 608031$) as a pseudo-random seed to calculate the attestation schedule for epoch 19002. However, since validator 56119's local state likely did not update after slot 608004, we conjecture and then verify that it actually used the state information til slot 608004 and calculated a "stale" attestation schedule. Indeed, according to the correct schedule, validator 56119 should attest in slot 608090, but instead, it follows its own stale schedule and attested in slot 608067. We then confirm that the explorer missed this edge case, and encoded the attestor in 56119's misplaced attestation using a non-stale attestation schedule, which turned out an innocent validator 4219.
(4) The above example suggests that in general, when a validator experiences network delay, it may locally compute a stale attestation schedule. However, the explorer may neglect this case and use an up-to-date schedule to encode validators, leading to erroneous records. We verify this conjecture on all 345 "phantom" attestations and were able to confirm 262 of them.
(5) We hypothesize that the remaining 83 "phantom" attestations also come from the same validator encoding error, as they also feature stale target roots. However, we cannot directly test them since the stale target roots were themselves orphaned, so we can no longer fetch the corresponding state information to replay what schedules those involved validators used.[15]

[15]When a consensus node syncs from peers, orphaned blocks are not transmitted. Unless one happens to be running a node during the event times (and happens to receive the misplaced attestations), such information is forever gone.

4 CONCLUSION

We uncover and explain an edge case in Ethereum's consensus layer that has been mishandled by a leading explorer. In addition to helping improve it, we hope our findings could also bring more attention to the reliability of major information providers, many of which provide critical supports to high-value DeFi applications.

Our investigation into slashing outcomes also adds to an emerging literature on the incentive analysis of BFT-based consensus protocols (e.g. [16], [3] and [4]). These economic analyses in turn build on a large computer science literature starting from [18], who formulated the Byzantine generals problem, with a practical solution first provided by [9]. More recent developments in BFT protocols include [8], [6], [25], [28], etc. See [27] for a summary.[16] Specific to Ethereum's PoS blockchain, see [26]. Lastly, our empirical results also relate to forensic studies of blockchains.[17]

ACKNOWLEDGMENTS

We thank Ian Gao for excellent research assistance as well as seminar participants at Tsinghua University and Luohan Academy for helpful comments. We thank helpful comments from the Ethereum community, including Paul Hauner, Hudson Jameson, Raul Jordan, Stefan Kobrc, and Barnabé Monnot. Zhiguo He and Jiasun Li are grateful for research grants from the Paris-Dauphine Partnership Foundation. Zhiguo He acknowledges financial support from the John E. Jeuck Endowment at the University of Chicago Booth School of Business.

REFERENCES

[1] Arash Aloosh and Jiasun Li. 2019. Direct evidence of bitcoin wash trading. *Available at SSRN 3362153* (2019).
[2] Dan Amiram, Evgeny Lyandres, and Daniel Rabetti. 2020. Competition and Product Quality: Fake Trading on Crypto Exchanges. *Available at SSRN 3745617* (2020).
[3] Yackolley Amoussou-Guenou, Bruno Biais, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. 2020. Committee-based Blockchains as Games Between Opportunistic players and Adversaries. (2020).
[4] Alon Benhaim, Brett Hemenway Falk, and Gerry Tsoukalas. 2021. Scaling Blockchains: Can Elected Committees Help? *Available at SSRN 3914471* (2021).
[5] Dan Boneh, Ben Lynn, and Hovav Shacham. 2004. Short signatures from the Weil pairing. *Journal of cryptology* 17, 4 (2004), 297–319.
[6] Ethan Buchman. 2016. *Tendermint: Byzantine fault tolerance in the age of blockchains.* Ph. D. Dissertation.
[7] Vitalik Buterin. 2014. Proof of stake: how I learned to love weak subjectivity. *Ethereum blog* (2014).
[8] Vitalik Buterin and Virgil Griffith. 2017. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437* (2017).
[9] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine fault tolerance. *Proceedings of the third symposium on Operating systems design and implementation* (1999), 173–186.
[10] Panagiotis Chatzigiannis, Foteini Baldimtsi, Igor Griva, and Jiasun Li. 2022. Diversification across mining pools: Optimal mining strategies under pow. *Journal of Cybersecurity* 8, 1 (2022), tyab027.
[11] Lin William Cong, Zhiguo He, and Jiasun Li. 2021. Decentralized mining in centralized pools. *The Review of Financial Studies* 34, 3 (2021), 1191–1235.
[12] Lin William Cong, Xi Li, Ke Tang, and Yang Yang. 2020. Crypto Wash Trading. *working paper* (2020).
[13] Neil Gandal, JT Hamrick, Tyler Moore, and Tali Oberman. 2017. Price Manipulation in the Bitcoin Ecosystem. (2017).

[16]For a sample of studies more broadly related to the economic analysis of blockchain, see [21], [11], [23], [10], [22], [15], and [19], etc.
[17]For example, [14] relate the 2017 bitcoin bubble to Tether issuance from a single large bitcoin address; [20] explore the extent of illicit transactions on Ethereum; [13] relate the 2013 Bitcoin bubble to price manipulation on the now defunct Mt.Gox Bitcoin exchange, while [1] point to direct evidence of volume-inflating wash trading Mt.Gox. [12] and [2] develop techniques to statistically infer wash trading, while [24] provide direct evidence of pump-and-dump schemes in the cryptocurrency market using communication records on Telegram.

[14] John M Griffin and Amin Shams. 2020. Is Bitcoin really untethered? *The Journal of Finance* 75, 4 (2020), 1913–1964.
[15] Yang Guo, Jiasun Li, Mei Luo, and Yintian Wang. 2022. Illiquid bitcoin options. *Available at SSRN 4149934* (2022).
[16] Hanna Halaburda, Zhiguo He, and Jiasun Li. 2021. *An Economic Model of Consensus on Distributed Ledgers.* Technical Report. National Bureau of Economic Research.
[17] Zhiguo He, Jiasun Li, and Zhengxun Wu. 2023. Don't Trust, Verify: The Case of Slashing from an Ethereum Explorer. *Available at SSRN 4344299* (2023).
[18] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4, 3 (1982), 382–401.
[19] Jiasun Li. 2021. DeFi as an Information Aggregator. In *Financial Cryptography and Data Security. FC 2021 International Workshops. Lecture Notes in Computer Science*, Vol. 12676. Springer, 171–176.
[20] Jiasun Li, Foteini Baldimtsi, Joao P Brandao, Maurice Kugler, Rafeh Hulays, Eric Showers, Zain Ali, and Joseph Chang. 2021. Measuring Illicit Activity in DeFi: The Case of Ethereum. In *Financial Cryptography and Data Security. FC 2021 International Workshops. Lecture Notes in Computer Science*, Vol. 12676. Springer, 197–203.
[21] Jiasun Li and William Mann. 2018. Digital tokens and platform building. (2018).
[22] Jiasun Li and William Mann. 2021. Initial coin offerings: Current research and future directions. *The Palgrave Handbook of Technological Finance* (2021), 369–393.
[23] Jiasun Li and Guanxi Yi. 2019. Toward a factor structure in crypto asset returns. *The Journal of Alternative Investments* 21, 4 (2019), 56–66.
[24] Tao Li, Donghwa Shin, and Baolian Wang. 2019. Cryptocurrency pump-and-dump schemes. *Available at SSRN 3267041* (2019).
[25] Rafael Pass and Elaine Shi. 2018. Thunderella: Blockchains with optimistic instant confirmation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 3–33.
[26] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse. 2022. Three attacks on proof-of-stake ethereum. In *Financial Cryptography and Data Security: 26th International Conference, FC 2022, Grenada, May 2–6, 2022, Revised Selected Papers.* Springer, 560–576.
[27] Elaine Shi. 2020. *Foundations of Distributed Consensus and Blockchains.* Book manuscript, Available at https://www.distributedconsensus.net.
[28] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2018. HotStuff: BFT consensus in the lens of blockchain. *arXiv preprint arXiv:1803.05069* (2018).

A RECORDED SLASHING INCIDENTS

We present a sample of slashed violations, separating proposer and attester violations:[18] Due to page limits, we list 15 incidents per category. A full list is available in the appendix of a companion working paper [17] (with SSRN link here).

(1) proposer violations: For each slashing incident, we list the slashed proposer's ID, the slashing proposer's ID, the locations of the slashing message (i.e. the block number at which the slashing message is included in the Beacon chain), and the slashable proposal's location (for which block the slashable proposal was made).

(2) attester violations: For each slashing incident, we list the slashed attester's ID, the slashing proposer's ID, the locations of the slashing message (i.e. the block number at which the slashing message is included in the Beacon chain), and the slashable vote's content (for which block the slashable vote was cast).

B UNDETECTED SLASHABLE VIOLATIONS

According to the explorer's records, all double proposals within the first 1.75 million slots were successfully slashed. The same is not true for attestations, and there are both unslashed double votes and unslashed surround votes. We again present 15 incidents for each category, with full lists linked after anonymous reviews.

Table 1: Examples of Slashed Proposer Violations

slashed proposer	slashing proposer	slashing message location	proposal location
20075	11313	6669	6668
18177	21106	22374	22373
25645	11117	40772	40771
38069	24876	138164	138163
38089	10010	138731	138730
38130	4156	140313	140312
38129	33452	140559	140558
38065	33153	140811	140810
38128	14011	140845	140844
38117	31339	140895	140894
38114	23929	141174	141173
45871	32686	248186	248185
40892	55778	343133	343132
63338	35018	476904	476903
169440	103269	1510279	1510278

Table 2: Examples of Slashed Attester Violations

slashed proposer	slashing proposer	slashing message location	proposal location
4259	19030	17112	17090
4100	19030	17112	17090
21574	19030	17112	17090
4110	10689	17206	17078
13869	10689	17206	17064
4102	10055	17188	17082
4086	10055	17188	17084
4390	11111	17184	17072
4451	11398	17227	17073
18249	11398	17227	17073
7635	17942	43920	43917
1644	21844	102389	102388
23241	15703	118136	118135
38061	10063	138194	138163
38105	10063	138194	138163

(1) "unslashed" double votes. For each violation, we list the committing attester's ID, the locations of conflicting votes (i.e. the block number at which each vote is included in the Beacon chain), and the vote content (for which block the conflicting votes were cast).

(2) "unslashed" surround votes. For each vote within a surround vote violation, we list the committing attester's ID, the location of the vote (i.e. the block number at which the vote is included in the Beacon chain), and the vote content (for which block, as well as for which source and target epochs the vote were cast).

[18] After the initial circulation of our paper, beaconcha.in began to return 502 bad gateway for attestation information for slots 1 - 2500250 in early December 2022, so some links may no long work (however, it went back online as of December 9).

Table 3: Examples of Unslashed Double Votes

attester	vote locations	vote content
237	[1041100, 1041108]	1041099
487	[1041103, 1041108]	1041102
2787	[1041107, 1041108]	1041106
3167	[1041104, 1041108]	1041103
3644	[1267818, 1267822]	1267817
4021	[1054792, 1054796]	1054791
4034	[1267816, 1267822]	1267815
4098	[1041104, 1041108]	1041103
4219	[608067, 608085]	608065
4220	[1041104, 1041108]	1041103
4826	[1041100, 1041108]	1041099
4993	[918915, 918922]	918914
5129	[1054791, 1054791, 1054796]	1054790
5194	[1054788, 1054788, 1054796]	1054787
5228	[1041098, 1041108]	1041097

Table 4: Examples of Unslashed Surround Votes

attester	vote location	vote content		
		block	source_epoch	target_epoch
4155	918882	918881	28714	28715
4155	918922	918914	28713	28716
4219	608052	608051	19000	19001
4219	608059	608051	19000	19001
4219	608067	608065	18999	19002
4993	918901	918900	28714	28715
4993	918922	918914	28713	28716
6666	988955	988954	30903	30904
6666	988965	988961	30902	30905
7412	918883	918882	28714	28715
7412	918922	918916	28713	28716
9018	988931	988930	30903	30904
9018	988965	988960	30902	30905
10740	918900	918899	28714	28715
10740	918922	918917	28713	28716