Total Eclipse of the Enclave: Detecting Eclipse Attacks From Inside TEEs

Haofan Zheng*
UC Santa Cruz
hzheng6@ucsc.edu

Tuan Tran*
UC Santa Cruz
atran18@ucsc.edu

Owen Arden
UC Santa Cruz
owen@soe.ucsc.edu

***Abstract*—Enclave applications that rely on blockchains for integrity and availability are vulnerable to eclipse attacks. In this paper, we present an approach for reliably detecting extended eclipse attacks, even when the adversary controls all network connectivity. By monitoring changes to the *difficulty* parameter in Proof-of-Work (PoW) protocols, our algorithm detects suppression of new blocks, as well as difficulty-lowering attacks that attempt to force an enclave client onto a malicious fork mined solely by an attacker. We present analysis that attackers have negligible probability of evading our block monitoring algorithm, and demonstrate its robustness to most historical fluctuations in difficulty on the Ethereum blockchain, resulting in a very low false-positive rate.**

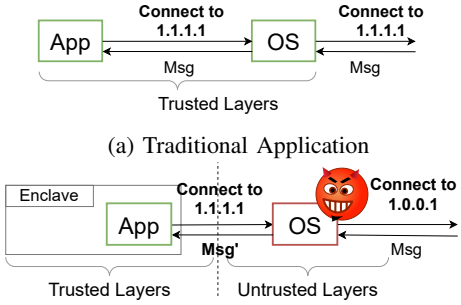
I. Introduction

One of the benefits permissionless blockchains offer is *censorship resistance*: adversaries cannot suppress transactions from clients to the blockchain, nor can they suppress the publication of new transactions. In most Proof-of-Work (PoW) blockchains, this guarantee holds as long as adversaries do not control a majority of the computational power in the network. An additional, but often overlooked, requirement is that clients must be able to create reliable network connections with at least some honest blockchain nodes.

Heilman et al. [1] discuss how an adversary can monopolize these connections to perform *eclipse attacks* that suppress communication between targeted clients and blockchain nodes. Most defenses against eclipse attacks involve diversifying connections to peers in the hopes that at least one connection is not controlled by the attacker.

Enclave applications run in a special restricted mode where even privileged code (such as the OS) cannot inspect or modify the enclave's code or memory, making it possible to run confidential applications on untrusted hosts that produce high-integrity outputs. The reliance on the OS for inputs and outputs also means that enclave mechanisms alone cannot provide any availability guarantees. Furthermore, since the enclave depends exclusively on the (potentially malicious) host for all network communication, mitigating eclipse attacks by creating more connections is not possible.

Even though an attacker can only directly access coins it possesses the keys for, eclipse attacks can have serious consequences. For example, an attacker may try to suppress incoming blocks containing time-sensitive transactions (e.g., revocation notifications, payment channel closures, etc.) published to the blockchain. Alternatively, the attacker may try to lower the difficulty to a level where it can generate malicious blocks within the expected block time. Using double spend attacks, the attacker can try to extract value from the enclave without spending coins on the real chain. In both cases, shutting down the enclave prevents it from operating on incomplete or malicious information.



(a) Traditional Application

(b) Enclave Application

Fig. 1: Threat Model Comparison

Figure 1 compares a threat model of a traditional application with an enclave application, with both applications wanting to communicate with the remote node on IP address `1.1.1.1`. The OS layer establishes the connection and forwards the message; however, an untrustworthy OS could intercept the message and manipulate or suppress it in the enclave application threat model. Cryptographic protocols such as TLS can protect the message content, but cannot guarantee delivery.

The main contribution of this paper is a novel algorithm that can reliably detect eclipse attacks both inside and outside of the enclave environment. Our algorithm continuously monitors changes in block difficulty to detect an eclipse attack in the enclave. We evaluate this algorithm by running it against all historical blocks in Ethereum.

II. Related Works

A. Eclipse Attacks in General

Castro et al. [2] introduced eclipse attacks on structured peer-to-peer overlay networks. Heilman et al. [1] then realized

* Authors contributed equally.

978-0-7381-1420-0/21/$31.00 ©2021 IEEE

this style of attack in Bitcoin, wherein an attacker can seize control of a sufficient number of IP addresses of peers of the victim node. Upon a successful eclipse attack, several other attacks can then be performed, such as selfish mining and double-spending transactions. The authors suggest diversifying the peer connection to mitigate this attack, but this relies on at least some of those connections to be outside the control of the attacker. In an enclave application, all connections may be intercepted by the host.

B. Eclipse Attack with Enclaves

Ekiden [3] addresses eclipse attacks in blockchain applications built with enclaves using a Proof-of-Publication (PoP) protocol. This approach requires the host to publish an enclave-generated nonce to the blockchain within a limited time. A successful PoP shows that the block containing the nonce is fresh and valid. However, relying solely on PoP for ensuring timely block delivery is too expensive since each PoP requires a blockchain transaction fee.

Tesseract [4] defines a time threshold by which the enclave must receive a new block from the Bitcoin network; otherwise, it waits for n more confirmation blocks to ensure the block was not mined by the attacker during an eclipse attack. Tesseract does not monitor the block difficulty, so an attacker could take advantage of drops in the difficulty to continue mining blocks in a malicious chain during the confirmation period. Additionally, blockchains with faster block arrival rates (such as Ethereum) typically have higher variance in arrival times, so choosing a time threshold that minimizes both false positives and negatives may be more difficult.

III. System Overview

A. Threat Model

We assume the attacker *physically* and *effectively* controls no more than some percentage q of the total network hash power, Q. The biggest Ethereum mining pool at the time of writing has around $26\%Q$ [5]. To make our exposition more concrete, we choose $q = 30\%$ as an upper bound, but this percentage is configurable. A lower percentage may be sufficient for many applications; a higher percentage may be required for smaller Ethereum-based blockchains (e.g., Ethereum Classic). To simplify our model, we assume q is the attacker's effective hash rate during the eclipse attack. Withholding their hash power could potentially lower the difficulty of the network overall (rather than just the enclave), making it easier for the attacker to mine blocks on a malicious fork, but new miners joining to take advantage of the lower difficulty could undermine the effectiveness of this technique.

The host executing the enclave may be malicious, and controls all other software layers including the network stack. Thus, a malicious OS could hijack the network connection or suppress messages.

B. Difficulty Monitoring

Our primary mechanism for detecting eclipse attacks is based on monitoring the current difficulty level, θ, set by the PoW protocol. During normal operation, θ is typically around $13 \cdot Q$, which targets an expected arrival time of 13 seconds.

One subtle point is that this arrival time accounts both for the hash power spent on mining the next primary block as well as *uncle blocks*, which are valid blocks that arrived later than the previous primary block. Based on Ethereum's Difficulty Adjustment Algorithm (DAA) formula [6], [7], primary blocks that have uncle blocks may arrive within 18 seconds, instead of 9 seconds, to avoid lowering the difficulty value. However, splitting the hash power in half between uncle blocks and primary blocks to get twice the time does not result in strategic advantage. Therefore, we assume the attacker spends its hash power generating blocks on a single malicious fork and produces no uncle blocks.

During an eclipse attack, the attacker must generate blocks every 9 seconds to avoid a difficulty drop.[1] Initially, this is difficult for the attacker, but each time the attacker fails, Ethereum's DAA lowers the difficulty by a maximum of 5%. For an attacker with $30\% \cdot Q$ hash power, θ will eventually approach $2.7 \cdot Q$. At this point, the attacker can expect to mine blocks fast enough to maintain the difficulty level indefinitely. We define the difficulty at which the attacker can maintain a block arrival rate of 9 seconds as $\theta_{min} = 9 \cdot (q \cdot Q)$.

While θ_{min} is the boundary at which the attacker has complete control over the chain, applications may wish to set a lower difficulty drop as a cutoff point. The lower the θ_{min}, the more confirmations are necessary to have high assurance that the most recent blocks were not mined by the attacker. Setting a more conservative bound on the drop will require fewer confirmations, at the price of potential false positives: blocks that trigger our algorithm while the system is not under an eclipse attack. We evaluate this tradeoff in detail in Section V. Once the difficulty nears θ_{min}, however, no number of confirmations can eliminate the possibility of an attack.

Ethereum's DAA adjusts the difficulty value from block to block,[2] so we must monitor difficulty drops over time to prevent attackers from incrementally dropping the difficulty value. The difficulty level adjusts naturally as miners leave or join the network, so we need to establish a stable difficulty level for a given time span, and adjust this level over time.

Beginning at the genesis block, the chain is split into fixed checkpoint ranges of N_c blocks. We set the current difficulty level as the median of the blocks in the previous (complete) checkpoint range. N_c should be large enough to make it impossible for an attacker to maintain a θ higher than θ_{min} after N_c blocks but small enough to avoid false positives.

Current enclave platforms do not provide a trusted world clock time, but some (e.g., Intel SGX [8], [9]) provide trusted elapsed time, which is sufficient for our purposes. We only require that the block time, calculated based on the block timestamps, is reasonably close to the elapsed time. Even if the timestamp is malicious, Ethereum bounds the drift of timestamps, so the influence on difficulty values is small.

[1] Given Ethereum's DAA [7], if the current block arrived within 9 seconds of its parent, then the next block's difficulty will not decrease

[2] In contrast, Bitcoin adjusts the difficulty every 2016 blocks.

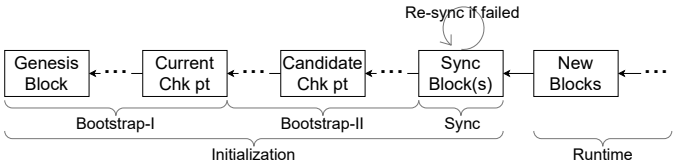


Fig. 2: Difficulty Monitor initialization

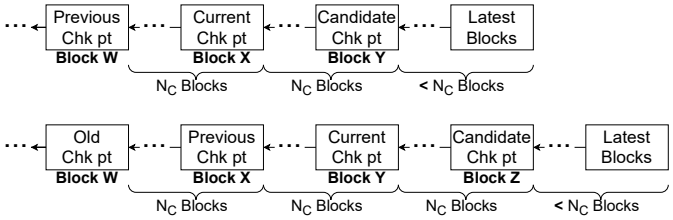


Fig. 3: Checkpoint Update

C. Accepting transactions in the enclave

When clients and enclaves communicate, they compare the hash of the most recent checkpoint to ensure both parties are following the same fork of the blockchain. Applications may also use N_c as an upper bound on the number of confirmations needed to consider a block finalized. This upper bound is conservative since our monitor prevents *sustainable* forks of the blockchain during an eclipse attack.

Within N_c blocks, attackers may have a non-negligible chance of mining a sequence of malicious blocks, but the chance of sustaining that sequence for N_c blocks without detection is negligible. It is possible that a smaller number of confirmations would be sufficient to guard against temporary forks, but we leave this question for future work.

IV. Difficulty Monitor

A. Initialization

Our difficulty monitor has three initialization phases, shown in Figure 2, which ensure the legitimacy of the existing blocks.

a) Bootstrap-I Phase: The monitor loads and processes all blocks starting from the *genesis block* to the latest *checkpoint block*. Blocks associated with well-known false positives (such as hard forks), are classified as *confirmed benign blocks* and included when determining the difficulty of the checkpoint range they appear in.

b) Bootstrap-II Phase: Next, the monitor loads all blocks published after the checkpoint block. Difficulty changes will be checked with the same process used at runtime, but without restrictions on the elapsed time.

c) Sync Phase: Finally, the monitor synchronizes with the blockchain by publishing a sync message, similar to the PoP used by Ekiden. Unlike Ekiden, we only require a sync message at initialization. This prevents attackers from feeding a pre-mined malicious fork to the monitor. Thus, it ensures the freshness of both the *Sync block* and following blocks. Sync messages contain a nonce generated by the monitor, and a block containing the sync message with the matching nonce must be received by the enclave within the expected block time, which is around 13 seconds in Ethereum. If the sync message fails to be published within the restricted time limit, the host will need to perform a re-sync by publishing a new nonce until a sync block is accepted.

B. Runtime Difficulty Monitoring

New blocks received by the monitor are first validated as normal. Then the monitor determines the nominal block time based on the block timestamps. If the difference between the trusted elapsed time and the nominal block time is greater than 15 seconds, the enclave rejects the block to prevent the attacker from either lowering the difficulty in primary chain by manipulating timestamps, or maintaining the difficulty in malicious chain with pre-dated timestamps. Temporary forks in the chain require each branch to be monitored simultaneously. Our documentation discusses further details [10].

Once the new block has passed the initial validation, the monitor compares θ_{new} presented in the new block with the θ_{min} calculated based on the formulas shown below.

$$\theta_{\text{ref},i} = \text{Median}([\theta_{(i-1)\times N_c}, \cdots \theta_{i\times N_c}]) \quad (1)$$

$$\theta_{\text{min},i} = \theta_{\text{ref},i} \times (1-p) \quad (2)$$

The monitor first gets the median of difficulty values among the blocks in the previous checkpoint window, which have been checked and are finalized.[3] By using the median as a reference value, we minimize false positives and false negatives caused by outliers. Next, the monitor uses θ_{ref} to calculate the θ_{min}, where p is the percentage drop allowed (set by the application). The new block is accepted if θ_{new} is higher than θ_{min}.

The attacker may choose to provide no blocks at all, preventing the monitor from determining how much the difficulty value has dropped. To address this issue, the monitor will periodically estimate the current difficulty value, θ_{est}, by using the trusted elapsed time, and compares θ_{est} with θ_{min}. Additionally, the maximum allowable block time in Ethereum's DAA (around 900 seconds) prevents the difficulty value from dropping more than 5% in each block. This value also serves as the maximum wait time ($T_{\Delta\text{Max}}$), and blocks arriving later than this time will trigger a shutdown.

C. Checkpoint Update

The monitor will automatically update the checkpoint every N_c blocks. As shown in Figure 3, the candidate checkpoint is the block that has less than N_c blocks ahead and is too new to be considered finalized. Once there are N_c blocks found ahead of the candidate checkpoint, the candidate will become the new checkpoint, and the N_cth block ahead of the new checkpoint will become the next candidate checkpoint.

[3]Finalized blocks are blocks having large number of successors and negligible probability of being reorganized.

Test Case Index	θ_{min}	N_c	Num of False Detections	False Negative Rate
1	$50\% \cdot \theta$	1270	2	$< 2^{-128}$
2	$60\% \cdot \theta$	880	2	$< 2^{-128}$
3	$70\% \cdot \theta$	620	3	$< 2^{-128}$
4	$80\% \cdot \theta$	430	5	$< 2^{-128}$
5	$80\% \cdot \theta$	610	7	$< 2^{-256}$
6	$90\% \cdot \theta$	275	17	$< 2^{-128}$
7	$90\% \cdot \theta$	420	23	$< 2^{-256}$

TABLE I: Evaluating the difficulty detection algorithm under various checkpoints sizes against the entire Ethereum database.

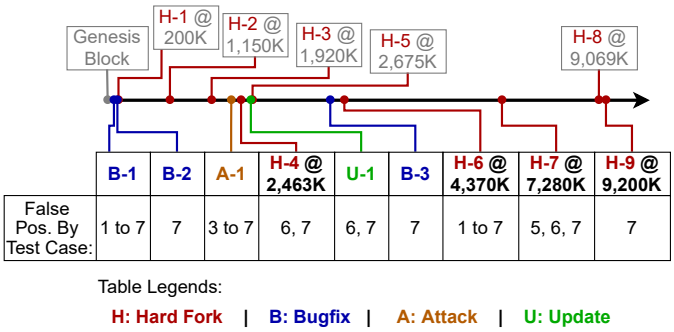


Fig. 4: Timeline of major events that cause false detections. The hardforks are [13], [14], [15], [16], [17], [18], [19], [20], and [21], respectively. The bugs are [22], [23], and [24], respectively. The attack is [25]. The major API update is [26].

V. Evaluation Against Historical Blocks

We implemented our methodology on top of Geth [11], the official Go implementation of the Ethereum client, since it is the most mature and widely used client. To evaluate the false positive rate under different parameters, we ran the monitor's algorithm against 10,900,373 historical blocks in Ethereum. We released our custom Geth database connector, experiment code and more technical details at [12], [10].

A. False Negative Rate

The false negative rate, as shown in Table I, represents the possibility of an attacker maintaining the difficulty value at the level higher than θ_{min} within N_c blocks. It is calculated based on the exponential distribution [4] established with attacker's hash power and network difficulty [10].

All combinations of N_c and θ_{min} that we have examined result in extremely low false negative rates: equivalent to brute-forcing a 128 or 256 bit key.

B. Difficulty Monitoring and Checkpoint Sizes

Table I also shows false positives that triggered our difficulty monitoring algorithm under various θ_{min} and N_c. For all configurations, the algorithm resulted in multiple false detections. We have investigated all of these false positives and found that, as shown in Figure 4, Most of these events correspond to hard forks. During hard forks, network hash power fluctuates significantly due to miners upgrading their nodes. There are also instances where the detections were caused by DoS attacks [25] addressed with software upgrades. In both cases, enclave components would need to be shutdown and replaced by one built with the latest protocol anyway, so the monitor's behavior is not as concerning.

When θ_{min} is set to a high value, such as $90\%\theta$, the monitor becomes more sensitive to smaller events, such as bugfixes and API updates. In the 7th test case, there is one false positive caused by a 10% difficulty drop that we could not tie protocol-related events but could be due to miners leaving for Ethereum Classic, which was increasing in value at the time.

When θ_{min} is set to a higher value, the monitor needs a smaller checkpoint size (i.e., N_c) to confirm that the recent N_c blocks have a negligible chance to be mined by the attacker. As N_c increases, so does the number of false positives since the checkpoint captures a larger time interval surrounding hard forks. Thus there is a tradeoff between N_c and θ_{min}. The application may set a larger θ_{min} or N_c to make malicious forks more difficult, but may incur more false detections since difficulty drops accumulate over a longer period.

Therefore, setting a reasonable θ_{min} is a key to avoid false positives, and then configuring a reasonable N_c to reach the desired false negative rate level. For instance, with θ_{min} set to $80\%\theta$, all those false positives can be avoided if the enclave operator times the upgrade correctly; a lower N_c with a $70\%\theta$ or lower will also have the same sensitivity to difficulty drops.

C. Block times

Currently, the average block time in Ethereum is around 13 seconds. To find a proper value for the $T_{\Delta\text{Max}}$, which is used to prevent attackers from remaining silent forever, we record the top 100 block times from historical blocks in Ethereum. Our results show that while the majority of the blocks do arrive within the expected time, all of the top 100 blocks took over 200s to arrive, with 91 blocks taking less than 300 seconds, 7 blocks less than 400 seconds, 1 block less than 500 seconds, and one outlier that took 13013 seconds. Upon closer inspection, almost all of these blocks were mined very close to the Byzantium hardfork [18]. The reason that outlier block took almost four days to be mined is because of a consensus issue (which is B-1 in Figure 4) in Geth that caused miners to have to switch to alternative clients[22]. Given these results, choosing the maximum time given in Ethereum's DAA (around 900s) should give no false positives during normal operation; for applications that are very time-sensitive, a reasonable $T_{\Delta\text{Max}}$ would be 400 seconds.

VI. Conclusion

Blockchain applications built with an enclave are vulnerable to eclipse attacks. In this paper, we present a novel solution to this problem using a PoW difficulty monitor that is inexpensive, immune to long-range attacks, and provides the maximum reaction time. Based on the evaluation against historical blocks, choices for θ_{min} and N_c with negligible false negative rates are unlikely to trigger false positives.

REFERENCES

[1] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Aug. 2015, pp. 129–144. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman

[2] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, p. 299–314, Dec. 2003. [Online]. Available: https://doi.org/10.1145/844128.844156

[3] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 185–200. [Online]. Available: https://doi.org/10.1109/EuroSP.2019.00023

[4] I. Bentov, Y. Ji, F. Zhang, L. Breidenbach, P. Daian, and A. Juels, "Tesseract: Real-time cryptocurrency exchange using trusted hardware," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1521–1538. [Online]. Available: https://doi.org/10.1145/3319535.3363221

[5] PoolWatch.io, "Best ethereum mining pools for 2020," 2020. [Online]. Available: https://www.poolwatch.io/coin/ethereum

[6] V. Buterin, "A next generation smart contract & decentralized application platform," Ethereum Foundation, Tech. Rep., 2013, ethereum White Paper.

[7] go-ethereum, "consensus.go," https://github.com/ethereum/go-ethereum/blob/4b2ff1457ac28fb2894485194e0e344e84c2bcd7/consensus/ethash/consensus.go, 2020. [Online]. Available: https://github.com/ethereum/go-ethereum/blob/4b2ff1457ac28fb2894485194e0e344e84c2bcd7/consensus/ethash/consensus.go

[8] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '13. ACM, 2013, pp. 10:1–10:1. [Online]. Available: http://doi.acm.org/10.1145/2487726.2488368

[9] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proceedings of the 2Nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '13. ACM, 2013, pp. 10:1–10:1. [Online]. Available: http://doi.acm.org/10.1145/2487726.2488368

[10] H. Zheng and T. Tran, "Decentdiffmonitorexpr," 2020. [Online]. Available: https://github.com/zhenghaven/DecentDiffMonitorExpr

[11] Ethereum Team, "Go ethereum: Official go implementation of the ethereum protocol," https://geth.ethereum.org/. [Online]. Available: https://geth.ethereum.org/

[12] H. Zheng and T. Tran, "Gethdbreader," 2020. [Online]. Available: https://github.com/zhenghaven/GethDBReader

[13] S. Tual, "Ethereum protocol update 1," https://blog.ethereum.org/2015/08/04/ethereum-protocol-update-1/, 2015. [Online]. Available: https://blog.ethereum.org/2015/08/04/ethereum-protocol-update-1/

[14] J. Wilcke, "Homestead release," https://blog.ethereum.org/2016/02/29/homestead-release/, 2016. [Online]. Available: https://blog.ethereum.org/2016/02/29/homestead-release/

[15] V. Buterin, "Hard fork completed," https://blog.ethereum.org/2016/07/20/hard-fork-completed/, 2016. [Online]. Available: https://blog.ethereum.org/2016/07/20/hard-fork-completed/

[16] H. Jameson, "Faq: Upcoming ethereum hard fork," https://blog.ethereum.org/2016/10/18/faq-upcoming-ethereum-hard-fork/, 2016. [Online]. Available: https://blog.ethereum.org/2016/10/18/faq-upcoming-ethereum-hard-fork/

[17] ——, "Hard fork no. 4: Spurious dragon," https://blog.ethereum.org/2016/11/18/hard-fork-no-4-spurious-dragon/, 2016. [Online]. Available: https://blog.ethereum.org/2016/11/18/hard-fork-no-4-spurious-dragon/

[18] Ethereum Team, "Byzantium hf announcement," https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/, 2017. [Online]. Available: https://blog.ethereum.org/2017/10/12/byzantium-hf-announcement/

[19] H. Jameson, "Ethereum constantinople/st. petersburg upgrade announcement," https://blog.ethereum.org/2019/02/22/ethereum-constantinople-st-petersburg-upgrade-announcement/, 2019. [Online]. Available: https://blog.ethereum.org/2019/02/22/ethereum-constantinople-st-petersburg-upgrade-announcement/

[20] ——, "Ethereum istanbul upgrade announcement," https://blog.ethereum.org/2019/11/20/ethereum-istanbul-upgrade-announcement/, 2019. [Online]. Available: https://blog.ethereum.org/2019/11/20/ethereum-istanbul-upgrade-announcement/

[21] ——, "Ethereum muir glacier upgrade announcement," https://blog.ethereum.org/2019/12/23/ethereum-muir-glacier-upgrade-announcement/, 2019. [Online]. Available: https://blog.ethereum.org/2019/12/23/ethereum-muir-glacier-upgrade-announcement/

[22] J. Steiner, "Security alert [consensus issue]," https://blog.ethereum.org/2015/08/20/security-alert-consensus-issue/, 2015. [Online]. Available: https://blog.ethereum.org/2015/08/20/security-alert-consensus-issue/

[23] ——, "Security alert – [implementation bug in go clients causing increase in difficulty – fixed – miners check and update go clients]," https://blog.ethereum.org/2015/09/03/security-alert-implementation-bug-in-go-clients-causing-increase-in-difficulty-fixed-miners-check-and-update-go-clients-if-necessary/, 2015. [Online]. Available: https://blog.ethereum.org/2015/09/03/security-alert-implementation-bug-in-go-clients-causing-increase-in-difficulty-fixed-miners-check-and-update-go-clients-if-necessary/

[24] Ethereum Team, "Release version 0.4.14 ethereum/solidity," https://github.com/ethereum/solidity/releases/tag/v0.4.14, 2017. [Online]. Available: https://github.com/ethereum/solidity/releases/tag/v0.4.14

[25] J. Wilcke, "The ethereum network is currently undergoing a dos attack," https://blog.ethereum.org/2016/09/22/ethereum-network-currently-undergoing-dos-attack/, 2016. [Online]. Available: https://blog.ethereum.org/2016/09/22/ethereum-network-currently-undergoing-dos-attack/

[26] Ethereum Team, "Release let there be light (v1.5.0) ethereum/go-ethereum," https://github.com/ethereum/go-ethereum/releases/tag/v1.5.0, 2016. [Online]. Available: https://github.com/ethereum/go-ethereum/releases/tag/v1.5.0